A Generic Scheduling Architecture for Service-Oriented Distributed Computing Infrastructures

Introducing New Concepts based on Automated Negotiation of Electronic Contracts

Dissertation

zur Erlangung des Grades eines

Doktors der Ingenieurwissenschaften

der Technischen Universität Dortmund
an der Fakultät für Informatik

von
Philipp Wieder

Dortmund
Januar, 2013

Tag der mündlichen Prüfung: 07/01/2013
Dekanin: Prof. Dr. Gabriele Kern-Isberner
Gutachter: Prof. Dr. Ramin Yahyapour, Prof. Dr. Dieter Kranzlmüller

Acknowledgements

Finally, this work is completed. This would have never been possible without the advise, support, and help from numerous colleagues and friends. It is not possible to express my thankfulness appropriately, but I can try: thank you all.

Special gratitude I want to express to my supervisor, Prof. Dr. Ramin Yahyapour, who helped me with his experience and knowledge to conduct my work, and who guided me through the challenge of finalising it. I would also like to thank Prof. Dr. Dieter Kranzlmüller for supervising my work and providing advice and directions.

Special thanks got to: Ali Anjomshoaa, Owen Appleton, Ranieri Baraglia, Dominic Battré, Dirk Breuer, Peter Chronz, Dietmar Erwin, Mike Fehse, Ralf Gruber, Peer Hasselmeyer, Matti Heikkurinen, Sebastian Hudert, Gregor von Laszewski, Dirk Lentzen, Daniel Mallmann, Ahmed Shiraz Memon, Mohammad Shahbaz Memon, Henning Mersch, Paolo Missier, Vincent Keller, Bastian Koller, Costas Kotsokalis, Ariel Oleksiak, Alexander Papaspyrou, Regine Pfeiffer, Volker Sander, Nicola Tonellotto, Rizos Sakellariou, Achim Streit, Andreas Savva, Oliver Wäldrich, and Wolfgang Ziegler.

Last, but not least, I want to thank my family for their patience and help. Ingrid, Friederich, Julie, and Hinni: case closed.

Annette and Juri, the gain in time is yours ...

Abstract

In state-of-the-art distributed computing infrastructures different kinds of resources are combined to offer complex services to customers. As of today, service-oriented middleware stacks are the work-horses to connect resources and their users, and to implement all functions needed to provide those services. Analysing the functionality of prominent middleware stacks, it becomes evident that common challenges, like scalability, manageability, efficiency, reliability, security, or complexity, exist, and that they constitute major research areas in information and communication technologies in general and distributed systems in particular.

One core issue, touching all of the aforementioned challenges, is the question of how to distribute units of work in a distributed computing infrastructure, a task generally referred to as scheduling. Integrating a variety of resources and services while being compliant with well-defined business objectives makes the development of scheduling strategies and services a difficult venture, which, for service-oriented distributed computing infrastructures, translates to the assignment of services to activities over time aiming at the optimisation of multiple, potentially competing, quality-of-service criteria.

Many concepts, methods, and tools for scheduling in distributed computing infrastructures exist, a majority of which being dedicated to provide algorithmic solutions and schedulers. We approach the problem from another angle and offer a more general answer to the question of 'how to design an automated scheduling process and an architecture supporting it'. Doing so, we take special care of the service-oriented nature of the systems we consider and of the integration of our solutions into IT service management processes.

Our answer comprises a number of assets that form a comprehensive scheduling solution for distributed computing infrastructures. Based on a requirement analysis of application scenarios we provide a concept consisting of an automated scheduling process and the respective generic scheduling architecture supporting it. Process and architecture are based on four core models as there are a model to describe the activities to be executed, an information model to capture the capabilities of the infrastructure, a model to handle the life-cycle of service level agreements, which are the foundation for elaborated service management solutions, and a specific scheduling model capturing the specifics of state-of-the-art distributed systems.

We deliver, in addition to concept and models, realisations of our solutions that demonstrate their applicability in different application scenarios spanning grid-like academic as well as financial service infrastructures. Last, but not least, we evaluate our scheduling model through simulations of artificial as well as realistic workload traces thus showing the feasibility of the approach and the implications of its usage.

