FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Survey and Taxonomy of Autonomic Large-scale Computing

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Gabriel Kittel
Matrikelnummer 9027972

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung
Betreuer: Univ.-Prof. Dr. Schahram Dustdar
Mitwirkung: Univ.-Ass. Mag. Dr. Ivona Brandic

Wien, 29.09.2010

(Unterschrift Verfasser) (Unterschrift Betreuer)

Gabriel Kittel
Eduard-Sueß-Gasse 8/5
1150 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 29.09.2010

_____
(Unterschrift Verfasser)

Contents

List of Figures

List of Tables

Kurzfassung

Verschiedene Ansätze im Bereich der verteilten Systeme hatten bereits das Ziel, Rechnerkapazität als eine öffentliche Versorgungsleistung vergleichbar mit Wasser, Gas, Elektrizität und Telefonie anzubieten. Cloud Computing ist der aktuellste dieser Ansätze, bei dem virtuelle Rechnerinfrastruktur, Softwareentwicklungsplattformen und Anwendungen nach Bedarf über das Internet bereitgestellt werden. Anbieter von Cloud Computing wie Amazon und Google nutzen Virtualisierungstechniken in ihren bestehenden Großrechenzentren, um Rechnerdienste im Rahmen eines Geschäftsmodells bereitzustellen, das auf ressourcenbezogenem Preismanagement basiert. Damit Unternehmen betriebsnotwendige Rechnerleistungen Cloud-Anbietern anvertrauen, muss jedoch ein zuverlässiger Zugriff auf diese ausgelagerten Kapazitäten gewährleistet sein, der die Zuverlässigkeit der bestehenden öffentlichen Versorgungseinrichtungen erreicht oder sogar übertrifft. Um dieser Anforderung zu begegnen, versprechen Anbieter von Cloud Computing eine Verfügbarkeit von 99,9 % oder höher. Während die Verfügbarkeit von Diensten mit Hilfe etablierter Technik wie Hochverfügbarkeitsclustern und Proxy-Rechnern zur Serverlastverteilung erreicht werden kann, setzt die Gewährleistung anderer wichtiger Dienstgüteparameter wie Antwortzeit oder Datentransferrate die laufende Anpassung von Diensten und Rechnerinfrastruktur an sich ständig ändernde Rahmenbedingungen wie Verfügbarkeit von Systemressourcen oder Anzahl von Anfragen voraus.

Autonomic Computing ist ein Ansatz, um der aus der Dynamik von Änderungen der Ressourcenverfügbarkeit und Systemlast resultierenden Komplexität der Systemadministration zu begegnen, indem Rechnersysteme entwickelt werden, die sich auf Grundlage vorgegebener Richtlinien selbsttätig an sich ändernde Rahmenbedingungen anpassen. Die Möglichkeiten dieser Selbstanpassung werden üblicherweise mit den Begriffen Selbstkonfiguration, Selbstoptimierung, Selbstheilung und Selbstschutz beschrieben. Autonomic Computing wurde bereits im Bereich des Grid Computing angewandt, das den gemeinsamen, organisationsübergreifenden Zugriff auf heterogene Systemressourcen über standardisierte Schnittstellen ermöglicht. Grid Computing wird vorwiegend im wissenschaftlichen Bereich eingesetzt, um den verteilten Zugriff auf Höchstleistungsrechner im Rahmen von organisationsübergreifenden, rechenleistungsintensiven Forschungsprojekten zu ermöglichen. Da Grid Computing üblicherweise über lang laufende Aufträge (Jobs) im Stapelbetrieb genutzt wird, die oftmals die Übertragung großer Datenbestände einschließen, ist deren zuverlässige Abarbeitung trotz der Komplexität der organisationsübergreifenden Systemadministration in besonderem Maße notwendig.

Das Projekt *Foundations of Self-managing ICT Infrastructures (FoSII)* an der TU Wien beabsichtigt, die Unterstützung von Selbstmanagement in bestehenden service-

orientierten Architekturen, die den aktuellen Implementierungen von Cloud-Computing und Grid-Computing zugrundeliegen, zu verbessern. Ein strukturierter Überblick über bestehende Projekte, die Autonomic Computing in Rechnersystemen großen Maßstabs wie Grids und Clouds einführen, und eine Taxonomie, die Klassifikationskriterien für solche Systeme anbietet, würden es ermöglichen, den aktuellen Stand der Forschung aufzubereiten, indem Kriterien vorgeschlagen werden, aufgrund derer Anwendungsbereiche, Teilprobleme und Lösungsansätze in diesem Bereich identifiziert werden können. Ein solcher Überblick mit Taxonomie existiert derzeit jedoch nicht.

Ziel dieser Diplomarbeit ist es, den aktuellen Stand der Technik im Bereich des Selbstmanagements mittels einer Taxonomie des Autonomic Large-Scale Computing systematisch zu untersuchen. Die Arbeit präsentiert einen Überblick über Projekte und theoretische Arbeiten in diesem Bereich und schlägt eine Taxonomie vor, um das Forschungsgebiet des Autonomic Computing in verteilten Systemen großen Maßstabs zu klassifizieren. Sie identifiziert die Funktionsbereiche Anwendungsmanagement, Datenmanagement, Entwicklung, Dienstgütemanagement und Ressourcenmanagement, in denen Selbstmanagement im Rahmen von verteilten Systemen großen Maßstabs bereits eingesetzt wurde. Weiters beschreibt sie auf Regelkreisen basierende, agentenbasierende und proxybasierende Architekturen im Bereich des Autonomic Computing.

Die Anwendung dieser Taxonomie auf die in der Übersicht enthaltenen Projekte ergibt einen Kriterienkatalog, der Forschungsschwerpunkte identifiziert, welche im autonomen Ressourcenmanagement im Bereich von Grid-Systemen und im Dienstgütemanagement im Bereich von Cloud-Systemen liegen. Einige Projekte untersuchen bestimmte Teilaspekte autonomer verteilter Systeme großen Maßstabs, wie Optimierung oder maschinelles Lernen. Überblick und Taxonomie erlauben es, den aktuellen Stand der Forschung im Bereich autonomer verteilter Systeme großen Maßstabs zu bewerten und ermöglichen dadurch weitere Fortschritte in diesem Bereich.

Abstract

In the area of distributed systems, several approaches have emerged with the objective to deliver computing power as a public utility like water, gas, electricity and telephony. Cloud computing is the latest of those approaches where virtual computing infrastructure, software development platforms and applications are provisioned on demand over the Internet. Cloud providers like Amazon or Google use virtualization technology within their existing large-scale data centers in order to deliver computing services following a business model based on resource pricing. In order to have organizations entrust mission-critical computing services to cloud providers, reliable possibility of access to those outsourced capacities needs to be ensured which meets and possibly exceeds the reliability of the existing public utilities. In order to address this requirement, cloud operators promise 99.9 % or higher availability. While service availability can be achieved using mature technology like high-availability clusters and load-balancing proxies, a prerequisite for delivering quality of service related to other attributes like response time or data transfer rate is the adaptation of services and computing infrastructure to constantly changing environmental conditions like availability of system resources or number of service requests.

Autonomic computing is a computing paradigm that promises to deliver systems adapting themselves to environmental changes by employing self-management mechanisms guided by policies as an effort to address the management complexity that arises from the dynamics of resource availability and system load in large-scale computing systems. Capabilities of autonomic systems are usually described by the properties of self-configuration, self-optimization, self-healing and self-protection. Autonomic computing has previously been introduced to grid computing, which is a large-scale distributed computing paradigm that allows sharing of heterogeneous computing resources within a virtual organization using standard interfaces. Grid computing is prevalent in the scientific domain, where high-performance computing resources hosted at a research institution are made available for organizations participating on computation-intensive research projects. Since the usual operation mode of grid computing is to submit long-running batch jobs possibly involving the transfer of large volumes of data, reliable completion of those jobs in spite of the system management complexity resulting from the involvement of multiple organizations is a crucial requirement.

The *Foundations of Self-managing ICT Infrastructures project (FoSII)* at TU Vienna intends to enhance self-management support in existing service-oriented architectures which provide the basis of current cloud computing and grid computing implementations. A survey of existing projects that introduce autonomic computing to large-scale computing systems like grids and clouds, and a taxonomy that provides classification

criteria for that research field would allow to assess the current state of research by suggesting criteria to help identify application areas, subproblems and approaches within that field. However, such a survey and taxonomy have not yet been proposed to this day.

The goal of this thesis is to systematically investigate the state of art of self-management by providing a taxonomy of autonomic large-scale computing. It presents a survey of projects and theoretical work in that field and proposes a taxonomy that classifies autonomic large-scale computing. It identifies the functional areas of application management, data management, development, quality of service management and resource management where self-management has been applied in large-scale computing projects, and the autonomic computing architectures of loop-based systems, agent-based systems and proxy-based systems.

Applying the taxonomy to the surveyed projects results in a criteria catalog showing that research focuses on autonomic resource management in grid computing, and autonomic quality of service management in cloud computing, while several projects explore specific autonomic system capabilities like optimization or machine learning in large-scale distributed computing systems. Survey and taxonomy allow to assess the current state of research in autonomic large-scale distributed systems, thus supporting further advancements in the field of autonomic large-scale distributed computing.

1 Introduction

The vision that computing power will one day be a public utility like water, gas, electricity and telephony has been perceived as early as 1969 [111]. Utility computing [153], where computing power shall be available to private and business consumers on demand as a service, requires service providers to establish a massively scalable infrastructure and mechanisms that allow the automated provisioning of resources according to standards. Cloud computing is the latest large-scale computing paradigm that promises to deliver the vision of utility computing by dynamically provisioning computing resources based on service-level agreements established through negotiation between service providers and consumers [41]. Cloud computing is related to the more established paradigms of cluster computing [149, 36] and grid computing [69], since all of these concepts provide access to high-performance computing resources. While the packaging of software into services is a well-established technology, the potential of providing computing power as a service currently is restricted due to a lack of dynamism and adaptivity in current service-oriented architectures. The *Foundations of Self-Governing ICT Infrastructure (FoSII)* project [73] at TU Vienna currently investigates this research issue.

Autonomic computing [140] is a paradigm that promises to deliver self-managed computing systems by providing mechanisms that allow self-adaptation of services in reaction to changes in environmental conditions guided by high-level policies. Currently, several publications and projects exist with the goal to build autonomic large-scale distributed computing systems. With a growing body of research, the need arises for a taxonomy that allows to categorize concepts used in research, relates the newly identified concepts to established concepts from other fields, and possibly allows the identification of new research issues [80].

This thesis proposes a taxonomy of autonomic large-scale computing that identifies common concepts and categories in that field in order to support the development of self-managing systems in the field of utility computing and other related areas. A survey of autonomic large-scale systems results in a catalog that identifies concepts in autonomic large-scale computing established in current projects.

Section 1.1 further outlines the FoSII project and describes why the taxonomy of autonomic large-scale computing is suitable to address the problems investigated by that project. Section 1.2 provides background about the problem field the thesis addresses by giving a short introduction to the concepts of grid computing, cloud computing, autonomic computing, and other related concepts. Section 1.3 describes the scientific contribution of this thesis which consists of a taxonomy and survey of autonomic large-scale systems. Section 1.4 finally outlines the organization of the thesis.

1.1 Motivation

Although cloud computing promises to deliver the vision of computing power as a public utility besides water, gas, electricity and telephony, among the obstacles to the acceptance of cloud computing [21] is reliability. In order to be accepted by the public [147], reliability of cloud services needs at least to match, or possibly even exceed reliability of the other public utilities.

Currently, public cloud systems like Amazon AWS [10] or Google Apps [83] promise to deliver highly available and well-performing service backed by the reputation of their data centers. However, for businesses to be willing to entrust their mission-critical applications to a cloud provider, non-functional requirements like availability and response time need to be guaranteed by specifying quality of service attributes and formally establishing service-level agreements. Most public cloud providers issue SLAs that guarantee availability and offer credit in case that availability cannot be achieved. Although a high degree of infrastructure availability currently can be attained using well-established and mature technology like redundant high-availability clusters and load-balancing proxies, in order to take account of other QoS attributes like response time or data transfer rate, systems need to adapt to dynamically changing patterns of service usage. While it traditionally has been the task of system administrators to reconfigure systems in order to adapt to changing requirements, manual reconfiguration is not feasible at cloud computing centers due to their size and complexity. Thus, large-scale computing systems need to support self-adaptation to changing environmental conditions without human intervention.

Self-adaptation of a single system can be achieved by introducing a system management component that collects information from the system and its environment, decides if the system configuration needs to be changed based on that information and applies configuration changes resulting from that decision back to the system. In the case of distributed systems consisting of several interconnected nodes, each node needs to employ its own management component, and the communication protocol employed by the distributed system needs to provide a means for the management components to exchange information. Contemporary cloud computing systems usually are based on web services, implementing a service-oriented architecture. However, current service-oriented architectures lack support for self-adaptation, thus restricting the possibilities for self-management.

The *Foundations of Self-Governing ICT Infrastructures project (FoSII)* [73] which is conducted at TU Vienna addresses the problem of lacking dynamism and adaptivity in current service-oriented architectures by developing methods for self-management and communication between ICT (Information and Communications Technology) systems with the goal to allow for delivering computing power as a service.

In order to achieve this goal, the FoSII project employs the paradigm of autonomic computing [107], where systems adapt themselves in reaction to changes in their environment in order to reach or maintain a state defined by a set of high-level policies issued by human operators. Within the FoSII project, the autonomic computing model is augmented by introducing self-governing principles that may improve or produce

the rules that are part of a policy. Currently, the project focuses on negotiation and monitoring of service-level agreements (SLA), deriving high-level SLAs from low-level system parameters, and on managing compliance requirements regarding security and privacy.

Although there is already a growing body of work outside the FoSII project that explores self-management in large-scale distributed systems, it has not yet been systematically investigated. A comprehensive survey of that work would allow to derive a taxonomy providing a classification by criteria observed in the surveyed projects. Such a taxonomy would allow to identify common patterns and methods for employing self-management in large-scale computing systems in order to advance the field of autonomic large-scale computing and to support the introduction of autonomic large-scale computing technologies in the FoSII project and other projects with similar goals. However, a taxonomy on self-adaptation in large-scale distributed computing has not yet been proposed to this day.

1.2 Problem field

Large-scale distributed computing investigates systems that exceed other available systems in their ability to solve computing problems of a large size, handle large amounts of data, or serve a large quantity of users. Compared with other systems available in the same period they differ not in functionality but in scale. In this work the term covers the interrelated paradigms of cloud computing and grid computing which will be further outlined in the following sections. Other paradigms covered by large-scale computing are high performance computing, volunteer computing, and global distributed computing.

High performance computing or cluster computing [36] investigates computer systems that are equipped with extraordinary processing power compared with other systems available at the time of their creation. A system type studied in high performance computing are single high-performance computers called supercomputers which are based on a custom architecture. Supercomputer architectures have employed vector register processors where single instructions operate on an array of data, and massive parallel processing where many independent processors interconnected with a high-speed network operate on the same task [24]. During the last two decades, supercomputers have been challenged by clusters [60] consisting of a large number of independent nodes, often commodity hardware, that are grouped together by cluster software that supports inter-node communication by employing the message-passing interface (MPI) [132]. High performance computing centers are often operated by scientific institutions and nowadays usually support access to their resources by means of another large-scale distributed system like a grid.

In volunteer computing [13], users contribute their idle computing resources to a large-scale distributed computing system, mostly in a non-dedicated fashion. A common resource type in volunteer computing is the processing power of desktop computers during idle periods, where detection of those periods may be triggered by a screen-

saver application. Global distributed computing projects like SETI@home [12] use those contributed computing resources to solve specific computation-intensive problems.

Short introductions to grid computing, cloud computing, and autonomic computing are presented in Sections 1.2.1, 1.2.2 and 1.2.3, respectively, in order to provide a background on the problem field addressed by this thesis.

1.2.1 Grid computing

Grid computing [69] is a special field of distributed computing that enables organizations to share resources like computing power, data, and storage in accordance with policies and sharing rules [71]. Computing nodes available on the grid often are expensive high-performance clusters that are shared on a cooperative basis within a scientific community.

Grids allow the sharing of computational and storage resources, network resources, code repositories, and database catalogs. Since potential consumers of these resources often are located in a different control domain than the resources, the concept of a virtual organization (VO) has been introduced. This concept allows physical organizations like universities or companies to enter or leave virtual organizations in accordance with sharing rules established within the virtual organization, in order to participate in joint projects.

The Open Grid Services Architecture (OGSA) [70] is based on a service-oriented architecture and introduces the concept of grid services as an extension of web services that allows to create, destruct and statefully invoke services on the grid. Other concepts in grid computing include desktop grids, also called enterprise grids, that allow to share the resources of idle desktop computers within an actual organization or a virtual organization. Desktop grids provide special means to remove resources from the grid when they are accessed by the local user of a desktop computer.

The usual mode of operation in computational grids is to submit a computational job on the grid and later collect results. Jobs may be dependent on existing data repositories, resulting in the need for providing information about how to access those repositories. Some grids allow to submit scientific workflows consisting of multiple interdependent jobs. Thus, resource management, scheduling, and workflow management are important research topics within grid computing. Grid systems are optimized to process jobs and workflows that contain very computation-intensive scientific calculations with the need to access very large data repositories or streams of real-time data.

Reference applications for grid computing include distributed aircraft engine diagnostics, earthquake engineering, management and analysis of distributed data in astronomy, medicine and high-energy physics, *in silico* experiments in bioinformatics, and enterprise resource management [69].

Section 3.1 gives a detailed introduction to the discipline of grid computing.

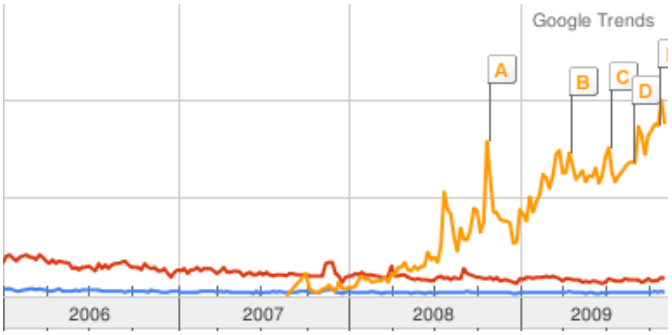

Figure 1.1: Google search trends for cluster computing (bottom), grid computing (middle) and cloud computing (top as of 2009) (adapted from [41])

1.2.2 Cloud computing

Cloud computing [41] allows data centers to offer computing resources based on interconnected virtualized computers to users on the Internet as a service. Key characteristics of cloud systems include their massive scale, the single ownership, the deliverance of services and provisioning of unified computing resources on demand, and the goal of establishing service-level agreements. Current clouds are based on web services and are commercially offered within a utility pricing model.

Cloud computing depends on the existing technologies of virtualization and cluster resource management. Virtualization allows multiple virtual machines to be employed on a single physical machine. Since each of the virtual machines provides an independent operating environment secured from other virtual machines, this model allows to offer computing services that are tailored to the individual needs of multiple customers in cloud computing. Also, resource management is a crucial point for the success of cloud computing, since service requirements of users change over time and service-level agreements need to be met.

Buyya et al. [41] provide a comparison of cloud computing with the related paradigms of grid computing and cluster computing. All of these distributed computing paradigms allow to combine computational resources of multiple systems in order to provide remote access to high-performance resources. However, while clusters usually group together centrally owned and managed commodity computers in order to provide resources that may be used within an organization, and grids interconnect high-end computers including clusters belonging to different organizations using open standards and supporting decentralized management, clouds combine commodity computers and high-end servers with single ownership in order to provide resources in form of virtual machines that are provisioned on demand. Figure 1.1 displays the Google search trends for cluster computing, grid computing, and cloud computing in order to show the popularity of the related paradigms over time.

Commercial services currently offered on the cloud include applications like Google Apps [83] that allow to directly use applications, software platforms like the Google

App Engine [82] that offer an API against which applications can be developed, and infrastructure offerings like Amazon Web Services [10] that provide direct access to virtual machine images.

Section 3.2 gives a detailed introduction to the discipline of cloud computing.

1.2.3 Autonomic computing

Autonomic computing [107, 140] is a concept that introduces self-management capabilities to computing systems in order to reduce or possibly eliminate the human intervention needed for maintaining those systems. It is related to the concept of autonomous agents [74, 156] in allowing programs to self-adapt to changing environment conditions. However, while an agent senses environment conditions and acts on these conditions according to its own rules, the concept of autonomic computing proposes a hierarchy of systems with self-management capabilities that act according to high-level policies using a control loop.

IBM [99] envisions that autonomic computing will allow customers to specify high-level business policies and have systems act according to those policies without the need of human intervention. In order to achieve this goal, characteristics like self-awareness, self-configuration, self-optimization, self-healing, self-protection, adaptation to the environment, interaction with other autonomic systems according to established standards, and anticipation of resource usage are defined.

The term *autonomic computing* has been chosen in order to stress the resemblance of the concept with the functions of the autonomous nervous system of the human body. The autonomous nervous system subconciously regulates bodily functions like blood pressure and pulse in reaction to environmental variables like the outside temperature, freeing the mind for pursuing higher-level tasks. In the same vein, autonomic systems shall relieve data center operators from the burden of manually adjusting operational parameters of individual computing systems, reducing their responsibility to the task of formulating guiding policies.

Section 3.3 gives a detailed introduction to the discipline of autonomic computing.

1.3 Research issues and scientific contribution

This thesis contributes a comprehensive survey of autonomic large-scale distributed computing, a taxonomy based on that survey which is to our knowledge the first taxonomy that specifically addresses the field of autonomic large-scale distributed computing, and finally a catalog of autonomic large-scale distributed computing systems classified by that taxonomy.

The survey of autonomic large-scale distributed computing intends to include all work that investigates self-management in the disciplines of grid computing and cloud computing. It provides a summary for each work describing overall functionality and autonomic capabilities of each project.

Taxonomies, by providing a classification scheme based on the examination of existing items in a field, help to organize knowledge in that field, identify classes of existing items and their prevalence and sometimes predict the discovery or creation of new items [80]. Thus, taxonomies not only allow to systematically present the current knowledge within a field, they also help to identify new research opportunities. If a taxonomy is accepted within a research community, it also helps to standardize the usage of terms and concepts, which is part of the maturing process of a research field. A taxonomy based on a survey of current work in autonomic large-scale computing systems thus helps to systematically present the current state of autonomic large-scale computing, suggests a standardized use of terms and concepts, and possibly helps to identify new areas in large-scale computing where self-management may be applied.

Finally, the thesis presents a catalog of autonomic large-scale distributed computing projects classified by terms and concepts established within the taxonomy, that allows to correlate research issues in large-scale distributed computing and autonomic computing methods addressing those issues.

1.4 Organization of the thesis

This section presents the structure of the thesis and summarizes the contents of the following chapters.

Chapter 2 presents existing taxonomies related to the fields of grid computing, cloud computing and autonomic computing as related work.

Chapter 3 describes concepts and terminology relevant to this thesis. It provides an introduction to the topics of grid computing, cloud computing, and autonomic computing and then introduces as reference some non-autonomic grid and cloud projects and autonomic projects that explore other fields than large-scale computing.

Chapter 4 conducts a comprehensive survey of autonomic large-scale computing projects and introduces the taxonomy of autonomic large-scale computing that has been derived from that survey.

Chapter 5 presents a catalog of the surveyed autonomic large-scale computing systems. Here the taxonomy that has been derived from the autonomic large-scale computing projects surveyed in Chapter 4 is reapplied to those projects in order to create the catalog.

Chapter 6 finally presents the conclusions drawn from creating the taxonomy and catalog of autonomic large-scale computing systems and suggests future work in the field of autonomic large-scale computing.

2 Related Work

This chapter presents related taxonomies in the areas of grid computing, cloud computing, and autonomic computing. Section 2.1 presents related taxonomies that are concerned with aspects of grid computing. Section 2.2 introduces taxonomies that are related to the area of cloud computing. Section 2.3 finally presents taxonomies related to autonomic computing.

2.1 Taxonomies of grid computing

Within the field of grid computing, several taxonomies have already been proposed that cover the field as a whole, subtopics of grid computing like resource management or workflow management, or special concerns.

Section 2.1.1 presents a taxonomy of grid resource management systems [113]. Section 2.1.2 introduces a taxonomy of market-based resource management systems [188]. Section 2.1.3 describes a taxonomy of resource submission [27]. Section 2.1.4 presents a taxonomy of data grid systems [182]. Section 2.1.5 describes a taxonomy of grid monitoring systems [193]. Section 2.1.6 presents a taxonomy of desktop grids [184]. Section 2.1.7 introduces a taxonomy for grid applications [168]. The next three sections are concerned with grid workflow management, where Section 2.1.8 presents a taxonomy of grid workflow management systems [192], Section 2.1.9 describes a taxonomy of grid workflow verification and validation [49], and Section 2.1.10 outlines a taxonomy of grid workflow scheduling [187]. Finally, Section 2.1.11 presents a taxonomy of grid computing security [47].

2.1.1 Taxonomy of grid resource management systems

Krauter et al. [113] present a survey and taxonomy of resource management systems in grid computing in order to identify possible architectural approaches in grid resource management. The taxonomy consists of a grid system taxonomy, a resource management taxonomy based on an abstract resource management system model, and a taxonomy of scheduling and resource allocation.

The remainder of this section introduces the taxonomies of grid systems, resource management, and scheduling, that together form the taxonomy of resource management systems, and presents results from the survey of grid resource management systems.

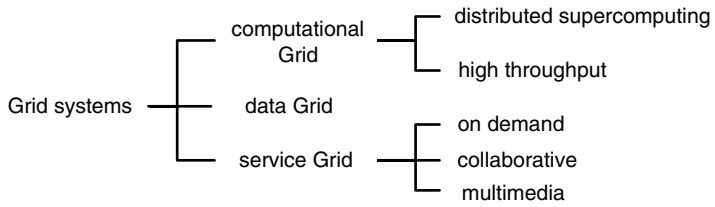

Figure 2.1: Taxonomy of grid systems [113]

Grid system taxonomy

Grid systems are grouped by their design objective, resulting in the categories of computational grid, data grid, and service grid. The grid systems taxonomy based on this categorization is shown in Figure 2.1 [113].

Computational grid systems are designed to improve application performance and offer computational capacity for single applications that is higher than any of the system capacities of the grid member systems. Computational grid systems are subdivided into distributed supercomputing systems that are typically used for parallelized computation of grand challenge problems in order to reduce job completion time. High throughput systems on the other hand are optimized to increase the completion rate of a stream of applications.

Data grid systems provide an infrastructure for data access and storage management to applications, in order to allow extracting information from large-scale data repositories. While some computational grid systems also implement their own data management functionality, data grid systems provide a specialized infrastructure for this purpose. Data grid systems are used for data mining purposes, in order to correlate information from various data sources.

Service grid systems provide composite services that are not provided by any single machine on the grid. This category is subdivided into on-demand, collaborative, and multimedia grid systems. On-demand service grids dynamically provide new services composed from base services. Collaborative service grids enable real-time interaction between users by providing a virtual workspace. Multimedia service grid systems provide an infrastructure for real-time multimedia applications, requiring provisions to support quality of service across multiple machines [133].

The survey indicates that most grid resource management systems fall into one of the categories described above, since the development of a general-purpose grid system supporting multiple or all of those categories is a hard problem.

Resource management taxonomy

The taxonomy of resource management systems is based on an abstract model shown in Figure 2.2 [113]. The model is based on a multilayer resource management system that allows to interconnect schedulers at different levels and provides four interfaces and

several functional units. The resource consumer interface connects with actual applications or higher-level resource management systems. The resource provider interface similarly allows access to actual resources or lower-level resource management systems. The resource manager support interface is used for access to support functions like naming or security. The resource manager peer interface finally allows interconnection with other resource management systems in order to provide functionality like resource discovery and co-allocation.

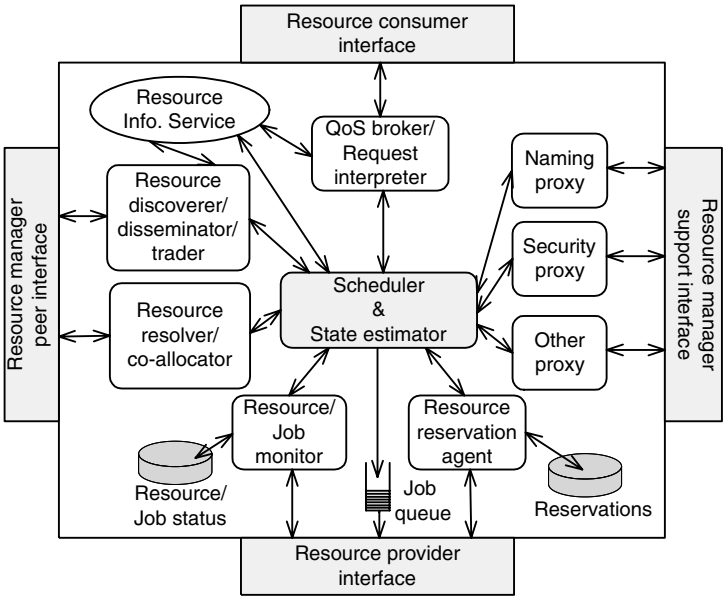

Figure 2.2: Resource management system model [113]

Based on the abstract resource management system model described above, a taxonomy of grid resource management systems is developed. Figure 2.3 (adapted from [113]) shows the top levels of the resource management system taxonomy.

The resource model refers to the description of grid resources by applications and resource management systems. A schema-based approach uses a description language and possibly a query language to describe resources. An object model defines operations on resources as part of the resource model. Both schema-based and object model approaches are subdivided in fixed and extensible models, with the extensible types allowing the extension of the resource model.

The resource namespace describes the organization of the names that are used to access resources on the grid. A relational namespace uses concepts borrowed from relational databases to access resource names that are organized in tuples. A hierarchical namespace organizes resources by hierarchical structures that typically reflect the physical or logical network structure. On the third level, the taxonomy also defines relational/hierarchical hybrids where the contents of a relation are hierarchically

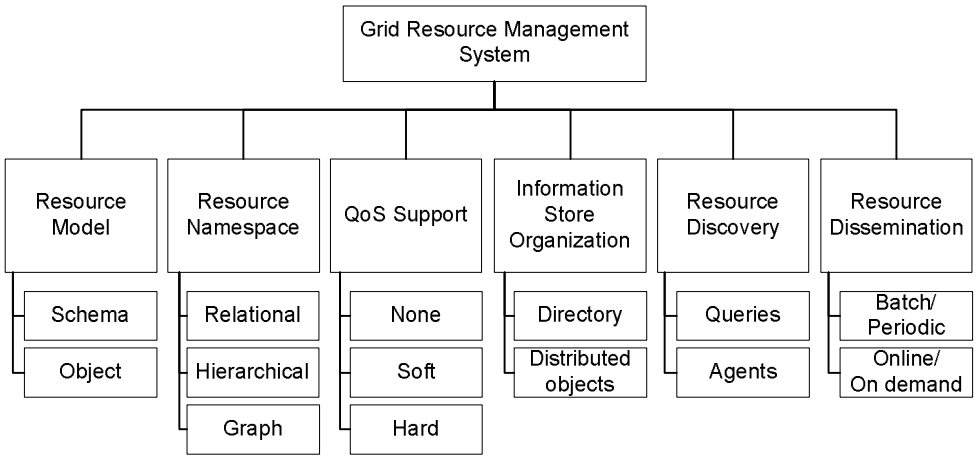

Figure 2.3: Taxonomy of grid resource management systems, compiling the resource management system taxonomies introduced in [113].

organized for the purpose of distribution. A graph namespace finally organizes names using nodes and pointers.

The QoS support policy defines quality of service levels offered by the systems. Some systems do not provide any QoS support. Soft QoS support refers to systems that support admission control by providing explicit QoS attributes for resource requests, but do not suport policing, since they cannot enforce service level agreements. Hard QoS support on the other hand refers to systems that provide both admission control and policing.

Information store organization determines the implementation of persistence offered by the resource management system. The distributed objects category is subcategorized into object model-based and language-based types. The former refers to language-independend object models like CORBA, the latter to language-specific persistence layers like those offered in Java. Network directory information stores are based on a relational database, while offering different interfaces to the database that constitute the subcategories within the taxonomy. X.500/LDAP type directories offer the interface and schema provided by the respective standard to interface with the data. Relational directories allow operations on the relational database using other query interfaces. The third subcategory is named "other" and refers to network directory access methods that do not fit in the other two categories.

Resource discovery is initiated by an application in order to find a resource on the grid, while resource dissemination is initiated by resource in order to find an application that can use it. Resource discovery is either query-based or agent-based. Query-based resource discovery operates either on centralized or distributed directories. While query-based systems use queries in order to operate on a directory, agent-based approaches send code fragments across machines that are interpreted locally. Resource

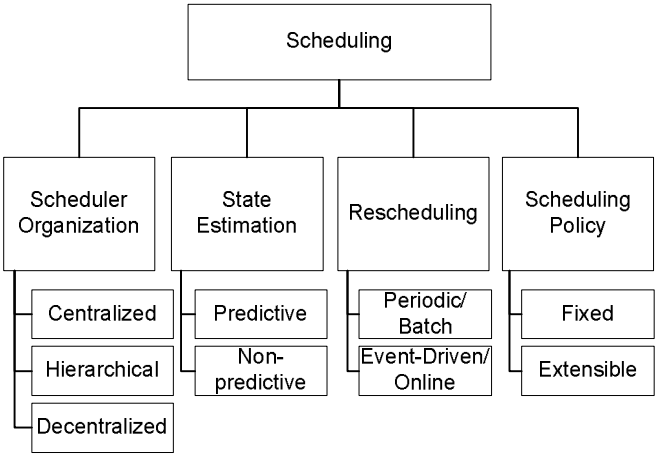

Figure 2.4: Scheduling in grid resource management systems, compiling the scheduling taxonomies introduced in [113].

dissemination approaches defined within the taxonomy are the batch/periodic approach where resource information is made available on each machine on the grid before being disseminated periodically, and the online/on demand approach where resource information is disseminated directly by the originating machine. In the batch/periodic approach, the taxonomy distinguishes between push and pull techniques.

Scheduling and resource allocation taxonomy

The taxonomy of scheduling in grid resource management systems examines the properties of scheduler organization, state estimation, rescheduling, and scheduling policy. The scheduling taxonomy is displayed in Figure 2.4 [113].

Scheduler organization is either centralized, decentralized, or hierarchical. In a centralized organization, there exists a single scheduling controller with system-wide responsibilities in decision making, leading to a simple and robust design with the disadvantage of having scalability issues. In a hierarchical organization, higher-level controllers manage a larger set of resources than lower-level controllers. A decentralized organization addresses issues like fault-tolerance, scalability, and multi-policy scheduling, but introduces problems in coordination between schedulers and usage tracking.

State estimation, which in grid systems is always based on partial or stale information, is categorized into predictive and non-predictive models. Predictive models use both current and historical data for state estimation and are based either on heuristics, pricing models, or machine learning. Predictive heuristics use predefined rules for state estimation. In a pricing model approach, resources are bought and sold according to market dynamics. Machine learning techniques uses learning schemes for state estimation. Non-predictive state estimation techniques use either heuristics based on

the current resource status and job information or a probability distribution model based on expected characteristics of the current job.

Rescheduling refers to the temporal characteristics of reexamining and possibly reordering the job scheduling queue. In periodic or batch rescheduling, resource requests and system events are grouped and then processed at intervals that are either periodic or triggered by events. In event-driven online models, rescheduling occurs immediately in reaction to resource requests or system events.

The scheduling policy determines reordering of jobs and requests during rescheduling. Fixed scheduling policies are subcategorized into system oriented and application oriented approaches and refer to systems with policies that are predetermined by the resource manager. System-oriented fixed policies maximize for system throughput, while application-oriented fixed policies strive to minimize application runtime or to optimize other application-specific metrics. Extensible scheduling policies allow the changing of policies by external entities and are subcategorized into ad-hoc and structured schemes. Ad-hoc extensible scheduling policies implement a fixed policy but allow an external agent to change the resulting schedule. Structure extensible schemes provide models of the scheduling process that allow external agents to override the default behavior of the scheduler.

Survey of resource management systems

A survey of grid resource management systems examines the following grid projects and maps them to the taxonomy: 2K, AppLES, Bond, Condor, Darwin, European DataGrid, Globus, Javelin, GOPI, Legion, MOL, NetSolve, Nimrod/G, Ninf, and PUNCH.

The survey shows that all projects except European DataGrid are either computational or service grid projects. Most systems support an extensible resource model and employ a periodic push approach to resource dissemination. Resource discovery approaches and resource models are correlated: While object model systems often use agents for resource discovery, schema-based systems employ queries. Generally, most aspects of the taxonomy have been explored in the different systems.

2.1.2 Taxonomy of market-based resource management systems

Yeo and Buyya [188] present a taxonomy and survey of market-based resource management systems for utility-driven cluster computing. The taxonomy examines the properties of cluster resource management systems and differentiates between clusters, distributed databases, grids, parallel and distributed systems, peer-to-peer systems and the world wide web as computing platforms. In the context of grid computing, grid resources are often located on clusters, with grid schedulers submitting and monitoring jobs on cluster systems through interaction with the cluster resource manager.

The taxonomy for utility-driven cluster computing is based on the five perspectives of market model, resource model, job model, resource allocation model, and evaluation model, which are shown in Figure 2.5 [188] and discussed in the remainder of this section. Then, a summary of the market-based resource management systems survey is given.

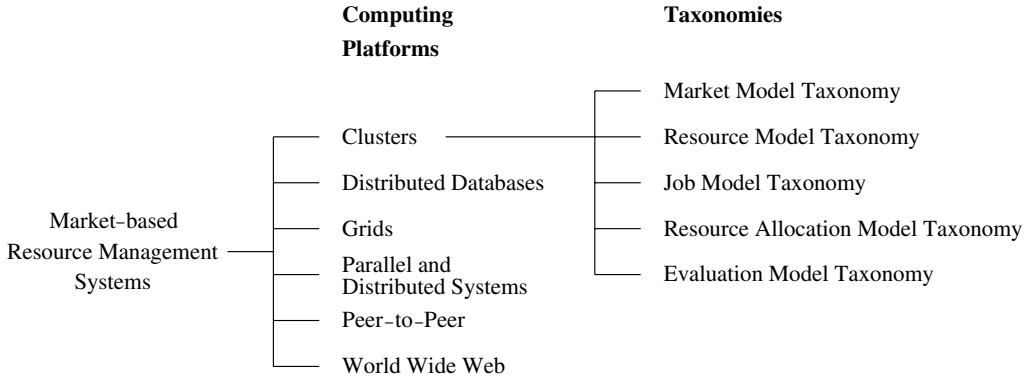

Figure 2.5: Categorization of market-based resource management systems [188]

Market model taxonomy

Market model refers to the representation of real-world economic concepts in the resource management system. It comprises the sub-taxonomies of economic model, participant focus, trading environment, and QoS attributes.

The economic model [37] establishes the allocation of resources based on market interaction between producers and consumers. Possible economic models within the taxonomy are commodity market, posted price, bargaining, tendering/contract-net, auction, bid-based proportional resource sharing, community/coalition/bartering, and monopoly/oligopoly.

The participant focus identifies the party that shall benefit from resource management and may be either consumer, producer, or facilitator. Systems with a consumer focus aim to optimize properties of the jobs issued by the consumers, while those with a producer focus try to maximize producer benefits from resource usage. Systems with a facilitator focus introduce a facilitator that draws gains from coordinating and negotiating resource allocations between producers and consumers.

Trading environment describes the motive of trading between participants and is either cooperative or competitive. Cooperative trading has the participants working together for a collective benefit like a common resource sharing environment, while in a competitive trading each participant acts according to individual goals like allocating resources.