The work at hand therefore offers a blueprint for developers of scheduling solutions for state-of-the-art distributed computing infrastructures. It contributes essential building blocks to realise such solutions and provides an important step to integrate them into IT service management solutions.

Contents

List of Tables

List of Figures

Acronyms

AWRT	Average Weighted Response Time
APS	Application Service Provisioning
AWS	Amazon Web Service
CRM	Customer Relationship Management
CIS	Common Information Service
CPU	Central Processing Unit
DCI	Distributed Computing Infrastructure
EC2	Elastic Compute Cloud
EPR	End-point Reference
HPC	High Performance Computing
ITIL	Information Technology Infrastructure Library
JSDL	Job Submission Description Language
LSF	Load Sharing Facility
OGF	Open Grid Forum
OCCI	Open Cloud Computing Interface
SDT	Service Description Term
SLA	Service Level Agreement
SWF	Standard Workload Format
S3	Simple Storage Server
UDAP	Universal Dynamic Activity Package
UNICORE	Uniform Interface to Computing Resources
UUID	Universally Unique Identifier
XML	Extensible Markup Language
WSAG4J	WS-Agreement for Java

WSDL Web Service Definition Language

WSLA Web Service Level Agreement

WSRF Web Services Resource Framework

XaaS Everything-as-a-Service

Part I.

Introduction and Problem Space

1. Introduction

With the advent of networked computers evolved the desire to combine the power of multiple computers into a single, more powerful system: the idea of *distributed systems* emerged. Nowadays such systems are commonly used and fulfil a variety of tasks, for example distributed data storage, distributed information provisioning, or distributed data collection. However, there is a common characterisation given by Tanenbaum and Van Steen to describe such systems, which perfectly fits the scope of our work: 'A distributed system is a collection of independent computers that appears to its users as a single coherent system.' [131].

In addition to the examples of distributed systems given above, and many more already used by the Internet society, one specific kind of distributed system has been in the focus of research for a long time: the *distributed computing system*. Following Tanenbaum and Van Steen, the purpose of a distributed computing system is to provide the combined compute power of multiple computers as a single coherent system with more CPUs or nodes or Flops, depending on the metric applied. This concept is very often also referred to as high-performance computing (HPC), with *cluster computers* and *grids* as the most prominent representatives.

But a distributed system does not necessarily have to fulfil the needs of high-performing applications or have to exclusively provide compute power to qualify as a distributed 'computing' system. Current systems, although having the main purpose to supply remote users with computing power, include access to storage facilities, high-speed networks, or even remote instruments. Therefore we talk about whole infrastructures, like those providing the IT facilities for the big e-Science experiments at CERN, which represent the evolution of distributed computers, and hence are called *distributed computing infrastructures*, or in short *DCIs*.

As it turns out when it comes to DCIs: the whole is not always more than sum of its parts. Not only is the measured maximum performance of all state-of-the-art high-performance computers much lower then their theoretical peak performance[1], but also scaling a parallel application to twice the number of CPUs or cores most likely does not imply half the runtime. For globally distributed computing infrastructures this comes more than true, as latency, heterogeneous hardware, or data dis-locality commit their share.

A large variety of projects, products, and initiatives have been created to compensate the various effects which prevent a distributed computing infrastructure to perform well and behave like a single coherent system.[2] The outcomes of such efforts contribute to the whole spectrum of hardware, software, algorithms, models, and methods which may comprise a DCI.

The entirety of software (excluding the operating system) that is used to set up and manage a distributed computing infrastructure is best known as middleware [80]. It is

[1] As proven by the current Top 500 list of supercomputers: TOP500 List - November 2010 (1-100), last visited February 18, 2011: http://www.top500.org/list/2010/11/100.

[2] A selection of these efforts is presented in Section 2.1.

situated between the application and the operating system and fulfils the purpose of combining the different resources, as there are computers, storage, network, and alike, into one single coherent system. Typical functions of such middleware are independent of the kind of DCI and include execution and control of computational activities, data transfer, authentication, authorisation, information provision, or scheduling.

This thesis' focus is on scheduling. It answers questions like what the purpose of a scheduling service in a state-of-the-art DCI actually is, what the essential links of a scheduling service to other parts of the middleware are, and how the scheduling process is executed. But the main contribution goes beyond this.

Contribution of this Thesis

The process of deciding who is allowed to use which resource when is called scheduling. Or to quote Pinedo: 'Scheduling concerns the allocation of limited resources to tasks over time. It is a decision-making process that has as a goal the optimization of one or more objectives.' [111]. Such optimisations are useful not only for DCIs, but are needed in many other areas as there are production processes, enterprise resource planning, mobile network routing, university room planning, or staff allocation. Examples from the IT world are also manifold: it is e.g. neither advisable to direct all Google query requests to one web service instance, nor does it make sense to start conditional parts of a scientific work-flow at once [139].

The challenge of scheduling in distributed computing infrastructures remains unsolved. Existing solutions mainly produce highly-optimised solutions for specific application scenarios, but fail to produce significant generic results. This is on the one hand due to the massive number of application scenarios with a wide range of scheduling requirements, on the other hand due to the current research either focussing on theoretical algorithmic work or producing highly-specialised scheduling systems. Extensive research has been done on scheduling algorithms in general [18, 81, 111, 134] and for distributed computing systems in particular [20, 28, 62, 92, 98, 101, 119, 142, 144, 146]. Furthermore, various schedulers have been developed for cluster computers and grids. But neither the algorithms nor the schedulers exist in isolation, they are part of an infrastructure and only as effective as the other parts of the infrastructure allow. Especially since DCIs become more and more service-oriented infrastructures and, with cloud computing being the latest trend, everything can be a service, schedulers become scheduling services that reside within a service environment and are part of complex service-level management solutions.

The general task of a scheduling service within a DCI is always the same independent of the underlying execution system: mapping the requirements derived from the activity description (or task as called by Pinedo) onto the capabilities of the resources that are controlled by the scheduling service. The result of this scheduling process is called a schedule, a (potentially ordered) list of alternatives of where (i.e. on which execution system) and when to execute an activity.

It is the purpose of this thesis to provide a broader view on scheduling in distributed computing infrastructures then most other comparable work. To achieve this, we evaluate horizontal challenges and solve a sub-set of them for state-of-the-art DCIs. As a first step, we evaluate a number of application scenarios that are related to scheduling and gather common requirements. Originating from them, we present the concept of a generic scheduling architecture for distributed computing infrastructures. Its design adheres to the principles

of service-oriented infrastructures and proposes a set of common services that encapsulate particular functions and logic, which are required for scheduling. Furthermore, the architecture relies on common DCI middleware services, like registries or information systems. Implementations following our design, consequently, form a scheduling framework within a DCI, thus fulfilling the requirements of the application scenarios and integrating with the middleware of an distributed computing infrastructure. In addition to services and their interrelations, we, too, define a scheduling process for DCIs that takes the common requirements into account.

Both, the architecture and the scheduling process are based on four models developed in the course of the thesis: (i) an activity model, (ii) an information model, (iii) a negotiation model, and (iv) a scheduling model governed by service-level agreements. The first takes the notion of a unit of work within a DCI to a different level. Not only jobs or work-flows, but also activities like database requests or web service calls are handled equally with respect to state tracing, resource monitoring, or even scheduling. In all cases, the activity is core to processing and monitoring, comprising all information related to it. Hence, with the activity model realised, there is no fragmentation and dispersion of information related to an activity, a situation that makes managerial tasks in current DCIs, like state and exception handling or real-time assessment, complicated and error-prone operations.

The second model tackles the still omnipresent challenge of modelling the capabilities of services and resources within a DCI. Historically, almost every DCI middleware features its own information model. This is one technical reason why especially the grid community finds themselves in a situation of insufficient interoperability. Through multiple initiatives and especially the GLUE standardisation effort [5], this problem currently diminishes as many of the grid middleware distributions adhere to GLUE. But with the focus on cloud infrastructures and the requirement of service-level management becoming more important, it is essential to broaden the view on manageable entities within a DCI. This is why we propose an information model for distributed infrastructures based on the *Common Information Model* (CIM) standard [83]. It comprises descriptions for a large number concepts and the respective interrelations, which are needed to describe the capabilities of services, sites, and whole DCIs. The Common Information Model, furthermore, provides extension points to create customised, domain-specific, information models. One such model is the specific contribution of this thesis as it demonstrates how CIM can be applied to capture the resource model of the UNICORE[3] middleware.