QoS attributes describe consumer requirements that the producer needs to satisfy. Possible attributes are time, cost, reliability, and trust/security. QoS time may refer

to job execution time, data transfer time, or to a job completion deadline. The cost attribute can be expressed in monetary or non-monetary terms like resource consumption. Reliability represents the expected level of service guarantee, and trust/security identifies the level of security needed.

Resource model taxonomy

The resource model taxonomy categorizes the architectural style of a cluster system. It comprises the sub-taxonomies of management control, resource composition, scheduling support, and accounting mechanism.

Management control investigates the organization of resource management, which may be either centralized or decentralized. Centralized management control uses a single resource manager for the whole cluster, while decentralized management control uses multiple resource managers with each of them being responsible for a subset of resources.

Resource composition describes the layout of resources within the cluster. A cluster with homogeneous resource composition has multiple nodes with identical resource configuration, while in the case of heterogeneous resource composition, nodes may be configured individually.

Scheduling support refers to the scheduling capabilities of the underlying operating system of a cluster. Space-shared scheduling allows a single job per processor at any given time, while time-shared scheduling supports multiple jobs on a single processor.

The accounting mechanism is responsible for storing job execution information for the purpose of charging the user or planning future resource allocation decisions. It may be either centralized or decentralized. Centralized accounting uses a single accounting manager responsible for the whole cluster, while decentralized accounting uses multiple accounting managers, each being responsible for a subset of information.

Job model taxonomy

The job model taxonomy describes attributes of jobs that are executed on a cluster. It comprises the sub-taxonomies of job processing type, job composition, QoS specification, and QoS update.

The job processing type may be either sequential or parallel. With sequential job processing, jobs are processed independently from each other on a single processor. Parallel job processing requires the distribution of jobs to multiple processors before execution and speeds up processing for distributable jobs.

Job composition describes the mapping between tasks and jobs. In a single-task job composition, each job consists of a single task, while in a multiple-task job composition, a job may be composed from multiple tasks. Multiple-task compositions may consist of dependent or independent tasks. In the case of independent tasks, parallelization may speed up job execution.

QoS specification refers to the type of specification that may be associated with a job and includes constraint-based, rate-based, and optimization-based specifications.

A constraint-based specification includes a value or a range of values for each QoS attribute that satisfies the minimum QoS requirements. Rate-based QoS specification specifies a rate that signifies the required level of service over type. For instance, a credit deprecation rate can be specified so that the user pays less for longer job execution time. In the case of optimization-based QoS, the user specifies a specific QoS that shall be optimized in order to maximize the utility.

QoS update describes whether QoS requirements may be changed after job submission. In the case of static QoS update, such changes are disallowed, while dynamic QoS update allows to change QoS requirements during job lifetime.

Resource allocation model taxonomy

The resource allocation model taxonomy describes factors that influence the allocation decision of a resource management system. It comprises the sub-taxonomies of resource allocation domain, resource allocation update, and QoS support.

Resource allocation domain refers to the operational scope of a resource management system and may be internal or external. In case of an internal resource allocation domain, jobs may only be assigned within the cluster, while an external resource allocation domain enables the execution of jobs on remote cluster systems.

Resource allocation update identifies whether a resource management system is able to change an existing resource allocation in reaction to environmental changes. Adaptive resource allocation upates allows to adjust the resource allocation, while non-adaptive resource allocation uses a fixed resource allocation for each job.

The sub-taxonomy for QoS support is derived from the grid resource management systems taxonomy [113] described in Section 2.1.1 and defines soft QoS support for systems that allow the specification of QoS requirements but do not guarantee their satisfaction. Hard QoS support refers to systems that provide measures to ensure that specified QoS requirements are fullfilled. Hard QoS support usually depends on admission control that allows to reject jobs whose QoS requirements cannot be satisfied. Systems may provide both soft and hard QoS support in order to meet the needs of different user groups.

Evaluation model taxonomy

The evaluation model taxonomy describes how the efficiency and effectiveness of resource management systems may be assessed. It comprises the sub-taxonomies of evaluation focus, evaluation factors, and overhead analysis

The evaluation focus describes the point of view from which evaluation is performed and corresponds to the participant focus sub-taxonomy described above. A consumer focus evaluates the utility delivered to the consumer, while a producer focus or a facilitator focus measures the value gained by the producer or facilitator, respectively.

Evaluation factors may be system-centric or user-centric and describe the metrics that are used for evaluation. System-centric evaluation factors measure the overall

performance of the cluster system like average wait time or system throughput, while user-centric factors are used for assessment from the user's perspective.

Overhead analysis examines overheads introduced by the resource management system and is categorized into system overhead analysis and interaction protocol overhead analysis. System overhead refers to overheads that are introduced by the underlying system, while interaction protocol overhead means overhead introduced by the operating policies of the resource management system like communication between nodes.

Survey of market-based resource management systems

A survey investigates a number of market-based resource management systems based on clusters, grids, and peer-to-peer networks. Cluster systems included in the survey are Cluster-On-Demand, Enhanced MOSIX, Libra, REXEC, and Utility Data Center. In addition to these systems, the survey includes the grid projects Faucets, Nimrod/G, and Tycoon, and the Peer-to-Peer system Stanford Peers Initiative. The survey shows that at the time of its publication, most systems support only simple job models with sequential processing type, single-task job composition and static QoS update. Thus, the taxonomy outlines possible future enhancements of market-based resource management systems.

2.1.3 Resource submission taxonomy

A taxonomy of resource submission [27] categorizes types of submission in the context of systems where providers offer a platform that can be used by consumers. The taxonomy distinguishes between jobs, workflows, and arbitrary resources.

The resource type of job defines whether native jobs may be run at the provider's platform. Native jobs are specified using a job description language. A workflow may be submitted in the same manner as a simple job. Workflows are specified using a workflow language. In some environments, arbitrary resources may be submitted that are specified using a service specification language.

2.1.4 Taxonomy of data grids

Venugopal et al. [182] present a taxonomy of data grids, which are defined as systems that provide services for discovery, transfer and manipulation of very large datasets that are stored in distributed repositories. For this purpose, data grids manage replica catalogs and storage resources that can be accessed by compute resources.

The taxonomy of data grids first compares data grids with related research paradigms and then introduces the taxonomy elements of data grid organization, data transport, data replication and scheduling. The following sections provide a summary of the taxonomy. Then, results from the survey of data grid projects are presented.

The main differences between data grids and the other data-intensive research areas identified are heavy computational requirements, wider heterogeneity and autonomy,

the support for virtual organizations, and the traditional focus of data grids on scientific applications.

Figure 2.6 (adapted from [182]) shows the taxonomy of data grids.

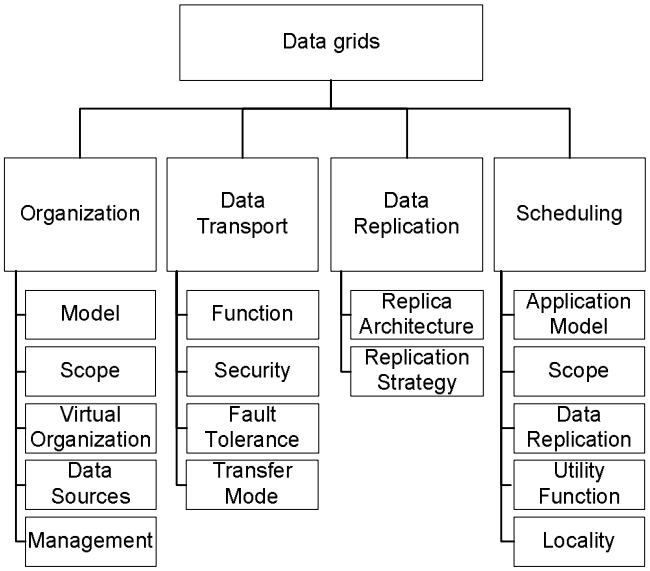

Figure 2.6: Taxonomy of data grids, compiling the taxonomies introduced in [182].

Data grid organization

The data grid organization taxonomy identifies the organizational characteristics of model, scope, virtual organization, data sources, and management. The model taxonomy describes ways of organizing data sources and distinguishes between monadic, hierarchical, federation, and hybrid models. Data grids following a monadic model gather all data into a central repository. Hierarchical models distribute data originating from a single source. Within a federation, each participating organization maintains its own data, and hybrid models finally merge the preceding approaches.

Scope is either intradomain or interdomain, where intradomain scope refers to restriction to a single scientific domain thus allowing specialization, and interdomain scope describes a generic infrastructure applicable to various domains.

Virtual organizations (VOs) reflect social organization and may be either collaborative, regulated, economy-based or reputation-based. Collaborative VOs are created by organizations collaborating on a single goal. Regulated VOs are controlled by a single organization. Access to economy-based VOs is guided by service-level agreements, and reputation-based VOs allow access for entities based on their provided level of service.

Data sources are transient or stable. Transient data sources (e.g., satellites) are available only at restricted time periods, while stable data sources are considered to be permanently available.

Finally, management is identified as being autonomic or managed. Managed grids require frequent human intervention in order to remain operable, while autonomic grids employ self-management as described in section 3.3.

Data transport

The taxonomy of data transport mechanisms describes issues around the actual movement of data by introducing the categories of function, security, fault tolerance, and transfer mode. The function of a data transport includes a transfer protocol, an overlay network, and a file I/O mechanism following a three-tier structure. Transfer protocol refers to the network protocol used for data transfer. The overlay network is an optional layer responsible for routing data. The file I/O layer finally provides transparent access to remote files.

Security aspects are authentication, authorization, and encryption. Authentication may occur either by passwords or by cryptographic keys. Authorization is coarse-grained or fine-graine, where coarse-grained authorization refers to methods provided by the underlying file system, and fine-grained authorization to methods that exceed file-system-based authorization. Encryption employs either SSL or is unencrypted.

Fault tolerance modes are restart transmission, resume transmission or cached transfers. Restart transmission describes the absence of fault tolerance. Resume transmission allows continuing a transmission starting with the last acknowledged byte. Cached transfers refers to the store-and-forward transfer mode of peer-to-peer overlay networks which do not require connectivity to the original data source.

Finally, supported transfer modes may be block, stream, compressed, and bulk modes. While the former three modes refer to the capabilities of traditional data transfer protocols, the bulk mode category groups together advanced capabilities like parallel data transfer and striped transfer.

Data replication and storage

The taxonomy of data replication and storage contains a replica architecture taxonomy and a replica strategy taxonomy. The replica architecture taxonomy consists of a model, a topology, storage integration, transfer protocols, metadata, update propagation, and catalog organization. The architectural model of a data replication is either centralized by employing a master replica being propagated to the other nodes, or decentralized employing many copies of the data that need to be synchronized. The topology of nodes in a replica management system may be flat, hierarchical or hybrid, employing different topologies at different levels. With regard to storage integration, tightly-coupled replication systems exercise control over local disk I/O, intermediately-coupled systems control the replication mechanism, and loosely-coupled systems have applications and users control replication using standard file transfer protocols. Transfer protocols may be open or closed with regard to the possibility of data access independent from the replication mechanism. Metadata may contain system-dependent and user-defined attributes, and may be updated actively by the replica management

system or passively by the user. Update propagation is synchronous like in databases or asynchronous, the latter being distinguished in epidemic (i.e., updating all replicas) and on-demand propagation. Catalog organization finally is based on a tree, on document hashes, or on a DBMS.

The replica strategy taxonomy differentiates by method, granularity, and objective function. Method is either static or dynamic regarding adaptations to changing network and storage conditions. Granularity of replication is categorized into dataset (i.e., multiple files), files, and fragments. Objective functions for replication finally are maximization of locality, exploiting popularity, economic objectives, minimization of update costs, preservation in case of failures, and effective propagation of new files.

Resource allocation and scheduling

The taxonomy of resource allocation and scheduling examines the application model, scope, data replication, utility function and locality of a data grid. The application model taxonomy distinguishes between process-oriented applications not structured into tasks, independent tasks, bags of tasks where all tasks must complete successfully, and workflows which define dependencies between tasks. Scope is individual or community-based. Scheduling with an individual scope occurs from the user's perspective, while community-based scheduling follows central policies. Data replication refers to the coupling of scheduling to replication of required data, with coupled schedulers replicating a permanent copy for future use, and decoupled schedulers working on a transient data copy. Utility functions in scheduling finally are minimization of computation time (makespan), load-balancing among the nodes, maximizing profit for the user, and observing quality of service requirements.

Survey of data grid systems

A survey of data grid systems examines the following data grids: LCG, EGEE, BIRN, NEESgrid, GriPhyn, Grid3, BioGrid, Virtual Observatories, Earth System Grid, GridPP, eDiaMoND, and the Belle Analysis Data Grid. In addition the data transport technologies GASS, IBP, FTP, SFTP, GridFTP, Kangaroo, Legion, and SRB are examined. These systems are then mapped to the taxonomy of data grids.

Observations from mapping the surveyed systems to the taxonomy include that most data grids follow the hierarchical or federated models using only a few well-established data sources, the identification of GridFTP as the standard data transfer protocol, and the dominance of manual static replication by system administrators in order to maximize locality and of scheduling strategies that reduce makespan.

2.1.5 Taxonomy of grid monitoring systems

Zanikolas et al. [193] present a taxonomy of grid monitoring systems that is based on the Grid Monitoring Architecture (GMA) [176] published by the Global Grid Forum. The taxonomy defines levels 0 to 3 of monitoring systems with additional

qualifiers. Figure 2.7 (adapted from [193]) shows the grid monitoring taxonomy and Figure 2.8 [193] further explains the levels of grid monitoring defined there.

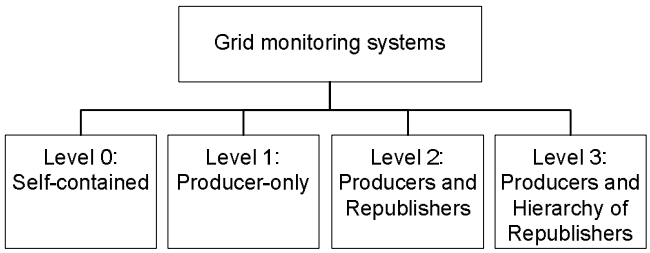

Figure 2.7: Taxonomy of grid monitoring systems, compiling the taxonomy introduced in [193].

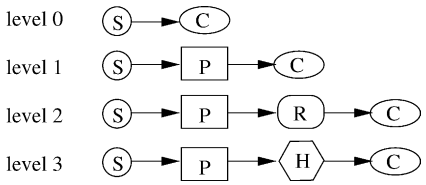

Figure 2.8: Levels of grid monitoring: S = sensor, P = producer, R = republisher, H = hierarchy of republishers, C = consumer. [193]

Level 0 characterizes self-contained systems that do not expose their functionality through a producer interface. Level 1 refers to producer-only systems, where sensors are separately implemented or provided by the producer. Systems on level 2 feature producers and republishers, with republishers providing only fixed functionality. Level 3 systems feature a producer and a hierarchy of republishers.

In addition to the levels defined above, the taxonomy uses qualifiers that further categorize the functionality of grid monitoring systems. The multiplicity qualifier applies to systems on level 2 and denotes whether republishers are centralized or distributed. The type of entities refers to the systems primarily monitored: hosts, networks, applications, availability, and general-purpose. The stackable qualifier refers to systems that support operation on top of another monitoring system.

A survey of grid monitoring systems examines the level 0 systems MapCenter and GridICE and the level 1 system Autopilot. On level 2, the survey examines a monitoring system based on the CODE framework and the systems GridRM, Hawkeye, HBM, Mercury, NetLogger, NWS, OCM-G, Remos, and SCALEA-G. Finally, the level 3 systems Ganglia, Globus MDS, MonALISA, Paradyn/MRNet, and RGMA are surveyed. The survey concludes that most considered systems maintain a database with an archive, which is exposed through an interface by some tools. Almost all systems

support host and network events. Data delivery models and event formats are implemented in various ways. Since there is no coordination between projects, functionality often overlaps, and interoperability is limited.

2.1.6 Taxonomy of desktop grids

Vladiou and Constantinescu [184] present a three-level hierarchical taxonomy for desktop grids which are grids consisting of resources provided by idle desktop computers. Figure 2.9 (adapted from [184]) shows the top two levels of the taxonomy.

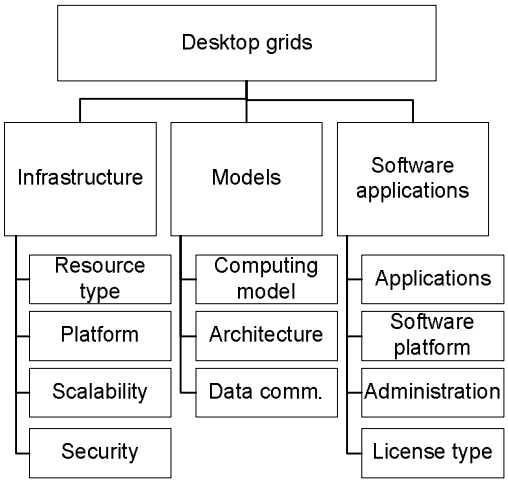

Figure 2.9: Taxonomy of desktop grids, compiling the taxonomies introduced in [184].

Desktop grids are categorized by infrastructure, models, and software. On the infrastructure level, the resource type is either volunteer and enterprise, the platform is web-based or middleware-based, scalability is Internet-based or LAN-based, and security is trust-based, authentication-based or sandbox-based. On the level of models, the computing model is either master-worker or based on parallel paradigms. The architecture is centralized, hierarchical, or peer-to-peer. Finally, the taxonomy identifies the data model types of middleware, data servers, and direct communication.

The software level examines applications, architecture, administration, and license type. The application type supported by a desktop grid may be limited to a set of dedicated applications, restricted to Java applets, or it may include one or more of legacy, script, compiled, and lightweight applications. The software platform specifies the operating system allowed for desktop grid nodes. Some systems, e.g. those implemented in Java are operating system independent, while others support one or more operating systems from the set of Windows, Linux, Unix, Solaris, and MacOS. Software administration examines whether systems distinguish between administrative and non-administrative users. The software license taxonomy finally distinguishes between closed and open source licenses.

A survey examines the following desktop grid systems and maps them to the taxonomy: distributed.net, Entropia, SETIhome, Bayanihan, Condor, XtremWeb, QADPZ, BOINC, SZTAKI LDG, and Javelin. The survey concludes, that desktop grids are a method of providing computing power to scientists that is easily available, and the taxonomy helps to choose the right tool for each scientific purpose.

2.1.7 Taxonomy of grid applications

Suciu and Potolea [168] present a taxonomy for grid applications that is based Flynn's taxonomy of computer architectures [66] and on a taxonomy of cryptographic and cryptanalytic algorithms for grid applications [167]. The adaptation of those taxonomies to grid applications results in the following categories: SPNF (single program no file), SPSF (single program single file), SPMF (single program multiple file), MPNF (multiple program no file), MPSF (multiple program single file), and MPMF (multiple program multiple file). The file category refers to the cardinality of input files that may be processed by a job, while the program category denotes the support for parallelization. Figure 2.10 (adapted from [168]) shows the taxonomy of grid applications.

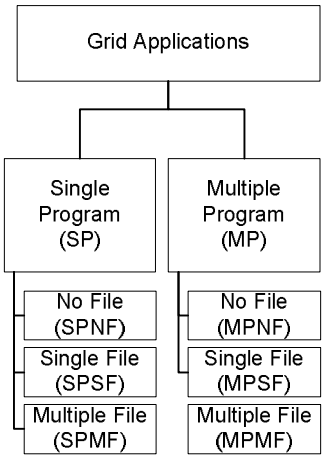

Figure 2.10: Taxonomy of grid applications, representing the taxonomy introduced in [168].

While applications in the SPNF and SPSF categories can process a single job at a time, the number of simultaneously running jobs in the SPMF, MPNF and MPSF categories is limited with the maximum number of programs and files, respectively. MPMF type applications support the highest degree of parallelism which is proportional to the number of programs and number of files. The SPSF, SPMF, MPSF, and MPMF optionally may support data parallelism (DP), which means that input files are split logically into slices which are processed simultaneously. In this case the maximum number of jobs increases with the supported number of data slices. On

the output side, the taxonomy distinguishes between single output (SO) and multiple output (MO) modes, where single output refers to the requirement of concatenating the outputs of multiple jobs into a single file.

The taxonomy provides a mathematical model in order to formally define the categories described above and gives example applications for each category. A survey of existing systems is not conducted, though.

2.1.8 Taxonomy of grid workflow management systems

Yu and Buyya [192] present a taxonomy of workflow management systems for grid computing that identifies five elements of a grid workflow management system: workflow design, information retrieval, workflow scheduling, fault tolerance, and intermediate data movement. The top levels of the taxonomy are shown in Figure 2.11 (adapted from [192]).

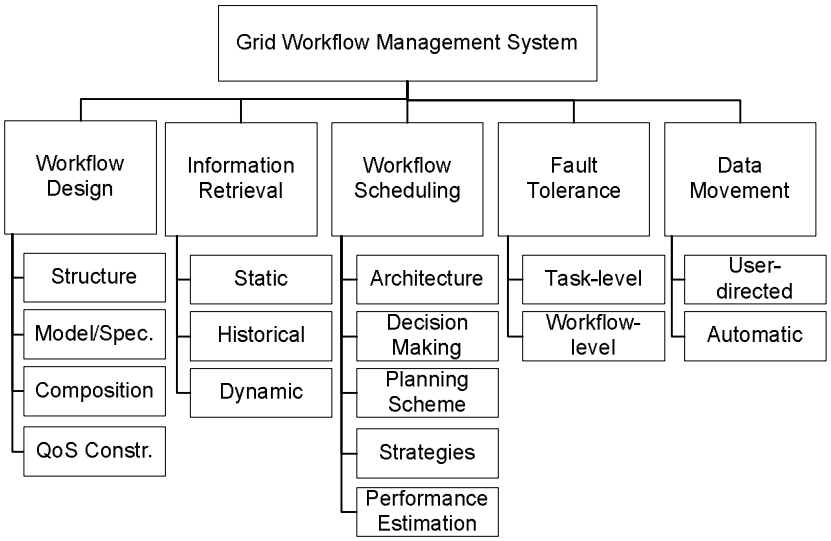

Figure 2.11: Taxonomy of workflow management systems, compiling the taxonomies introduced in [192]

Workflow design includes the key factors of structure, model, composition systems and QoS constraints. Workflow structure is represented as a directed acyclic graph (DAG) or as a non-DAG and classified as sequence, parallelism, choice, and in the case of a non-DAG additionally as iteration. Workflow models are either abstract or concrete models, while workflow composition systems are either user-directed or automatic. The taxonomy for user-directed workflow composition systems distinguishes between language-based or graph-based modeling and specifies a number of language types and representations for graphs. QoS constraints are derived from existing models for workflows based on web services [44, 122, 142] and specify the dimensions of time,

cost, fidelity, reliability and security which can be assigned at task level or workflow level.

The taxonomy of information retrieval in grid workflow management systems includes the three dimensions of static information, historical information, and dynamic information. Static information is related either to infrastructure, configuration, QoS, access, or to the user and may be used for pre-selection of resources during workflow initialization. While historical information is not further subdivided, dynamic information is categorized as being related to resources, the state of task execution or to the market.

Workflow scheduling is identified as being a case of global task scheduling [46] where the global scheduler coordinates with local management systems. Scheduling is discussed from the views of architecture, decision making, planning scheme, strategies, and performance estimation. The scheduling architecture may be centralized, hierarchical, or decentralized. Decisions are made either on a local or a global level. Planning schemes, which are methods for translating abstract workflows to concrete workflows include static and dynamic schemes, the former being user-directed or simulation-based, and the latter prediction-based or just-in-time. From the strategy viewpoint, heuristics developed to solve the NP-complete problem of workflow scheduling [65] are categorized into performance-driven, market-driven and trust-driven strategies. Finally, possible performance estimation techniques are simulation, analytical modeling, historical data, on-line learning, and hybrid approaches.

The taxonomy of fault tolerance in grid workflow management systems follows Hwang and Kesselman [98] in distinguishing between task-level and workflow-level techniques. Task-level techniques are catalogued into retry, alternate resource, checkpoint/restart, and replication. Workflow-level techniques are alternate task, redundancy, user-defined exception handling, and rescue workflow.

Intermediate data movement refers to the staging of input files that need to be accessed by the tasks of a workflow and of output files generated by that tasks to remote sites. The taxonomy of grid workflow management systems distinguishes between user-directed and automatic data movement. User-directed data movement refers to systems that require the user to manage data transfer as part of the grid workflow specification. Automatic data movement systems derive staging from a workflow specification and are categorized into centralized, mediated, and peer-to-peer approaches. A centralized approach for data movement defines a central point that collects execution results and transfers them to subsequent execution points. A mediated approach manages intermediate data locations using a distributed data management system like a catalog service, while in a peer-to-peer approach the executing nodes themselves transfer data to subsequent nodes.

A survey of grid workflow management systems examines the following grid workflow management projects and maps them to the taxonomy: Condor DAGMan, Pegasus, Triana, ICENI, Taverna, GridAnt, GrADS, GridFlow, Unicore, Gridbus workflow, Askalon, Karajan, and Kepler. The survey shows that many of these systems provide graph-based workflow editing environments that allow to compose a workflow using drag-and-drop techniques. Most projects use their own graphical workflow language

though, due to lack of standardization. On the other hand, quality of service is not addressed very well since most projects focus on system centric policies in resource allocation. Especially, market-driven strategies are largely ignored. While most projects include performance prediction services that are used for schedule optimization, fault handling techniques often are not implemented.

2.1.9 Taxonomy of grid workflow verification and validation

Chen and Yang [49] provide a taxonomy that categorizes methods for ensuring correctness of grid workflows. At the top level, correctness is ensured through verification and validation. Verification ensures that the specification and execution of a grid workflow is free of errors. Validation ensures that the grid workflow meets the requirements stated by the developer of the grid workflow. Figure 2.12 [49] shows the taxonomies of grid workflow verification and validation.

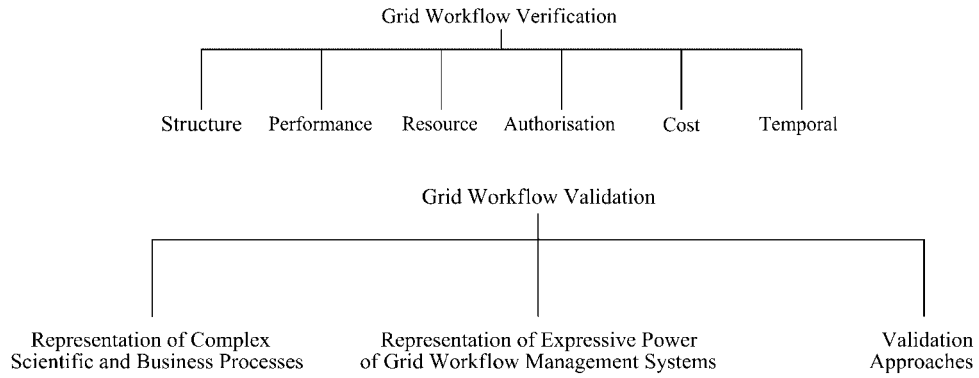

Figure 2.12: Taxonomies of grid workflow verification and validation [49]

The taxonomy of grid workflow verification consists of the elements of structure verification, performance verification, resource verification, authorization verification, cost verification and temporal verification. Structure verification verifies the consistency of the syntactic structure of the workflow, in order to avoid deadlocks, livelocks and similar conditions. The subcategories of structure verification are syntactic verification and semantic verification. Performance verification verifies if performance requirements are met. Possible approaches for performance computation include Markovian chain theory, queuing theory, and simulation tools. Resource verification verifies the absence of resource conflicts between different workflows and activities in the case of resource competitions. Authorization verification of the grid workflow specification checks the consistency between resource access policies and defined roles, while during grid workflow execution it checks for unauthorized access in order to trigger protective measures. Cost verification checks whether grid workflow specification and execution meets budget requirements, and temporal verification ensures that grid workflows are completed in time.

The categories within the grid workflow validation taxonomy are representation of complex scientific and business processes, representations of expressive power of grid workflow management system, and validation approaches. The representation of processes category states whether a system supports a defined means to formally represent the scientific or business process. Possible methods for process representation include Petri nets or process algebra. Representation of expressive power states which control structures are used to represent grid workflows. Current grid workflow management systems mostly support parallel, selective, sequential, and iterative control structures. Possible validation approaches result from the process representation and expressive power of grid workflow management systems. Successful validation means that a mapping exists between business process and workflow.

A survey examines the following grid workflow verification and validation projects and maps them to the taxonomy: DILIGENT, CROWN, Discovery Net, SwinDeW-G, and CAT. The survey shows that no project supports resource verification, authorization verification or cost verification. Besides CROWN that partially supports analysis of the expressive power of its workflow modeling language GPEL, no project currently supports any method for workflow validation.

2.1.10 Taxonomy of grid workflow scheduling

Within grid workflow management systems, scheduling – which is defined as the assignment of grid services to workflow tasks – needs to be optimized in order to minimize execution time, maximize job throughput, minimize cost or satisfy other criteria. The taxonomy of the multi-criteria grid workflow scheduling problem [187, 186] classifies existing approaches for optimization by two specific criteria and aims to provide a basis for developing generalized scheduling approaches that address multiple optimization criteria. Figure 2.13 [187] shows the top levels of the grid workflow scheduling taxonomy.

The taxonomy analyzes the following facets of the workflow scheduling problem: scheduling process, scheduling criteria, resource model, task model, and workflow model. The taxonomy for each of these facets is organized in three levels that match the subject-predicate-object notation of RDF[1].

The taxonomy of scheduling process includes the predicates of criteria multiplicity, workflow multiplicity, dynamism, and advance reservation. Criteria multiplicity is classified into single criterion and multiple criteria approaches. While single criterion approaches usually strive to minimize execution time and are rather common, multiple criteria approaches need to specify trade-offs between potentially contradicting criteria and thus introduce additional complexity. Workflow multiplicity describes the number of workflows that are subject to optimization and is classified either as single workflow or multiple workflows. Single workflow optimization optimizes a single workflow per scheduling process, while multiple workflows refers to the possibility of optimizing the execution of multiple workflows within a single scheduling process. Dynamism refers

[1]Resource Description Framework, `http://www.w3.org/RDF/`

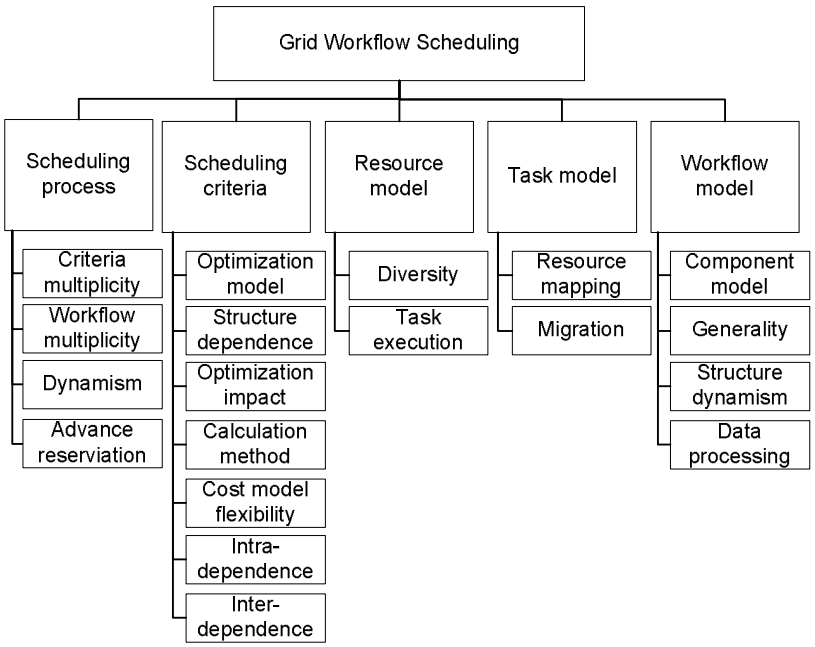

Figure 2.13: Taxonomy of grid workflow scheduling, compiling the taxonomies introduced in [187].

to the relation between workflow scheduling and workflow execution and is described in three classes: Just-in-time scheduling, where scheduling of a single task is performed immediately prior to execution, full-ahead planning, where the whole workflow is scheduled before its execution, and hybrid approaches that combine just-in-time scheduling and full ahead planning. Advance reservation finally refers to the possibility of delegating future resource capabilities to users. The taxonomy of scheduling process distinguishes between systems with reservation and without reservation.

The taxonomy of scheduling criteria includes the predicates of optimization model, workflow structure dependence, optimization impact, calculation method, cost model flexibility, intradependence, and interdependence. Optimization model describes the perspective from which the scheduling of workflows is optimized, which is either workflow-oriented or Grid-wide. Workflow structure dependence describes dependencies between individual tasks of a workflow and is classified in structure dependent and structure independent criteria, with structure dependent criteria being subclassed by the aggregation type of average, conjunctive, disjunctive, and mixed. Optimization impact is classified into objectives (maximized, minimized, and focused) and constraints (global or local). The calculation method may be additive, multiplicative or concave. Cost model flexibility is fixed or adaptive. Intradependence refers to dependence between individual scheduling decisions by a single criterion and is classified into intradependent and non-intradependent scheduling criteria, with intradependent

criteria being subclassed into partial cost related and aggregated cost related criteria. Interdependence describes the dependence between different scheduling criteria and is classified into interdependent and non-interdependent pairs of criteria.

The taxonomy of grid resources includes the predicates of diversity and task execution. The classes of diversity are homogenous and heterogenous. Homogenous resources have identical static and dynamic characteristics, while heterogeneous resources are subclassed into single type resources having resources of the same type but with different characteristics, and multiple type resources, where resources with different types are considered for scheduling optimization. Task execution refers to the assignment of resources to tasks and is classified into non-multiprogrammed models, where a resource is assigned to a single task, and multiprogrammed models, where multiple tasks can be scheduled on a single resource at the same time.

The taxonomy of workflow tasks includes the predicates of resource mapping and migration. Resource mapping is rigid, moldable, or malleable. Rigid resource mapping requires a task to use a predetermined number of resources. Moldable task mapping determines the required number of resources for a task at execution time, and malleable task mapping allows resources to be added or withdrawn from a job during execution. Migration is classified into migrative and non-migrative tasks. Migrative tasks may be checkpointed, preempted and resumed using a different set of resources.

The taxonomy of workflow model includes the predicates of component model, generality, atomic structure dynamism, and data processing. The component model is either task oriented or task and transfer oriented. While in the graph of a task oriented workflow model tasks are represented as nodes and data transfers as edges, within a task and data transfer oriented model both tasks and data transfers are represented as nodes. Generality is classified as specific or general digraph, where general digraph (directed graph) models may be formally described by a directed graph with the nodes and edges representing tasks and data transfers as described above, and specific models are described by a regular structure which is a well-defined subset of the general digraph model. Atomic structure dynamism refers to the possibility of modifying a given workflow structure during optimization and is classified into fixed and tunable models. Data processing distinguishes between single data set models and pipelined models, with the latter class referring to the processing of a stream consisting of multiple data sets during multiple workflow executions.

The taxonomy is applied to the following Grid systems: GrADS, Vienna Grid Environment, PEGASUS, ASKALON, K-WfGRid, Instant-Grid and to several approaches by other authors. The study identifies the workflow scheduling approach as not yet being fully addressed by existing work and states that the taxonomy may facilitate the development of new scheduling approaches based on the problem classes identified by the taxonomy.

2.1.11 Taxonomy of grid computing security

Chakrabarti et al. [47] present a taxonomy of grid computing security that identifies security issues in grid systems. At the top level, grid security issues are categorized

into host level, architecture level, and credential level issues. Figure 2.14 [47] shows the taxonomy of grid computing security.

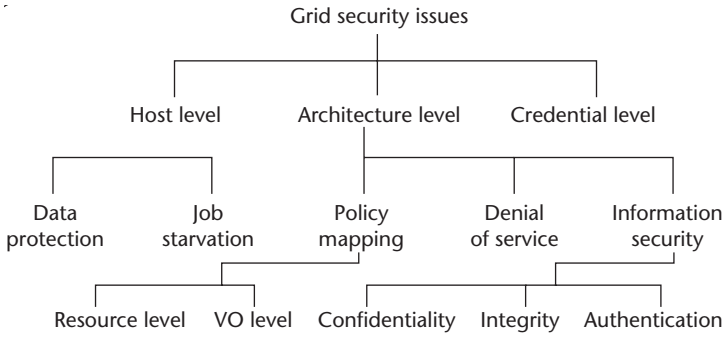

Figure 2.14: Taxonomy of grid computing security [47]

The host level issues category refers to security threats originating from new, untrusted hosts that are added to the grid. Host level issues are classified into data protection and job starvation issues. Data protection refers to the problem that a job might be submitted to the grid that contains a worm or virus. Solutions to the data protection problems are application-level sandboxing, virtualization, user-space sandboxing, and flexible kernels. Job starvation refers to remote jobs starving local jobs in a system. Countermeasures to job starvation are advanced reservation and priority reduction.

The architecture level issues category refers to security problems that affect the grid system as a whole. Architecture level issues are classified into information security, policy-mapping, and denial-of-service (DoS). Information security refers to concerns that arise regarding the communication between grid nodes and includes the issues of secure communication, authentication, and single sign-on. Information security is addressed by the grid security infrastructure (GSI)[2] standard. GSI implements secure communication at the transport and message levels using web services standards. Authentication is implemented in GSI using certificates. Single sign-on and delegation in GSI are implemented through a proxy that employs a certificate for delegation. Policy-mapping occurs either at resource level or at the level of a virtual organization (VO). The denial of service (DoS) issue can be addressed either using preventive solutions or reactive solutions.

The credential level issues category refers to the initiation, storage, renewal, translation, delegation, and revocation of credentials within a grid system. Credential management systems are categorized into credential repositories and credential federation systems. While the former systems are concerned with the storage of credentials, the latter ones help manage credentials across multiple systems, domains, and realms.

[2]Grid Security Infrastructure, `http://www.globus.org/toolkit/docs/latest-stable/security/`

2.2 Taxonomies of cloud computing

This section presents existing taxonomies of cloud computing. Since cloud computing is a relatively new field of distributed computing, no taxonomy that specifically addresses cloud computing has yet been proposed. However, because cloud systems are implemented using existing technologies like cluster computing and virtual machines, taxonomies of these areas are relevant to cloud computing.

In particular, the taxonomy of market-based resource management systems for utility-driven cluster computing [188] previously described in Section 2.1.2, and the taxonomy of resource submission that has been introduced in Section 2.1.3 are relevant to grid and cloud computing. Section 2.2.1 describes a taxonomy of virtual machines [164].

2.2.1 Virtual machine taxonomy

Smith and Nair [164] propose a virtual machine (VM) taxonomy as part of an overall description of virtual machine architectures. Virtual machines are created by adding a software layer to a physical machine in order to support multiple system architectures. Figure 2.15 [164] shows the taxonomy of virtual machines.

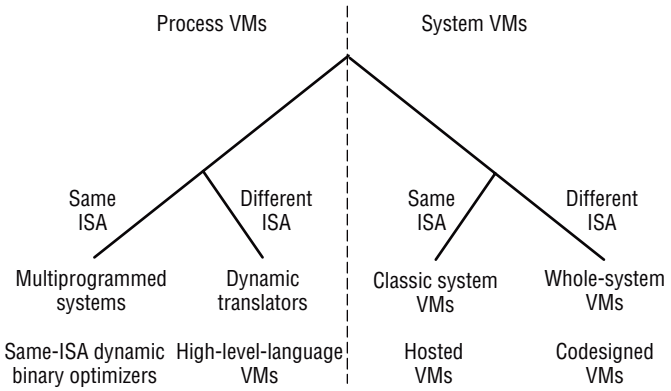

Figure 2.15: Taxonomy of virtual machines [164]
(ISA = instruction set architecture)

On the top level, the taxonomy distinguishes between process VMs and system VMs. While process VMs provide an application binary interface or application programming interface for applications, a system VM offers a complete environment for an operating system and applications.

The main categories of the process VM taxonomy are multiprogrammed systems and dynamic translators. Multiprogrammed systems refers to the ability of almost all contemporary operating systems to simultaneously run multiple processes on a

single processor, while providing a separate environment for each process. A dynamic translator allows to execute program binaries that have been compiled for a machine type differing from the executing machine.

The main system VM taxonomy categories are classic system VMs and hosted VMs. Classic VMs employ a fully privileged virtual machine manager that is placed on top of the hardware and manages multiple virtual machines with reduced privileges. Hosted VMs execute a virtualization software on top of an existing operating system, that allows the execution of multiple virtual machines.

2.3 Autonomic computing taxonomies

This section presents related taxonomies that are relevant to the field of autonomic computing. The taxonomies presented in this section are concerned with software adaptation, which is a prerequisite for autonomic computing, since the execution of change plans requires the modification of running programs.

Section 2.3.1 presents a taxonomy for system adaptation [175]. Section 2.3.2 introduces a taxonomy of compositional adaptation [127]. Section 2.3.3 describes a taxonomy of dependable and secure computing [23].

2.3.1 Taxonomy for system adaptation

System adaptation refers to the capability of a system to adapt itself to environmental changes with the goal of stability. Tianfield and Unland [175] propose a taxonomy that classifies system adaptation into passive adaptation, parametric adaptation, and mission-oriented adaptation.

Passive adaptation establishes a simple control loop that includes a sensor, a regulator and an effector. The passive adaptation scenario includes both manually regulated and automated systems. The example of a water leveling plant is given, with a valve forming the regulator. Sensor and effector may be provided either by a human operator's vision and hands, respectively, or by an automated regulation device.

Parametric adaptation introduces a parametric control loop that allows the update of tunable parameters using feedback control. The feedback control loop consists of the steps of determining the tunable parameters from a set of perceivable parameters that includes all recent changes in the environment, determining updates on the tunable parameters, and enforcing the determined updates into the system. Updates on the parameters are determined using some optimization algorithm.

Mission-oriented adaptation establishes a set of control loops designated for separate control missions in order to address the complexity of a system that has too many parameters to be managed by a single control loop. In most cases, the control loop set is organized as a hierarchy, thus establishing a hierarchical control architecture suitable for large complex systems [174].

The concepts of system adaptation outlined in the taxonomy may be observed both in the discipline of feedback control which is concerned with continuous parameters of

systems that follow physical laws, and in computing systems that are artificial systems mostly governed by human definitions in terms of discrete states and events. While the specific models and algorithms of feedback control thus are not directly applicable to autonomic computing, these systems can take advantage of the general frameworks and methodologies for feedback control and adaptive control.

2.3.2 Compositional adaptation taxonomy

McKinley et al. [127] present a taxonomy of compositional adaptation. Compositional adaptation is an approach for software adaptation where structural or algorithmic system components are exchanged in order to fit a program into a changing environment. It contrasts with parameter adaptation, where program variables are modified for the same purpose. The prerequisites for compositional adaptation are separation of concerns between business logic and crosscutting concerns, computational reflection where a program has access to its own structure, and component-based design that allows composing applications from preexisting components.

Figure 2.16 (adapted from [127]) shows the compositional adaptation taxonomy that categorizes existing methods for supporting compositional adaptation by the dimensions of software recomposition technique (how), composition time (when), and composition location (where).

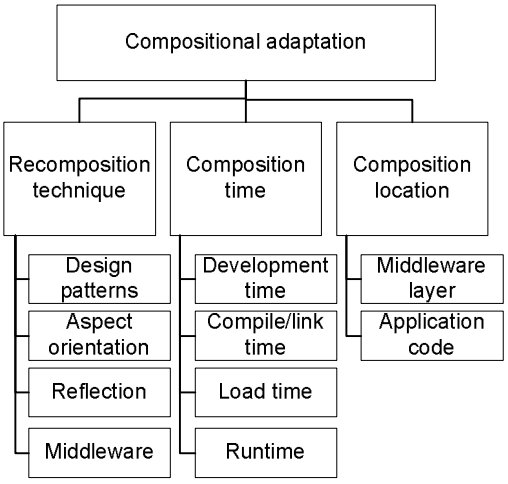

Figure 2.16: Taxonomy of compositional adaptation, representing the taxonomy introduced in [127].

Software recomposition techniques [6] form the first dimension of the taxonomy and create an additional level of indirection in order to support software composition. Some techniques realize this indirection by applying specific design patterns like function pointers, wrappers, proxies, the strategy pattern or the virtual component pattern. Others use aspect-oriented programming where auxiliary functions are separated from

business logic and isolated into a specialized software layer, reflection that allows a program to access and possibly modify its own structure, or both. Middleware techniques finally modify the interaction between the application and its middleware, with middleware interception being transparent to the application, and integrated middleware providing adaptive services that are explicitly invoked by the application.

The second dimension of the taxonomy categorizes approaches by the time of composing adaptive behavior with business logic. Later composition time introduces additional flexibility, thus allowing more powerful adaptation methods, but also complicates the problem of maintaining consistency within the adapted program. Composition may occur at development time with adaptive behavior being hardwired into the program, at compile or link time which leads to an application that is customizable for different environments, at load time for configurable applications, or at runtime. While the first three approaches are referred to as static composition methods, runtime composition is also called dynamic composition and is further categorized into tunable software composition that prohibits modification of code but allows the fine-tuning of cross-cutting concerns in response to changing environment conditions, and mutable software composition that permits the exchange of the program's function thus allowing for the recomposition of a running program into a functionally different one.

Composition location finally is the third dimension of the taxonomy of compositional adaptation that investigates where the adaptive code is located. Adaptive code may be inserted either at the middleware layer or directly into the application program. Middleware layer approaches may support adaptation at the host-infrastructure layer in form of an additional interprocess communication layer or by using facilities provided by a virtual machine. Adaptive behavior may also be introduced in higher middleware layers, allowing portability across virtual machines. In this case, middleware components typically support adaptation either by intercepting and modifying or redirecting remote method invocation messages, or by providing explicit calls to adaptive middleware services. Approaches that implement adaptation in the application program are either supported by the programming language itself or weave adaptive code into the functional code at compile time or later.

A survey identifies research projects, commercial packages, and standard specifications that provide compositional adaptation. Examples of surveyed projects are AspectJ [61] that introduces adaptive behavior into Java programs by implementing aspect-oriented programming at compile time within the application layer, and the Open ORB platform [53] that supports reflection at runtime within the middleware layer.

2.3.3 Taxonomy of dependable and secure computing

Avižienis et al. [23] present a taxonomy of dependable and secure computing which unifies the previously separate concerns of dependability which is defined as the ability of delivering service that can justifiably be trusted by avoiding service failures that are more frequent and more severe than is acceptable, and security which is defined

as satisfying all of the conditions of availability for authorized actions only, no disclosure of information (confidentiality), and absence of unauthorized system alterations (integrity). Figure 2.17 [23] shows the top levels of the dependability and security taxonomy.

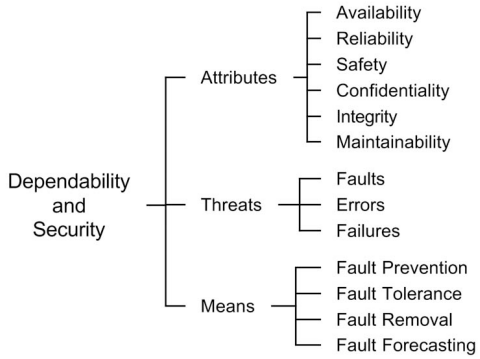

Figure 2.17: Taxonomy of dependable and secure computing [23]

The following sections describe the top categories of the taxonomy which are attributes of dependability and security, threats to dependability and security, and means to attain dependability and security.

Attributes of dependability and security

The identified attributes of dependability and security are availability, reliability, safety, confidentiality, integrity, and maintainability. Availability is defined as readiness for correct service, reliability as continuity of correct service, safety as absence of catastrophic consequences on users and environment, integrity as absence of improper (i.e. unauthorized) system alterations, and maintainability as ability to undergo modifications and repairs. Confidentiality is the absence of unauthorized disclosure of information.

Empirically, dependability may be identified by the condition of other systems actually depending on the observed system. In this case the dependability of systems that depend on the observed system is affected by the dependability of the observed system. The condition of dependence on another system being accepted constitutes trust.

Threats to dependability and security

Threats to dependability and security are faults, errors, and failures. An error is defined as a deviation of a system's external state from the correct state, while a fault is the adjudged or hypothesized cause of an error. A failure is an event where the delivered service deviates from the correct service, with a service being defined as a sequence of external states.

Faults are classified by eight binary viewpoints of creation phase (development or operation), system boundaries (internal or external), cause (natural or human-made), dimension (hardware or software), objective (malicious or non-malicious), intent (deliberate or non-deliberate), capability (accidental or incompetence), and persistence (permanent and transient). Combination of these viewpoints results in 256 theoretical manifestations, of which 31 are identified as being likely. For example, malicious faults are always deliberate, and natural faults always affect hardware and are non-malicious and non-deliberate. These 31 fault classes are assigned to the partially overlapping groupings of development faults that occur during development, physical faults that affect hardware, and interaction faults that include all external faults.

Failures are classified into service failures, development failures, and dependability failures. *Service failures* result from deviation of delivered service to correct service and are characterized by the viewpoints of failure domain (content or timing), detectability (signaled to the user, or not signaled), consistency (consistent, i.e. perceived identically by all users, or inconsistent), and consequence (between minor and catastrophic). Systems that are designed and implemented to fail only in specified modes to a specified limited extend are called fail-controlled systems. A system where only minor failures may occur is called a fail-safe system. *Development failures* may be complete or partial, with complete development failures resulting in the termination of the development process and the system not being placed into service, and possible results of a partial development failure being budget overrun, schedule overrun, or downgrading of functionality, performance or predicted dependability and security. *Dependability and security failures* occur due to service failures that are more frequent or more severe than acceptable according to a previously agreed-on dependability and security specification.

Means to attain dependability and security

Means to attain dependability and security are fault prevention, fault tolerance, fault removal, and fault forecasting. *Fault prevention* aims to prevent the occurrence or introduction of faults. It is a goal of software and hardware development methodologies and is not further discussed within the taxonomy.

Fault tolerance aims to prevent service failures in the presence of faults. The categories of fault tolerance are error detection and system recovery. *Error detection* identifies the presence of errors and occurs as *concurrent detection* which takes place during normal service delivery, and *preemptive detection* that takes place while normal service delivery is suspended. *System recovery* transforms the system into a state without detected errors or faults that can be activated again, and is subcategorized into *error handling* that eliminates errors from the system and *fault handling* that prevents faults from being activated again.

Fault removal reduces the number and severity of faults. In the development cycle, the steps of fault removal are *verification* that a system complies to a set of given properties, diagnosis of the faults detected during verification, and performing necessary corrections. Verification is categorized into *static verification*, where the program

is not executed and *dynamic verification* that executes the program to be verified. Dynamic verification of a program with real input is called *testing*. Fault removal in the system use phase is either *preventive maintenance* that removes faults that have not yet produced errors or *corrective maintenance* that removes faults after errors have been reported.

Fault forecasting finally estimates the present number, future incidence and consequences of faults. The categories of fault forecasting are ordinal or *qualitative evaluation* that aims to identify and classify possible failure modes and probabilistic or *quantitative evaluation* that aims to evaluate the probability of system attributes (measures) being satisfied.

3 Concepts and terminology

This chapter provides an introduction to the concepts and terminology relevant to the survey and taxonomy of autonomic large-scale computing. Section 3.1 introduces the field of grid computing. Section 3.2 gives an introduction to cloud computing, and Section 3.3 finally describes autonomic computing.

3.1 Grid computing

The Grid [69] is a distributed computing infrastructure that has originally been created in order to enable organizations to share computing resources for the purpose of scientific computing in order to solve computationally intensive problems. The term *grid computing* refers to the electric power grid and suggests that from a user perspective, the location and other details of the used resources are as transparent as the origin of the electric power that is consumed.

Foster et al. [71] define the grid problem as

> coordinated resource sharing and problem solving in dynamic, multi-institutional virtual organizations.

This definition outlines that the sharing of resources – such as computing power, storage, or data – has to be coordinated. Since users and resources typically live within different control domains (e.g., companies, university sites), resource providers need to employ sharing policies for specific user groups. A set of organizations with associated resource sharing rules is called a virtual organization (VO). Actual (physical) organizations are entitled to enter or leave virtual organizations subject to the sharing rules established within the VO.

The concept of a virtual organization enables competitors in some industry field to collaborate within a joint project by sharing a subset of their resources, since membership of an actual organization in a VO may be temporary, e.g. limited to the lifetime of some project. Figure 3.1 [71] shows the relationship between actual organizations and VOs, where the ovals denote actual organizations, and P and Q are virtual organizations.

The following kinds of resources may be shared [71]:

- Computational resources
- Storage resources
- Network resources

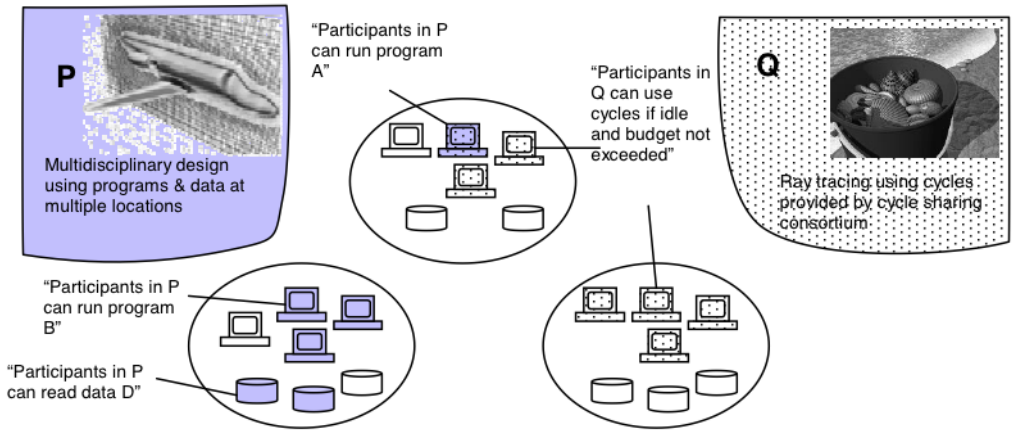

Figure 3.1: Sharing relationships within virtual organizations [71]

- Code repositories
- Database catalogs

Section 3.1.1 provides a high-level architectural description of the grid [71] and introduces the Open Grid Services Architecture (OGSA) that implements that high-level architecture. Section 3.1.2 presents a selection of grid projects.

3.1.1 Grid architecture

A high-level architectural description of the grid [71] consists – from bottom to top – of a fabric layer which provides resource-level operations, a connectivity layer for data exchange between fabric-level resources (including security aspects), a resource layer for obtaining information about and negotiating access to resources, a collective layer for operations on a collection of resources, and finally an application layer which contains the user applications.

The grid architecture follows a "hourglass" model, with the resource layer being the hourglass's neck. So, while the other layers are allowed to offer a wide range of protocols, the protocol set offered in the resource layer should be restricted.

The grid architecture and its relation to the Internet protocol architecture are shown in Figure 3.2 [71].

Fabric layer The fabric layer comprises the resources which are shared on the grid. Resources may be physical resources like storage or computing power which are accessed directly by the respective protocols of the grid fabric layer. Logical resources include computer clusters and distributed file systems which are accessed by the means of another protocol like a cluster resource manager or NFS. These protocols are not part of the grid architecture.

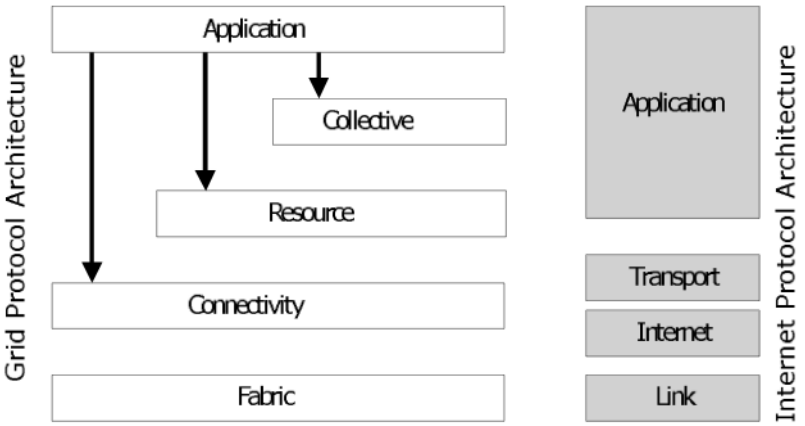

Figure 3.2: Grid architecture [71]

Since different resource implementations offer different capabilities, it is recommended that the protocol includes introspection mechanisms to have the higher level protocols query the respective capabilities. For example, some resource implementations offer advance reservation, which enables applications (e.g. workflow management systems) to plan for resource utilization.

Connectivity layer The connectivity layer defines communication protocols that are used for data exchange between fabric-layer resources, and authentication protocols to verify the identity of users and resources. While for communication, usually the protocols of the IP stack are used, the grid architecture allows for adaptation, if the need for new protocols should arise.

Authentication solutions within the connectivity layer should be based on existing standards. Since they operate on a large scale – potentially on a global scale – they should support single sign-on and delegation of rights, and they should integrate with existing local authentication solutions like Unix authentication and Kerberos.

Resource layer The resource layer defines protocols for the sharing of individual resources. Information protocols on the resource layer provide information about a resource's state and structure, while management protocols allow for the negotiation of access to a resource.

The resource layer forms the neck of the grid architecture's "hourglass" model. So the protocol set offered on this layer should be small and focused.

Collective layer The collective layer offers protocols for handling collections of resources. This includes directory services which allow to discover resources, software discovery services, co-allocation services that allocate a group of resources, data replication, monitoring and diagnostics.

Services on the collective layer may either be general-purpose or developed for the specific need of some virtual organization or application domain.

Application layer The application layer finally comprises the user applications that are running on a grid and invoke the protocols and services described above. Applications may call protocols and services provided by frameworks in the grid application layer.

The Open Grid Services Architecture

The Open Grid Services Architecture (OGSA) [70] is based on a service-oriented architecture and defines a set of protocols and interfaces that are implemented as a de facto standard in the Globus Toolkit [68], which is described in Section 3.1.2. OGSA introduces the concept of a grid service, which is an extension of a web service that allows for creation, destruction and stateful invocation of services. Thus, concepts originally developed for web services can be applied to grid services.

Grid services implement one or more interfaces that consist of a set of operations and offer some capabilities. A grid service may implement functions on resource or collective level.

3.1.2 Reference projects

This section introduces selected grid infrastructure and middleware projects that form the basis of a large part of currently existing grid infrastructure. While these projects do not feature autonomic capabilities themselves, some of the autonomic grid projects that will be surveyed in Section 4.1 are based on one of the middleware stacks presented here.

Globus toolkit

The Globus toolkit [68] is a widespread software toolkit that provides tools and an API to build applications for. The current version 4 implements the Open Grid Services Architecture (OGSA) described in Section 3.1.1 and is built on top of the Web Services Resource Framework (WSRF)[1].

The Globus toolkit includes service implementations such as the GRAM (Grid Resource Allocation and Management) execution manager and GridFTP for data access and data movement between nodes, containers for user-developed services written in various programming languages, a security infrastructure, and tools for building new grid services, and a set of client libraries and command line programs that allows accessing these services and capabilities from various environments. Figure 3.3 [68] shows the architecture of the Globus toolkit, some of its components and possible interactions with other components of a grid system.

[1] OASIS Web Services Resource Framework, `http://www.oasis-open.org/committees/wsrf/`

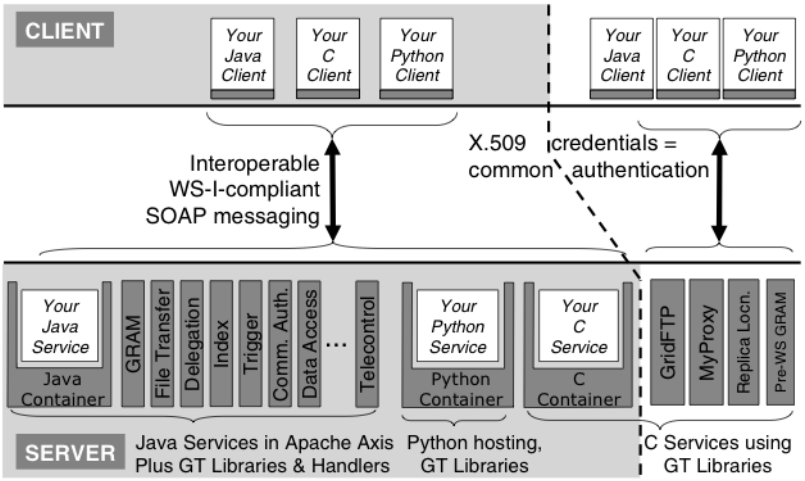

Figure 3.3: Globus toolkit architecture [68]

From a functional point of view, the components of the Globus Toolkit cover execution management, data management, information services, and security. Execution management is provided by the Grid Resource Allocation and Management (GRAM) service mentioned in the previous section. GRAM provides a web services interface for managing execution of computations like individual tasks, services, or subtasks on remote computers. Data management components of the Globus Toolkit are the GridFTP service that provides high-performance memory-to-memory and disk-to-disk data movement, the reliable file transfer (RFT) service that manages multiple GridFTP transfers, the replica location service (RLS) that provides access to location information about replicated files and data sets, and data access and integration tools (OGSA-DAI) for access to relational and XML data.

Several grid projects are based on the infrastructure and protocol stack provided by the Globus Toolkit. Projects directly based on the Globus Toolkit include NSF TeraGrid [154] and the cancer Biomedical Informatics Grid (caGrid) [162]. Grid infrastructure that is based on a different protocol stack often provides gateways to Globus-based grids, making the Globus Toolkit a de facto standard for grid computing.

UNICORE

Unicore (Uniform Interface to Computing Resources) [62] is a grid platform operated by Unicore Forum e.V., a consortium of European high-performance computing centers. It originated as a software infrastructure interconnecting computing centers in Germany over the Internet, in order to allow users to submit batch jobs with automatic data staging to be executed on heterogeneous systems at multiple remote sites. After computing paradigm had emerged, the follow-up “Unicore Plus” project enhanced the existing software into a grid middleware system that now includes client

and server software, is based on standards like the Open Grid Services Architecture and is implemented in Java for platform independence.

Unicore is based on a three-layered architecture consisting of a client layer, a service layer, and a system layer. The client layer provides several types of user interfaces like a command line client called *ucc*, an Eclipse[2] plugin, a high level API (HiLA), and a web portal. These clients support the management of batch jobs that allocate resources at multiple sites. The job preparation agent which is a client component is responsible for ensuring syntactical correctness of the jobs before submission. The service layer consists of an authentication gateway that authenticates requests using X.509 certificates and passes them to a network job supervisor which maps abstract requests into concrete jobs. The concrete jobs then are handed to target system interface daemons forming the interface to the system layer. The system layer finally consists of the target hosts which execute the user jobs.

Besides job management, Unicore supports storage access, file transfer, workflow management, security and information services. While Unicore is based on its own infrastructure, it is interoperable with Grid infrastructure based on the Globus Toolkit described above.

Alchemi

Alchemi [121] is a grid computing framework developed at the University of Melbourne, Australia. While conventional grids are usually formed by interconnecting Unix-based high-performance computers, Alchemi allows to build a grid from idle resources of Windows-based desktop computers. Such a system is called an enterprise grid or desktop grid. Alchemi is impemented on top of the Microsoft .NET framework, allowing grid applications to be written in a language supported by .NET like C#.

Together with the Gridbus Grid Service Broker [40] which has been extended with an Alchemi actuator besides the existing Globus actuator, the Alchemi framework allows to execute jobs both on Globus and Alchemi resources. Figure 3.4 [121] shows the relation between the Globus toolkit and Alchemi within a grid architecture integrating both high performance computing and desktop resources.

The basic deployment scenario of Alchemi is a cluster of desktop computers consisting of a manager node that manages multiple executor nodes. However, within a multi-cluster environment, a manager may connect to other managers in a hierarchical fashion. Within the global grid scenario described in the previous section, the top Alchemi manager node is managed by a cross-platform manager. A grid broker connects both to Globus-based resources and the Alchemi cross-platform manager. Desktop computers that share resources may execute jobs either when triggered by a screen saver, or periodically. Alchemi uses .NET remoting for communication between its components, and web services for the public interface.

[2]Eclipse IDE, `http://www.eclipse.org`

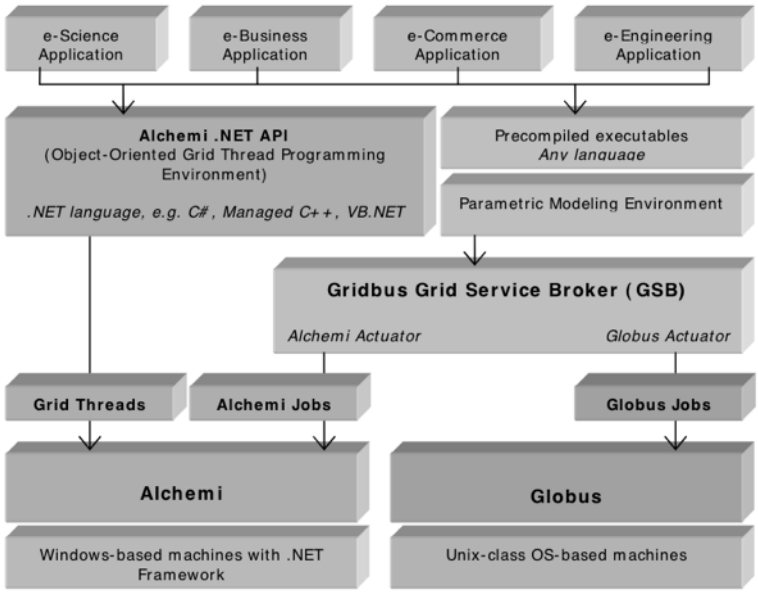

Figure 3.4: Layered grid architecture integrating Globus and Alchemi [121]

NAREGI

NAREGI [125] is a project instituted in Japan under which a grid middleware stack, a networking infrastructure, and scientific applications have been developed. Figure 3.5 [125] shows the NAREGI grid middleware stack.

The lower and middle-tier layer of the middleware is responsible for resource management and scheduling. It consists of a superscheduler that supports resource management across virtual organizations, the GridVM local resource controller that offers a virtual layer of computing resources, and information services for resource discovery. The programming environment supports message passing (GridMPI) and remote procedure call (GridRPC) mechanisms adapted for grid applications. Ninf-G [171] is a reference implementation for GridRPC developed within the NAREGI project, that operates using basic services from the Globus toolkit like GRAM and MDS. The application environment of NAREGI finally includes a visual workflow tool that allows managing distributed jobs independent of specific Grid middleware, the GridPSE problem solving environment that allows collaboration between distributed simulation applications, and a real-time visualization system that allows to present simulation results. The networking subsystem consists of a network function infrastructure that supports bandwith control and policy-based QoS routing, and a communication protocol infrastructure.

The NAREGI middleware is designed to interoperate with resources managed by Globus or the Unicore system. It uses the functions provided by the Globus toolkit for tasks like basic security checking, job launching, and file transfer.

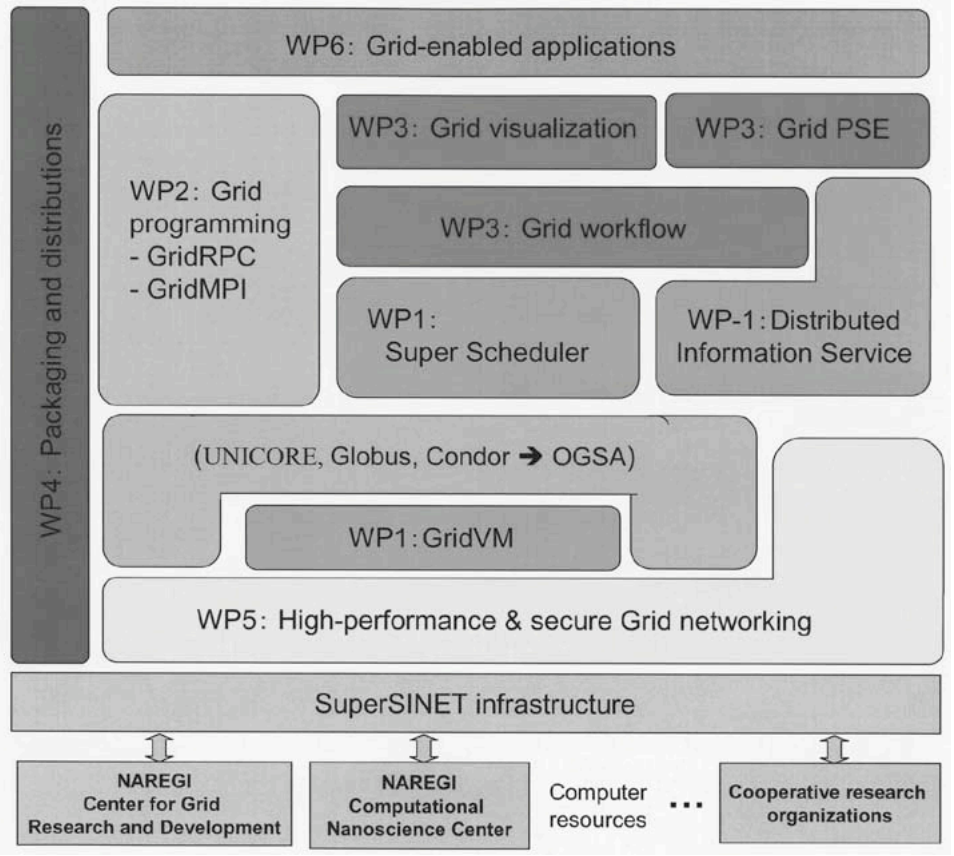

Figure 3.5: NAREGI grid middleware stack [125]

3.2 Cloud computing

Cloud computing has emerged as a new computing paradigm in the field of parallel and distributed computing, where applications, software platforms, infrastructure and hardware resources are provided for a fee, and are accessible globally over the Internet. The term *cloud computing* denotes that from a user perspective, the applications and infrastructure reside on a "cloud" on the Internet and are accessible on demand from any location.

As cloud computing is a novel concept, there has not yet been an agreement on a single definition. Several definitions have been proposed [181], and two of those definitions will be presented here. Buyya et al. [41] propose the following definition:

> A Cloud is a type of parallel and distributed system consisting of a collection of inter-connected and virtualized computers that are dynamically provisioned and presented as one or more unified computing resource(s) based on service-level agreements established through negotiation between

the service provider and consumers.

As the definition suggests, virtualization is a prerequisite technology for cloud computing. Providers often are commercial operators who own large data centers, where they set up a hypervisor and offer access to virtual machines (VM) to the public.

Clouds resemble grids in that they enable access to high-performance remote computing resources. However, where grids originate from the research community and offer access to resources within a community called virtual organization, clouds at the moment are usually offered by commercial operators and provide resources to the public based on a resource pricing model.

Foster et al. [72] offer another definition for cloud computing:

> [Cloud computing is] a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet.

This definition stresses that unlike in the case of grid computing, where a defined set of resources is shared within a virtual organization, clouds are massively scalable and deliver services to customers on the Internet.

Section 3.2.1 presents an ontology which categorizes the services currently offered on the cloud into several layers. Section 3.2.2 describes the high-level architecture of a typical cloud system.

3.2.1 Cloud ontology

Since cloud computing is a new field, little standardization has been done yet. However, cloud systems can be categorized as falling into one of five layers [190], which are (from bottom to top):

- a firmware/hardware layer (hardware as a service, HaaS),
- a cloud-specific software kernel,
- an infrastructure layer consisting of
  - computational resources (infrastructure as a service, IaaS),
  - storage resources (data-storage as a service, DaaS) and
  - communication facilities (communication as a service, CaaS),
- a software environment layer (platform as a service, PaaS), and finally
- an application layer (software as a service, SaaS).

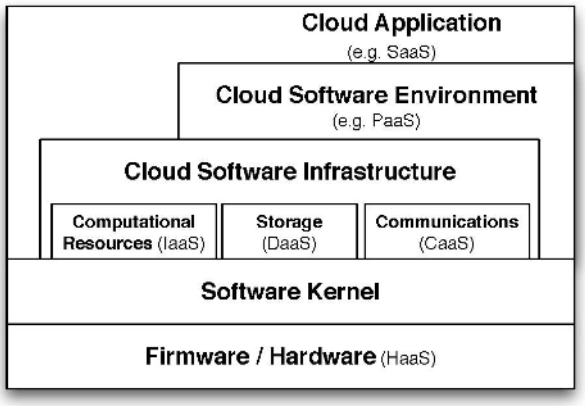

Figure 3.6: Layers within a cloud ontology [190]

These layers are depicted in Figure 3.6 [190].

Existing cloud systems may be categorized as operating on one of the layers proposed. For example, while Google Apps [83] provides software as a service (SaaS) and thus is situated on the application layer, the engine behind Google Apps [82], which is offered as a service itself, falls in the platform as a service (PaaS) category and thus is situated on the software environment layer.

The following paragraphs give a description of the cloud ontology layers.

Infrastructure as a Service (IaaS) The Infrastructure as a Service category contains cloud projects, where infrastructure operators directly provide physical or virtual hardware resources through a software stack. This layer may be further subdivided [190] into Infrastructure as a Service (IaaS) projects in the narrow sense, Data-storage as a Service (DaaS[3]) – sometimes called Disk as a Service – and Hardware as a Service (HaaS). The difference between these cloud service types basically lies in the kind of resources being made available, the protocols used to access the services, and the availability of intermediate management layers.

Platform as a Service (PaaS) Cloud projects in the Platform as a Service category offer a development platform for applications, mainly consisting of a runtime environment and an API. This allows developers of cloud applications to use the same cloud environment for development, where the finished application finally will be deployed, while avoiding setup costs of the development environment and eliminating a source of possible application errors.

Software as a Service (SaaS) Software as a Service denotes applications hosted on cloud systems which are available to the end-user usually via a web interface. For

[3]The abbreviation DaaS is sometimes used to denote Database as a Service, which will be handled as a special case of Platform as a Service (PaaS) here.

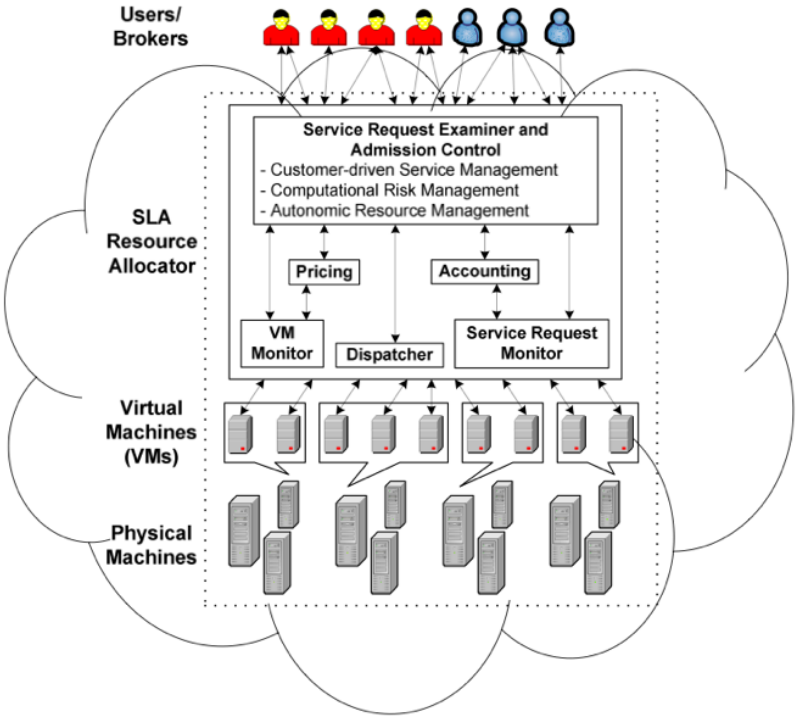

Figure 3.7: High-level cloud architecture [41]

the end user, this model has the advantage that application and data are available through any Internet connection, and that there is no need of installing, configuring and maintaining office applications on local systems. Service providers also save development and maintenance cost, since software only needs to be deployed within the provider's data center, thus reducing the number of test cases in the software testing process.

3.2.2 Cloud architecture

Buyya et al. [41] describe a high-level cloud architecture consisting of four entities involved when providing resources on the cloud. Within this architecture, cloud systems offer multiple virtual machines that are operated on top of a group of physical machines using a hypervisor. The virtual machines are managed by a SLA resource allocator that offers functionality for service request examination and admission control, pricing, accounting, VM availability monitoring, dispatching and monitoring of accepted service requests.

Figure 3.7 [41] shows the high-level architecture briefly described in the previous paragraph, which will be further outlined in the remainder of this section.

Users/brokers The top layer displayed in the aforementioned figure shows *users* who submit service requests to the cloud, and *brokers* acting on their behalf.

SLA resource allocator The SLA resource allocator consists of a *service request examiner and admission control* which is responsible for accepting or rejecting service requests submitted by users or brokers based on QoS requirements, and passes accepted requests to the *pricing* mechanism that decides how the request will be charged, the *accounting* mechanism maintaining the actual resource usage, the *VM monitor* tracking availability of VMs and their assigned resources, the *dispatcher* responsible for starting the requested services on allocated VMs, and the *service request monitor* that keeps track of service execution.

Virtual machines Services offered on the cloud are provided using virtual machines (VMs). Multiple VMs may be run concurrently on a single physical machine in isolation from each other, where different operating environments may be installed on the virtual machines running on the same physical machine, in order to provide a customized environment for each service offered on the cloud.

Physical machines Multiple physical machines are operated within the data center in order to provide scalability. Machine types may range from commodity hardware to specialized high-end servers.

3.2.3 Reference projects

This section presents some reference cloud computing projects that cover one or more of the cloud service layers introduced in Section 3.2.1. Table 3.1 shows which cloud service layers are covered by the reference projects.

Industrial projects

Several cloud projects are actively marketed by the computing industry. The following paragraphs present a selection of commercial cloud projects.

Amazon Web Services Amazon EC2 (elastic compute cloud), part of Amazon Web Services [10], is based on the Xen hypervisor[4] and provides a computing environment in form of virtual machines, called Amazon machine images (AMI). Users may either choose from one of the pre-configured virtual machines or configure and upload their own. Amazon EC2 uses Amazon S3 (simple storage service) for persistence of the virtual machine images. Amazon S3 falls in the Data-storage as a Service (DaaS) subcategory of IaaS according to the definitions provided in Section 3.2.1.

While Amazon S3 is priced for used disk space, data transfer and number of requests, EC2, besides charging for data transfer, offers pricing per usage hour (i.e. virtual

[4]Xen hypervisor, `http://www.xen.org`

	IaaS	PaaS	SaaS
Industrial projects			
Amazon Web Services	x	x	
Google Apps		x	x
HP Flexible Computing Services	x	x	x
IBM Smart Business Services	x	x	x
Microsoft Windows Azure		x	
Microsoft Office Live			x
Salesforce.com		x	x
Academic projects			
Aneka		x	
Eucalyptus	x		
Nimbus	x		
RESERVOIR	x		

Table 3.1: Cloud service layers covered by reference projects

machine instance uptime) or pricing per year. Amazon also provides a monitoring service called Amazon CloudWatch as an add-on to EC2 which might be used for implementing autonomic capabilities in the future.

Besides EC2 and S3, Amazon offers simple database, queueing, and MapReduce services within their web services [10].