The third model addresses multi-phase negotiation in a distributed computing infrastructure in which service-level agreements are applied. The development of this model is a result of scenarios requiring the option to automatically negotiate the quality of service provision not only in a basic accept/reject fashion, but execute multiple, iterative, steps to reach an agreement between service consumer and service provider. Following a number of draft approaches, an Open Grid Forum community effort, to which we contribute, has released the *WS-Agreement Negotiation* (WSAGN) proposed recommendation. It fulfils the requirement of multi-step negotiation and thus allows services within a DCI to automatically negotiate service levels, responsibilities, and constraints.

The fourth model outlines the use of service-level agreements as the foundation for scheduling. Ever since the transition of DCIs towards service-oriented infrastructures has been noticeable, it became apparent that service-level management would be an integral

[3]UNICORE, last visited: January 25, 2013. http://unicore.eu/.

part of modern DCIs. With scheduling being a fundamental part of the management of DCIs, it was also evident that the scheduling process, now comprising the co-ordination of services and resources, would be based on quality-of-service constraints. One possibility which we evaluated was the scheduling of service-level agreements. They comprise not only the description of an activity, which is common practice for example in grids, but also capture service-levels, guarantees, and constraints. With such an service-level agreement being scheduled, one can provision the whole activity-specific set of services and resources dynamically when needed. SLAs also offer the enforcement of business objectives, dynamic assessment of quality-of-service parameters, and many more. Our research resulted in a scheduling model based on WS-Agreement [6], the standard on which the aforementioned WSAGN is also based upon.

Each of the four models constitutes a specific contribution to the development of a generic scheduling architecture for DCIs, but the core contribution of this thesis is the integration of all the models into one consistent design and their application to the scheduling process. The purpose of the integration will become visible throughout the thesis, nevertheless, by way of illustration, we refer to a few implications here. SLAs, for example, are not merely used as contracts that govern scheduling, but they are the back-bone of service-level management within a DCI. It is therefore essential that they are well-integrated with the information management. In particular, this implies that mechanisms exist which allow services to effectively exploit information about entities within a DCI and map this information to service-levels in order to execute managerial tasks like SLA-compliance assessment, adjustment of service provisioning, or scheduling of activities. Another example is the integration of activity management into the scheduling process. Although the introduction of the activity model is an achievement on its own, it is the integration with the scheduling process that reveals its true value. Having all information related to an activity in once place, the scheduling service can conveniently harness the data and can take it into consideration for making scheduling decisions. This provides a much broader foundation for scheduling and will promote the development of algorithms that operate on a wider set of parameters and are more suitable for the application to the generic scheduling process.

As an addition to the models, we also contribute a number of implementations. Although we do not provide a full implementation of the the whole generic scheduling architecture, a venture that is almost equivalent to the realisation of a complete distributed computing infrastructure, a number of prototypical service frameworks have been realised in an academic context. There, the focus has been on the core functions of the scheduling process while ‘auxiliary’ services like security, billing, accounting and alike have not been considered.

The implementations offer a wide range of possibilities to evaluate the application of the models to the scheduling process. As we do not target specific algorithms or domain-specific schedulers, we decided that it would be of benefit to the community if we evaluated the delegation of scheduling requests. We therefore simulated a DCI environment with a number of scheduling services which apply various delegation strategies. It is the analysis of these strategies which contributes to the discussion of outsourcing activities in case of overloaded systems, a situation common to distributed computing infrastructures and of growing importance in cloud-like set ups.