Amazon Elastic MapReduce is based on Apache Hadoop[5] and offers an implementation of Google's MapReduce programming model [56]. MapReduce requires an Amazon EC2 instance and is priced per hour. Amazon SimpleDB offers a database service which eliminates the need of defining a database schema and indexes by offering a pre-defined data model based on domains which consist of attribute-value pairs. It is priced per hour, data transfer, and storage size. Amazon Simple Queue Service (SQS) provides the user with message passing functionality and is priced based on number of requests and data transfer.

Amazon MapReduce, SimpleDB, and SQS offer services to application developers in order to avoid the need to set up and maintain the respective systems on their own site. Thus, they fall into the Platform as a Service category.

Google Apps Google Apps [83] provide the user with e-mail, calendar, office applications like word processing or spread sheets, instant messaging, and web publishing tools. Documents from existing office applications may be uploaded and edited using Google apps. Documents may either be made publicly available, or be shared with selected other Google account owners in order to enable collaboration.

Thus, Google Apps provides a standard office environment, eliminating the need

[5]Apache Hadoop, `http://hadoop.apache.org/`

of configuring (or even owning) a personal computer at the cost of increasing the dependence on an available Internet connection and requiring trust in the application provider regarding data integrity, availability, and confidentiality. Legal concerns may also apply, since potentially confidential data may be migrated across country borders.

The Google App Engine [82] provides the API used by Google Apps in order to support the development of custom applications which are interoperable with Google Apps, e.g. by requiring the user to sign in using a Google account. Currently, either Python or Java may be used as programming languages. A simple application server is provided to allow offline development.

Google solicits SDK patches from developers in order to improve their API. Applications may be published in a gallery, in order to be deployed by other users.

HP Infrastructure Provisioning Service Infrastructure Provisioning Service (IPS) is an offering within HP's Flexible Computing Services (FCS) [96], which provides access to server, storage, and operating system platforms and associated system management tools [102].

The HP Infrastructure Provisioning Service plus Scheduling (IPS+) is part of Hewlett-Packard's Flexible Computing Services [96] and offers the hardware and operating system services described above plus grid-management, scheduling, compilers, and other development software [102].

The Application Provisioning Service (APS) is an option provided on top of the infrastructure and platform services offerings. Besides allowing the installation of customer-provided applications, applications for computer aided engineering (CAE) are offered, which target the oil and gas industries, and financial services [102].

IBM Smart Business Services IBM offers a cloud computing platform within their Smart Business Services [101]. Infrastructure offerings include the Smart Business Desktop Cloud, which offers a virtual operating system environment (Windows or Linux) targeted at desktop users. The Smart Business Test Cloud allows customers to deploy cloud services within their private network, using IBM CloudBurst hardware. Thus, applications can be migrated between the customer's private network and IBM's data center.

Microsoft Windows Azure and Office Live Microsoft Windows Azure [129] is a cloud-based operating system and development platform. The Windows Azure platform allows to develop applications using existing Microsoft technologies like .NET[6] that can be deployed either locally or on the cloud. Thus, existing applications may be migrated to the cloud.

The Azure services platform consists of the Windows Azure environment, AppFabric (formerly called .NET services) that offers a distributed infrastructure for developers, SQL Azure providing database access, and the codename "Dallas" service that offers access to content collected from public data sources in a central location using an API

[6]Microsoft .NET, `http://www.microsoft.com/NET/`

based on representational state transfer (REST), which is an architectural style for building web services.

Office Live [128] is another cloud platform from Microsoft that provides an online data store where users may save documents, organize them within workspaces, share workspaces for collaboration, and view and edit documents in a web browser. Document types supported include Microsoft Office documents and Microsoft Outlook items.

Salesforce force.com Salesforce.com [159] is a customer relationship management (CRM) tool offered as a service. Salesforce.com consists of a sales and a service cloud, each denoting the respective business process within customer relationship management.

The force.com [158] platform allows for the development of enhancements to applications provided within the Salesforce.com customer relationship management platform. Besides customizing the database and user interface, business workflows may be created, and business applications may be developed.

Academic projects

The following paragraphs present a selection of cloud projects that have been initiated by academic institutions. This includes projects where academic institutions cooperate with the computing industry.

Aneka Aneka [52, 41] is a software platform for cloud computing based on .NET. Originating from enterprise grid computing[7], it has been adapted to support the cloud paradigm, enabling free resources of individual desktop computers to form an enterprise cloud. To achieve this goal, an Aneka container is configured on each desktop computer to provide basic functionality like persistence, security, and communication. The container is able to host additional services in order to support functionality like indexing or scheduling. Requesters communicate with the Aneka cloud through the Gridbus broker [183], which also enables advance reservation and negotiation on service level agreements.

Aneka supports different application development models like grid task, grid thread, and MapReduce [56], and allows for the definition of custom models.

Eucalyptus Eucalyptus [139] is an open source cloud infrastructure project targeted at researchers who want to explore cloud capabilities. It can be set up on commodity hardware – ranging from a single laptop to a cluster – in a non-dedicated fashion, allowing the usage of available hardware. The design of Eucalyptus is modular in order to allow the replacement of components for research purposes. It emulates the API of Amazon EC2 described on page 64 in order to allow using EC2's command line tools.

[7] An enterprise grid is a grid that provisions the resources of desktop computers.

Nimbus The Nimbus science cloud [136] developed at University of California is a public cloud for the scientific community. Its focus is to provide a cloud infrastructure to domain scientists familiar with grid computing. Nimbus supports Amazon EC2 and Globus WSRF interfaces, thus allowing usage of both Amazon's client tools and existing grid computing tools. Virtual machines can be scheduled using the Portable Batch Scheduler (PBS). Components of Nimbus are designed to be replaceable.

RESERVOIR The RESERVOIR project – Resources and Services Virtualization without Barriers [155] – is a result from a cooperation between IBM, SAP, Telefónica I+D, and three European universities. It addresses the limited scalability of single-provider clouds, the current lack of interoperability among cloud providers and supports Business Service Management (BSM) which is a strategy for measuring IT services from a business perspective by managing service-level agreements.

In order to achieve the goals outlined above, an architecture is employed which has infrastructure providers operate one or more virtual execution environments (VEE) per site. Service providers distribute the execution of their applications among the VEE hosts that may be located at the same or at different sites. Applications that are deployed on the RESERVOIR cloud include a service manifest that formally defines an SLA between the respective infrastructure provider and service provider. The service manifest specifies parameters like minimum and maximum number of CPUs, memory size, and application instances. RESERVOIR distinguishes between explicit and implicit modes of capacity provisioning, where the former mode has the service provider explicitly state their capacity needs (sized applications), and the latter one lets the infrastructure provider estimate the resource requirements and allocate resources according to a high-level policy which usually includes the goal to avoid over-provisioning.

Components of the RESERVOIR cloud include the service manager which is responsible for interaction with the service providers, deploying and provisioning VEEs, and ensuring SLA compliance by adjusting application resource usage. Another component is the virtual execution environment manager (VEEM) that assigns VEEs to VEE hosts subject to policies specified by the service manager. If permitted by policy, VEEMs may place VEEs at remote sites operated by a different cloud provider, thus creating a federation of clouds. The virtual execution environment host (VEEH) finally is responsible for managing individual VEEs on a specific virtualization platform.

3.3 Autonomic computing

The autonomic computing paradigm has been proposed [107] as a solution for handling the complexity involved in managing large-scale distributed computing systems and applications. The term *autonomic computing* has been chosen for the concept's resemblance to the autonomic nervous system in biology, which is responsible for maintaining a stable state of an organism (e.g. the human body) by adapting parameters like blood pressure or pulse as a reaction to a changing environment (e.g. outside

temperature) on a subconscious level. Similarly, autonomic computing systems adapt to changing environment conditions (e.g. system load) by tuning system parameters without intervention of a system operator in order to maintain a stable system state. Human operators are solely responsible for providing high-level guidance in form of policies, from which actions of the autonomic system are derived.

The remainder of this section is organized as follows: Section 3.3.1 introduces the characteristics that define an autonomic system. The general architecture of an autonomic system is discussed in Section 3.3.2. An adoption model for transitioning existing systems into autonomic systems is given in Section 3.3.3. Section 3.3.4 finally introduces some autonomic projects.

3.3.1 Defining characteristics of an autonomic system

Autonomic computing may be viewed as a next step in relation to existing technologies like fault-tolerant systems and proprietary system management solutions that currently help system administrators maintain systems operable. Thus, the following eight characteristics have been proposed to define an autonomic system, as opposed to existing technologies [99, 91, 79]:

Self-awareness Autonomic systems need to be aware of their capabilities, current status, allocation of resources and connections to other systems.

Context awareness Autonomic systems shall be aware of their operating environment, available resources and other systems to interact with.

Openness Since autonomic systems need to interact with other systems, they shall be implemented using open standards.

Anticipativeness Autonomic system shall support proactive management of their resources by anticipating future environment changes and usage profiles.

The remaining four aspects of self-management are also known as *self-** or *self-star properties* [107]:

Self-configuration Autonomic systems shall configure themselves according to high-level guidelines. Typical self-configuration actions include software upgrades (including the possibility of a rollback in case of problems with the new software version) or installation of new components.

Self-optimization Autonomic systems shall continually seek ways to optimize themselves, e.g. by adjusting tunable software parameters. Optimization shall happen proactively, i.e. under the current environment conditions, or in reaction to degrading performance.

Self-healing Autonomic systems shall detect failures and issue appropriate measures to maintain an operable state by correcting or circumventing the underlying problem.

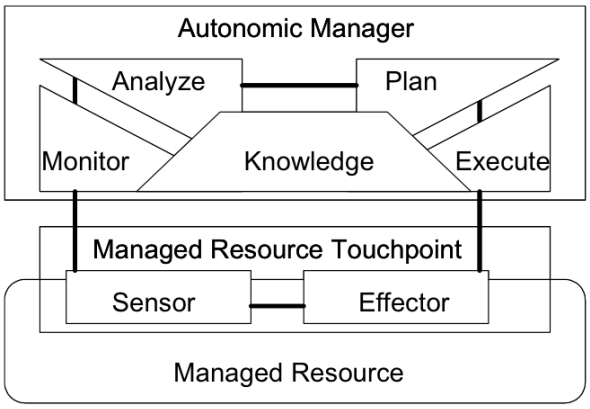

Figure 3.8: Autonomic element with control loop [104]. An autonomic system is composed of autonomic elements.

Self-protection Autonomic systems shall be able to detect and protect from internal and external attacks, viruses or unauthorized access.

3.3.2 Architecture of an autonomic system

On a conceptual level, an autonomic system is composed of one or more *autonomic elements* interacting with each other, where the scope of an autonomic element may be determined by the underlying system architecture. For example, in a service-oriented architecture an autonomic element may correspond to a service.

Figure 3.8 [104] shows that each autonomic element consists of an *autonomic manager* and a *managed resource*. While the former implements autonomic behavior of the autonomic element, the latter implements the service's functionality. This architecture allows introducing autonomic behavior to existing services by adapting the existing service into a managed resource and then adding the autonomic manager.

Autonomic manager

The autonomic manager shown in the aforementioned figure implements the *autonomic control loop* (sometimes called MAPE-K loop) which consists of monitoring, planning, analysis, and execution steps and is guided by knowledge. While these steps describe the functionality required of the autonomic control loop and suggest its design, they do not prescribe a control flow, though. So for example it is possible for planning to call monitoring in order to get additional information.

The following paragraphs describe the individual steps of the autonomic control loop.

Monitoring Monitoring collects data using the sensors of the managed resource. The monitoring step includes filtering and aggregation of raw data. Monitoring may occur

in a continuous mode (e.g. using counters) or by request (e.g. execution of queries).

Analysis Analysis provides mechanisms to correlate data from monitoring. Forecasting techniques may be applied in order to identify problems that may be passed to planning. Analysis may request additional data from monitoring if needed.

Planning Planning provides actions based on analysis that allow achieving system goals. Planning is guided by policies described below and results in actions passed to the execution step.

Execution Execution carries out the mechanisms required to execute a plan using the capabilities provided by the managed resource's effector.

Knowledge The autonomic control loop is guided by knowledge accessible by each of its four steps. Knowledge consists of configuration and management data like symptoms, policies, requests for change, and change plans [100].

Policies may be specified at different levels of abstractions which are shown in Figure 3.9 and described below.

Managed resource

A managed resource is a hardware or software component or a collection of components that is managed by the autonomic manager. The autonomic manager controls the managed resource by collecting data from the *sensor* and issuing configuration changes through the *effector*.

Sensor The sensor provides mechanisms to collect information about the managed resource, either using methods which can be called by the autonomic manager, or by sending messages that may be collected.

Effector The effector allows changing the state or configuration of the managed resource. Possible implementations of an effector include a configuration API, message passing, or a web service.

Policies

According to the defining characteristics of an autonomic system as described in Section 3.3.1, an autonomic system shall be aware of its capabilities and current state, which has to be represented as some form of knowledge. In order to represent knowledge, the policy types described in the following paragraphs and presented in Figure 3.9 [108] have been proposed [108, 97, 156]:

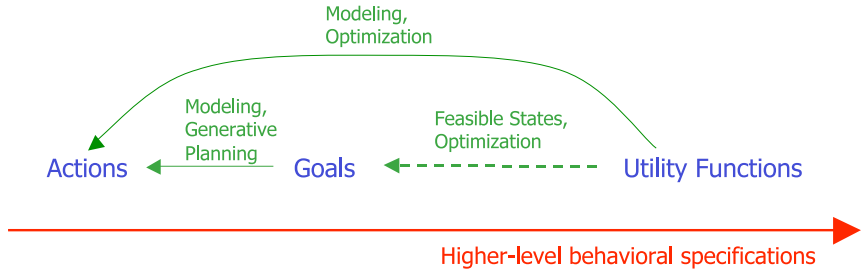

Figure 3.9: Knowledge representation policies [108]

Action policies Action policies express what action shall be taken if a system currently is in a given state. Action policies are specified using rules that follow the general pattern of *if condition then action*, where the condition implicitly represents the current system state and the action the desired transition from this state. However, action policies do not specify the target state that will be reached by executing an action. Action policies typically are formulated using a policy language. Many policy languages express policies using event-condition-action (ECA) rules, where the event triggers the evaluation of the condition which may result in execution of the action. Within those policy languages, event and condition together represent the current system state.

Action policies are executed immediately and thus consume few system resources. However, conflicts may occur between different policies in an action policy set. Methods for resolving such conflicts include adding meta-policies to the policy set that explicitly specify actions for conflicting states, eliminating overlapping conditions from the policy set so that at most one policy matches each possible condition, or assigning priorities to individual policies so that in case of a conflict, only the policy with the highest priority is executed. For many applications including autonomic computing it is furthermore required that the action policy set covers the whole state space and provides a single unique action for each state. Goal policies and utility functions which are described in the following paragraphs are not subject to conflicting policies.

Goal policies Goal policies specify a desired system state and leave it to the system to derive actions in order to make a transition from the current to the desired state. More than one desired state may be given explicitly or by specifying criteria, among which the system may chose at random. In order to translate goal policies to action policies, a system needs to have a model of itself that specifies possible system states and a mapping from actions to state transitions. Based on that model, generative planners [156] may translate goal policies to action policies.

As long as the goals themselves are not contradictory, goal policies are not subject to policy conflicts since the system is responsible for taking appropriate actions in order to meet the goals. However, goal policies do not specify what action a system shall take if a desired state cannot be reached. While possible actions in that case include

	Action policies	Goal policies	Utility policies
Conflicting policies impossible	–	+	+
Unspecified states impossible	+	–	+
Maintainable for human operator	+	+	–

Table 3.2: Policy types in autonomic computing

giving up on a subset of goals in order to allow satisfaction of the complementary subset, or for quantifiable goals like server response times to allow a small deviation from all of the goals, goal policies do not allow to express such tradeoffs, since they only distinguish between acceptable and unacceptable system states.

Utility policies Policies based on utility functions assign an ordinal value to each possible state, allowing the system to choose the highest-ranked state that can be currently reached. This overcomes the limitation of goal policies not to give directions for the case that no desired system state is feasible. Instead of a binary distinction between desired and undesired system states, a utility value is mapped to each system state. Optimization algorithms may be applied to utility policies in order to translate them to goal policies. In order to distinguish between feasible and not feasible states during optimization, a system model is usually required. Since the system model includes knowledge about actions that result in a given state, the step of generating intermediate goal policies may be bypassed.

Utility policies in theory are the most powerful class of policies since unlike goal policies they provide a result for each possible system state, and unlike in the case of action policies there is no potential for conflicting policies. However, they leave the burden to specify an adequate utility function to the system operator which often is too difficult for practical purposes. Utility policies may be combined with goal policies that set constraints in order to prevent the selection of states with too low utility values.

Table 3.2 summarizes and compares the characteristics of the policy types outlined in the previous sections, with a plus sign denoting an advantage of a specific policy type, while a minus sign stands for a disadvantage.

Composition of autonomic systems

While the autonomic architecture described above focuses on the structural and behavioral characteristics of a single autonomic element, this section introduces approaches for building an entire autonomic system.

Kephart et al. [107] envision an autonomic system that is composed of multiple autonomic elements interacting with each other. Within an hierarchy of autonomic elements, higher-level autonomic elements shall manage lower-level autonomic elements based on higher-level policies.

A life cycle of relationships between autonomic elements shall consist of the following stages: specification, which may include registration with a public service registry and description of the element's capabilities using an ontology; location of other autonomic elements based on name or function; negotiation for a service agreement; provisioning of resources; operation under the negotiated agreement; and finally termination of the agreement, when the service is no longer needed.

Negotiation processes between autonomic elements may be as simple as a demand for service, where a lower-level element unconditionally provides service to a higher-level element if possible based on resource availability. Within a first-come, first-served negotiation, service requests are honored in the received order, as long as resources are available. In posted-price negotiation schemes, the service provider sets a price and the service requester decides whether to consume the resource at that price. Complex negotiation schemes may include multiple rounds of negotiation over multiple attributes.

Brazier et al. [30] state that systems composed of multiple autonomic elements resemble multi-agent systems that interact with services as employed in service-oriented computing. An autonomic element shall be modeled as a rational, goal-directed agent that is able to reason about its intentions and to plan its actions. An autonomic system composed of autonomic elements thus is analogous to a multi-agent system composed of agents. Autonomic systems may benefit from an agent interaction model called stigmergy, swarm intelligence or emergent system, where agents by following a simple behavior pattern jointly change an environment, leading to more complex behavior emerging from the multi-agent system. The challenge of that approach is to achieve a well-defined behavior of the overall system conforming to pre-defined requirements. A possible approach for verifying system behavior of an emergent system is to run simulations.

3.3.3 Adoption of autonomic systems

IBM [100] proposes an adoption model which gradually introduces autonomic capabilities into existing systems. The adoption model is shown in Figure 3.10 [100].

The dimensions given in Figure 3.10 denote the following adaptations to existing systems:

- The *functionality dimension* given on the x-axis characterizes increasing autonomic functionality, from (1) manual level (i.e. no autonomic features), to (2) monitor only, (3) monitor and analysis, (4) full control loop (monitor, analysis, plan, and execute), and finally (5) control loop guided by policies, which counts as full implementation of autonomic functionality.

- The *control scope dimension* given on the y-axis characterizes the level of management, starting from (A) individual subcomponents (e.g. an individual application) to (B) single instances (e.g. a whole application server), (C) multiple instances, i.e. all resources of the same type, to (D) multiple resources of different types, and finally (E) the whole business system.

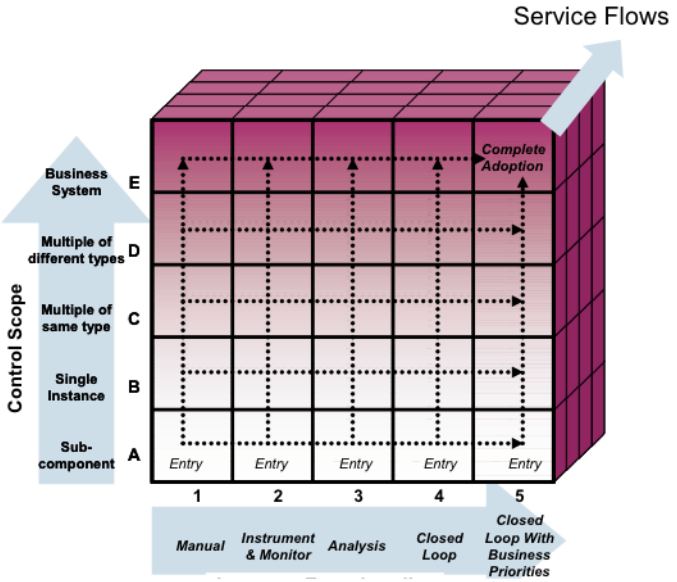

Figure 3.10: The autonomic computing adoption model [100]

- The *service flow dimension* along the z-axis denotes the introduction of autonomic capabilities across different IT processes, such as configuration management or security management.

3.3.4 Reference projects

This section introduces selected reference projects for autonomic computing. While these projects do not explicitly address large-scale computing and thus are not part of the survey, they are included here in order to present a view on possible applications for autonomic computing. In addition, most projects described here provide prerequisite technologies for large-scale computing.

Applications of autonomic computing presented in this section are the power management of data centers which is an optimization problem between low power consumption and performance, network security where malicious access needs to be identified, and runtime reconfiguration of applications.

Power management

The minimization of power consumption is a requirement of growing relevance in the field of distributed computing. For handheld and mobile devices, battery lifetime is not just a distinguishing characteristic for competition in the marketplace, but a key factor in order to allow adding complexity and reducing the size and weight of the devices. For data centers, the same rationales apply on a larger scale: minimization of power consumption improves competitiveness by reducing total cost of ownership,

but may be even required in order to allow further expansion of a data center due to power shortage. The requirement of low power consumption may also result from government regulations either in form of subsidies for energy-saving measures or by regulation that sets an upper border to power consumption. Since mobile endpoints and data centers both are required for large-scale projects, successful and efficient power management is a key factor for further establishment of large-scale computing.

Khargharia et al. Khargharia et al. [109, 110] model the power management problem as an optimization problem, where power consumption shall be minimized given that QoS requirements are met. Key factors are the startup times of systems that are in down state, and resume times of systems that are in suspended state. They use a hierarchy of autonomic managers for autonomic power management of a data center consisting of a web server cluster as front end tier, an application server cluster as mid tier, and a database cluster as back end tier in order to address the trade-off between minimization of power consumption and meeting QoS requirements described above.

The system is modeled and optimized using DEVS discrete event system modeling and simulation [194]. A global component manager is responsible to minimize power consumption of the front end tier cluster based on the size of the global service queue, such that the global request loss rate and global average wait time constraints are met. A local component manager performs power and performance optimization on server level based on the local queue size.

Mastroleon et al. Mastroleon et al. [124] formulate the autonomic power management problem described in the introduction to this section using a dynamic programming (DP) approach [25]. The work is based on a model of a single application server shown in Figure 3.11 [124], which comprises a single job buffer with an initial number of jobs, a pool of available CPUs, an allocation environment of CPUs available for job scheduling, and a thermal environment that models the environmental effects of CPU usage. At each step in a decision process based on a time-homogeneous Markov chain, costs are associated with a job backlog, CPU allocation, increasing environment temperature, and reconfiguration of CPU usage.

The objective is to minimize costs while serving all the jobs initially stored in the buffer. The solution of the DP is a function of the current system state consisting of the job backlog, the current temperature, the number of available CPUs and the number of allocated CPUs. The optimal solution does not depend on past states and thus may be implemented using stationary policies. Since solving a DP is computationally intensive, several heuristics are proposed, including the allocation of no cpus in the case of an empty job buffer, at least one cpu in the case of a non-empty buffer, and increasing the number of cpus with increasing queue size.

Network management

In the area of network management, the autonomic computing paradigm is applied for adapting network infrastructure to configuration changes, and to detect and cir-

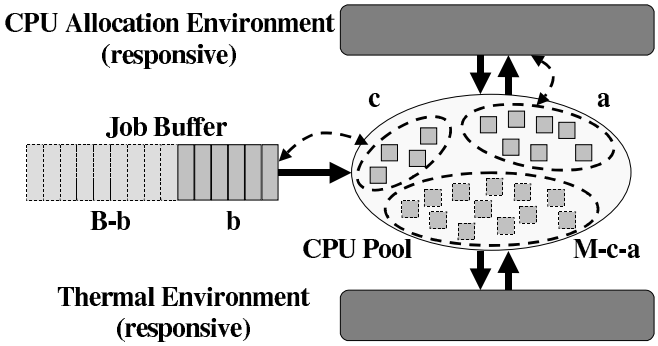

Figure 3.11: System model for autonomic power management [124]

cumvent unauthorized access. The following paragraphs present the FOCALE project which is an example for the former application realizing the self-configuration property, and ML-IDS as an example for the latter one which addresses self-protection capabilities.

FOCALE The FOCALE [106] project is an autonomic networking project developed at the Waterford Institute of Technology in Ireland. It adapts the overall architecture of autonomic computing systems for the task of network management by providing a model-based translation layer which maps vendor-specific commands and data like SNMP to vendor-neutral commands and data. Figure 3.12 [106] shows the functional architecture of an autonomic management element in FOCALE associated with a managed resource.

FOCALE comprises a distributed architecture that allows an autonomic manager to manage entities ranging from a single network device to a complete subnetwork. Autonomic managers use a maintenance control loop when the system is in normal state, and an adjustment control loop when reconfiguration actions are to be performed.

ML-IDS In the area of network security, the autonomic computing paradigm is realized within an anomaly-based multi-level intrusion detection system [7, 150, 131] that implements the self-protection property. Anomaly-based intrusion detection systems gain information by observing system behavior, as opposed to pattern-based systems that monitor network traffic to identify signatures of an attack. System metrics are collected in order to model overall system behavior to be in normal state, uncertain state, or abnormal state. Local control loops manage the behavior of individual system resources, whereas the global control loop is responsible for managing the whole system. An autonomic runtime system consisting of common web services and an autonomic runtime manager is responsible for establishing autonomic properties. Since current QoS protocols do not distinguish between normal and attacking network traffic, a quality of protection (QoP) framework is established.

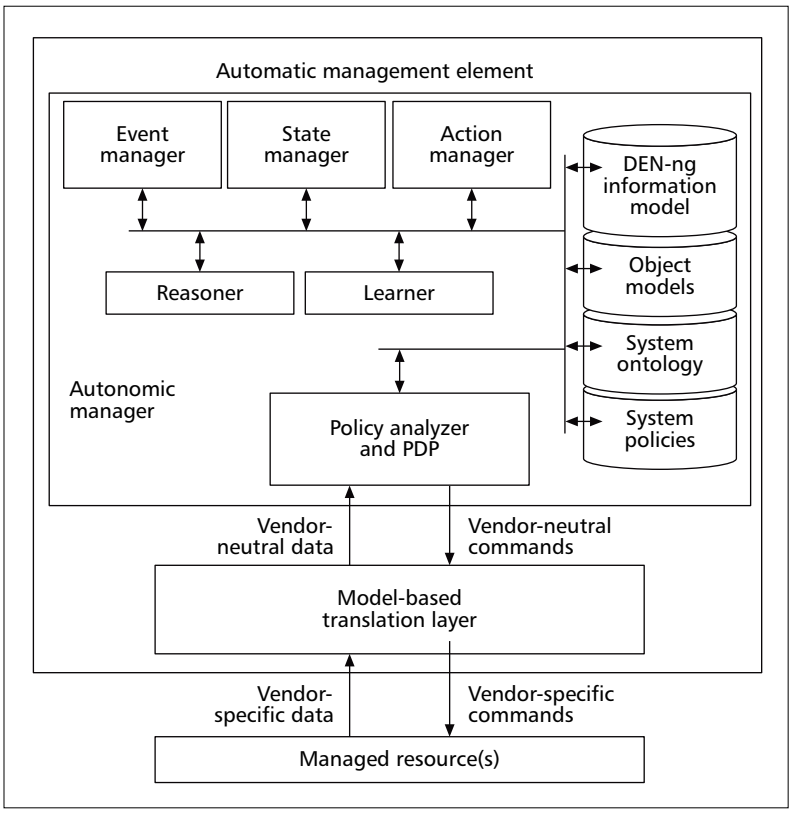

Figure 3.12: FOCALE autonomic management element architecture [106]

False positives are a problem in intrusion detection system that prevent activation of autonomic actions based on monitoring and analysis. Thus, an adaptive learning algorithm and a multilevel analysis consisting of protocol behavior analysis, rule-based behavior analysis, and payload behavior analysis has been implemented, which are combined using least squares technique. An action translation module translates the desired high level action into action steps for the managed network modules.

Runtime reconfiguration

Runtime reconfiguration refers to allowing dynamic reconfiguration of running systems without the need of stopping or restarting of application components.

Kheiron Kheiron [86, 87] is a framework that allows introducing runtime reconfiguration support in applications which run in a managed execution environment like Microsoft's CLR or the Java VM that provide a management API. In the case of CLR, the profiler APIs are used to collect information about parameters like memory usage of the application. An execution monitor catches module load and unload events,

while the bytecode and metadata transformer allows the replacement of bytecode, thus triggering a JIT recompile of the modified method. For native applications which have been written in languages like C, a management API is not available. Therefore, profiling and debugging information generated by the compiler is used to collect information, and the Dyninst API[8] for generating runtime code.

[8]Dyninst, `http://www.dyninst.org/`

4 Survey and taxonomy of autonomic large-scale computing

This chapter conducts a comprehensive survey of autonomic large-scale computing. It includes projects that explicitly support autonomic capabilities in order to address large-scale computing problems and presents a taxonomy derived from properties of those projects.

Section 4.1 presents work in the domain of autonomic large-scale computing that is part of the survey. Section 4.2 examines considerations to be taken into account when building taxonomies and presents the taxonomy of autonomic large-scale computing.

4.1 Survey of autonomic large-scale computing projects

This section presents a survey of large-scale computing projects with self-management capabilities. Work that extends existing large-scale computing projects with autonomic capabilities is listed under the respective project name. Theoretical contributions and unnamed projects are listed under the author's names.

Section 4.1.1 introduces autonomic grid resource managers and schedulers. Section 4.1.2 presents desktop grid and peer-to-peer systems with self-management support. Section 4.1.3 describes autonomic grid middleware projects that do not fall into one of the preceding categories. Section 4.1.4 introduces work that investigates autonomic computing within cloud computing platforms. Section 4.1.5 presents work that examines the autonomic management of quality of service parameters and service level agreements in large-scale computing systems. Section 4.1.6 lists autonomic scientific workflow management systems. Section 4.1.7 finally introduces projects that support the development of autonomic large-scale computing systems.

4.1.1 Grid resource managers and schedulers

This section describes projects that use autonomic capabilities in order to address the problems of resource management and scheduling in grid computing, excluding desktop grids which will be described in the next section. A taxonomy of grid resource management systems has been presented in Section 2.1.1.

Dasgupta et al.

Dasgupta et al. [55] present an architecture for run-time fault handling in grid job-flow management which targets the self-healing capability of autonomic computing. A prototype implementation consists of two domains using different implementations and different internal architecture, but mapping to the same conceptual architecture consisting of a job flow manager responsible for sequencing individual jobs and maintaining concurrency, and a meta-scheduler that selects resources and executes jobs. The meta-scheduler contains logic to decide between local and remote job execution.

In order to implement self-healing capabilities, a generic proxy intercepts job flows and adds run-time fault handling based on rules to detect failures and policies for recovery actions. In addition, the proxy maintains a job flow repository and fault-tolerant patterns that specify common actions like re-submitting the job with modified parameters or to a different domain.

Dynaco framework

The Dynaco framework [34] is designed to introduce dynamic adaptation into grid resource management in order to react to changing resource availability during job execution. Buisson et al. [35] use the principles of autonomic computing by employing a control loop consisting of observe, decide, plan, and execute sub-functions while delegating the actual sub-functions to plug-ins. This design allows to implement various methods for each sub-function, e.g., to have the decide (analyze) step being delegated to a program implemented in an imperative language like Java, to use decision rules which are evaluated by some expert system, or to employ a genetic algorithm.

Job management in Dynaco is designed to refrain from stopping and restarting a job in order to avoid the associated performance penalty. Instead, it requires the jobs to be "malleable" by design, i.e. to allow for acquisition and release of resources during job execution. Ultimately, job management strategies may be submitted by the user as part of the job data.

Dynaco is built on top of the Fractal [32] component system.

Jarvis et al.

Jarvis et al. [105] present an autonomic middleware service that may be integrated with standard Grid middleware tools like the Condor scheduler [75] and Globus information services [68]. It is based on the PACE [138] performance analysis toolkit.

On a local (intra-domain) level, the Titan resource manager [166, 165] is used to manage resources and to schedule jobs to Condor one by one, with resource usage directions added which are based on PACE predictions. Thus, the jobs are queued within Titan, not Condor. On the queued jobs, Titan employs a genetic algorithm for resource selection that calculates multiple schedules, throws away unsuccessful schedules, and replaces the current best schedule, if a better schedule has been found. The genetic algorithm uses status information obtained from Condor for adaptation of the queued job's resource allocations.

For multi-domain task management, the intra-domain resource usage and performance data is published using a performance information service based on the monitoring and discovery service from the Globus toolkit. A network of agents [42] uses this performance data for deciding on each incoming job request, if it should be fulfilled locally or directed to another site.

The GridFlow workflow management system presented in Section 4.1.6 is related to this project, since it is also based on the Titan scheduler, the PACE performance analysis toolkit, and the ARMS agent-based resource management system.

Perez et al.

Perez et al. [148] combine reinforcement learning and utility functions [172, 173] for job scheduling on the EGEE grid infrastructure introduced in Section 3.1.2. The reinforcement learning framework uses a time-utility-function that represents user satisfaction as a decreasing function of job completion time and models "fairness" as the difference between actual and previously agreed-on resource allocation between participants of a virtual organization.

A reward is calculated from time-utility and fairness that is used to determine the reward associated with the selection. The scheduler uses the SARSA (State-Action-Reward-State-Action) algorithm [169] with neural network training for reinforcement learning.

4.1.2 Desktop grids

This section presents autonomic desktop grid systems which are also called enterprise grids, and peer-to-peer systems. Desktop grids are grids consisting of idle resources of desktop computers. Grid computing in a peer-to-peer network employs a homogeneous set of nodes that are at the same time providers and consumers of grid resources. A taxonomy of desktop grids has been presented in Section 2.1.6.

CoordAgent

CoordAgent [76] is an enterprise grid middleware based on mobile agents. In order to locate resources of desktop computers within a grid, a structure similar to Internet-based discussion groups is established, where a moderator creates a group that may be joined by desktop computers. The group itself may be distributed among several computers. When a user submits a job, the group information is used to locate suitable resources for job execution, and a mobile agent is created on the requester's behalf.

CoordAgent supports process migration in case a desktop computer becomes unavailable during job execution. Checkpointing occurs by periodically storing a snapshot on computers selected using a quorum-based protocol. This requires that applications are written in C/C++ or Java, and that they are being preprocessed beforehand using a JavaCC/Antlr-based preprocessor. In case of C/C++ programs, the preprocessor inserts setjmp and longjmp instructions that allow saving the execution envi-

ronment including the program counter, but only supports process migration among homogenous machines. Since Java does not support manipulation of the program counter, state-capturing statements are inserted that divide the user code in functions that are executed before and after migration, respectively.

Inter-process communication allows location of a migrated process and takes into account that sites may be located behind a firewall or within a private network. A middleware-unique ID is assigned to each job which gets translated into an IP address on each process migration. Agent transfer uses the http or https port of the TCP protocol stack in order to accommodate to the common network security practice of closing all but a few common ports.

The mobile agent described above is responsible for maintaining the process state by initiating process migration in case of resource unavailability and finally returns the result to the requester.

InteGrade

InteGrade [81] is a desktop grid middleware based on CORBA which allows to use the idle resources of desktop computers in a shared or dedicated manner. It organizes desktop computer resources into a hierarchy of clusters with a cluster manager node responsible for managing cluster nodes and for communication between clusters. The problem of idle desktop resources suddenly becoming busy or unavailable is addressed by collecting usage statistics and determining the probability of being available for each node, in combination with a checkpointing mechanism that periodically saves the current state of a computation. InteGrade implements the bulk-synchronous parallel (BSP) model [178] of parallel computation which separates the concerns of inter-process communication and scheduling but requires frequent synchronization.

Figure 4.1 [81] shows the architecture of a single cluster within the InteGrade middleware.

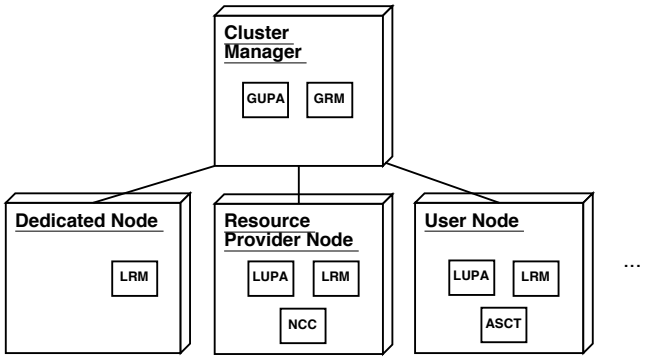

Figure 4.1: InteGrade's intra-cluster architecture. The inter-cluster architecture connects several cluster manager nodes to form a hierarchy of clusters. [81]

AutoGrid [161] is an autonomic grid middleware built using the Adapta [160] framework which allows the development of self-adapting component-based distributed applications. It is based on the InteGrade enterprise grid middleware described above. AutoGrid implements context-awareness, self-optimization, self-healing and self-configuration capabilities by providing a monitoring service as sensor and a dynamic reconfiguration service as effector.

Possible self-configuration actions in AutoGrid include replacement of an application algorithm by using the lazy object replacement approach of the Adapta framework, and changing of application parameters. Self-healing is based on the fault-tolerance mechanisms supported by the Integrade middleware, namely retrying, replication, and checkpointing. AutoGrid currently supports replication and allows dynamic change of the number of replicas. Self-optimization provides means for replacing the scheduling algorithm.

Neto et al. [134] employ a model based on interceptors as an alternate approach for adding self-management capabilities to the InteGrade middleware. Interceptors are used to modify non-functional properties of a system. The interceptor model employed here consists of a monitor and several implementor components, each of them implementing a non-functional property. Since the dynamic interceptor model is implemented on the communications layer, applications written for the InteGrade middleware need not to be modified as it is the case with AutoGrid which uses additional components in its infrastructure.

Organic Grid

Chakravarti et al. [48] address the problem of scheduling within a desktop grid. Desktop grids require the grid middleware to adapt to a frequently changing environment, since the resources of desktop computers are only available on the grid, if the computer is powered on and not used for interactive applications.

The Organic Grid middleware uses an approach that combines decentralized scheduling and strongly mobile agents. Strong mobility in agents allows computational tasks to disregard mobility issues, leading to simpler program code. Additionally, the agents support forced mobility, which means that agent migration can be initiated from an external thread, i.e. the grid middleware on behalf of a user reclaiming their desktop's resources.

If an agent migrates to another node, the destination becomes a child node of the source. The agent's knowledge about the grid environment is organized using a tree-based overlay network with the task origin as root, the child nodes as vertices, and the edges weighted with the children's perceived performance. Initial migration is supported by a "friend's list". If an agent is started with a task that is divisible into subtasks, it executes one of the tasks and sends requests for work to other nodes. A node that receives a request for work sends a clone of its agent to the requester, with the requester becoming the child of this node. Each node periodically informs its parent of the performance of its children. The parent may add new grandchildren to its list of children in order to contact them directly in future. This leads to nodes

with good performance moving closer to the root of the overlay network.

So-Grid

Forestiero et al. [67] present So-Grid, a self-organizing information system for the management of grid resources in a peer-to-peer network that employs bio-inspired algorithms for replication of resource descriptors and discovery of those descriptors. Agents pick and drop resource descriptors according to probability functions in order to replicate and reorganize available resources. Agents switch between copy mode for replication and move mode for reorganization autonomously, i.e. based on their local information. The discovery algorithm directs searches to regions of the grid where the appropriate resource type is accumulated. In order to direct searches, grid hosts that accumulate a large number of descriptors for a given class are elected as representative peers that attract queries for that class type. Parameters that influence the agent's decisions for resource selection include the average connection time of a peer and the update interval of resources.

4.1.3 Other grid middleware

This section describes autonomic grid middleware projects that are part of this survey and do not fall into one of the categories described in the two previous sections, which have been resource management and scheduling, and desktop grids. Problems addressed by the projects in this section using autonomic computing methods include client access, service discovery and data transfer optimization.

AutoMAGI

AutoMAGI [157] is an autonomic mobile-to-grid middleware which is designed to enable access from mobile devices like mobile phones and PDAs to grid resources. In order to achieve mobile grid access, limited client resources and a highly unreliable network connection have to be taken into account. Thus, the middleware is organized into autonomic components which operate as semantic web services.

Components in AutoMAGI include a client communication interface responsible for the decision of whether a loss of connection detected during monitoring has been intentional (i.e. requested by the user) or unintentional (e.g. client-side lack of network coverage). A job broker service is responsible for scheduling the job on the grid on behalf of the client (which may be offline at that point). Job results are stored within the knowledge component described below, and finally passed to the mobile client when it reconnects. Security and support for multiple instances of the middleware are provided.

The knowledge management component of AutoMAGI uses semantic web technologies to organize knowledge passed by the autonomic elements, like context information, system logs, performance metrics and policies. Since each autonomic element may have its own knowledge model, the knowledge management component is also responsible

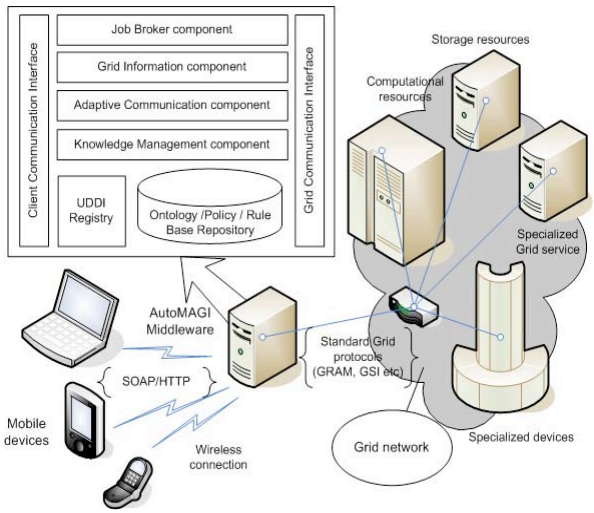

Figure 4.2: AutoMAGI architecture and deployment model [157]

for data and knowledge integration and for resolving conflicts that may arise due to conflicting goals.

A security infrastructure allows to specify different levels of trust among the agents, and provides cryptographic protocols for encryption of communication between agents.

The middleware supports the deployment of multiple instances of the AutoMAGI gateway. An arbitrator component allows to share information between the gateway instances according to high-level guidance. In case a client disconnects and reconnects to a different gateway instance, the client sends the last used gateway ID upon reconnection, allowing the gateway that processes the current connection to obtain information from the gateway where the client has previously been connected.

GATES

The GATES middleware (grid-based adaptive execution of streams) [50, 51] uses existing grid standards and tools for real-time processing of data streams that are generated from data collection instruments or received through WAN communication in a rate faster than the respective disk write speeds, disallowing temporary storage. A self-adaptation algorithm allows optimization of processing accuracy under the real-time constraint. The middleware requires applications to use a specific API to expose adjustment parameters like the sampling rate. This API is called periodically by the stream processor in order to increase or decrease the values of certain parameters based on current status of the processing system.

For example, if incoming data cannot be processed in real time at the current sampling rate due to high system load – which can be determined by observing the processing queue length – the middleware increases the application's sampling rate, thus

reducing accuracy while maintaining stability.

The adaptation algorithm's objective is to keep the average queue size between configurable low and high thresholds. An open problem is how to maintain system stability by appropriately weighing past and current queue lengths.

Gounaris et al.

Gounaris et al. [84] present an approach for self-optimization in grids based on web services, that tunes the block size of a response to a OGSA-DAI web service (database interface) in order to minimize total response time of database calls. It uses an autonomic controller that resides on the client side and thus has limited information about the service. So, self-management occurs solely in reaction to the currently observed actual response time of a database call. Optimization of the response time must take into account noise in the performance graph resulting from the client-side observation and the moving optimum resulting from dynamic system behavior. In addition, fast convergence is required.

Two optimization algorithms are presented in order to solve the problem outlined above. The first one is a runtime optimization algorithm based on Newton's method with averaging applied to mitigate the algorithm's sensitivity to noise. The second algorithm is based on switching extremum control. For the application of finding the optimal block size for a data transfer, fast convergence and low overshooting – i.e. avoidance of intermediate solutions far away from the optimal point – is required. Experimental evaluation shows that the algorithm based on switching extremum control outperforms the algorithm based on Newton's method. While the case study is conducted in the context of OGSA-DAI web services, the presented methods are applicable for similar optimization problems.

GridSim

GridSim [39] is a toolkit that allows the simulation of resource allocation and scheduling mechanisms. Assuncão et al. [22] provide an extension framework for GridSim that allows to model policies for the provisioning and negotiation of resources and services. The policies developed with this framework can be used by autonomic managers. Example acquisition and provisioning policies based on the Catallaxy economic model are given.

Guan et al.

Guan et al. [88, 89] introduce an autonomic service discovery middleware for pervasive devices built on semantic web technology. The web ontology language for services (OWL-S) [123] is used to describe grid services by extending its service profile with grid-specific service parameters. For example, a service type is specified in order to distinguish between an information access scenario, where a pervasive client is used to retrieve information, and a work assistant scenario, where a pervasive client initiates the execution of a program on the grid.

A service discovery algorithm uses the ontology described above to match requests with services based on the taxonomic relation of concepts. The algorithm distinguishes between strict and non-strict requirements. After selecting all services which do match all of the strict requirements, a score is calculated to represent the matching of non-strict requirements. Calculation of the score is based on the semantic distance between requirement and service, which is a function of their class relationship within the ontology.

Guo et al.

Guo et al. [90] use intelligent agents to achieve self-organization of autonomic elements in a grid environment. Autonomic elements publish their capabilities and interests based on ontologies and first order predicate logic, and they establish acquaintance relationships with each other. Autonomic managers are agents, and their managed resources are either application grid services, management grid services, or lower-level agents. Composite services are services which are composed of lower-level services that are integrated according to a guideline called recipe.

The collaboration life cycle between autonomic elements consists of invitation, negotiation, affirmance, request, monitoring, exception handling, and termination. Elements pass information to other elements with matching interests using notifications. Verification rules are employed to decide if a piece of information or an event matches some interest. The control loop of each element first decides which management work is to be done and then either performs it or establishes collaboration with other elements that have matching capabilities.

ICENI

The Imperial College e-Science Networked Infrastructure (ICENI) is middleware framework which adapts the component model of software engineering to grid computing. Applications and resources are modeled as components with component metadata expressed in Component XML (CXML) [77]. Within the Open Grid Services Architecture, ICENI is situated between the application and collective layers with the goal to provide application abstraction in the scientific computing field.

Hau et al. [92] use ontological annotation in order to provide autonomic service adaptation within ICENI. The annotation of components published in the ICENI framework is extended with semantic information which the scheduler can use for semantic matching of components. For example, `a+b` has the same semantics as `sum(a,b,0)`, allowing to add up two numbers using a component with addition functionality that requires three parameters. However, autonomic service adaptation is not part of the current version of the ICENI architecture. [126].

OptimalGRID

Almaden OptimalGRID [59, 118] is a middleware developed at the IBM Almaden Research Center which automates the problem partitioning, problem deployment and

run-time management for large-scale, parallel distributed applications.

First, a connected parallel problem is partitioned into original problem cells (OPC) with connections that are represented using a map. Then, the OPCs from a region of the map are statically aggregated into OPC collections. One or more OPC collections are dynamically assigned to a variable problem partition (VPP) for execution on a grid node. Communication between OPCs on the edges of neighboring OPC collections is minimized, since it consumes network resources. A tuple-space communication system based on TSpaces [117] is used to implement a distributed whiteboard model that allows client access to one or more shared global message boards. The optimal number of whiteboards depends on the problem size and is determined using autonomic methods described in the following paragraph.

The first component on the OptimalGRID system involved in application execution is the problem builder that determines the optimal number of nodes needed to solve the problem. The autonomic program manager (APM) is assigned to one of the grid nodes and collects performance and diagnostic data from the VPPs. Pluggable policy modules may be added to the APM in order to support specific optimization goals. Self-configuration occurs during the initial problem assignment to specific nodes based on their computing power and memory sizes, and the network latency in order to minimize overall problem solving time. Self-optimization and self-healing is supported by writing the state of all OPCs to the whiteboard periodically with a configurable interval length, in order to allow recovery from a failed node.

Zhao et al.

Zhao et al. [197] present an autonomic grid data management system based on the Grid Virtualized File System (GVFS) [196]. An autonomic data scheduler service maintains parameters like the client-side disk cache size using a utility function based on the size of the client-side data set, computing power of a node, and deadline requirements. A data replication service maintains a number of replicas. Since increasing the number of replicas of a given data set enhances data availability but increases replication cost, a benefit-cost-analysis is done based on available knowledge of reliability values of the nodes. Additionally, data sets of applications with high priority and data sets of active applications are given a higher value. Data replica regeneration in case of data server failures is also supported. The grid data management system is supported by client-side and server-side autonomic file system services. Functionality supported includes autonomic cache configuration, data replication, and session redirection.

4.1.4 Cloud computing systems

This section presents work that explicitly describes self-management in cloud computing systems. Work presented in this section either addresses a problem that is specific to cloud computing or investigates a general problem in that context.

AbdelSalam et al.

AbdelSalam et al. [1] present a scheme that minimizes energy consumption of a cloud computing center and identifies suitable time-slots for change management, while satisfying processing requirements of interactive applications based on service-level agreements and previous usage data. The system model assumes a group of identically configured servers with an operation frequency manageable by hardware-based power-management techniques. Client SLAs are translated to computing power requirements by executing test query sets provided by the customer.

Assuming a cubic relationship between operating frequency and power consumption, the optimal number of servers and their operating frequencies are determined for each time period based on the sum of the client computing power requirements. [2] Time slots with low demand, where the optimum number of servers is less than the total number of servers are identified as being suitable for change management purposes.

Dai et al.

Dai et al. [54] present a framework for self-diagnosis and self-healing in cloud systems that uses multivariate decision diagrams (MDD) in order to determine the severity level of a problem, and a Naïve Bayes classifier (NB) to identify the category of a problem and possible remedies. Figure 4.3 [54] shows the self-diagnosis and self-healing process employed by the framework.

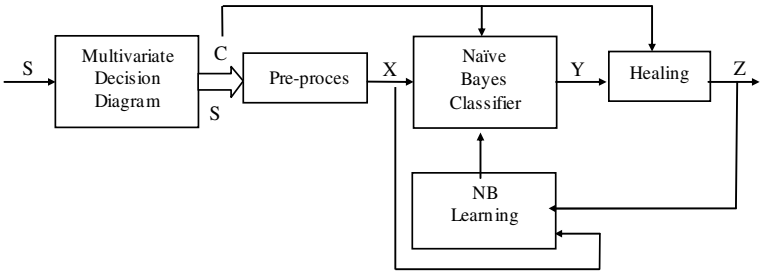

Figure 4.3: Self-healing with MDD and the Naïve Bayes Classifier [54]

A multivariate decision diagram, which is a directed acyclic graph extending a binary decision diagram for the case of n-valued variables, thus having up to n sink nodes represents the overall severity level of the current system state. The severity level obtained by applying the MDD is used to determine if some immediate action like suspending or killing suspect processes needs to be performed before running the actual and possibly time-consuming diagnosis process.

Diagnosis assesses the probability for a problem class based on the observed attribute values, applying Bayes' theorem to the actual values and their probabilities known from history data. Problem identification includes a list of possible remedies and their predicted consequences. The result of applying each remedy is checked against

the predicted consequence in order to determine success. Parameters of the original problem and the successful remedy are used as input for the Naïve Bayes learning network in order to adjust parameters for future diagnosis.

Iqbal et al.

Iqbal et al. [103] address the problem of guaranteeing response times of cloud applications. They present a prototype based on the Eucalyptus [139] cloud system introduced in Section 3.2.3 which allows cloud systems to adapt to the condition of heavy use by autonomically requesting additional resources. The architecture of the prototype consists of a load balancing proxy and several virtual machine images containing instances of the cloud application. Monitoring examines the proxy log file in order to detect sessions with an average response time exceeding a certain threshold. If such sessions exist, a new cloud application instance is created and the load balancer configuration is updated to include the newly created virtual machine.

Planned future enhancements of the prototype are to allow dismissing unused virtual machines in periods of low traffic, and the prediction of high traffic periods.

Libra+$Auto

Libra+$Auto [189] is a mechanism for advance reservation of cloud resources that employs autonomic resource pricing, allowing commercial cloud providers to increase revenue by applying price discrimination, which is the economic term for the practice of charging different customers with different prices for the same good, with the good in question being computing resources.

While most commercial cloud computing systems currently offer static unit pricing of resource usage, pricing in Libra+$Auto is a function of expected workload demand and resource availability. Since resources are reserved in advance, expected workload demand is known to the system. Providers may configure a static base price with a fixed weight that allows the charging of a minimum price, and a unit price with its weight being adjusted automatically based on node availability. The price per computing node is calculated as the weighted sum of the base and unit prices, and the total price as the sum of the prices of the reserved computing nodes at the current time slot.

Libra+$Auto is implemented within the Aneka cloud computing platform described in Section 3.2.3. The implemented reservation system allows to select nodes within independent time slots for sequential applications, or a set of nodes within the same time slot for parallel applications.

Paton et al.

Paton et al. [143] present an approach for adaptive workload management on cloud systems based on utility functions, with the goal of allowing to handle unpredictable variations in workload resulting from the composition of cloud services. Utility functions combined with optimization algorithms are identified as a paradigm for workload

execution management. After describing a methodology for developing a utility-based workload management approach, the applications of autonomic workflow execution and autonomic query workload execution are studied.

The methodology states that, after identifying a property to optimize, a utility function and a cost model shall be defined. While utility is expressed in terms of the chosen property, the cost model predicts the workload performance while taking into account adaptation costs. After designing a representation for the assignment of workload components to computational resources (e.g. a vector of tasks) and selecting an algorithm for optimization, the control loop of the autonomic controller may be implemented.

The adaptive workload management methodology is applied to the case of autonomic workflow execution, where workflows are mapped to cloud resources. Here, optimization occurs by the utility properties of response time and profit. In the case of autonomic query workload execution, utility properties are response time and number of QoS targets met. The utility-based approach is shown to outperform action-based adaptation strategies in the context of cloud workload management.

4.1.5 Quality of service frameworks

This section introduces projects that examine the management of non-functional requirements expressed in terms of quality of service parameters, and the negotiation of service level agreements in large-scale computing systems. Work described in this section presents algorithms or research prototypes that target QoS management and SLA negotiation in the context of large-scale computing systems.

Koller et al.

Koller et al. [112] present an SLA management proxy that can be used to add SLA management to existing services. The proxy maps services to virtual services that are accessed by the consumer, who calls the virtual services offered by the proxy instead of the original ones. Virtual services are responsible for communicating with the respective original services. Besides passing through the functionality of the original service, virtual services add monitoring of the original services – which requires the services to offer monitoring capabilities – and provide status reports derived from monitoring that may be accessed by the consumer.

During the communication setup phase, the proxy identifies services that fulfill the requested QoS parameter and maps a physical service to the virtual one. When communication is established, the proxy supports preventive monitoring that allows to notify a system operator if QoS parameters exceed a certain threshold so that there is a risk of violating a service-level agreement in the near future.

In order to resolve conflicts of interest between producers and consumers, the proxy approach allows for neutral evaluation and management of SLAs, provided that the proxy system is operated by a third party. If the proxy is operated by the service

provider instead, it allows the agreed-on SLAs to be stored at a third party notary site.

PAWS

PAWS - Processes with Adaptive Web services [16, 146] is a framework for the execution of business processes based on web services which has been developed at Politecnico di Milano. An optimizer component within that framework is responsible for composing web services such that quality of service constraints are satisified [20].

Anselmi et al. [14] propose an algorithm for selecting web services (and, by extension, grid services) for composition subject to quality of service requirements. In order to minimize the computation power required to solve the web service composition problem, the approach presented optimizes the mapping between abstract and concrete web services for a set of requests simultaneously.

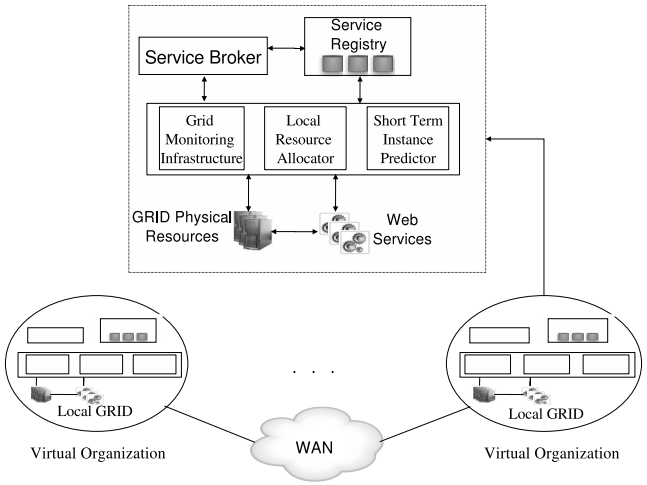

Figure 4.4: Reference grid environment for service composition [14]

A reference environment shown in Figure 4.4 [14] consisting of multiple virtual organizations (VO) is given that share their resources which are represented by concrete web services and can be accessed by an end-user. Each VO contains a service registry and a service broker which is responsible for the mapping process described above. An NP-hard optimal solution based on mixed integer linear programming (MILP) and a greedy heuristic algorithm are given that solve the multi-instance web service composition (MI-WSC) problem by discretizing periodic QoS profiles that are stored in the service registry.

Ardagna et al. [19, 18, 17] present a reference framework for web service selection and resource allocation in autonomic grid environments. Models are presented for the end user and the provider perspective. While the end user's goal is to maximize QoS,

the provider wants to maximize SLA revenues and to minimize resource management costs.

In order to maximize SLA revenues, the local resource allocator uses a short-term workload predictor, historical workload statistics, and data from the grid monitor to assign a fraction of the available capacity to each web service invocation. QoS maximization for the end user occurs on the grid broker level, where preferences and constraints are specified.

VieSLAF framework

The VieSLAF framework [29] addresses the problem of service negotiation between producers and consumers which do not have matching QoS templates, e.g., prices denoted in different currencies. Non-matching QoS templates are a realistic assumption in heterogenous Grids or Clouds, since the publishing of services is usually not coordinated across different organizations. In order to allow negotiation with non-matching templates, a meta-negotiation process is introduced which uses meta-negotation documents consisting of pre-requisites (e.g. negotiation terms or security requirements), a list of supported negotiation protocols, and terms of an agreement (e.g. proposed 3rd party arbitrator).

SLA templates and meta-negotiation documents are categorized by application domain and stored in a knowledge base where they may be accessed using a GUI. The user then may add SLA mappings to the remote templates. During meta-negotiation, SLA mappings and XSLT transformation are applied, and eventually a service method is selected and invoked. Finally, public SLA templates are adapted based on the submitted SLA mapping.

Brandic et al. [28] apply the principles of autonomic computing in order to monitor SLA parameters and for the adaptation of meta-negotiation documents. Measurements of parameters which have to be monitored periodically are stored in a parameter pool and returned to the user on request, while other parameters are measured on request and the measurement result returned immediately. Adaptation of SLA templates may be initiated by producers or consumers. A property issued by the registry administrator defines how often a request for a specific SLA parameter has to be submitted in order to be accepted. On acceptance of a new SLA parameter, a new revision of the SLA template is published and all existing SLA mappings are assigned to the new revision. SLA templates which have not been requested for a specified amount of time may be removed from the registry.

Subsequent work [27] investigates the life cycle of self-manageable cloud services in the context of the VieSLAF framework and defines a resource submission taxonomy which is presented in Section 2.1.3 of this thesis. Then, after discussing the architecture of self-manageable cloud services, its application to negotiation bootstrapping and service mediation within the VieSLAF framework is studied. The life cycle of a self-manageable cloud service consists of the phases of meta negotiation, negotiation, execution, and post-processing. While in the meta negotiation phase, negotiation protocols, SLA specification languages and similar issues are negotiated, during nego-

tiation – i.e. after agreement on the mentioned issues – specific terms of contract like execution time and price are negotiated. Service execution includes job submission, job monitoring and similar actions, while post-processing releases resources after job execution.

Possible self-management actions during job execution and post processing include job migration in case of failures, and early release of resources in case of abnormal job termination. Self-management during meta-negotiation consists of a monitoring phase that selects candidate services, an analysis phase that queries the knowledge base for bootstrapping strategies, a planning phase that allows users to define new bootstrapping strategies if missing, and an execution phase that starts negotiation with the selected strategy. Self-management during negotiation similarly consists of a monitoring phase, where inconsistencies between SLA templates may be discovered, an analysis phase that queries for applicable SLA mappings, a planning phase, where the user may define new SLA mappings if existing SLAs cannot be applied, and an execution phase that applies SLA mappings in order to allow to begin negotiations between heterogeneous providers and consumers.

Weng et al.

Weng et al. [185] present an agent negotiation model based on the concept of a market-oriented grid [31, 38], where access to resources is associated with cost and negotiated between resource owners and consumers that are represented through agents. A reasoning model based on fuzzy cognitive map (FCM) theory is employed that consists of a set of concept state values and a set of associated weights and maps negotiation issues like QoS to concepts. During negotiation, values of issue concepts are exchanged between agents, while weights and the degree of satisfaction are kept private.

4.1.6 Workflow management systems

This section presents scientific workflow management systems in large-scale computing. Scientific workflow management systems manage collections of interdependent scientific computation jobs. Tasks provided by workflow management systems include selecting computing resources, providing access to data repositories, scheduling jobs, and handling faults. A taxonomy of grid workflow management systems [192] is introduced in Section 2.1.8.

Gridbus Workflow Engine

The Gridbus Workflow Engine (GWFE) [85] is part of the Gridbus Toolkit [40] developed at the GRIDS Laboratory, University of Melbourne. Workflows in GWFE are defined using an XML-based workflow language called GWFE workflow language (xWFL). A decentralized just-in-time scheduler is employed, where each task has its own task manager, and the task managers communicate using events. Tuple spaces are used to store and retrieve events [191].

A self-managing scheduling algorithm employed within the Gridbus Workflow Engine is based on a reputation system for selecting a grid site for workflow execution [152]. Figure 4.5 shows the scheduling architecture which at each site consists of a Grid Autonomic Scheduler (GAS) that obtains reputation information from a peer-to-peer coordination space based on a distributed hash table (DHT).

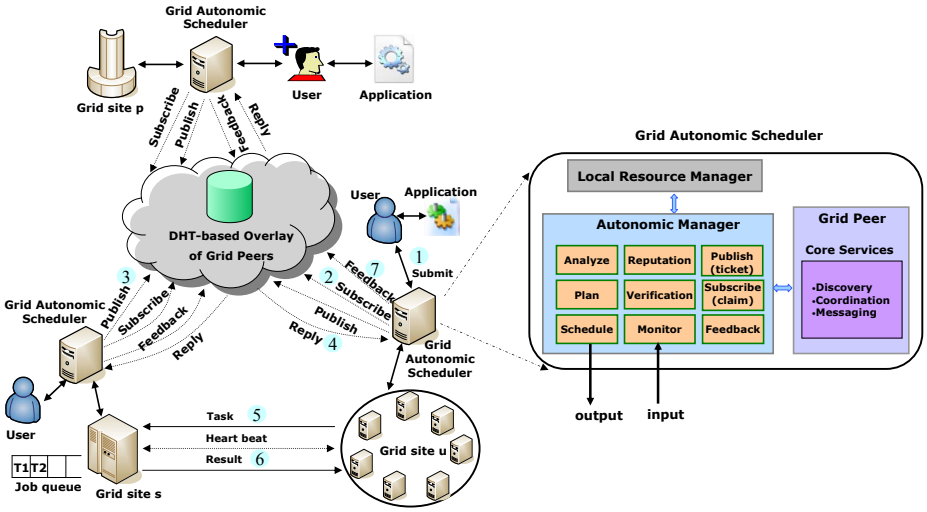

Figure 4.5: GWFE reputation-based dependable scheduling architecture [152]

A research proposal of the same authors uses self-management in order to address the problem of conflicting schedules that occur, if multiple workflow management systems schedule their workflows within the same pool of resources [151].

GridFlow

GridFlow [43] is a workflow management system built on top of ARMS [42], a resource management system based on mobile agents. GridFlow distinguishes between local sub-workflows, which are collections of related tasks expected to be executed on resources belonging to one physical organisation, and a global workflow consisting of one or more sub-workflows.

Each sub-workflow is managed by exactly one agent within ARMS. An agent consists of the data which is to be processed by a task, instructions how to process the data, and information on how to migrate to another site after sub-workflow completion. Thus, scheduling is decentralized in order to avoid the workflow engine becoming a bottleneck.

Checkpointing is used to allow a workflow to be restarted from after the last known completed task in case of failure. GridFlow supports local checkpointing, where an archive copy of the current state is left at each site after sub-workflow completion, and remote checkpointing, where a central checkpoint location exists.

Workflow scheduling in GridFlow is designed to minimize overall execution time, with estimated performance data provided by the PACE toolkit [138]. GridFlow uses Titan [166] for resource management.

An autonomic extension to GridFlow [135] adds self-healing, self-optimization and self-configuration capabilities. In order to achieve these capabilities, each agent is wrapped with pre- and post-execution performance monitoring instructions that allow the extension to react to failures by retrying, replicating, or checkpointing. Since retrying is unacceptable in workflows that require once-only execution, the user may restrict the allowed failure handling strategies at workflow creation time, thus providing a management policy. Failure handling strategies are designed to minimize the bull-whip effect that occurs when rescheduling of a sub-workflow results in rescheduling of other sub-workflows.

Section 4.1.1 describes a related grid middleware service which is also based on the ARMS resource manager, the PACE toolkit, and the Titan scheduler.

JOpera

JOpera [63] is a service composition tool originally targeted for web service composition, which has been extended to support distributed scientific workflows in grid computing.

The distributed workflow engine of JOpera [144, 145] employs a number of navigator threads to determine the tasks within a workflow that are ready to be scheduled, and several dispatcher threads responsible for actual task invocation. These tasks communicate using tuple spaces. In order to achieve good performance, the number of navigator and dispatcher threads needs to be properly configured. Figure 4.6 [93] shows the architecture of the JOpera workflow engine.

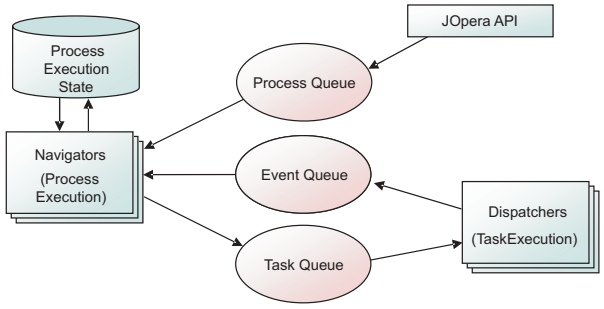

Figure 4.6: JOpera workflow engine architecture. Each queue corresponds to a tuple space. [93]

An autonomic extension to JOpera [94] uses self-tuning and self-configuration capabilities in order to determine the optimal thread pool sizes at run-time. Self-tuning employs an information strategy that uses the tuple space size as indicator to establish

a configuration where the number of navigator and dispatcher threads are balanced. A selection strategy within self-tuning then determines nodes suitable for a configuration change (e.g. idle nodes) taking into account the costs of a configuration change. The self-tuning component then submits change requests to self-configuration, which is responsible for actually applying configuration changes, i.e. starting additional threads, stopping threads or killing threads. When killing a dispatcher thread, the task associated with it has to be repeated. Thus, only threads associated with repeatable tasks (as defined by the application) may be killed. Other threads may only be stopped, that is, they wont accept new tasks but will finish currently executing tasks. Self-healing capabilities allow reacting to external events like failing nodes by monitoring cluster node availability and state information and comparing monitoring data with configuration data submitted by previously started threads. Self-healing restarts tasks handled by failing dispatcher threads and reroutes events affected by navigator thread failures.

Subsequent work [93] presents two zero-configuration policies for the controller described above that rely only on observable internal system parameters like current queue sizes. A policy based on a feedback loop controller adjusts the number of dispatchers such that the sum of process queue and event queue sizes equals the task queue size but still needs to be tuned in order to avoid oscillations. A balancing policy observes the message production and consumption rates in order to determine the number of navigators and dispatchers.

Pegasus

Pegasus [58] is a workflow management framework developed within the GriPhyN project [57] that is based on directed acyclic graphs (DAG) and targeted at data-intensive applications [192]. Pegasus maps abstract workflows that are derived from user-provided partial workflow descriptions or defined as a directed acyclic graph using DAG XML descriptions to executable workflows in an incremental but statical fasion. Autonomic Pegasus [115] is an extension which provides dynamic workflow adaptation capabilities using autonomic computing principles.

When mapping abstract workflows to executable workflows, Pegasus strives to minimize the overall workflow execution time using heuristics. Since resource availability changes over time, Pegasus optionally uses a scheduling horizon for executable tasks and a mapping horizon for workflow planning as limits, which can be set by the user based on the costs (e.g. data stage-in/stage-out) for scheduling and mapping. Applying scheduling and mapping horizons results in incremental compilation of the abstract workflow into an executable workflow.

Lee et al. [116, 115] present an autonomic manager for Pegasus that provides a dynamic workflow adaptation strategy, enabling a more efficient resource utilization in a changing execution environment (e.g. additional workload on nodes) than the incremental but static scheduling strategies originally supplied with Pegasus, thus providing *self-configuration* and *self-optimizing* capabilities. The autonomic manager is based on the full MAPE (monitoring, analysis, planning, and execution) cycle with

sensors and effectors, as described in Section 3.3.

Sensors Pegasus uses Condor DAGMan [75] for workflow execution. DAGMan logs the relevant events, like queueing of a job, job execution, and job termination, in a log file. Autonomic Pegasus uses a log sensor which polls the DAGMan log file in order to extract information relating to these events and their respective time stamps.

Monitoring In the monitoring phase of the autonomic cycle, the events from the log sensor are identified, filtered and passed to the analyzing phase. The following events are considered to be of interest in this phase:

- *Job queue:* occurs when Condor submits a task to the remote scheduler.
- *Execute:* indicates that execution of the task has been started by the remote scheduler.
- *Termination:* occurs when the task has been completed.

Analysis In analysis, the events supplied by the monitoring phase are grouped using the CQL continuous query language [15] and then compared to previous predictions regarding job waiting and execution times. If analysis detects a substantial increase or decrease of these parameters, the planner is notified.

Planning Planning proposes an alternative schedule, if analysis has indicated that rescheduling might be appropriate. The Pegasus planner is called to generate a new schedule based on current queue time and execution time estimates. In order to save work already done by the running tasks, intermediate results are stored in the replica catalogue. Rescheduling uses an algorithm which takes average queue times into account.

Then, planning compares the proposed new schedule with the current schedule – taking into account the rescheduling costs – in order to determine if an overall improvement is to be expected from rescheduling. Only if such an overall performance improvement can be achieved, the execution stage is called.

Execution Execution stops the currently running workflow and deploys a new one based on the schedule the planner has submitted.

Effectors Pegasus commands that stop a running workflow and start a new one are effectors in the sense of the autonomic computing model.

Evaluation shows, that the adaptive scheduling policy based on the autonomic model results in a significant performance improvement compared with the static incremental scheduler of Pegasus, especially with workflows that contain many tasks that can be parallelized.

TigMNS workflow management

Zhang et al. [195] developed an autonomic workflow management engine that uses event-condition-action (ECA) rules to dynamically specify workflows. Its architecture consists of a matrix-based discrete-event controller [130] which employs a task-sequencing matrix, an autonomic service component matrix and an input matrix which are basically adjacency matrices of the corresponding dependency relation trees.

The autonomic controller supports self-configuration by modifying the matrices as a reaction to changing user requirements, node failures or availability of additional nodes, and self-optimization capabilities as a reaction to changing workload.

The autonomic workflow management engine has been implemented as part of a grid-based traffic information system (TigMNS) [163] which supports different types of clients and networks (e.g. mobile phone over GSM, PC notebook over WLAN).

WorkflowML

Nordstrom et al. [137] developed a modeling language for workflow management in grid environments called WorkflowML, which was created using the GME and MetaGME modeling frameworks [114]. These frameworks implement the model-integrated computing (MIC) [170] approach. Models expressed in this language are transformed into a directed acyclic graph, upon which an algorithm performs predictive analysis.

Based on this model, the authors present the initial state of an autonomic design implementing the self-analysis property: If during workflow execution a job is in faulty state, predictive analysis is performed in order to determine if the workflow can be completed under this condition. This analysis forms the basis for reacting to a not-completable workflow, like suspending the current workflow in favor of a workflow that can be completed. However, these capabilities are not yet implemented.

4.1.7 Development frameworks

This section presents tools and frameworks that support the development of autonomic large scale systems.

ASSIST

ASSIST [9, 180] – a software development system based upon integrated skeleton technology – is a programming environment that allows the development of parallel programs on top of a component model consisting of modules that are interconnected with streams. ASSIST supports sequential modules and parallel modules (parmod). Parmods are implemented using a coordination language that is syntactically derived from C and the CORBA interface description language. They may contain a number of sequential functions that can be run using virtual processes (VP) that are triggered by items arriving from the input stream.

Virtual processes may put items onto the output streams where they are processed according to output rules. Figure 4.7 [9] shows in its upper part the relation between

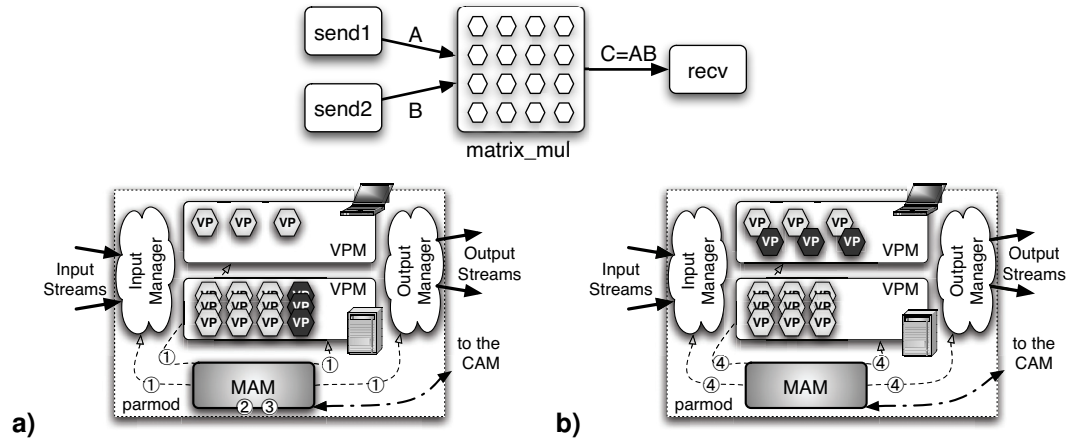

Figure 4.7: ASSIST parallel module (parmod) example and reconfiguration: a) before and b) after, dark VPs being migrated [9]

a parmod, its virtual processes, associated input and output streams and output rules using matrix multiplication as an example: the matrices A and B are sent on input streams, and the results of the virtual processes calculating the inner products of the row and column vectors are sent on the output stream where they are assembled according to the output rule.

The ASSIST software architecture consists of a grid abstract machine (GAM) that interfaces with standard grid middleware like Globus [68]. An application manager (AM) interfaces with the GAM and supports decentralization according to various strategies including a hierarchical decentralization, where the root node is assumed to be highly available. The AM consists of module application managers (MAM) that are responsible for managing a single module, and a component application manager (CAM) that acts according to a global strategy for a component. AMs, MAMs, and CAMs are automatically generated by the ASSIST compiler according to a management policy in form of a quality of service contract provided by the application's programmer. A virtual process manager (VPM) is also generated by the compiler and is responsible for managing all the virtual processes on a single node.

Autonomic self-management in ASSIST allows satisfaction of quality of service goals like processing bandwidth for stream-based applications and completion time for non-stream computations. Autonomic behavior of the MAM basically consists of load balancing according to historical performance data of the VPs and VPMs. Figure reffig-assist shows in its lower part the control loop of the MAM which consists of (1) *monitoring* that collects data like the VPM's execution times, (2) *analysis* that verifies its performance goal and builds historical performance data, (3) *planning* that generates a sequence of reconfiguration actions in case of a broken performance goal detected by analysis, and (4) *execution* that redistributes VPs among the VPMs and possibly

requests more resources from the CAM. The CAM may also trigger restructuring in reaction to global performance variations.

AutoMate

AutoMate [4, 5, 141] is an autonomic Grid framework which extends OGSA to enable the development of autonomic Grid applications. The architecture of AutoMate consists of an application layer, a component layer, and a system layer. The AutoMate project includes the Accord programming framework and the Accord composition engine (ACE) as sub-projects.

Accord introduces the concept of an autonomic service which is basically a grid service extended with a control port and a service manager.

Accord programming framework Accord [120] is a programming framework designed to support the development of autonomic applications. Accord defines autonomic elements with a functional port, a control port and an operational port. The functional port defines the functionalities provided by the autonomic element, the control port exports sensors, actuators and constraints for autonomic management, and the operational port allows management of rules, which are formulated using if–then expressions.

Conflicting rules are resolved during rule execution by constructing a precondition and disabling rules that would change the precondition (sensor-actuator-conflict). Remaining conflicts are resolved by relaxing the precondition according to user-defined strategies (actuator-actuator-conflict).

Autonomic elements may be dynamically composed using a multi-agent infrastructure, where a central composition manager controls the autonomic elements using interaction rules. The interaction rules are generated using workflow patterns [179] which are mapped to rule templates. The rule templates are subsequently used by the composition manager to decompose the user-supplied application workflow into interaction rules. Although the interaction rules are defined centrally by the composition manager, the actual interactions are controlled by the autonomic element managers in a decentralized manner. Elements may be added, deleted, or dynamically replaced by other elements with compatible functional ports. Finally, Interaction relationships may be changed by the element managers in reaction to new or changed interaction rules.

In addition to the rule-based self-management described above, Accord has been extended to support model-based control strategies [26] using a limited look-ahead controller [3], where optimization occurs at each time-step within a prediction horizon in order to satisfy QoS requirements. Accord supports the self-configuration, self-optimizing, self-healing and self-protecting properties of autonomic systems.

Accord has been implemented as a prototype in C++ and the Message Passing Interface (MPI), and as another prototype within the CCAFFEINE CCA framework.

Rudder coordination framework The Rudder coordination framework [119, 141] of project AutoMate provides a coordination infrastructure for autonomic elements. The architecture of Rudder consists of an agent framework of context-aware agents, and a decentralized tuple space.

Agent framework An agent framework of context-aware agents manages context information, where the context consists of device profiles like CPU or memory, network resources, and software metrics like processing capabilities. The agents use this context information to select a plan based on predefined policies. Component agents are responsible for managing computations performed locally within the components, and system agents represent their collective behavior. Composition agents are transient unlike the other agent types that exist as system services. They are responsible for the composition of autonomic components.