We believe that the entirety of the artefacts, as there are the generic architecture, the scheduling process, the models, the implementations, and the evaluation of delegation

strategies, provide a relevant contribution to the area of scheduling in distributed computing infrastructures. Especially the inclusion of service-level management-related aspects and the introduction of the activity model are important considerations we have to make with service-orientation and cloud computing currently changing the whole DCI landscape.

To sum it up, this thesis contributes the following assets to the discussion on scheduling in DCIs:

- A list of scheduling requirements.
- A generic scheduling architecture for DCIs.
- An automated scheduling process for DCIs.
- A model to describe activities.[4]
- An information model applicable to DCIs.[5]
- An SLA negotiation model.[6]
- A scheduling model for DCIs.
- Implementations of various parts of the architecture and the scheduling process.[7]
- An evaluation of the scheduling model.

Thesis' Structure

This thesis comprises 11 chapters which are organised into six logical blocks:

1. Introduction to topic and problem space.
2. Requirements gathering and conception.
3. Core models.
4. Implementation.
5. Evaluation.
6. Conclusion.

Following this chapter, which provides a condensed introduction to the area of DCIs and to the contributions of this thesis, we show in Chapter 2 'Motivation' the need for a generic scheduling architecture, based on unsolved challenges and current state-of-the-art. These two chapters together form the first logical block. Chapter 3 'Requirements' then introduces scheduling scenarios, gathers common requirements, and evaluates them in the

[4] We developed the initial model together with colleagues from the NextGRID project. Furthermore, we proposed the model to the Open Grid Forum for further uptake.

[5] Our information model is a domain-specific instantiation of the Common Information Model standard.

[6] This model has as been specified with our participation through the Open Grid Forum community process. The resulting WS-Agreement Negotiation specification has been published as GFD.193 [138].

[7] The implementations result from various projects and research activities realising parts of the ideas contributed by this work.

light of service-oriented DCIs. All this is the foundation for the second part of the 'requirements gathering and conception' block, which is Chapter 4 'Concept'. There, we present services that fulfil the common requirements and form a generic framework for scheduling in distributed infrastructures. Furthermore, we show the different steps of the scheduling process and how they benefit from the generic architecture. The remainder of this chapter then motivates the following two blocks which provide in-depth descriptions of the 'core models' and the 'implementation'. The former comprises four models as there are the 'Activity Model' (cf. Chapter 5), the 'Information Model' (cf. Chapter 6), the 'Service-Level Agreements Model' including the specific contribution to the negotiation of SLAs (cf. Chapter 7), and the model for 'Scheduling with SLAs' (cf. Chapter 8). The latter presents different implementations that realise the models and that are instantiations of the services that form a scheduling framework within a service-oriented and distributed infrastructure (cf. Chapter 9 'Realisation'). Subsequent to the various implementations, we revive in Chapter 10 'Simulation and Evaluation' the delegation of scheduling requests, an application scenario which has been introduced in the 'Requirements' section. This case represents the core characteristics of SLA-governed scheduling within DCIs and we therefore use it as the foundation for a simulation scenario and the subsequent evaluation of various delegation strategies. The final logical block consists of Chapter 11 'Conclusion', in which we summarise briefly our findings and contributions, describe there effects, and discuss open issues.

2. Motivation

This chapter provides an introduction to the problem definition and motivates the work conducted for this thesis. To lead there, we first briefly describe the history and characteristics of distributed computing infrastructures. Next, we examine common challenges that designers, developers, and operators of distributed computing infrastructures face independent of domain or application. Then, we introduce the scheduling challenge in general and the DCI-related scheduling problem we aim to solve in particular, thus scoping the motivation for our work.

2.1. Distributed Computing Infrastructures

The crux of defining what a *distributed computing infrastructure* actually is appears obvious when looking into the variety of definitions that are available for distributed computing systems and its instantiations like grid computing or cloud computing[1]. The definition dilemma aggravates by bringing terms like web services, service-oriented architectures (or likewise service-oriented infrastructures), XaaS[2], high-performance computing, or alike onto the table.

The characterisation of distributed systems as cited in the introduction, though, refers to the 'coherent view' that should be offered to users independent of the number or kind of underlying services and resources. Approaching DCIs from this angle and putting aside the technology-, paradigm- or middleware-related definitions, we can focus on common research challenges of the different disciplines engaged in distributed computing research.