Tuple space A decentralized tuple space is used for agent-based system coordination. In extension to the capabilities usually supported by tuple spaces, the Rudder tuple space supports matching mechanisms and reactivity. Administrators and agents both can insert and modify policies and constraints using the tuple space.

The tuple space layer implements the tuple space capabilities described above, while the content-based routing layer maps tuples to peer nodes. The Rudder tuple space builds on top of the Meteor middleware described in the next section.

Meteor middleware Meteor is a middleware infrastructure that supports interactions based on message content, called the associative rendezvous model. The middleware consists of an overlay network layer which establishes an overlay network that is composed of rendezvous peer nodes, the Squid content-based routing infrastructure which is responsible for de-centrally discovering information, and the Associative Rendezvous Messaging Substrate which is a matching engine responsible for matching message profiles.

GCM - CoreGRID component model

The CoreGRID component model [8] implements autonomic behavior of components using behavioral skeletons, which can be used to implement autonomic managers. Behavioral skeletons in GCM are algorithmic skeletons that serve as templates for the implementation of parallel computing paradigms, which are tailored for the purpose of autonomic component management and may provide a number of pre-defined plans to achieve a self-management goal.

GCM distinguishes between passive components and active components. Passive components can be monitored using introspection, and reconfigured, while active components implement the full autonomic management process. GCM is based on the Fractal [32] component model.

Jade

Jade [177] is an autonomic management middleware for distributed software environments based on the Fractal [32] component model. Legacy software pieces are encapsulated into Fractal components. The control interfaces provided by the Fractal model are then used to implement autonomic management capabilities.

A generic self-optimizing and self-healing policy is implemented that adjusts the number of application instances based on monitored parameters like the system load. Self-healing with Jade is demonstrated within a DIET [45] grid application.

4.2 Taxonomy of autonomic large-scale computing

This section presents the taxonomy of autonomic large-scale computing derived from the projects surveyed in section 4.1. It is designed to explore the current state of implementing autonomic properties in large-scale computing systems.

The taxonomy is organized as follows: Each section describes an area within the taxonomy of autonomic large-scale computing. Subsections describe categories and subcategories within an area. Within each category, the first paragraph defines the category, optionally followed by a paragraph that discusses the described category's relation to other categories. Finally, a short description of subcategories is provided. At each bottom level category, a list with short descriptions of projects that match the respective category is given.

Section 4.2.1 provides background information on building taxonomies. Section 4.2.2 classifies large-scale computing systems by their area of self-management which is the set of functions where autonomic management is introduced. Section 4.2.3 introduces a classification by the model of autonomic computing.

4.2.1 Approach for building a taxonomy

A taxonomy is a classification scheme that consists of a controlled vocabulary – that is, an authoritative list of terms that identify concepts to be classified – and a hierarchical structure of relations that represent parent-child, or generalization-specialisation relationships between the terms in that controlled vocabulary [11]. A taxonomy may be represented by a directed acyclic graph with a set of nodes consisting of the terms in the controlled vocabulary and a set of edges consisting of the relations between the terms.

If a taxonomy conforms to a strict hierarchy, the taxonomy graph is a tree with the term identifying the domain that is being classified by the taxonomy as root. However, some taxonomies require a node to appear at more than one location within the taxonomy, which is referred to as a polyhierarchy and is represented as a connected graph. Another approach classifies the terms of the controlled vocabulary by multiple sub-taxonomies, called facets or aspects. This type of taxonomy maps to a forest or to a general ordered directed graph, if polyhierarchies are also used. Table 4.1 lists

Facets / *Polyhierarchy*	*no*	*yes*
no	tree	forest
yes	connected	general

Table 4.1: Taxonomies represented by directed acyclic graphs

the taxonomies and resulting graph types described above depending on the use of polyhierarchies and facets.

Glass and Vessey [80] suggest to introduce taxonomies that structure the body of knowledge in a field, in order to systematically describe that field, interpret aspects of relevance, and to allow predicting future areas of development. Developing a taxonomy involves establishing criteria for classification of items and specifying methods for assigning items to categories completely and unambiguously.

In case of a general-purpose taxonomy, classification criteria are usually driven by properties of the observed items themselves. Sometimes, the development of a taxonomy is driven by a special interest like the needs of an industry or conformance to some technology or standard, though. In this case, classification criteria may be driven by that interest, leading to a specialized taxonomy. In either case, the classification should provide accurate definitions of item descriptors and guidelines for placing items in categories, so that all items that occur in the field of interest may be unambiguously assigned to mutually exclusive categories.

The taxonomy of autonomic large-scale computing presented in this chapter is a general-purpose taxonomy without polyhierarchies which is based on facets in order to classify systems by functional and non-functional criteria.

4.2.2 Self-management areas in large-scale computing

The proposed taxonomy of autonomic large-scale computing classifies systems by the area of self-management in large-scale computing projects, which is the set of functions that is addressed by applying autonomic methods.

The areas of self-management that have been observed within the survey of autonomic large-scale computing are application management, data management, development of autonomic large-scale computing applications, quality of service, and resource management. These areas of autonomic management in large-scale computing systems form the top level of the taxonomy. Figure 4.8 shows the top two levels of the taxonomy of self-management areas in large-scale computing.

The remainder of this section expands the taxonomy by describing the areas of autonomic large-scale computing and their respective categories and subcategories, in order to allow the unambiguous assignment of large-scale computing projects to areas and categories within the taxonomy. The section concludes with a discussion of the taxonomy.

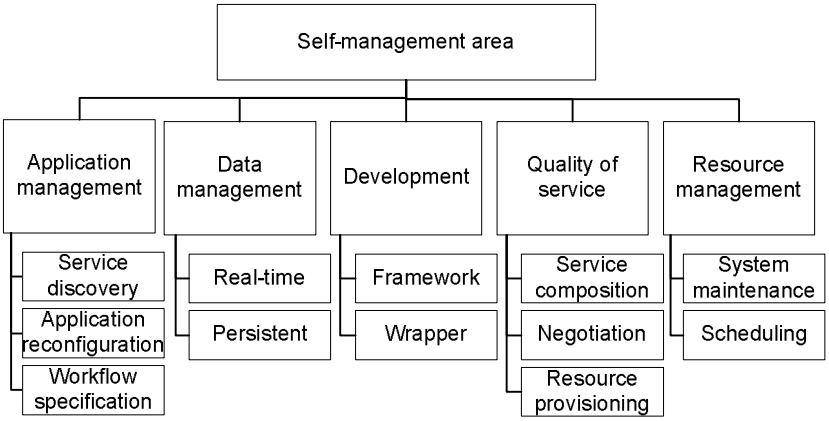

Figure 4.8: Top two levels of the proposed taxonomy of self-management areas in large-scale computing projects.

4.2.2.1 Application management

The application management area of autonomic large-scale computing includes all projects that investigate self-management of functional properties of applications executed on large-scale computing systems. Projects in this category address the problem that run-time characteristics of the executing system are not known at build time and allow the dynamic selection or modification of components and services in order to address availability and performance characteristics at run-time.

Projects that investigate self-management of non-functional application properties are assigned to the quality of service category. Projects that address the problem of dynamic run-time characteristics in large-scale computing systems by autonomically changing the resource allocation are assigned to the resource management category.

Figure 4.9 shows the taxonomy of the application management area of autonomic large-scale computing which consists of the service discovery, application reconfiguration, and workflow specification categories described in the following paragraphs.

Service discovery

Projects in the service discovery category of application management use autonomic methods to lookup services at application run-time that conform to a specification given at build time.

The projects included in the survey implement this functionality either by using agents or by ontologies.

Agents The agents subcategory of service discovery in application management includes projects that use software agents in order to implement self-managing service discovery at application run-time in large-scale computing systems.

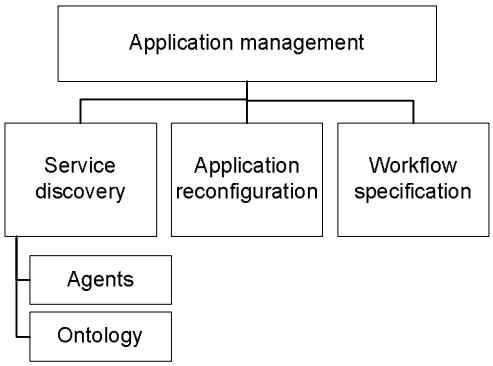

Figure 4.9: Taxonomy of application management in autonomic large-scale computing

The following project implements autonomic service discovery using agents:

- Guo et al. [90] propose a model of an autonomic grid computing system that implements autonomic elements described in Section 3.3.2 as self-organizing agents which announce their capabilities and interests. Other agents may delegate work based on the announced capabilities and interests. Based on this model, a mechanism of autonomic system management of a grid is described.

Ontology The ontology subcategory of service discovery in application management includes projects that use ontologies in order to implement self-managing service discovery at application run-time in large-scale computing systems. An ontology consists of a set of object descriptions using a controlled vocabulary and a set of relationships between the objects. It is used to present services using high-level descriptions in order to allow service selection using semantic criteria.

The following projects implement autonomic service discovery using an ontology:

- Guan et al. [89] implement semantic service matching based on a context ontology in order to facilitate grid access from pervasive devices like mobile phones.
- Hau et al. [92] implement autonomic service adaptation in the ICENI [78] grid middleware framework by using ontological annotation of service definitions. Based on that service ontology, the framework supports semantic matching and service adaptation at run-time.

Application reconfiguration

Projects in the application reconfiguration category of application management use autonomic methods in order to perform reconfiguration actions of an application that is executed on a large-scale computing system.

The following project impements autonomic application reconfiguration:

- AutoGrid [161] is an autonomic extension of the InteGrade [81] desktop grid middleware that allows changing application parameters and replacement of application algorithms at run-time based on resource availability and QoS requirements.

Workflow specification

Projects in the workflow specification category of application management use autonomic methods in order to allow the specification of a workflow at run-time based on a high-level description.

The following project implements autonomic workflow specification:

- Zhang et al. [195] present an autonomic grid workflow engine for the TigMNS traffic information system that maps high-level missions specified using a matrix-based distributed event controller (DEC) to a grid workflow based on run-time characteristics of the requesting client and the information system grid.

4.2.2.2 Data management

The data management area of autonomic large-scale computing includes all projects that use self-management for the processing of data maintained in large-scale computing systems.

The categories of data management are real-time data and persistent data.

Real-time data

Projects in the real-time category of data management in large-scale computing systems use autonomic methods in order to allow the processing of data streams arriving in real-time.

The following project implements autonomic management of real-time data:

- GATES [51] is an autonomic grid middleware that supports the processing of data streams by adjusting the data sampling rate in order to achieve the best accuracy under the real-time constraint.

Persistent data

Projects in the persistent data category of data management in large-scale computing systems use autonomic methods in order to manage persistent huge data sets used for large-scale computing applications.

The following project implements autonomic management of persistent data:

- Zhao et al. [197] propose an autonomic grid data management architecture for grid computing based on the grid virtualized file system (GVFS) [196] that optimizes data placement and data replica regeneration for grid applications.

4.2.2.3 Development

The development area of autonomic large-scale computing includes projects that investigate tools and methods for the development of autonomic large-scale computing applications.

The categories of autonomic large-scale computing development are frameworks and wrappers.

Frameworks

The frameworks category of autonomic large-scale computing development includes projects that provide development frameworks for autonomic large-scale computing applications.

The following projects provide a development framework for autonomic large-scale computing applications:

- Project AutoMate [141] provides a framework for the development of autonomic applications for the gridby extending the Open Grid Services Architecture (OGSA) described in Section 3.1.1 in order to provide autonomic capabilities.
- The GCM Grid Component Model [8] provides a framework based on behavioral skeletons that allows to implement autonomic control in Grid applications.

Wrappers

The wrappers category of autonomic large-scale computing development includes projects that support the development of software wrappers for the enhancement of legacy applications with autonomic capabilities.

The following project supports the development of autonomic large-scale computing applications using wrappers:

- Jade [177] is a development framework that provides a component model for the deployment and reconfiguration of software environments and control loops for autonomic behavior. Jade allows to develop autonomic wrappers for legacy software like grid middleware systems.

4.2.2.4 Quality of service

The quality of service area of autonomic large-scale computing includes projects that investigate self-management for the specification and maintenance of QoS properties in large-scale computing systems. Quality of service refers to non-functional properties like response time and availability. A set of QoS properties may be formally specified as a service-level agreement (SLA) in order to guarantee non-functional criteria.

Figure 4.10 shows the taxonomy of quality of service management in autonomic large-scale computing which is categorized into service composition, negotiation, and resource provisioning.

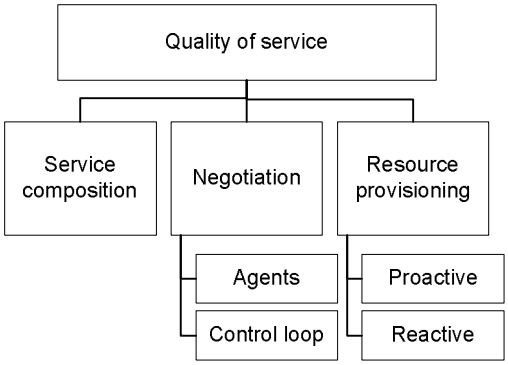

Figure 4.10: Taxonomy of autonomic quality of service management in large-scale computing

Service composition

Projects in the service composition category of QoS in autonomic large-scale computing use autonomic methods in order to compose services such that non-functional criteria are met. Self-management of functional criteria is categorized into the appropriate category of the application management area instead.

The following project implements autonomic service composition for QoS:

- Within the PAWS [16] framework for the composition of web services subject to QoS requirements, Ardagna et al. [18] investigate the service selection and resource allocation problem from the user's and the provider's perspective. Anselmi et al. [14] provide algorithms to solve the web service composition problem.

QoS negotiation

Projects in the negotiation category of QoS in autonomic large-scale computing use autonomic methods in order to guide a negotiation process that selects a service from a set of services such that given QoS requirements are satisfied. The projects included in the survey implement QoS negotiation either as an agent-based system or using an autonomic control loop.

Agents The agents subcategory of QoS negotiation in autonomic large-scale computing includes all projects that implement self-managed QoS negotiation using software agents.

The following project implements autonomic QoS negotiation using agents:

- Weng et al. [185] present a negotiation model in a market-oriented grid, where resource providers and consumers are represented by agents.

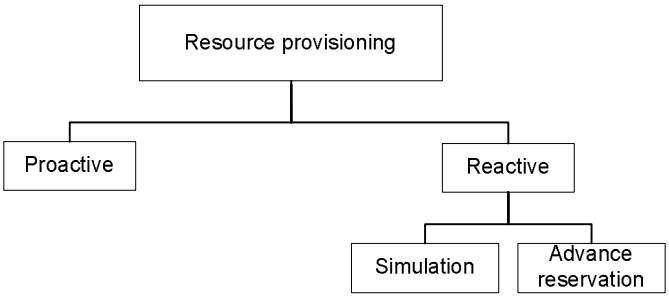

Figure 4.11: Taxonomy of autonomic resource provisioning in large-scale computing

Control loop The control loop subcategory of QoS negotiation in autonomic large-scale computing includes all projects that implement self-managed QoS negotiation based on an autonomic control loop as described in Section 3.3.2.

The following projects implement autonomic QoS negotiation using a control loop:

- Koller et al. [112] propose a proxy-based architecture in order to manage service level agreements as a third party between service provider and consumer. The architecture supports negotiation, enactment, and monitoring of SLAs.
- The VieSLAF [29] framework implements negotiation bootstrapping which is the negotiation about parameters like negotiation protocols or SLA specification languages, and service mediation which is the mapping between inconsistencies in SLA templates using an autonomic control loop that triggers the application of bootstrapping strategies and the application of SLA mappings.

Resource provisioning

Projects in the resource provisioning category of QoS in autonomic large-scale computing use autonomic methods for the provisioning of resources subject to QoS requirements.

Figure 4.11 shows the taxonomy of autonomic resource provisioning which is categorized into proactive and reactive provisioning explained in the following paragraphs.

Proactive provisioning Proactive provisioning refers to resource provisioning in advance such that QoS requirements are expected to be met in future. The projects included in the survey implement proactive provisioning either by simulation or using advance reservation.

Simulation Proactive provisioning by simulation refers to the simulation of autonomic large-scale computing systems in order to gather a system profile based on load variations. Results of the simulation allow to determine the number of resources required to meet QoS requirements in advance.

The following project implements autonomic proactive resource provisioning by simulation:

- Assunção et al. [22] provide a simulation framework implemented using GridSim [39] in order to evaluate policies for resource allocation in utility computing environments.

Advance reservation Proactive provisioning by advance reservation refers to systems where users are required to reserve resources before usage, such that the system provider may provision enough resources in order to meet QoS requirements.

The following projects implement autonomic proactive resource provisioning by advance reservation:

- AbdelSalam et al. [1] propose a change-management scheme that uses advance reservation in order to optimize a cloud computing system such that resources may be made available for change management while the QoS requirements specified by advance reservations are met.

- Libra+$Auto [189] uses advance reservation in order to provide autonomic resource pricing for utility computing services based on resource availability and user application and service requirement.

Reactive provisioning Projects that employ autonomic reactive provisioning use self-management techniques in order to detect expected or actual failures to meet QoS requirements and apply appropriate reconfiguration in order to reach a system state that again satisfies QoS requirements.

The following project implements autonomic reactive resource provisioning:

- Iqbal et al. [103] propose an architecture for management of SLAs promising a certain service response time which is implemented within the Eucalyptus [139] cloud. It is based on a load balancer and a component that monitors the load balancer's log files. On detection of SLA violations, the manager triggers the provisioning of additional resources.

4.2.2.5 Resource management

The resource management area of autonomic large-scale computing includes projects that investigate self-management of large-scale computing resources.

Figure 4.12 shows the taxonomy of autonomic resource management in large-scale computing which is categorized into system management where self-management is applied in the context of internal management purposes, and scheduling where autonomic resource management is used for scheduling of applications, jobs, or workflows.

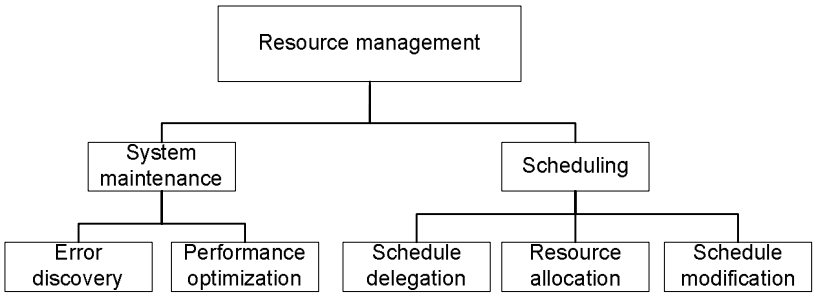

Figure 4.12: Taxonomy of autonomic resource management in large-scale computing

System management

Autonomic resource management in the context of system management of large-scale computing systems includes work that employs self-management in order to keep a large-scale computing system operable according to its specification.

The system management category includes the subcategories of error discovery and performance optimization.

Error discovery The error discovery subcategory of autonomic resource management in the context of system management of large-scale computing systems includes projects that employ autonomic resource management for the purpose of detecting and correcting errors. Work included in this subcategory implements the autonomic self-healing capability described in Section 3.3.1.

The following project implements autonomic error discovery:

- Dai et al. [54] propose an architecture for self-diagnosis and self-healing in a cloud computing system that uses a multiple-valued decision diagram in order to determine the severity of an error, and a Naïve Bayes classifier for the classification of the detected error.

Performance optimization The performance optimization subcategory of autonomic resource management in the context of system management of large-scale computing systems includes projects that employ autonomic resource management for system performance optimization.

The following projects implement autonomic performance optimization:

- JOpera [94] is a distributed workflow engine that employs self-management of the number of navigator threads and dispatcher threads in order to minimize task runtime.
- Gounaris et al. [84] present optimization algorithms that are applied for minimizing the response time of a grid-based database service by modifying its block size.

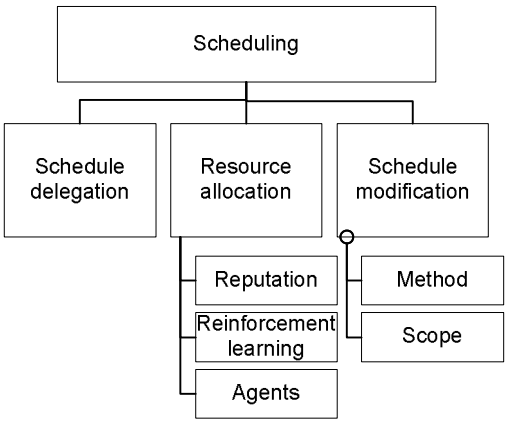

Figure 4.13: Taxonomy of autonomic scheduling in large-scale computing

Scheduling

The scheduling category of resource management in autonomic large-scale computing includes projects that investigate autonomic scheduling of jobs and workflows in large-scale systems.

While projects in the system management category of resource management described above autonomically adapt resources in order to process a given job schedule, work in this category autonomically adapts job schedules for a given set of resources. Thus, system management and scheduling may be seen as complementary categories of resource management.

Figure 4.13 shows the taxonomy of scheduling in autonomic large-scale computing which is categorized into schedule delegation, resource allocation, and schedule modification and described in the following paragraphs.

Schedule delegation The schedule delegation subcategory of autonomic scheduling includes all projects where a client delegates the scheduling of jobs or workflows to an autonomic large-scale computing system.

The following project implements autonomic scheduling by schedule delegation:

- AutoMAGI [157] is an autonomic grid middleware that supports access from mobile devices with unreliable connectivity to grid services by providing means to store identification data and job results of disconnected clients into the autonomic knowledge component, in order to initiate transfer of the results of completed jobs to the client upon reconnection.

Resource allocation The resource allocation subcategory of autonomic scheduling includes all projects that employ self-management for the allocation of resources to jobs or workflows. Based on the resource selection method, the taxonomy distinguishes

between reputation-based approaches where resources are selected based on historical performance data, reinforcement learning approaches where resources are allocated based on a reward received from previous resource allocations, and agent-based approaches where agents represent both resource providers and resource consumers.

Reputation-based resource allocation This subcategory includes projects that employ a reputation-based approach for resource allocation. Reputation is information associated with a resource that is established based on historical information about properties like resource availability and performance.

The following project implements resource allocation for autonomic scheduling using a reputation-based approach:

- Rahman et al. [151] introduce a reputation-based autonomic scheduling architecture for the Gridbus Workflow Engine (GWFE) [85] consisting of several grid sites managed by an autonomic scheduler that are connected by an overlay network. A reputation system calculates the reputation score of a grid site based on the feedback of previous interactions with that site.

Resource allocation by reinforcement learning This subcategory includes projects that employ an approach for resource allocation based on reinforcement learning. Reinforcement learning is a machine learning scheme where a system chooses actions that are expected to be associated with an award based on previous experience.

The following project implements resource allocation for autonomic scheduling using an approach based on reinforcement learning:

- Perez et al. [148] combine reinforcement learning with utility functions in order to provide an autonomic scheduler at site level of the EGEE [64] grid infrastructure. The reinforcement learning framework expresses user satisfaction as a utility function decreasing with the expected job completion time.

Resource allocation using agents This subcategory includes projects that use agents representing jobs for resource allocation.

The following projects implement resource allocation for autonomic scheduling using an agent-based approach:

- OrganicGrid [48] employs strongly mobile agents for decentralized scheduling of independent identical subtasks of an application in a self-organizing peer-to-peer desktop grid.

- So-Grid [67] is another self-organizing agent-based peer-to-peer desktop grid that uses bio-inspired algorithms for autonomic resource allocation and agent replication.

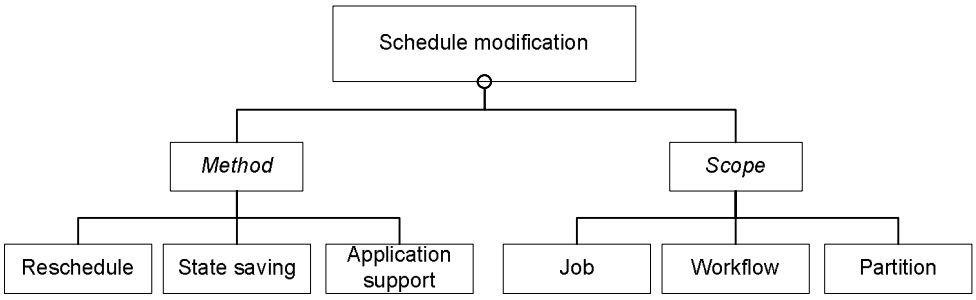

Figure 4.14: Taxonomy of autonomic schedule modification in large-scale computing. Each project shall be categorized by method and by scope.

Schedule modification The schedule modification subcategory of autonomic scheduling includes all projects that employ self-management in order to modify schedules in reaction to changing availability or performance characteristics of a large-scale computing system.

The taxonomy of schedule modification which is shown in Figure 4.14 categorizes projects by the method of schedule modification and by its scope. The scope of schedule modification may be a single job, a workflow consisting of multiple jobs and their interdependencies, or a partition of a job. Methods of schedule modification are to reschedule jobs, job partitions or those parts of a workflow that have not yet been executed with a different resource allocation, state saving or checkpointing of a job or job partition followed by suspending and resuming with a different resource allocation, or requiring application support for dynamically changing the resource allocation of a running job.

Rescheduling Rescheduling refers to executing a failed job or job partition again from the beginning using a different set of resources.

The following projects implement schedule modification by rescheduling:

- Jarvis et al. [105] employ a genetic algorithm on jobs in a queue of a grid system that continuously searches for a better schedule. Thus they implement rescheduling on job level.

- Dasgupta et al. [55] implement rescheduling on job level for autonomic management of job-flows in a grid environment. When job monitoring detects a failing job, it is resubmitted to an alternate location.

- Pegasus [58] is a grid workflow management system. Autonomic workflow management in Pegasus [115] implements rescheduling on workflow level by employing a scheduling strategy that allows to reschedule the workflow based on current resource performance data.

- Paton et al. [143] present a methodology for adaptive workload management on cloud systems using utility functions that implements rescheduling on workflow level. Autonomic workflow execution is given as an example application, where an optimization algorithm explores alternative assignments of a given schedule, possibly leading to rescheduling of the workflow to a different site.

- The ASSIST [9] component framework partitions a problem into parallel modules (parmod) using a coordination language. Parmods are distributed among virtual processors (VP) which are able to synchronize with each others. Virtual process managers (VPM) contain a group of VPs and are distributed among grid nodes. An autonomic resource management component of the ASSIST framework redistributes VPs among VPM on detection of violation of a previously defined performance goal (goal based mode) or a better distribution strategy (best effort mode). Since the ability of redistribution of already executing parmods is not explicitly mentioned, it is assumed here that redistribution is restricted to non-executing parmods. Thus, ASSIST implements rescheduling on partition level.

State saving State saving refers to resuming a failed job or job partition using a different set of resources based on an intermediate state of execution that has been saved before.

The following project implements schedule modification by state saving on job level:

- CoordAgent [76] is a desktop grid middleware based on mobile agents that represent client users and coordinate job execution and allocation of desktop resources. Agents migrate to other resources using checkpointing, when a desktop computing resource suddenly becomes unavailable (e.g. being reclaimed by its interactive user).

The following projects implement schedule modification by state saving on workflow level:

- Nichols et al. [135] propose a model for autonomic workflow execution within the GridFlow [43] grid workflow execution engine using mobile agents. The model implements schedule modification by allowing agents to migrate autonomically between resources. Local and remote checkpointing allows continuation of the workflow in case of a resource failure.

- Nordstrom et al. [137] present a model based on the WorkflowML language which employs predictive self-analysis of workflow execution in order to detect workflows that may not be completed due to failed resources. Such workflows may be suspended using checkpointing in favor of workflows that may be completed, which matches the definition for the taxonomical category schedule modification by state saving on workflow level.

The following project implements schedule modification by state saving on partition level:

- OptimalGrid [59] is a self-managing desktop grid middleware designed for solving connected parallel problems using a finite element model that partitions a computation problem among grid nodes. OptimalGrid implements self-configuration by determining the optimal number of grid nodes and self-optimization by selecting a node configuration based on properties like network latency or computing power. Intermediate results are saved on a distributed whiteboard and are reused in case of a node failure.

Application support Application support refers to systems that require jobs to support changes of the resource allocation at run-time.

The following project implements schedule modification by application support:

- DYNACO [34] implement self-adapting grid resource management using a control loop that detects and reacts to changes in the execution environment. The actual management functions are delegated to plugins. DYNACO requires jobs to adapt to resource allocation changes at run-time. This constitutes schedule modification by application support on job level.

4.2.2.6 Discussion of the large-scale computing area taxonomy

The area of autonomic large-scale computing taxonomy intends to categorize autonomic large-scale computing by the problem that is addressed using self-management. It has identified five functional areas of autonomic large-scale computing on its top level. Application management refers to the selection of services or components for applications. Data management refers to self-adaptation of data processing within large-scale computing systems. Development refers to projects that help for the development of autonomic large-scale computing systems and applications. Quality of service management refers to work that provides self-management within the process of establishing and maintaining QoS in large-scale distributed computing systems. The resource management category finally includes work that provides self-management within the process of maintaining and assigning system resources to applications.

Applying the taxonomy to the surveyed projects shows that projects that usually are considered to be related to each other (e.g. workflow management systems) appear in different categories since the categories are based on self-management functionality, not project functionality or technology. Similarly, agents appear as subcategory of service discovery in application management, negotiation in quality of service, and resource allocation in scheduling, since agents have been used to implement those different kinds of functionality.

These observations show the need for a complementary taxonomy covering non-functional aspects, which will be presented in the following section.

4.2.3 Approaches in autonomic large-scale computing

The approach of autonomic large-scale computing taxonomy describes the methods and technologies that have been used in order to develop an autonomic large-scale computing system. On its top level, the taxonomy distinguishes between work that presents a whole autonomic large-scale computing system, and work that explores how to autonomically manage certain aspects of large-scale computing.

Figure 4.15 shows the taxonomy of autonomic large-scale computing approaches.

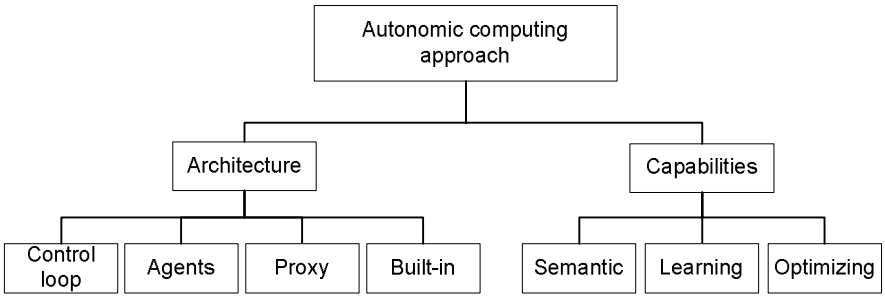

Figure 4.15: Approaches in autonomic large-scale computing

4.2.3.1 Architecture of autonomic large-scale computing systems

The architecture of autonomic large-scale computing systems taxonomy classifies systems by their autonomic architecture. The main architectures observed are based on control loops or on agents, employ a self-management proxy or have autonomic capabilities built into the main application.

Control loop based systems

The category of control loop based autonomic large-scale computing systems includes all projects that use a control loop in order to provide autonomic functionality. Particularly this includes systems that implement the autonomic computing architecture described in Section 3.3, which proposes a system composed of autonomic elements, each of them consisting of an autonomic manager that manages a resource using a loop decomposed into monitor, analyze, plan, and execute phases (MAPE loop).

The following projects fall into the category of control loop based autonomic large-scale computing systems:

- AutoMAGI [157] is a grid middleware for mobile device access that is composed of autonomic components implementing the MAPE loop described above.
- Guo et al. [90] employ agent-based autonomic managers that manage grid services using a MAPE control loop. Grid services managed by autonomic man-

agers are categorized into management services for system management work, application services for human demand, and lower-level agents.

- Dynaco [34] is a framework that extends grid resource management systems in order to support adaptable applications. Adaptability in Dynaco [35] is decomposed into observe, decide, plan, and execute steps delegating their functionality to external applications.

- Zhao et al. [197] provide autonomic data management for grid applications. The system architecture consists of services for data scheduling, replication, and file system operations. Each service is implemented as an autonomic element containing a MAPE loop.

- VieSLAF [29] is an SLA management framework supporting negotiations using non-matching SLA templates. It uses an autonomic MAPE control loop for negotiation bootstrapping and service mediation [28] which are the initial steps of a negotiation process that is also applicable to self-managing cloud services [27].

- Autonomic management within the Gridbus Workflow Engine [151] is implemented by employing an autonomic manager comprising a MAPE loop for each participating grid site. The autonomic managers communicate with each other using a tuple space [191].

- JOpera [94] is a web service composition tool supporting grid workflows that implements autonomic management of the number of executor and navigator threads, and recovery of failed nodes using a closed-feedback loop controller.

- Workflow management within the grid-based TigMNS traffic information system [195] uses a MAPE control loop for workflow specification.

- An autonomic extension for the Pegasus grid workflow management system [115] by providing an autonomic manager implementing the MAPE control loop for workflow adaptation.

- WorkflowML [137] is a workflow specification language for large-scale computing environments that supports predictive workflow analysis for autonomic workflow management. It currently supports the monitoring and analysis steps of the MAPE control loop.

- ASSIST [9] is a development framework for grid applications under the control of a module application manager that implements the MAPE control loop for self-adaptation of resource allocations.

- GCM [8] is a component model for autonomic grid applications that distinguishes between passive components supporting introspection and dynamic reconfiguration, and active components defining methods for implementing the full MAPE control loop.

- Jade [177] is a framework that wraps legacy applications including grid middleware systems into autonomic managers supporting a feedback control loop.

Agent-based systems

The category of agent-based autonomic large-scale computing systems includes all projects that use agents in order to provide autonomic functionality.

This section first describes the projects that employ agents. Then, a taxonomy of agents is given, and the agent-based projects are categorized based on that taxonomy.

The following projects fall into the category of agent-based autonomic large-scale computing systems:

- CoordAgent [76] is a desktop grid middleware based on mobile agents that represent client users and coordinate job execution and allocation of desktop resources.
- Guo et al. [90] employ agents representing autonomic elements for system management work in grid computing environments. Agents communicate with each other based on their capabilities and interests in order to establish relationships of acquaintance, collaboration, or notification. Capabilities and interest which are represented using first-order predicate logic and ontologies, are evaluated using logic reasoning and semantic cooperation.
- Jarvis et al. [105] present an autonomic grid middleware service that employs a network of peer-to-peer agents for multi-domain task management. Each domain participating in the grid is represented by an agent that acts as a resource broker by publishing information about the local task queue and resource utilization using the Monitoring and Discovery Service (MDS) from Globus Toolkit [68].
- OrganicGrid [48] is a desktop grid middleware employing strongly mobile agents that form an overlay network for autonomic scheduling. Agents represent a large computational task consisting of subtasks that may be delegated to other agents. In addition, agents are able to execute subtasks from other agents. Agents maintain parent-child relationships with each other and may choose to adopt grandchildren based on performance metrics.
- So-Grid [67] is a desktop grid middleware based on a system of agents that use bio-inspired algorithms for resource discovery and allocation which implement stochastic pick and drop operations for grid resource descriptors.
- Weng et al. [185] have agents represent consumers and producers in a market-oriented grid environment based on a fuzzy cognitive map (FCM) model that represents negotiation issues like quality of service.
- GridFlow [43] is a grid workflow management system built on top of the ARMS [42] resource management system based on mobile agents. Self-management in GridFlow [135] uses agents that support checkpointing and migration for decentralized workflow execution, where each agent represents a distinct sub-workflow.

Each agent contains pre- and post-execution performance monitoring instructions.

- AutoMate [141] is a framework for the development of autonomic grid computing systems. Within AutoMate, autonomic elements are modeled as components and composed using the Accord [120] composition engine. Composition plans are executed using the Rudder coordination middleware [119] that forms the coordination layer of the AutoMate project. Context-aware agents in Rudder maintain information about system resources and act based on pre-defined policies.

Several taxonomies for agents exist. Franklin et al. [74] classify agents by properties that include being communicative, learning, or mobile. Brustoloni's [33] taxonomy categorizes software agents as regulating, planning, and adaptive agents. Figure 4.16 shows a taxonomy of agents that is based on properties observed in the surveyed projects.

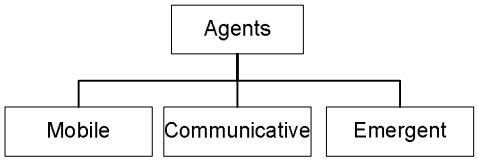

Figure 4.16: Taxonomy of agents in autonomic large-scale computing

Communicative agents Communicative agents are agents that are able to interact with other agents in order to achieve a goal. Agents may coordinate their behavior, or they may negotiate in order to establish a contract.

Communicative agents are used in CoordAgent [76], by Guo et al. [90], Jarvis et al. [105], OrganicGrid [48], Weng et al. [185], GridFlow [135], and AutoMate [141].

Mobile agents Mobile agents are agents that are able to move between different execution sites.

Mobile agents are used in CoordAgent [76], OrganicGrid [48], by Weng et al.. [185], and in GridFlow [135].

Multi-agent systems with emergent behavior A multi-agent system with emergent behavior is composed of a large number of simple agents that together produce complex behavior. The principle of emergence which is also called swarm intelligence, is inspired by the behavior of organisms like ants or termites.

Multi-agent systems with emergent behavior are employed in the OrganicGrid [48] and So-Grid [67] projects.

Proxy-based systems

Proxy-based autonomic large-scale computing systems employ a proxy for communication between components which exercises self-management functionality.

The following projects fall into the category of proxy-based autonomic large-scale computing systems:

- Dasgupta et al. [55] employ a transparent proxy for grid job-flow management that triggers an autonomic recovery component on detection of a failure.

- Iqbal et al. [103] minimizes the response time of an cloud-based web applications by monitoring a load balancer log file in real-time and instantiating additional virtual machines if needed.

- Koller et al. [112] present an SLA management architecture that employs a proxy for autonomic SLA negotiation.

Systems with built-in autonomic behavior

This category comprises all systems that have autonomic behavior built into their main functionality. Built-in autonomic behavior includes system architectures where self-management work is triggered by application-specific events or implied by the application's main purpose.

The following projects fall into the category of large-scale computing systems with built-in autonomic behavior:

- AutoGrid [161] is an autonomic desktop grid middleware that extends the InteGrade [81] middleware using the Adapta [160] framework which describes adaptable elements and reconfiguration actions using an XML-based language. Neto et al. [134] present an alternate approach that uses dynamic interceptors in order to provide dynamic adaptation for InteGrade. Both approaches employ functionality provided by the CORBA middleware.

- Based on the GridSim [39] grid simulation toolkit, Assunção et al. [22] present a framework for the modeling and simulation of policies for resource management, negotiation and service provisioning.

- Jarvis et al. [105] present an autonomic scheduling middleware service that besides employing agents for multi-domain task management as described above, employs a co-scheduler on the job queue in order to optimize the current schedule.

- OptimalGrid [59] employs self-configuration and self-optimization after each step in the calculation of a parallel problem.

4.2.3.2 Autonomic large-scale computing capabilities

The taxonomy of autonomic large-scale computing capabilities classifies capabilities of large-scale computing systems that result in autonomic behavior.

The categories of autonomic large-scale computing capabilities are semantic capabilities, machine learning, and optimization.

Semantic capabilities

The category of semantic capabilities in autonomic large-scale computing systems includes all projects that use semantic web technologies in order to produce autonomic behavior.

The following projects present autonomic large-scale systems based on semantic web technologies:

- Guan et al. [89] present a grid service discovery middleware based on the OWL-S semantic language in order to provide access for pervasive devices. The autonomic large-scale computing functionality provided by this work is service discovery.
- Hau et al. [92] present an autonomic service adaptation framework for the ICENI [78] grid middleware that provides semantic service matching based on OWL-S. The autonomic large-scale computing functionality provided by this work is service adaptation.

Machine learning

The category of machine learning in autonomic large-scale computing systems includes all projects that employ learning in order to produce autonomic behavior.

The following projects present autonomic large-scale systems based on machine learning:

- Perez et al. [148] present a grid scheduler that uses reinforcement learning based on a utility function for schedule optimization. The autonomic large-scale computing functionality provided by this work is scheduling.
- Dai et al. [54] present a framework for self-diagnosis and self-healing in cloud systems based on multivariate decision diagram and a Naïve Bayes classifier. The autonomic large-scale computing functionality provided by this work is error recovery.

Optimization

The category of autonomic optimization in large-scale computing includes all projects that explore how to use autonomic computing methods in order to optimize existing functionality of a large-scale computing system.

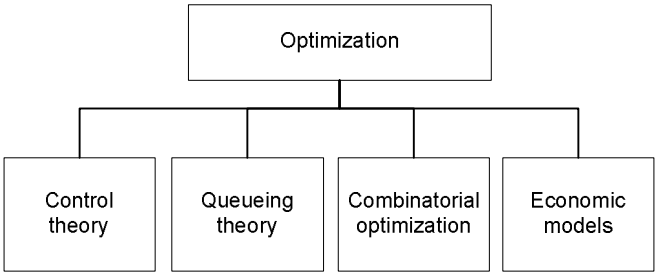

Figure 4.17: Taxonomy of optimization in autonomic large-scale computing

Figure 4.17 shows the taxonomy of autonomic optimization in large-scale computing.

The surveyed projects apply optimization using methods from queueing theory, control theory, combinatorial optimization and economic modeling.

Queueing theory The subcategory of queueing theory in autonomic large-scale computing optimization includes all projects that apply self-management for the optimization of a set of properties using methods from the field of queueing theory.

The following project uses methods from queueing theory in order to implement self-optimization in large-scale computing:

- GATES [50] optimizes the processing of real-time data streams for grid computing by self-adaptation of the sampling rate such that the length of the input queue remains within defined boundaries, which is an indicator for the grid system in question being able to process the data at the current rate.

Control theory The subcategory of control theory in autonomic large-scale computing optimization includes all projects that apply self-management for the optimization of a set of properties using methods from the field of control theory.

The following project uses methods from control theory in order to implement self-optimization in large-scale computing:

- Gounaris et al. [84] present algorithms based on runtime optimization and switching extremum control in order to minimize the total response time of an OGSA-DAI database service.

Combinatorial optimization The subcategory of combinatorial optimization in autonomic large-scale computing includes all projects that apply self-management for the optimization of a set of properties using methods from the field of combinatorial optimization.

The following projects use methods from combinatorial optimization in order to implement self-optimization in large-scale computing:

- AbdelSalam et al. [2] provide a model for power management in a cloud computing center based on the bin packing problem in order to compute the optimal number of servers and their running frequencies.

- Work in the PAWS project [14, 20] describes the optimization of quality of service in web service selection and composition in the context of grid computing. The large-scale computing property optimized by this project is quality of service.

Economic modeling The subcategory of economic modeling in autonomic large-scale computing includes all projects that apply self-management for the optimization of a set of properties using economic models.

The following projects use economic models in order to implement self-optimization in large-scale computing:

- Libra+$Auto [189] is a pricing mechanism that allows charging variable prices for resource usage based on expected workload demand in a cloud computing environment that employs advance reservations. The large-scale computing property optimized by this project is the resource price.

- Paton et al. [143] present a methodology for designing adaptive workload execution schemes based on utility functions. The large-scale computing property that is to be optimized is to be specified during the design process proposed by the methodology.

5 Catalog of autonomic large-scale computing projects

This chapter provides a catalog of autonomic large-scale computing projects classified by the criteria established by applying the taxonomy developed in Chapter 4.

Section 5.1 lists general information about the projects in the catalog. Section 5.2 classifies the projects by the taxonomy that has been established in Chapter 4. Section 5.3 finally summarizes the results of the classification within this catalog.

5.1 Project information

Table 5.1 lists project information for the work presented in this catalog. Some projects in this catalog are autonomic large-scale computing projects that have been given a name. Other work adds autonomic capabilities to a large-scale computing project that is already known by a name. Such work is listed under that name. Other publications either present conceptual work in the field of autonomic large-scale computing that is not associated with specific projects or describe autonomic large-scale computing projects but do not assign a name to them. Such work that cannot be associated with a project name is listed under the authors of a publication that represents the work.

In addition to the project's or publication's name, the organization that hosts the project or the affiliation of the first author of the publication representing the work is given in order to help identify the project. Finally, the table states a reference to the section within this thesis that provides a project description.

The project information table has the following columns:

Name The name of the autonomic large-scale computing project, or authors and date of the publication, as described above.

Organization The organization that hosts the autonomic large-scale computing project, or the affiliation of the first author of the publication, as described above.

Grid The grid column states whether the project covers the discipline of autonomic grid computing. A project covers autonomic grid computing if a publication exists that investigates the application of self-management to grid computing within the context of the project.

Cloud The cloud column states whether the project covers the discipline of autonomic cloud computing. A project covers autonomic grid computing if a publication

Name	Organization	Grid	Cloud
Grid Middleware			
AutoMAGI	Kyung Hee University, South Korea	x	
CoordAgent	University of Washington, Bothell, USA	x	
Dasgupta et al.	IBM Corporation, South Dakota State University, Florida International University	x	
Dynaco	Institut National de Recherche en Informatique, France	x	
GATES	Ohio State University, Columbus, USA	x	
Gounaris et al.	University of Manchester, UK	x	
GridSim	University of Melbourne, Australia	x	
Guan et al.	University of Southampton, UK	x	
Guo et al.	Zhejiang University, China	x	
ICENI	Imperial College of Science, Technology and Medicine, UK	x	
InteGrade	Federal University of Maranhão, Brazil	x	
Jarvis et al.	University of Warwick, Coventry, UK	x	
OptimalGRID	IBM Corporation	x	
Organic Grid	Ohio State University, USA	x	
Perez et al.	CNRS and Université Paris-Sud, France	x	
So-Grid	Institute for High Performance Computing and Networking (ICAR-CNR), Italy	x	
Zhao et al.	University of Florida, USA	x	
Cloud computing systems			
AbdelSalam et al.	Old Dominion University, Norfolk, USA		x
Dai et al.	University of Electronic Science and Technology of China		x
Iqbal et al.	Asian Institute of Technology, Thailand		x
Libra+$Auto	University of Melbourne, Australia		x
Paton et al.	University of Manchester, UK		x
Quality of service frameworks			
Koller et al.	Höchstleistungsrechenzentrum Stuttgart, Germany	x	x
PAWS	Politecnico di Milano, Italy	x	
VieSLAF	Vienna University of Technology, Austria	x	x
Weng et al.	Nanyang Technological University, Singapore	x	
Workflow management systems			
Gridbus	University of Melbourne, Australia	x	
GridFlow	University of Warwick, Coventry, UK	x	
JOpera	Swiss Federal Institute of Technology (ETHZ)	x	
Pegasus	University of Southern California, USA	x	
TigMNS	Tongji University, Shanghai, China	x	
WorkflowML	Vanderbilt University, Nashville, USA	x	
Development frameworks			
ASSIST	University of Pisa, Italy	x	
AutoMate	Rutgers State University of New Jersey, USA	x	
GCM	University of Pisa, Italy	x	
Jade	Institut National Polytechnique de Toulouse, France	x	

Table 5.1: Project information for autonomic large-scale systems

exists that investigates the application of self-management to cloud computing within the context of the project.

5.2 Project classification within the taxonomy

This section applies the taxonomy of autonomic large-scale computing that has been proposed in Section 4.2 to the autonomic large-scale computing projects listed in Table 5.1. The taxonomy allows to classify the projects by the functional area of large-scale computing that is addressed using methods of autonomic computing, and by the autonomic computing approach employed for this purpose.

A classification of the large-scale computing projects of this survey by their self-managed functional area is given in Section 5.2.1. A classification by the autonomic computing approach is given in Section 5.2.2.

5.2.1 Area of autonomic large-scale computing

In this section, the projects included in the survey will be classified by their area of autonomic large-scale computing as defined in Section 4.2.2. Section 5.2.1.1 presents the catalog of autonomic large-scale computing systems classified by large-scale computing areas. Then, after providing rationales for the assignment of projects within the taxonomy in Section 5.2.1.2, a discussion of the classification is presented in Section 5.2.1.3.

5.2.1.1 Classification

The tables in this section apply the area of autonomic large-scale computing taxonomy to the projects included in the survey. Each table represents an area of autonomic large-scale computing that has been defined in the taxonomy. The categories within each area and their respective subcategories are displayed in the respective table.

Table 5.2 classifies autonomic large-scale computing projects in the application management area. Projects with autonomic data management capability are listed in Table 5.3. Projects that cover the development of autonomic large-scale computing systems are listed in Table 5.4. Table 5.5 lists large-scale computing projects with autonomic capabilities in the quality of service area. Table 5.6 finally categorizes large-scale computing projects with autonomic capabilities in the resource management area.

5.2.1.2 Rationales for classification

This section provides rationales for the establishment of the categories within the taxonomy of autonomic large-scale computing areas, and for the assignment of projects to categories.

	Application management			
	Service discovery		Application	Workflow
	Agents	Ontology	reconfiguration	specification
Guan et al.		x		
Guo et al.	x			
ICENI		x		
InteGrade			x	
TigMNS				x

Table 5.2: Application management area

	Data management	
	Real-time data	Persistent data
GATES	x	
Zhao et al.		x

Table 5.3: Data management area

	Development	
	Framework	Wrapper
AutoMate	x	
GCM	x	
Jade		x

Table 5.4: Development area

	Quality of service					
		Negotiation		Resource provisioning		
	Service composition			Proactive		Reactive
		Agents	Control loop	Simu-lation	Advance reservation	
GridSim				x		
AbdelSalam et al.					x	
Iqbal et al.						x
Libra+$Auto					x	
Koller et al.			x			
PAWS	x					
VieSLAF			x			
Weng et al.		x				

Table 5.5: Quality of service area

	Resource management											
	System management		Scheduling									
				Resource allocation			Schedule modification					
							Method			Scope		
	Error discovery	Performance optimization	Schedule delegation	Reputation	Reinforcement learning	Agents	Reschedule	State saving	Application support	Job	Workflow	Partition
AutoMAGI			x									
CoordAgent						x		x		x		
Dasgupta et al.							x			x		
Dynaco									x	x		
Gounaris et al.		x										
Jarvis et al.							x			x		
OptimalGrid								x				x
OrganicGrid						x						
Perez et al.					x							
So-Grid						x						
Dai et al.	x											
Paton et al.							x				x	
Gridbus WFE				x								
GridFlow								x			x	
JOpera		x										
Pegasus							x				x	
WorkflowML								x			x	
ASSIST							x					x

Table 5.6: Resource management area

Application management area The application management area intends to cover all projects that use autonomic capabilities to manage the functionality of the application. While other areas of the taxonomy described further below describe how autonomic computing is used to enable execution of previously defined applications, the application management area describes how autonomic computing is used to specify or compose an application before execution.

Service discovery is a well-defined research field within service-oriented computing that investigates methods for locating and selecting services based on functional or non-functional criteria. Since in service-oriented computing applications are composed of services, the location of services determines the composition of the application, or in case of selection by functional criteria even the functionality of the application. Thus, in this taxonomy, service discovery is a category of application management.

Application reconfiguration allows to change the configuration of an already executing application. Reconfiguration actions range from changing a configuration parameter to changing a complete module. In contrast to the service discovery category described above which is based on the service-oriented computing paradigm, the application reconfiguration category implies a component-oriented model.

Workflow specification allows to specify or compose a scientific workflow using autonomic methods. This category implies that the workflow constitutes an application which is composed of the workflow's jobs.

Guan et al. [89] employ an ontology for service discovery based on semantic matching, Guo et al. [90] have agents representing services publish and discover their capabilities. Hau et al. [92] implement ontological annotation in the ICENI grid middleware [78] for semantic matching of services. AutoGrid [161] which is an autonomic extension of the InteGrade [81] grid middleware, supports application reconfiguration by object replacement within a running application. Workflow specification in TigMNS [195] is employed by mapping high-level missions to a grid workflow.

Data management area The data management area intends to cover all projects that use autonomic capabilities in order to manage data in a large-scale computing environment. Data management is categorized into real-time data and persistent data which each are defined by the properties of a single grid computing project.

GATES [51] employs autonomic real-time data management by modifying the processing accuracy of a stream such that it can be processed by a grid system in real time.

Zhao et al. [197] employ autonomic management of persistent data within the grid virtualized file system (GVFS) by maintaining parameters like the client size disk cache in order to optimize data access from grid applications.

Development area The development area intends to cover all projects that support the development of autonomic large-scale computing applications.

While AutoMate [141] and GCM [8] both present frameworks supporting the development of autonomic grid applications, Jade [177] uses an approach for wrapping

legacy applications into self-managing components.

Quality of service area The quality of service area includes all large-scale computing projects that investigate self-management of QoS properties. QoS refers to non-functional system properties like response time or reliability which may be formally defined using a service-level agreement (SLA).

The taxonomy of QoS in autonomic large-scale computing distinguishes on its top level between service composition, negotiation, and resource provisioning. While service composition for QoS composes services such that QoS properties are optimized, QoS negotiation has a service provider and a service consumer negotiate for QoS requirements, resulting in a service-level agreement. Resource provisioning finally has a service provider provision resources such that given QoS requirements are met.

Service composition is investigated within the PAWS [16] project by composing services such that QoS properties like runtime are minimized.

Within QoS negotiation, the taxonomy distinguishes between agent-based and control loop-based negotiation. While Weng et al. [185] employ negotiating agents, Koller et al. [112] and work within the VieSLAF project [29] employs loop-based controllers for negotiation.

Resource provisioning may either be proactive, which means that future provisioning of resources is planned such that QoS requirements are met, or reactive, where a system actively monitors resource usage for adaptation in reaction to a detected QoS degradation. Proactive resource management may be employed by simulation or by requiring advance reservation of resources. While Assunção et al. [22] present a simulation framework for autonomic service policies within the GridSim [39] toolkit, AbdelSalam et al. [1] and Libra+$Auto [189] both require advance reservation of resources, the former for optimizing resource usage, the latter for optimizing resource pricing. Reactive provisioning is employed by [103], where additional resources are provisioned upon detection of QoS degradation.

Resource management area The resource management area includes work that investigates autonomic resource management in large-scale computing systems. On the top level, the taxonomy distinguishes between two perspectives on resource management. The first one, system management, refers to self-management that is employed in order to keep system resources in a desired state. It may also be seen as a system-internal perspective since the system's own resources are maintained, or a resource-centric perspective since resources are adapted according to some higher-level specification. The second perspective on resource management, scheduling, refers to the selection of resources for scheduling using autonomic methods. It may also be seen as a system-external perspective since the resources of external systems are maintained, or a schedule-centric perspective since schedules are adapted based on resource properties.

System management is categorized into error discovery and performance optimization. Dai et al. [54] use autonomic methods for detecting and classifying errors, fol-

lowed by application of remedies. JOpera [94] supports self-management of thread pool sizes in order to minimize response time. Gounaris et al. [84] finally optimize the response time of a grid-based database service.

Scheduling is categorized into schedule delegation, resource allocation and schedule modification. The schedule delegation category is established by a single project, AutoMAGI [157] where mobile clients with limited resources and unreliable connectivity delegate the process of scheduling to a gateway. Resource allocation refers to autonomic methods for allocating resources before running an application or independent thereof (e.g. to maintain a list of resources), schedule modification allows changing the schedule of an already running application in reaction to changes in resource properties. Resource allocation is further categorized by the method of determining suitable resources, namely reputation-based schemes, reinforcement learning, and agents. Schedule modification is further categorized by the method of schedule modification and by its scope, that is, whether the schedule of a complete workflow, a single job that may be part of a workflow, or a part of a job called partition is modified. Methods of schedule modification are rescheduling of failed items, state saving such that a scheduled item may be resumed from some state of execution other than the beginning, and application support, where a system requires applications to support dynamic modification of the current schedule. The properties described above are implemented in the surveyed systems as described in Section 4.2.2.5 and listed in Table 5.6.

5.2.1.3 Discussion

The tables referenced above show five projects in the application management area, two data management projects and three development projects. Eight projects are categorized into the quality of service area, and 18 projects investigate autonomic resource management in large-scale computing systems. The number of projects within each area is reflected in the number of categories and subcategories within those areas. Half of the projects focus on autonomic resource management, followed by quality of service management, application management, development and data management. Within each area, one or more categories exist that are defined by a single project. The data management area itself is defined by two projects, each of them establishing a category within that area.

The classification shows that the problems in large-scale computing that are most frequently addressed by applying the autonomic computing paradigm are resource management and quality of service management. All of the cloud computing projects included in the survey belong to one of these two areas, while the remaining three areas are exclusively populated by grid projects.

Within the resource management taxonomy, 14 of 18 projects are assigned to the scheduling category, 13 thereof to one of the subcategories of resource allocation or schedule modification which are distinguished from each other by the point of time where self-management is applied rather than the research problem that is being addressed. This leads to the conclusion that the problem most frequently investigated in large-scale computing projects by applying autonomic computing methods is the

assignment of system resources to scheduled or running applications.

5.2.2 Autonomic computing approach

In this section, the projects included in the survey will be classified by the approach of autonomic computing that has been chosen, as defined in Section 4.2.3. Section 5.2.2.1 presents the catalog of autonomic large-scale computing systems classified by autonomic computing approach. Then, after providing rationales for the assignment of projects within the taxonomy in Section 5.2.2.2, a discussion of the classification is presented in Section 5.2.2.3.

5.2.2.1 Classification

This section applies the autonomic computing approach taxonomy to the projects included in the survey. A single table is used to present the results of the classification with its columns mapping the taxonomy introduced in Section 4.2.3.

Table 5.7 classifies the projects included in the survey by their autonomic computing approach.

5.2.2.2 Rationales for classification

This section provides rationales for the establishment of the categories within the taxonomy of autonomic computing approaches, and for the assignment of projects to categories.

The top level of the autonomic computing approaches provides a classification by the architecture chosen to implement autonomic behavior, and by the capabilities provided by self-management. Usually, autonomic capabilities of a project are classified by the properties of self-configuration, self-optimization, self-healing and self-protection introduced in Section 3.3.1. Herrmann et al. [95] state that the differences between these properties are somewhat fuzzy, though: is a system with poor response time faulty and thus subject to self-healing or simply performing sub-optimally and thus subject to self-optimization? The projects included in the survey confirm this point by actually implying slightly different meanings of the properties, especially regarding the difference between self-configuration and self-optimization, since optimization often is accomplished by applying configuration actions.

Based on the observations stated above, this taxonomy prefers to provide on its second level a classification by capabilities actually observed in the surveyed projects.

Autonomic computing architecture The autonomic computing architecture category describes the architectural approach that the projects included in the survey follow to implement autonomic behavior. Most projects use an architecture where self-management is encapsulated into components distinguished from components providing system functionality. While several projects follow the autonomic computing architecture described in Section 3.3.2 which is composed of elements managed by

	Autonomic computing approach											
	Architecture						Capabilities					
		Agents							Optimizing			
	Control loop	Mobile	Communicat.	Emergent	Proxy	Built-in	Semantic	Learning	Control theory	Queueing th.	Combinatorial	Economic mod.
Grid middleware												
AutoMAGI	x											
CoordAgent		x										
Dasgupta et al.					x							
Dynaco	x											
GATES									x			
Gounaris et al.										x		
GridSim						x						
Guan et al.							x					
Guo et al.	x		x									
ICENI							x					
InteGrade						x						
Jarvis et al.			x			x						
OptimalGrid						x						
OrganicGrid		x	x	x								
Perez et al.								x				
So-Grid				x								
Zhao et al.	x											
Cloud computing systems												
AbdelSalam et al.											x	
Dai et al.								x				
Iqbal et al.					x							
Libra+$Auto												x
Paton et al.												x
Quality of service frameworks												
Koller et al.					x							
PAWS											x	
VieSLAF	x											
Weng et al.			x									
Workflow management systems												
Gridbus WFE	x											
GridFlow		x	x									
JOpera	x											
TigMNS	x											
Pegasus	x											
WorkflowML	x											
Development frameworks												
ASSIST	x											
AutoMate			x									
GCM	x											
Jade	x											

Table 5.7: Approach of autonomic large-scale computing

autonomic managers that consist of a control loop decomposed into monitor, analyze, plan, and execute steps, others implement autonomic behavior using software agents. Agents may be mobile, that is they move between different nodes of a large-scale computing system. They may be communicative by sending messages to and receiving messages from other agents. Multi-agent systems finally may be emergent by providing complex behavior that emerges from simple behavior patterns of multiple agents.

A third architectural approach observed in the survey is to encapsulate self-management into a proxy that intercepts and modifies communication within a large-scale distributed computing system. The fourth category, built-in autonomic behavior, refers to system architectures where autonomic capabilities are tightly integrated with system functionality.

Several projects explore specific capabilities of autonomic computing but do not present a fully operating autonomic large-scale computing system and thus cannot be associated with a system architecture. Often the explored capabilities may be implemented in a system following an arbitrary architectural style.

Rationales for the classification of individual projects within the taxonomic category of autonomic computing architecture are given in Section 4.2.3.1.

Autonomic computing capabilities The taxonomy of capabilities in autonomic large-scale computing systems is targeted at projects exploring a specific capability intended to be part of an autonomic large-scale computing system. As stated in the introduction to this section, the capability model followed here is to be distinguished from the self-management capabilities of an autonomic system introduced in Section 3.3.2.

The service discovery mechanism presented by Guan et al. [89] and autonomic service adaptation in the ICENI grid infrastructure [92] introduce autonomic behavior to large-scale computing systems using semantic web technologies. Perez et al. [148] and Dai et al. [54] provide self-management by machine learning.

Several projects present optimization mechanisms of autonomic large-scale computing systems, which are further categorized by the optimization approach. Gounaris et al. [84] employ an optimization approach based on control theory for access to a database service. The technique used in the GATES project [51] for optimizing real-time data transfer originates in queueing theory. AbdelSalam et al. [2] and work within the PAWS project [14, 20] apply combinatorial optimization techniques for resource provisioning and quality of service, respectively. Optimization within the Libra+$Auto resource pricing mechanism [189] and the adaptive workload execution methodology introduced by Paton et al. [143] finally is based on economic models.

5.2.2.3 Discussion

This section discusses the results of the surveyed projects' classification by the approach of autonomic large-scale computing taxonomy.

Of the 36 projects included in the survey, 26 present an autonomic large-scale computing system that can be classified by its architecture, while 10 projects explore

specific capabilities. 13 of the 26 autonomic large-scale computing system projects employ an architecture based on a control loop, as suggested by the autonomic computing model introduced in Section 3.3.2. Eight projects are based on software agents that introduce self-management but are not structured like autonomic elements. Three projects employ a self-management proxy, and three projects have autonomic behavior integrated into system functionality. The remaining ten projects explore specific capabilities, six thereof by optimization, two by machine learning, and two using semantic technologies.

Of the 13 projects employing an architecture composed of autonomic elements, Guo et al. [90] model the autonomic elements as communicative software agents, and AutoMAGI [157] employs an architecture composed of autonomic elements operating as semantic web services. The remaining 11 projects employ a single central autonomic manger.

Of the ten projects that investigate specific autonomic capabilities, six projects target specific optimization problems. Two projects investigate semantic web technologies and two others machine learning in the context of autonomic large-scale computing.

The conclusion drawn from the classification is that while several projects employ a central loop-based autonomic controller, and several others use software agents, the combination of these technologies suggested by Brazier et al. [30], while being employed by Guo et al. [90] for grid system management purposes, remains to be addressed by future work in other areas of autonomic large-scale computing.

5.2.3 Autonomic approaches by large-scale computing areas

This section combines the taxonomy of large-scale computing areas introduced in Section 4.2.2 with the taxonomy of autonomic computing approaches introduced in Section 4.2.3 in order to determine if there exist approaches that are preferred within certain areas of large-scale computing.

Section 5.2.3.1 describes the classification criteria of the combined taxonomy. Section 5.2.3.2 investigates rationales for establishing the combined taxonomy. Section 5.2.3.3 finally discusses results of the combined classification.

5.2.3.1 Classification

Table 5.8 shows the combined taxonomy of large-scale computing area and autonomic computing approach. On the top two levels, projects are categorized into the four major architectural styles of autonomic computing, namely control loop, agents, proxy, and built-in support or into the three major autonomic capabilities, namely machine learning, optimization, and semantics. Each of those architectures or capabilities is subcategorized into the five major areas identified by the large-scale computing area taxonomy, namely application management, data management, development, quality of service management and resource management, provided that at least one project is included in the survey that applies the architecture or capability within that area.

	Autonomic computing approach and large-scale computing areas																			
	Architecture														Capabilities					
	Control loop					Agents				Proxy		Built-in			L	Opt.				S
	App. mgmt.	Data mgmt.	Development	QoS	Resource mgmt.	App. mgmt.	Development	QoS	Resource mgmt.	QoS	Resource mgmt.	App. mgmt.	QoS	Resource mgmt.	Resource mgmt.	Data mgmt.	QoS	Resource mgmt.	App. mgmt.
Grid middleware																				
AutoMAGI					x															
CoordAgent									x											
Dasgupta et al.											x									
Dynaco					x															
GATES																x				
Gounaris et al.																		x		
GridSim													x							
Guan et al.																			x	
Guo et al.	x					x														
ICENI																			x	
InteGrade												x								
Jarvis et al.						x						x								
OptimalGrid														x						
OrganicGrid									x											
Perez et al.															x					
So-Grid									x											
Zhao et al.		x																		
Cloud computing systems																				
AbdelSalam et al.																	x			
Dai et al.															x					
Iqbal et al.										x										
Libra+$Auto																	x			
Paton et al.																		x		
Quality of service frameworks																				
Koller et al.										x										
PAWS																	x			
VieSLAF				x																
Weng et al.								x												
Workflow management systems																				
Gridbus WFE					x															
GridFlow									x											
JOpera					x															
TigMNS	x																			
Pegasus					x															
WorkflowML					x															
Development frameworks																				
ASSIST			x																	
AutoMate							x													
GCM			x																	
Jade			x																	
No. of projects	2	1	3	1	6	2	1	1	4	2	1	2	1	1	2	1	3	2	2	

Table 5.8: Areas of large-scale computing within approaches of autonomic computing
L = machine learning, S= semantic technologies

5.2.3.2 Rationales for classification

Rationales for classification of the individual projects within the taxonomies of large-scale computing areas and autonomic approaches have been given in Section 5.2.1 and 5.2.2, respectively. The purpose of the classification within the combined taxonomy is to investigate if there are correlations between classes of large-scale computing problems and autonomic computing approaches that address them.

5.2.3.3 Discussion

This section discusses results of the classification of the projects included in the survey by the combined taxonomy of autonomic computing approaches in large-scale computing areas. While the surveyed projects represent a too small sample with regard to the number of classes to apply statistical methods, some combinations of large-scale computing functionality and autonomic technology are noteworthy.

Table 5.8 shows a total of six projects that investigate resource management using an approach based on an autonomic control loop. Four projects use an agent-based approach for resource management. Three projects support the development of autonomic large-scale computing applications that use an approach based on control loops, and another three projects investigate optimization in the quality of service area. Two projects each explore control loops in application management, agents in application management, quality of service management using a proxy, application management using an architecture with integrated autonomic support, machine learning in resource management, optimization in resource management and semantic technologies in application management. The remaining combinations are each explored by a single project.

Of the six loop-based resource management projects, four are in the area of workflow management systems. Three of the four agent-based projects in the resource management area are desktop grid systems. While managing a workflow using a central controller and representing the nodes of a desktop grid as independent agents seems to be an obvious decision at the first glance, both resource management problems are in fact quite similar, though, since in both cases the problem is the assignment of a sequence of jobs or job partitions with interdependencies to resources that dynamically change their availability or performance. The difference between the two operational models of workflow management and desktop grids seems to be that in workflow management systems like Pegasus, the original concept was a static assignment of jobs to resources that were supposed to be reliable, and dynamic resource management has been added at a later time. In the case of desktop grids, it has been a requirement from the beginning, that desktop computers may reclaim their resources or simply disappear without causing job failure. In other words, in workflow management systems, disappearance of resources traditionally has been considered a fault, while in desktop grid systems it has always been regarded as a normal mode of operation. Thus, grid workflow management systems that originally have lacked self-adaptation capabilities have been a natural target for applying the autonomic computing archi-

tecture described in Section 3.3.2 which has been designed for adoption of autonomic capabilities by legacy systems.

While resource management and QoS management are the functional areas with the most projects and thus have been implemented using all of the four architectural styles defined within the taxonomy, some autonomic capabilities seem to target only a few functional areas. Both projects that employ machine learning target the resource management area. While work from Perez et al. applies machine learning for future decisions on the selection of resources provided by other systems, machine learning as employed by Dai et al. has the goal to improve fault diagnosis of a system's own resources.

Another special capability, the application of semantic technologies for self-management is targeted at the functional area of application management, specifically at service discovery as shown in Table 5.2. Both projects in those categories, work from Guan et al. and ICENI employ semantic service annotation based on ontologies in order to allow service discovery and service selection based on the service's semantics.

Finally, optimization with regard to quality of service management has been studied by three projects. Table 5.7 shows that two of these projects, work of AbdelSalam et al. and work within the PAWS project employ methods from the field of combinatorial optimization. The third of these projects, the Libra+$Auto pricing mechanism applies optimization using an economic model based on utility functions.

5.3 Results of the classification

This chapter presented a catalog of autonomic large-scale computing that has been created by classifying the autonomic large-scale computing projects of the survey conducted in Section 4.1 using the taxonomy proposed in Chapter 4. Each project has been assigned to a subcategory and category within an area of large-scale computing, as defined in Section 4.2.2 and to an autonomic computing approach as defined in Section 4.2.3. Finally, a combined taxonomy showing which approach of autonomic computing has been used to address which problem in large-scale computing has been presented.

The catalog showed that the topics that are most frequently investigated by within the field of autonomic large-scale computing are resource management and quality of service management. On the other hand, the area of data management in grid systems has been addressed only by two projects. The relative majority of the surveyed projects employs an autonomic computing architecture based on autonomic elements composed of managed elements and autonomic managers, the latter employing a control loop decomposed into monitor, analyze, plan and execute functions. Other architectural styles for autonomic computing that have been employed by the surveyed projects are agent-based systems, systems employing a proxy for autonomic management, and systems with self-management tightly integrated into the application's functionality.

The main conclusion drawn from the catalog of autonomic large-scale computing systems is that, while the particular strength of the concept of autonomic comput-

ing presented in Chapter 3 is adding autonomic capabilities to non-autonomic legacy systems, for large-scale computing projects that incorporate self-management capabilities from the beginning, alternate architectural styles like agent-based or proxy-based systems are equally suitable.

6 Conclusions and future work

This thesis explored the field of autonomic large-scale distributed computing and presented autonomic computing as an approach to provide self-management in current large-scale distributed computing systems. It has conducted the first attempt on a survey and taxonomy of autonomic large-scale distributed computing.

While several interrelated large-scale computing paradigms exist, like grid computing which provides sharing of system resources usually located at high-performance computing centers within scientific communities, and the more recent paradigm of cloud computing, where data centers provide system resources and software as a service over the Internet as an effort to realize the vision of utility computing [41], broad acceptance of these paradigms has been impeded by difficulties to provide reliable service expressed as quality of service and formalized within service level agreements, that originate in the complexity and dynamism of those systems.

Autonomic computing, where policy-based controllers provide self-management to computing systems with the goal of relieving human system administrators from manually adapting those systems to dynamically changing requirements and correcting faults of components has been identified as a possible solution to provide the required reliability, resulting in research projects like Foundations of Self-Governing ICT Infrastructures (FoSII) that explore self-management in large-scale distributed computing systems.

This thesis has presented a comprehensive survey of the existing body of work in autonomic large-scale distributed computing. It has identified 27 autonomic grid projects that provide middleware services, workflow management or enable application development, five autonomic cloud computing projects and four projects exploring self-management of quality of service in large-scale distributed computing systems. Based on those projects, a taxonomy of autonomic large-scale computing has been derived, which is organized by the two dimensions of autonomic large-scale computing area and approach of autonomic computing. While the taxonomy of autonomic large-scale computing areas intends to describe and classify the functional dimension of autonomic computing in large-scale distributed systems by identifying the areas of application management, data management, development, quality of service management, and resource management where autonomic computing has been applied in the large-scale computing projects included in the survey, the taxonomy of autonomic computing approaches describes autonomic computing technology like architectural styles and capabilities that have been applied to those functional areas. Autonomic computing architectures identified within the surveyed projects include control loop based architectures, agent-based architectures and proxy-based architectures. Autonomic capabilities include optimization, machine learning and semantics.

The survey showed that most work in autonomic large-scale distributed computing targets the field of grid computing, while only a few research projects specifically explore autonomic capabilities in cloud computing systems or investigate specific problems like quality of service management applicable to both fields. While this phenomenon largely originates in the higher degree of maturity of grid computing compared to the relatively new field of cloud computing, another reason for the dominance of grid projects within the survey is that inclusion of cloud systems had to be restricted to academic projects since commercial cloud operators like Amazon or Google up to this day do not have issued publications that would allow to assess self-management capabilities of their cloud infrastructures.

Application of the taxonomy to the surveyed projects resulted in a catalog of autonomic large scale distributed computing systems that identified functional areas and autonomic approaches explored by those projects. The research problem addressed most frequently within autonomic large-scale computing is self-adaptation in resource management, which includes the assignment of tasks to resources in scheduling. While autonomic resource management largely prevails in grid projects which dominate in the survey, autonomic cloud projects tend to focus on quality of service management, which includes the provisioning of resources. This difference in research focus apparently maps the different operation modes of grid and cloud systems. While within grid systems, resources are cooperatively shared between organizations or in the case of desktop grids even between individual users and thus tend to suddenly appear and disappear, leaving the burden to find and select suitable and reliable resources on the system issuing the job, with cloud computing resources are under the control of a single operator who is entitled to capacity planning with regard to the number of jobs processed and the expected quality of service.

With regard to the autonomic computing approaches taxonomy, the catalog showed that the autonomic computing architecture proposed by IBM which is composed of autonomic elements that employ a control loop consisting of the steps of monitoring, planning, analyzing and executing guided by knowledge, has been largely accepted within the field of large-scale computing and implemented by several projects. A number of grid projects followed the approach of adopting that model to existing projects. Architectures that are based on cooperating autonomic software agents have also been explored, especially in projects that have required autonomic capabilities from the beginning like desktop grid systems where traditional central system management modes are not feasible. System architectures based on self-management proxies allow a third party to employ self-management with existing systems, which has been identified as a suitable approach for quality of service management [112].

The existing body of work in the field of large-scale distributed computing systems shows that all of the autonomic computing architectures that have been described in this thesis are effective in providing self-management capabilities within different functional areas of large-scale distributed computing systems. Project classification within this thesis has already suggested that IBM's autonomic computing architecture is especially suited for adding autonomic capabilities to existing non-autonomic large-scale distributed computing systems. However, future work needs to further investigate

those architectures in order to compare advantages and disadvantages of the autonomic computing approaches presented in this thesis, with the goal of providing information to implementers of autonomic large-scale computing systems supporting their decision on an autonomic computing architecture for their specific requirements.

While capabilities of an autonomic system usually are described using the properties of self-configuration, self-optimization, self-healing and self-protection, the taxonomy presented in this thesis refrained from using those properties for classification. While providing a useful scheme for describing the general capabilities of an autonomic computing system, those properties lack standardization, leading to different usages in different projects, thus impeding direct comparison of projects by those criteria. Future work, possibly by a standardization body, may formally define the meanings of those properties in order to allow direct comparison of autonomic computing systems. On the other hand, the self-management properties may be regarded to be intentionally ambiguous, since for example the classification of a self-management action as self-optimization or self-healing depends on the classification of the underlying system behavior that is subject to self-management as suboptimal or faulty [95].

The success of cloud computing as a means to deliver the vision of utility computing heavily depends on its acceptance by the public. A key factor for that acceptance is reliability. While autonomic computing has been identified as a means for providing this reliability in a scalable fashion by enabling self-management of cloud computing resources, the field of autonomic large-scale distributed computing systems up to this date lacked a comprehensive survey and taxonomy of the existing body of work in that area. By presenting a survey and taxonomy of autonomic large-scale distributed computing for the first time, this thesis provides a tool that helps for further advancements in autonomic large-scale distributed computing research within projects like Foundations of Self-Governing ICT Infrastructures (FoSII) [73].

Bibliography

[1] Hady AbdelSalam, Kurt Maly, Ravi Mukkamala, Mohammad Zubair, and David Kaminsky. Towards energy efficient change management in a cloud computing environment. In *Scalability of Networks and Services*, volume 5637/2009 of *Lecture Notes in Computer Science*, pages 161–166. Springer Berlin / Heidelberg, 2009.

[2] Hady S. Abdelsalam, Kurt Maly, Ravi Mukkamala, Mohammad Zubair, and David Kaminsky. Analysis of energy efficiency in clouds. In *Computation World: Future Computing, Service Computation, Cognitive, Adaptive, Content, Patterns*, pages 416 –421, Nov. 2009.

[3] Sherif Abdelwahed, Nagarajan Kandasamy, and Sandeep Neema. A control-based framework for self-managing distributed computing systems. In *WOSS '04: Proceedings of the 1st ACM SIGSOFT workshop on Self-managed systems*, pages 3–7, New York, NY, USA, 2004. ACM.

[4] Manish Agarwal, Viraj Bhat, Hua Liu, Vincent Matossian, Venkatesh Putty, Cristina Schmidt, Guangsheng Zhang, Zhen Li, Manish Parashar, Bithika Khargharia, and Salim Hariri. AutoMate: enabling autonomic applications on the grid. In *Autonomic Computing Workshop, 2003*, pages 48–57, June 2003.

[5] Manish Agarwal and Manish Parashar. Enabling autonomic compositions in grid environments. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 34–41, Nov. 2003.

[6] Mehmet Aksit and Zièd Choukair. Dynamic, adaptive and reconfigurable systems overview and prospective vision. In *Distributed Computing Systems Workshops, 2003. Proceedings. 23rd International Conference on*, pages 84–89, May 2003.

[7] Youssif Al-Nashif, Aarthi Arun Kumar, Salim Hariri, Guangzhi Qu, Yi Luo, and Ferenc Szidarovsky. Multi-level intrusion detection system (ML-IDS). In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 131–140, June 2008.

[8] Marco Aldinucci, Sonia Campa, Marco Danelutto, Marco Vanneschi, Peter Kilpatrick, Patrizio Dazzi, Domenico Laforenza, and Nicola Tonellotto. Behavioural skeletons in GCM: Autonomic management of grid components. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 54–63, Feb. 2008.

[9] Marco Aldinucci, Marco Danelutto, and Marco Vanneschi. Autonomic QoS in ASSIST grid-aware components. In *Parallel, Distributed, and Network-Based Processing, 2006. PDP 2006. 14th Euromicro International Conference on*, pages 10 pp.–, Feb. 2006.

[10] Amazon.com. Amazon web services. `http://aws.amazon.com/`, July 2009.

[11] American Society for Indexing. About taxonomies & controlled vocabularies. `http://www.taxonomies-sig.org/about.htm`, Sep 2009.

[12] David P. Anderson, Jeff Cobb, Eric Korpela, Matt Lebofsky, and Dan Werthimer. SETI@home: an experiment in public-resource computing. *Commun. ACM*, 45(11):56–61, 2002.