We therefore first examine the two most prominent examples of paradigms that are used to build distributed computing infrastructures: grid and cloud[3]. Following that, we introduce a list of their common challenges later in this section. This list then serves as the foundation for the introduction of scheduling in such systems (cf. Section 2.2), leading to the problem definition of this thesis.

[1] The article 'Grid computing definition' (last visited: June 08, 2011. http://www.arjuna.com/node/53) lists and discusses a number of definitions for what a grid is including the very popular check-list by Ian Foster [47]. As for cloud computing, also various definitions exist, like for example those collected by the Cloud Computing Journal and published int the article 'Twenty-One Experts Define Cloud Computing' (last visited: June 08, 2011. http://cloudcomputing.sys-con.com/node/612375).

[2] Acronym for everything-as-a-service.

[3] Figure 2.1 shows selected events during the last 15 years that are related to the evolution of grid and cloud computing.

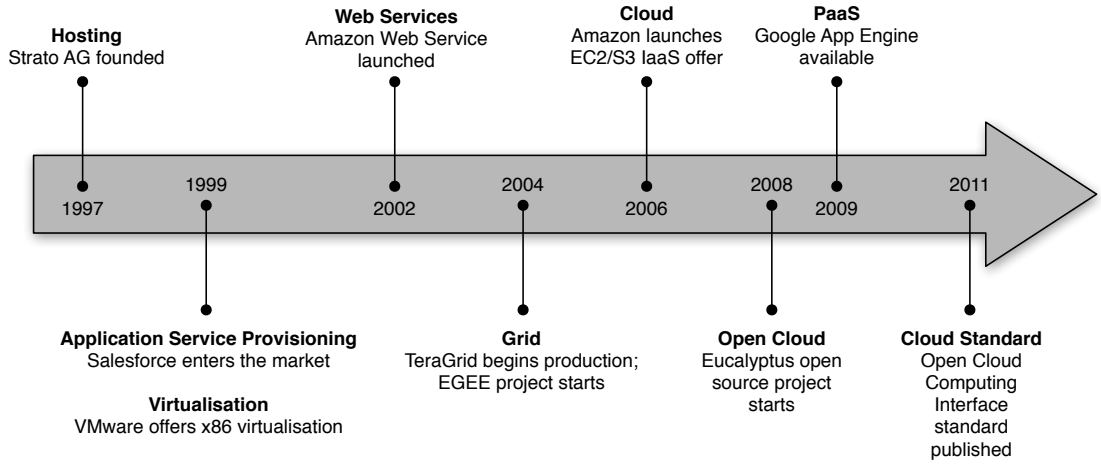

Figure 2.1.: Selected events related to the evolution of distributed computing infrastructures towards clouds

2.1.1. Grid Computing

Although the technological evolution of the previous decades is especially visible in the area of Information and Communication Technologies, computing power has always been a limited resource. The apparent reason for this is adaptation. Whenever faster processors or bigger systems are available, users either create new applications or increase the problem size respectively. Especially providers of high-performance computers observe this phenomenon with every new system generation: applications exploit more nodes, more cores, faster networks, or larger storage systems up to their limits. As of today, this can only lead to one conclusion: there is still a growing need for more computing power [97].

Besides increasing the performance of processors - and consequently single processor computer systems, the aggregation of processors to form high-performance computing systems is a suitable way to achieve better application performance. This makes HPC systems - not considering special purpose computers - the fastest machines according to benchmarks such as LINPACK[4], which is used as the metric to generate the list of the top 500 computers[5]. For quite some time now, the building of clusters is an interesting subject for high-performance computing. Linking single and autonomous computers like workstations or servers via a communication network to a cluster is a cost-efficient alternative for some but not all applications.

The evolution of the internet in the past two decades is an another development that dramatically changed the life of our, now to be called, information society. The ability to be connected and linked to the internet is omnipresent. The simple connection of our computing resources to the net is yet common standard and practice. The trend of connecting everything, not just computers but all kinds of wired resources, via a network has already reached commodity devices, leading to the term *internet of things* [24].