[13] David P. Anderson and Gilles Fedak. The computational and storage potential of volunteer computing. In *Cluster Computing and the Grid, 2006. CCGRID 06. Sixth IEEE International Symposium on*, volume 1, pages 73 – 80, May 2006.

[14] Jonatha Anselmi, Danilo Ardagna, and Paolo Cremonesi. A QoS-based selection approach of autonomic grid services. In *SOCP '07: Proceedings of the 2007 workshop on Service-oriented computing performance: aspects, issues, and approaches*, pages 1–8, New York, NY, USA, 2007. ACM.

[15] Arvind Arasu, Shivnath Babu, and Jennifer Widom. The CQL continuous query language: semantic foundations and query execution. *VLDB Journal: Very Large Data Bases*, 15(2):121–142, Jun. 2006.

[16] Danilo Ardagna, Marco Comuzzi, Enrico Mussi, Barbara Pernici, and Pierluigi Plebani. PAWS: A framework for executing adaptive web-service processes. *Software, IEEE*, 24(6):39–46, Nov.-Dec. 2007.

[17] Danilo Ardagna, Gabriele Giunta, Nunzio Ingraffia, Raffaela Mirandola, and Barbara Pernici. *QoS-Driven Web Services Selection in Autonomic Grid Environments*, volume 4276 of *Lecture Notes in Computer Science*, pages 1273–1289. Springer, 2006.

[18] Danilo Ardagna, Silvia Lucchini, Raffaela Mirandola, and Barbara Pernici. *Web Services Composition in Autonomic Grid Environments*, volume 4103 of *Lecture Notes in Computer Science*, pages 375–386. Springer, 2006.

[19] Danilo Ardagna and Barbara Pernici. *Global and Local QoS Guarantee in Web Service Selection*, volume 3812 of *Lecture Notes in Computer Science*, pages 32–46. Springer, Sep 2005.

[20] Danilo Ardagna and Barbara Pernici. Adaptive service composition in flexible processes. *Software Engineering, IEEE Transactions on*, 33(6):369 –384, june 2007.

[21] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the clouds: A Berkeley view of cloud computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[22] Marcos Dias de Assunção, Werner Streitberger, Torsten Eymann, and Rajkumar Buyya. Enabling the simulation of service-oriented computing and provisioning policies for autonomic utility grids. In *Grid Economics and Business Models*, volume 4685/2007 of *Lecture Notes in Computer Science*, pages 136–149. Springer Berlin / Heidelberg, 2007.

[23] Algirdas Avižienis, Jean-Claude Laprie, Brian Randell, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. *Dependable and Secure Computing, IEEE Transactions on*, 1(1):11–33, Jan.-March 2004.

[24] Gordon Bell and Jim Gray. What's next in high-performance computing? *Commun. ACM*, 45(2):91–95, 2002.

[25] Dimitri P. Bertsekas. *Dynamic programming and optimal control.* Athena Scientific, 1995.

[26] Viraj Bhat, Manish Parashar, Hua Liu, Mohit Khandekar, Nagarajan Kandasamy, and Sherif Abdelwahed. Enabling self-managing applications using model-based online control strategies. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 15–24, June 2006.

[27] Ivona Brandic. Towards self-manageable cloud services. In *Computer Software and Applications Conference, 2009. COMPSAC '09. 33rd Annual IEEE International*, volume 2, pages 128–133, July 2009.

[28] Ivona Brandic, Dejan Music, and Schahram Dustdar. Service mediation and negotiation bootstrapping as first achievements towards self-adaptable grid and cloud services. In *Grids meet Autonomic Computing Workshop 2009 - GMAC09. In conjunction with the 6th International Conference on Autonomic Computing and Communications*, Jun. 2009.

[29] Ivona Brandic, Dejan Music, Philipp Leitner, and Schahram Dustdar. VieSLAF framework: Increasing the versatility of grid QoS models by applying semi-automatic SLA-mappings. In *Grid Economics and Business Models*, volume 5745/2009 of *Lecture Notes in Computer Science*, pages 60–73. Springer Berlin / Heidelberg, 2009.

[30] Frances M.T. Brazier, Jeffrey O. Kephart, H. Van Dyke Parunak, and Michael N. Huhns. Agents and service-oriented computing for autonomic computing: A research agenda. *Internet Computing, IEEE*, 13(3):82–87, May-June 2009.

[31] James Broberg, Srikumar Venugopal, and Rajkumar Buyya. Market-oriented grids and utility computing: The state-of-the-art and future directions. *Journal of Grid Computing*, 6(3):255–276, 2008.

[32] Eric Bruneton, Thierry Coupaye, Matthieu Leclercq, Vivien Quéma, and Jean-Bernard Stefani. The FRACTAL component model and its support in java. *Software: Practice and Experience*, 36(11-12):1257 – 1284, Sep.-Oct. 2006.

[33] Jose C. Brustoloni. Autonomous agents: Characterization and requirements. Technical Report CMU-CS-91-204, Carnegie Mellon University, 1991.

[34] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. Performance and practicability of dynamic adaptation for parallel computing: an experience feedback from Dynaco. Research Report/Publication interne 1782, Institut National de Recherche en Informatique et en Automatique (INRIA), `http://hal.inria.fr/inria-00001087/en/`, 2006.

[35] Jérémy Buisson, Françoise André, and Jean-Louis Pazat. Supporting adaptable applications in grid resource management systems. In *Grid Computing, 2007 8th IEEE/ACM International Conference on*, pages 58–65, Sep 2007.

[36] Rajkumar Buyya. *High Performance Cluster Computing: Architectures and Systems*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 1999.

[37] Rajkumar Buyya, David Abramson, Jonathan Giddy, and Heinz Stockinger. Economic models for resource management and scheduling in grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1507–1542, 2002.

[38] Rajkumar Buyya and Kris Bubendorfer. *Market-Oriented Grid and Utility Computing*. Wiley Series on Parallel and Distributed Computing. Wiley, November 2009.

[39] Rajkumar Buyya and M. Manzur Murshed. GridSim: A toolkit for the modeling and simulation of distributed resource management and scheduling for grid computing. *Concurrency and Computation: Practice and Experience*, 14(13-15):1175–1220, 2002.

[40] Rajkumar Buyya and Srikumar Venugopal. The Gridbus toolkit for service oriented grid and utility computing: an overview and status report. In *Grid Economics and Business Models, 2004. GECON 2004. 1st IEEE International Workshop on*, pages 19–66, April 2004.

[41] Rajkumar Buyya, Chee Shin Yeo, Srikumar Venugopal, James Broberg, and Ivona Brandic. Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility. *Future Generation Computer Systems*, 25(6):599 – 616, 2009.

[42] Junwei Cao, Stephen A. Jarvis, Subhash Saini, Darren J. Kerbyson, and Graham R. Nudd. ARMS: An agent-based resource management system for grid computing. *Scientific Programming*, Volume 10(2):135–148, 2002.

[43] Junwei Cao, Stephen A. Jarvis, Subhash Saini, and Graham R. Nudd. GridFlow: workflow management for grid computing. In *Cluster Computing and the Grid, 2003. Proceedings. CCGrid 2003. 3rd IEEE/ACM International Symposium on*, pages 198–205, May 2003.

[44] Jorge Cardoso, Amit Sheth, John Miller, Jonathan Arnold, and Krys Kochut. Quality of service for workflows and web service processes. *Web Semantics: Science, Services and Agents on the World Wide Web*, 1(3):281 – 308, 2004.

[45] Eddy Caron, Frédéric Desprez, Frédéric Lombard, Jean-Marc Nicod, Laurent Philippe, Martin Quinson, and Frédéric Suter. A scalable approach to network enabled servers. In *Euro-Par 2002 Parallel Processing*, volume 2400/2002 of *Lecture Notes in Computer Science*, pages 239–248. Springer Berlin / Heidelberg, 2002.

[46] Thomas L. Casavant and Jon G. Kuhl. A taxonomy of scheduling in general-purpose distributed computing systems. *Software Engineering, IEEE Transactions on*, 14(2):141–154, Feb. 1988.

[47] Anirban Chakrabarti, Anish Damodaran, and Subhashis Sengupta. Grid computing security: A taxonomy. *Security & Privacy, IEEE*, 6(1):44–51, Jan.-Feb. 2008.

[48] Arjav J. Chakravarti, Gerald Baumgartner, and Mario Lauria. The organic grid: self-organizing computation on a peer-to-peer networkrganic rid: self-organizing computation on a peer-to-peer network. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 96–103, May 2004.

[49] Jinjun Chen and Yun Yang. A taxonomy of grid workflow verification and validation. *Concurrency and Computation: Practice and Experience*, 20(4):347–360, 2008.

[50] Liang Chen and Gagan Agrawal. Self-adaptation in a middleware for processing data streams. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 292–293, May 2004.

[51] Liang Chen, Kolagatla Reddy, and Gagan Agrawal. Gates: a grid-based middleware for processing distributed data streams. In *High performance Distributed Computing, 2004. Proceedings. 13th IEEE International Symposium on*, pages 192–201, June 2004.

[52] Xingchen Chu, Krishna Nadiminti, Chao Jin, Srikumar Venugopal, and Rajkumar Buyya. Aneka: Next-generation enterprise grid platform for e-science and

e-business applications. In *e-Science and Grid Computing, IEEE International Conference on*, pages 151–159, Dec. 2007.

[53] Geoff Coulson, Gordon S. Blair, Michael Clarke, and Nikos Parlavantzas. The design of a configurable and reconfigurable middleware platform. *Distributed Computing*, 15(2):109–126, 2002.

[54] Yuanshun Dai, Yanping Xiang, and Gewei Zhang. Self-healing and hybrid diagnosis in cloud computing. In *Cloud Computing*, volume 5931/2009 of *Lecture Notes in Computer Science*. Springer Berlin / Heidelberg, 2009.

[55] Gargi Dasgupta, Onyeka Ezenwoye, Liana Fong, Selim Kalayci, Sayed Masoud Sadjadi, and Balaji Viswanathan. Runtime fault-handling for job-flow management in grid environments. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 201–202, June 2008.

[56] Jeffrey Dean and Sanjay Ghemawat. Mapreduce: Simplified data processing on large clusters. In *Proc. 6th Symp. Operating System Design and Implementation (OSDI)*, pages 137–150. Usenix Assoc., 2004.

[57] Ewa Deelman, Carl Kesselman, Gaurang Mehta, Leila Meshkat, Laura Pearlman, Kent Blackburn, Phil Ehrens, Albert Lazzarini, Roy Williams, and Scott Koranda. GriPhyN and LIGO, building a virtual data grid for gravitational wave scientists. In *High Performance Distributed Computing, 2002. HPDC-11 2002. Proceedings. 11th IEEE International Symposium on*, pages 225–234, 2002.

[58] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia C. Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[59] Glenn Deen, Toby Lehman, and James Kaufman. The Almaden OptimalGrid project. In *Proceedings of the Autonomic Computing Workshop. Fifth Annual International Workshop on Active Middleware Services. AMS 2003*, pages 14–21. IEEE Computer Society, June 2003.

[60] Jack Dongarra, Thomas Sterling, Horst Simon, and Erich Strohmaier. High-performance computing: clusters, constellations, mpps, and future directions. *Computing in Science Engineering*, 7(2):51 – 59, Mar 2005.

[61] Eclipse Foundation. AspectJ project site. `http://www.eclipse.org/aspectj/`.

[62] Dietmar W. Erwin and David F. Snelling. UNICORE: A grid computing environment. In *Euro-Par 2001 Parallel Processing*, volume 2150/2201 of *Lecture Notes in Computer Science*, pages 825–834. Springer Berlin / Heidelberg, 2001.

[63] ETH Zürich. JOpera. `http://www.iks.ethz.ch/jopera/`, Jul. 2009.

[64] European Grid Initiative. EGEE: Enabling grids for e-science. `http://www.eu-egee.org`, Aug. 2009.

[65] David Fernández-Baca. Allocating modules to processors in a distributed system. *Software Engineering, IEEE Transactions on*, 15(11):1427–1436, Nov. 1989.

[66] Michael J. Flynn. Some computer organizations and their effectiveness. *Computers, IEEE Transactions on*, C-21(9):948–960, Sept. 1972.

[67] Agostino Forestiero, Carlo Mastroianni, and Giandomenico Spezzano. So-Grid: A self-organizing grid featuring bio-inspired algorithms. *ACM Trans. Auton. Adapt. Syst.*, 3(2):1–37, 2008.

[68] Ian Foster. Globus toolkit version 4: Software for service-oriented systems. *Journal of Computer Science and Technology*, 21(4):513–520, July 2006.

[69] Ian Foster and Carl Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2nd edition, 2004.

[70] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. The physiology of the grid: An open grid services architecture for distributed systems integration. In *Open Grid Service Infrastructure WG, Global Grid Forum*, 2002.

[71] Ian Foster, Carl Kesselman, and Steven Tuecke. The anatomy of the grid: Enabling scalable virtual organizations. In *First IEEE/ACM International Symposium on Cluster Computing and the Grid, 2001. Proceedings.*, pages 6–7, 2001.

[72] Ian Foster, Yong Zhao, Ioan Raicu, and Shiyong Lu. Cloud computing and grid computing 360-degree compared. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov. 2008.

[73] Foundations of Self-Governing ICT Infrastructures (FOSII). Project site. `http://www.infosys.tuwien.ac.at/linksites/FOSII/`, 2009.

[74] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *Intelligent Agents III Agent Theories, Architectures, and Languages*, volume 1193/1997 of *Lecture Notes in Computer Science*, pages 21–35. Springer Berlin / Heidelberg, 1997.

[75] James Frey, Todd Tannenbaum, Miron Livny, Ian Foster, and Steven Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5(3):237–246, July 2002.

[76] Munehiro Fukuda, Yuichiro Tanaka, Naoya Suzuki, Lubomir F. Bic, and Shinya Kobayashi. A mobile-agent-based pc grid. In *Autonomic Computing Workshop, 2003*, pages 142–150, June 2003.

[77] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, and John Darlington. A component framework for HPC applications. In *Euro-Par 2001 Parallel Processing*, volume 2150/2001 of *Lecture Notes in Computer Science*, pages 540–548. Springer Berlin / Heidelberg, 2001.

[78] Nathalie Furmento, Anthony Mayer, Stephen McGough, Steven Newhouse, Tony Field, and John Darlington. ICENI: Optimisation of component applications within a grid environment. *Parallel Computing*, 28(12):1753 – 1772, 2002.

[79] Alan Ganek. Overview of autonomic computing: Origins, evolution, direction. In Parashar and Hariri [140], chapter 1, pages 3–18.

[80] Robert L. Glass and Iris Vessey. Contemporary application-domain taxonomies. *IEEE Software*, 12(4):63–76, 1995.

[81] Andrei Goldchleger, Fabio Kon, Alfredo Goldman, Marcelo Finger, and Germano Capistrano Bezerra. Integrade: object-oriented grid middleware leveraging the idle computing power of desktop machines. *Concurrency and Computation: Practice and Experience*, 16(5):449–459, 2004.

[82] Google, Inc. Google App Engine. `http://code.google.com/appengine`.

[83] Google, Inc. Google Apps. `http://www.google.com/apps/business/index.html`.

[84] Anastasios Gounaris, Christos Yfoulis, Rizos Sakellariou, and Marios D. Dikaiakos. A control theoretical approach to self-optimizing block transfer in web service grids. *ACM Trans. Auton. Adapt. Syst.*, 3(2):1–30, 2008.

[85] Gridbus Project. Gridbus Workflow Engine. `http://www.gridbus.org/workflow/`, Jul. 2009.

[86] Rean Griffith and Gail Kaiser. A runtime adaptation framework for native C and bytecode applications. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 93–104, June 2006.

[87] Rean Griffith, Giuseppe Valetto, and Gail Kaiser. Effecting runtime reconfiguration in managed execution environments. In Parashar and Hariri [140], chapter 18, pages 369–387.

[88] Tao Guan, Ed Zaluska, and David De Roure. An autonomic service discovery mechanism to support pervasive device accessing semantic grid. In *Autonomic Computing, 2007. ICAC '07. Fourth International Conference on*, pages 8–8, June 2007.

[89] Tao Guan, Ed Zaluska, and David De Roure. An autonomic service discovery mechanism to support pervasive devices accessing the semantic grid. *International Journal of Autonomic Computing*, 1(1):34–49, 2009.

[90] Hang Guo, Ji Gao, Peiyou Zhu, and Fan Zhang. A self-organized model of agent-enabling autonomic computing for grid environment. In *Intelligent Control and Automation, 2006. WCICA 2006. The Sixth World Congress on*, volume 1, pages 2623–2627, 2006.

[91] Salim Hariri, Bithika Khargharia, Houping Chen, Jingmei Yang, Yeliang Zhang, Manish Parashar, and Hua Liu. The autonomic computing paradigm. *Cluster Computing*, 9(1):5–17, Jan 2006.

[92] Jeffrey Hau, William Lee, and Steven Newhouse. Autonomic service adaptation in ICENI using ontological annotation. In *Grid Computing, 2003. Proceedings. Fourth International Workshop on*, pages 10–17, Nov. 2003.

[93] Thomas Heinis and Cesare Pautasso. Automatic configuration of an autonomic controller - an experimental study with zero-configuration policies. In *Proceedings of the 5th IEEE International Conference on Autonomic Computing (ICAC'08)*, Chicago, USA, 2008.

[94] Thomas Heinis, Cesare Pautasso, and Gustavo Alonso. Design and evaluation of an autonomic workflow engine. In *Autonomic Computing, 2005. ICAC 2005. Proceedings. Second International Conference on*, pages 27–38, June 2005.

[95] Klaus Herrmann, Gero Mühl, and Kurt Geihs. Self-management: The solution to complexity or just another problem? *IEEE Distributed Systems Online*, 6, 2005.

[96] Hewlett-Packard Company. Hp flexible computing services. `http://www.hp.com/services/flexiblecomputing`, July 2009.

[97] Markus C. Huebscher and Julie A. McCann. A survey of autonomic computing—degrees, models, and applications. *ACM Comput. Surv.*, 40(3):1–28, 2008.

[98] Soonwook Hwang and Carl Kesselman. A flexible framework for fault tolerance in the grid. *Journal of Grid Computing*, 1(3):251–272, Sept. 2003.

[99] IBM. Autonomic computing: IBM's perspective on the state of information technology. `http://www.research.ibm.com/autonomic/manifesto/autonomic_computing.pdf`, 2001.

[100] IBM. *An architectural blueprint for autonomic computing.* IBM White Paper, 4th edition, June 2006.

[101] IBM. Smart business services. `http://www.ibm.com/ibm/cloud/smart_business/`, July 2009.

[102] InformationWeek. Hp launches formal utility computing service. `http://www.informationweek.com/news/windows/microsoft_news/showArticle.jhtml?articleID=174402582`, November 2005.

[103] Waheed Iqbal, Matthew Dailey, and David Carrera. SLA-driven adaptive resource management for web applications on a heterogeneous compute cloud. In *Cloud Computing*, volume 5931/2009 of *Lecture Notes in Computer Science*, pages 243–253. Springer Berlin / Heidelberg, 2009.

[104] Bart Jacob, Sudipto Basu, Amit Tuli, and Patricia Witten. *A First Look at Solution Installation for Autonomic Computing.* IBM Redbooks, 1st edition, July 2004.

[105] Stephen A. Jarvis, Daniel P. Spooner, Hélène N. Lim Choi Keung, Justin R. D. Dyson, Lei Zhao, and Graham R. Nudd. Performance-based middleware services for grid computing. In *Autonomic Computing Workshop, 2003*, pages 151–159, June 2003.

[106] Brendan Jennings, Sven van der Meer, Sasitharan Balasubramaniam, Dmitri Botvich, Mícheál Ó. Foghlú, William Donnelly, and John Strassner. Towards autonomic management of communications networks. *Communications Magazine, IEEE*, 45(10):112 –121, October 2007.

[107] Jeffrey O. Kephart and David M. Chess. The vision of autonomic computing. *IEEE Computer*, 36(1):41–50, 2003.

[108] Jeffrey O. Kephart and William E. Walsh. An artificial intelligence perspective on autonomic computing policies. In *Policies for Distributed Systems and Networks, 2004. POLICY 2004. Proceedings. Fifth IEEE International Workshop on*, pages 3–12, June 2004.

[109] Bithika Khargharia and Salim Hariri. Autonomic power and performance management of Internet data. In Parashar and Hariri [140], chapter 21, pages 435–469.

[110] Bithika Khargharia, Salim Hariri, and Mazin S. Yousif. Autonomic power and performance management for computing systems. *Cluster Computing*, 11(2):167–181, 2008.

[111] Leonard Kleinrock. A vision for the Internet. *ST Journal of Research*, 2(1):4–5, 2005.

[112] Bastian Koller and Lutz Schubert. Towards autonomous SLA management using a proxy-like approach. *Multiagent and Grid Systems*, 3(3):313–325, 2007.

[113] Klaus Krauter, Rajkumar Buyya, and Muthucumaru Maheswaran. A taxonomy and survey of grid resource management systems for distributed computing. *Software: Practice and Experience*, 32(2):135–164, Feb 2002.

[114] Akos Ledeczi, Miklos Maroti, Arpad Bakay, Gabor Karsai, Jason Garrett, Charles Thomason, Greg Nordstrom, Jonathan Sprinkle, and Peter Volgyesi. The generic modeling environment. In *Proceedings of IEEE International Workshop on Intelligent Signal Processing (WISP 2001)*, May 2001.

[115] Kevin Lee, Norman W. Paton, Rizos Sakellariou, Ewa Deelman, Alvaro A. A. Fernandes, and Gaurang Mehta. Adaptive workflow processing and execution in Pegasus. In *Third International Conference on Grid and Pervasive Computing Symposia/Workshops*, 2008.

[116] Kevin Lee, Rizos Sakellariou, Norman W. Paton, and Alvaro A. A. Fernandes. Workflow adaptation as an autonomic computing problem. In *WORKS '07: Proceedings of the 2nd workshop on Workflows in support of large-scale science*, pages 29–34, New York, NY, USA, 2007. ACM.

[117] Tobin J. Lehman, Alex Cozzi, Yuhong Xiong, Jonathan Gottschalk, Venu Vasudevan, Sean Landis, Pace Davis, Bruce Khavar, and Paul Bowman. Hitting the distributed computing sweet spot with TSpaces. *Computer Networks*, 35(4):457 – 472, 2001.

[118] Tobin J. Lehman and James H. Kaufman. OptimalGrid: middleware for automatic deployment of distributed FEM problems on an Internet-based computing grid. In *Cluster Computing, 2003. Proceedings. 2003 IEEE International Conference on*, pages 164–171. IEEE Computer Society, Dec. 2003.

[119] Zhen Li and Manish Parashar. Rudder: a rule-based multi-agent infrastructure for supporting autonomic grid applications. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 278–279, May 2004.

[120] Hua Liu and Manish Parashar. Accord: a programming framework for autonomic applications. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):341–352, May 2006.

[121] Akshay Luther, Rajkumar Buyya, Rajiv Ranjan, and Srikumar Venugopal. Alchemi: A .NET-based grid computing framework and its integration into global grids. Technical Report GRIDS-TR-2003-8, Grid Computing and Distributed Systems Laboratory, University of Melbourne, Australia, December 2003.

[122] Anbazhagan Mani and Arun Nagarajan. Understanding quality of service for web services. `http://www.ibm.com/developerworks/library/ws-quality.html`, January 2002.

[123] David Martin, Massimo Paolucci, Sheila McIlraith, Mark Burstein, Drew McDermott, Deborah McGuinness, Bijan Parsia, Terry Payne, Marta Sabou, Monika Solanki, Naveen Srinivasan, and Katia Sycara. Bringing semantics to web services: The OWL-S approach. In *Semantic Web Services and Web Process Composition*, volume 3387/2005 of *Lecture Notes in Computer Science*, pages 26–42. Springer Berlin / Heidelberg, 2005.

[124] Lykomidis Mastroleon, Nicholas Bambos, Christos Kozyrakis, and Dimitris Economou. Autonomic power management schemes for internet servers and data centers. In *Global Telecommunications Conference, 2005. GLOBECOM '05. IEEE*, pages 943–947, Nov.-2 Dec. 2005.

[125] Satoshi Matsuoka, Sinji Shinjo, Mutsumi Aoyagi, Satoshi Sekiguchi, Hitohide Usami, and Kenichi Miura. Japanese computational grid research project: Naregi. *Proceedings of the IEEE*, 93(3):522–533, March 2005.

[126] A. Stephen McGough, William Lee, and John Darlington. ICENI II architecture. In Simon J. Cox and David W. Walker, editors, *Proceedings of the UK e-Science Meeting*. EPSRC, Sep. 2005.

[127] Philip K. McKinley, Sayed Masoud Sadjadi, Eric P. Kasten, and Betty H. C. Cheng. Composing adaptive software. *Computer*, 37(7):56–64, July 2004.

[128] Microsoft. Office live. `http://www.officelive.com/`, July 2009.

[129] Microsoft. Windows azure. `http://www.microsoft.com/azure/`, July 2009.

[130] Jose Mireles, Jr. and Frank L. Lewis. Intelligent material handling: Development and implementation of a matrix-based discrete event controller. *IEEE Transactions on Industrial Electronics*, 48(6):1087–1097, 2001.

[131] ML-IDS. Multi-level intrusion detection system. `http://www.ece.arizona.edu/~hpdc/projects/mlids/`.

[132] The MPI Forum. Homepage. `http://www.mpi-forum.org/`.

[133] Klara Nahrstedt, Hao-hua Chu, and Srinivas Narayan. QoS-aware resource management for distributed multimedia applications. *Journal of High Speed Networks*, 7(3):229–257, 1998.

[134] Jesus José de Oliveira Neto and Fábio M. Costa. An interceptor model to provide dynamic adaptation in the InteGrade grid middleware. In *Anais do VII Workshop on Grid Computing and Applications (WCGA2009)*, pages 77–88, Recife, Brazil, May 2009. Sociedade Brasileira de Computação.

[135] Jason Nichols, Haluk Demirkan, and Michael Goul. Autonomic workflow execution in the grid. *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, 36(3):353–364, May 2006.

[136] Nimbus. Nimbus science cloud. `http://workspace.globus.org/clouds/nimbus.html`, July 2009.

[137] Steve Nordstrom, Abhishek Dubey, Turker Keskinpala, Rahul Datta, Sandeep Neema, and Ted Bapty. Model predictive analysis for autonomic workflow management in large-scale scientific computing environments. In *Engineering of Autonomic and Autonomous Systems, 2007. EASe '07. Fourth IEEE International Workshop on*, pages 37–42, March 2007.

[138] Graham R. Nudd, Darren J. Kerbyson, Efstathios Papaefstathiou, Stewart C. Perry, John S. Harper, and Daniel V. Wilcox. PACE–a toolset for the performance prediction of parallel and distributed systems. *International Journal of High Performance Computing Applications*, 14(3):228–251, 2000.

[139] Daniel Nurmi, Rich Wolski, Chris Grzegorczyk, Graziano Obertelli, Sunil Soman, Lamia Youseff, and Dmitrii Zagorodnov. The Eucalyptus open-source cloud-computing system. In *Cluster Computing and the Grid, 2009. CCGRID '09. 9th IEEE/ACM International Symposium on*, pages 124–131, May 2009.

[140] Manish Parashar and Salim Hariri, editors. *Autonomic Computing: Concepts, Infrastructure, and Applications*. CRC Press, Dec 2006.

[141] Manish Parashar, Hua Liu, Zhen Li, Vincent Matossian, Cristina Schmidt, Guangsheng Zhang, and Salim Hariri. AutoMate: Enabling autonomic applications on the grid. *Cluster Computing*, 9(2):161–174, 2006.

[142] Chintan Patel, Kaustubh Supekar, and Yugyung Lee. A QoS oriented framework for adaptive management of web service based workflows. In *Database and Expert Systems Applications*, volume 2736/2003 of *Lecture Notes in Computer Science*, pages 826–835. Springer Berlin / Heidelberg, 2003.

[143] Norman W. Paton, Marcelo A. T. Aragão, Kevin Lee, Alvaro A. A. Fernandes, and Rizos Sakellariou. Optimizing utility in cloud computing through autonomic workload execution. *IEEE Data Eng. Bull*, 32(1):51–58, 2009.

[144] Cesare Pautasso and Gustavo Alonso. JOpera: A toolkit for efficient visual composition of web services. *International Journal of Electronic Commerce*, 9(2):107–141, 2005.

[145] Cesare Pautasso, Thomas Heinis, and Gustavo Alonso. Autonomic resource provisioning for software business processes. *Information and Software Technology*, 49(1):65–80, 2007.

[146] PAWS. Processes with adaptive web services. `http://www.paws.elet.polimi.it/`.

[147] PCWorld. Google outage lesson: Don't get stuck in a cloud. `http://www.pcworld.com/article/164946/google_outage_lesson_dont_get_stuck_in_a_cloud.html`, May 2009.

[148] Julien Perez, Cécile Germain-Renaud, Balázs Kégl, and Charles Loomis. Utility-based reinforcement learning for reactive grids. In *Autonomic Computing, 2008. ICAC '08. International Conference on*, pages 205–206, June 2008.

[149] Gregory F. Pfister. *In search of clusters (2nd ed.)*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1998.

[150] Guangzhi Qu and Salim Hariri. Anomaly-based self protection against network attacks. In Parashar and Hariri [140], chapter 23, pages 493–521.

[151] Mustafizur Rahman and Rajkumar Buyya. An autonomic workflow management system for global grids. In *CCGRID '08: Proceedings of the 2008 Eighth IEEE*

International Symposium on Cluster Computing and the Grid, pages 578–583, Washington, DC, USA, 2008. IEEE Computer Society.

[152] Mustafizur Rahman, Rajiv Ranjan, and Rajkumar Buyya. Dependable workflow scheduling in global grids. In *Proceedings of the 10th IEEE International Conference on Grid Computing (Grid 2009)*, October 2009.

[153] Michael A. Rappa. The utility business model and the future of computing services. *IBM Systems Journal*, 43(1):32–42, 2004.

[154] Daniel A. Reed. Grids, the teragrid and beyond. *Computer*, 36(1):62–68, Jan 2003.

[155] Benny Rochwerger, David Breitgand, Eliezer Levy, Alex Galis, Kenneth Nagin, Ignacio M. Llorente, Ruben Montero, Yaron Wolfsthal, Erik Elmroth, Juan Cáceres, Muli Ben-Yehuda, Wolfgang Emmerich, and Fermín Galán. The reservoir model and architecture for open federated cloud computing. *IBM Systems Journal*, 53(4), 2009.

[156] Stuart Russel and Peter Norvig. *Artificial Intelligence: A Modern Approach*. Prentice Hall, second edition, 2003.

[157] Ali Sajjad, Hassan Jameel, Umar Kalim, Sang Man Han, Young-Koo Lee, and Sungyoung Lee. AutoMAGI - an autonomic middleware for enabling mobile access to grid infrastructure. In *Autonomic and Autonomous Systems and International Conference on Networking and Services, 2005. ICAS-ICNS 2005. Joint International Conference on*, pages 73–73, Oct. 2005.

[158] Salesforce.com. Force.com platform. `http://www.salesforce.com/platform/`, July 2009.

[159] Salesforce.com. Salesforce.com. `http://www.salesforce.com/`, July 2009.

[160] Marcio Augusto Sekeff Sallem and Francisco José da Silva e Silva. The Adapta framework for building self-adaptive distributed applications. In *Autonomic and Autonomous Systems, 2007. ICAS07. Third International Conference on*, pages 46–46, June 2007.

[161] Marcio Augusto Sekeff Sallem and Stanley Araújo de Sousa. AutoGrid: Towards an autonomic grid middleware. In *WETICE '07: Proceedings of the 16th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprises*, pages 223–228, Washington, DC, USA, 2007. IEEE Computer Society.

[162] Joel Saltz, Tahsin Kurc, Shannon Hastings, Stephen Langella, Scott Oster, David Ervin, Ashish Sharma, Tony Pan, Metin Gurcan, Justin Permar, Renato Ferreira, Philip Payne, Umit Catalyurek, Enirco Caserta, Gustavo Leone,

Michael C. Ostrowski, Ravi Madduri, Ian Foster, Subhashree Madhavan, Kenneth H. Buetow, Krishnakant Shanbhag, and Eliot Siegel. e-Science, caGrid, and translational biomedical research. *Computer*, 41(11):58–66, Nov. 2008.

[163] YouQun Shi, ZhaoHui Zhang, Yu Fang, and ChangJun Jiang. Build city traffic information service system based on grid platform. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 1, pages 278–282 vol.1, Oct. 2003.

[164] James E. Smith and Ravi Nair. The architecture of virtual machines. *Computer*, 38(5):32–38, May 2005.

[165] Daniel P. Spooner, Junwei Cao, Stephen A. Jarvis, Ligang He, and Graham R. Nudd. Performance-aware workflow management for grid computing. *The Computer Journal*, 48(3):347–357, 2005.

[166] Daniel P. Spooner, Stephen A. Jarvis, Junwei Cao, Subhash Saini, and Graham R. Nudd. Local grid scheduling techniques using performance prediction. *Computers and Digital Techniques, IEEE Proceedings*, 150(2):87–96, Mar 2003.

[167] Alin Suciu and Rodica Potolea. Cryptographic and cryptanalytic algorithms for grid applications. In *2007 IEEE International Conference on Intelligent Computer Communication and Processing*, 2007.

[168] Alin Suciu and Rodica Potolea. A taxonomy for grid applications. In *Automation, Quality and Testing, Robotics, 2008. AQTR 2008. IEEE International Conference on*, volume 3, pages 365–368, May 2008.

[169] Richard S. Sutton. Generalization in reinforcement learning: Successful examples using sparse coarse coding. In *Advances in Neural Information Processing Systems*, volume 8, pages 1038–1044. MIT Press, 1996.

[170] Janos Sztipanovits and Gabor Karsai. Model-integrated computing. *Computer*, 30(4):110–111, Apr 1997.

[171] Y. Tanaka, H. Nakada, S. Sekiguchi, T. Suzumura, and S. Matsuoka. Ninf-G: A reference implementation of RPC-based programming middleware for grid computing. *Journal of Grid Computing*, 1(1):41–51, 2003.

[172] Gerald Tesauro. Reinforcement learning in autonomic computing: A manifesto and case studies. *Internet Computing, IEEE*, 11(1):22–30, Jan.-Feb. 2007.

[173] Gerald Tesauro, Nicholas K. Jong, Rajarshi Das, and Mohamed N. Bennani. On the use of hybrid reinforcement learning for autonomic resource allocation. *Cluster Computing*, 10(3):287–299, 2007.

[174] Huaglory Tianfield. An innovative tutorial on large complex systems. *Artificial Intelligence Review*, 17(2):141–165, 2002.

[175] Huaglory Tianfield and Rainer Unland. Towards autonomic computing systems. *Engineering Applications of Artificial Intelligence*, 17(7):689 – 699, 2004. Autonomic Computing Systems.

[176] Brian Tierney, Ruth Aydt, Dan Gunter, Warren Smith, Valerie Taylor, Rich Wolski, Martin Swany, and the Grid Performance Working Group. A grid monitoring service architecture. White Paper GWD-GP-6-1, Global Grid Forum, `http://www-didc.lbl.gov/GGF-PERF/GMA-WG/papers/GWD-GP-6-1.pdf`, 2001.

[177] Mohammed Toure, Girma Berhe, Patricia Stolf, Laurent Broto, Noel Depalma, and Daniel Hagimont. Autonomic management for grid applications. In *Parallel, Distributed and Network-Based Processing, 2008. PDP 2008. 16th Euromicro Conference on*, pages 79–86, Feb. 2008.

[178] Leslie G. Valiant. A bridging model for parallel computation. *Commun. ACM*, 33(8):103–111, 1990.

[179] Wil M.P. van der Aalst, Arthur H.M. ter Hofstede, Bartek Kiepuszewski, and Alistair P. Barros. Workflow patterns. *Distributed and Parallel Databases*, 14(1):5–51, Jul 2003.

[180] Marco Vanneschi. The programming model of ASSIST, an environment for parallel and distributed portable applications. *Parallel Computing*, 28(12):1709 – 1732, Dec 2002.

[181] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A break in the clouds: towards a cloud definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[182] Srikumar Venugopal, Rajkumar Buyya, and Kotagiri Ramamohanarao. A taxonomy of data grids for distributed data sharing, management, and processing. *ACM Comput. Surv.*, 38(1):3, 2006.

[183] Srikumar Venugopal, Rajkumar Buyya, and Lyle Winton. A grid service broker for scheduling e-science applications on global data grids. *Concurrency and Computation: Practice and Experience*, 18(6):685–699, May 2006.

[184] Monica Vladoiu and Zoran Constantinescu. A taxonomy for desktop grids from users' perspective. In S. I. Ao, Len Gelman, David W. L. Hukins, Andrew Hunter, and A. M. Korsunsky, editors, *Proceedings of the World Congress on Engineering (WCE2008)*, pages 599–604. International Association of Engineers, Newswood Limited, 2008.

[185] Jianshu Weng, Chunyan Miao, and Angela Goh. Dynamic negotiations for grid services. In *Autonomic Computing, 2004. Proceedings. International Conference on*, pages 296–297, May 2004.

[186] Marek Wieczorek, Andreas Hoheisel, and Radu Prodan. Towards a general model of the multi-criteria workflow scheduling on the grid. *Future Generations Computer Systems*, 25(3):237–256, March 2009.

[187] Marek Wieczorek, Radu Prodan, and Andreas Hoheisel. Taxonomies of the multi-criteria grid workflow scheduling problem. Technical Report TR-0106, CoreGRID European Research Network, Aug. 2007.

[188] Chee Shin Yeo and Rajkumar Buyya. A taxonomy of market-based resource management systems for utility-driven cluster computing. *Software: Practice and Experience*, 36(13):1381–1419, Nov. 2006.

[189] Chee Shin Yeo, Srikumar Venugopal, Xingchen Chu, and Rajkumar Buyya. Autonomic metered pricing for a utility computing service. *Future Generation Computer Systems*, In Press, Corrected Proof:–, 2009.

[190] Lamia Youseff, Maria Butrico, and Dilma Da Silva. Toward a unified ontology of cloud computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, pages 1–10, Nov. 2008.

[191] Jia Yu and Rajkumar Buyya. A novel architecture for realizing grid workflow using tuple spaces. In *Grid Computing, 2004. Proceedings. Fifth IEEE/ACM International Workshop on*, pages 119–128, Nov. 2004.

[192] Jia Yu and Rajkumar Buyya. A taxonomy of workflow management systems for grid computing. *Journal of Grid Computing*, 3(3-4):171–200, Sep 2005.

[193] Serafeim Zanikolas and Rizos Sakellariou. A taxonomy of grid monitoring systems. *Future Generation Computer Systems*, 21(1):163 – 188, 2005.

[194] Bernard P. Zeigler, Hae Sang Song, Tag Gon Kim, and Herbert Praehofer. DEVS framework for modelling, simulation, analysis, and design of hybrid systems. In *Hybrid Systems II*, volume 999/1995 of *Lecture Notes in Computer Science*, pages 529–551. Springer Berlin / Heidelberg, 1995.

[195] Guangsheng Zhang, Changjun Jiang, Jing Sha, and Ping Sun. Autonomic workflow management in the grid. In *Computational Science – ICCS 2007*, volume 4489 of *Lecture Notes in Computer Science*, pages 220–227. Springer Berlin / Heidelberg, 2007.

[196] Ming Zhao and Renato J. Figueiredo. A user-level secure grid file system. In *Proceedings of International Conference for High Performance Computing, Networking, Storage and Analysis (SC07)*, November 2007.

[197] Ming Zhao, Jing Xu, and Renato J. Figueiredo. Towards autonomic grid data management with virtualized distributed file systems. In *Autonomic Computing, 2006. ICAC '06. IEEE International Conference on*, pages 209–218, June 2006.