Combining the need for computing power with the connection of all kind of IT resources via networks led to advanced concepts that have gained a lot of attention by research in the past decade.

[4]The LINPACK benchmark, last visited: January 27, 2011. http://www.top500.org/project/linpack.
[5]TOP500 Supercomputer Sites, last visited: January 27, 2011. http://www.top500.org/.

First of all, the term *meta-computing* was established in 1987 by Smarr and Catlett [125]. Here, the target resources are high performance computers, which are most likely situated in different locations and operated in different administrative domains. The users of so called *meta-computers* then ideally do not have to care about where a computing task is actually executed, thus implementing the former slogan by Sun Microsystem: "The network is the computer". The underlying approach of meta-computing has similarities to *cluster computing* as different computers are connected via a network. But in comparison, cluster computing terms an infrastructure where single servers, usually blades stacked in racks, but also workstations, are interconnected. And, in general, clusters belong to the same owner or administrative instance.

The term meta-computing was substituted by *grid computing* or just the *grid*. While the scope of meta-computing is limited to compute resources, grid computing takes a broader approach on the resources that are connected. Here, devices for visualisation as well as data storage are considered to be part of a grid. Meta-computing – besides several limited projects – never got common practice, whereas grid computing gained a lot of attention and evolved to a commonly used platform in scientific communities, mainly for compute and data-oriented tasks.

The term grid has been coined by Foster et al. [50] and draws an analogy with the electrical power grid, which delivers electrical power just on demand without requiring more knowledge from users then where to put in the plug. The analogy is to provide computational power on demand to all users without thorough knowledge about the underlying resources and how to use them.

Although breaking the promise of delivering a grid of computational and related resources to the non-expert user, grids became a common foundation for e-Science infrastructures. As many scientific disciplines need access to computing or data resources while collaborating with colleagues in their research area, the concept of resource sharing within *virtual organisations* [51] has been successful. As a result, many national and international grid initiatives and specific grid infrastructure for certain user communities emerged and are still maintained and extended. Well known examples are the European projects PRACE[6] and DEISA[7], but also grid-based infrastructures like EGEE[8], EGI[9], FutureGrid[10], and TeraGrid[11]. These grid infrastructures are currently in production and serve the computing demands of researchers around the globe.

2.1.2. Cloud Computing

Contrary to original expectations, grids did not fulfil the objectives set by the naming analogy: there is very small uptake beyond academic research, thus there is no "computing on demand" for the general user yet. There are various potential reasons why this uptake did no take place, but it is not the purpose of this thesis to speculate about them.

However, cloud computing turns out to be the new technological paradigm taking the

[6] Partnership for Advanced Computing in Europe, last visited: January 27, 2011. http://www.prace-project.eu/.
[7] Distributed European Infrastructure for Supercomputing Applications, last visited: January 27, 2011. http://www.deisa.eu/.
[8] Enabling Grids for E-Science, last visited: January 27, 2011. http://www.eu-egee.org/.
[9] European Grid Infrastructure, last visited: January 27, 2011. http://www.egi.eu/.
[10] FutureGrid Portal, last visited: January 27, 2011. https://portal.futuregrid.org/.
[11] TeraGrid, last visited: January 27, 2011. https://www.teragrid.org/.

challenge of providing distributed computing power to a broader public. It already has the buy-in from industry, resulting in terms like *scale out*, *multi tenancy*, and *pay-as-you-go* become increasingly popular to describe this new approach, showing on the one hand the business model behind the cloud offerings and on the other hand underlining its commercial character.

In comparison to the popular Application Service Provisioning (ASP), which also offers a pay-as-you-go model via web interfaces, cloud services are designed to be operated within a web infrastructure, with AJAX, Adobe AIR, or Microsoft Silverlight on the client side and multi tenancy service offers on the back-end [16]. According to a popular layering, we distinguish three different layers when it comes to clouds and their respective services: infrastructure-as-a-service, platform-as-a-service, and software-as-a-service, often subsumed under the term *everything-as-a-service*. The different layers including some of their most common characteristics are shown in Figure 2.2.

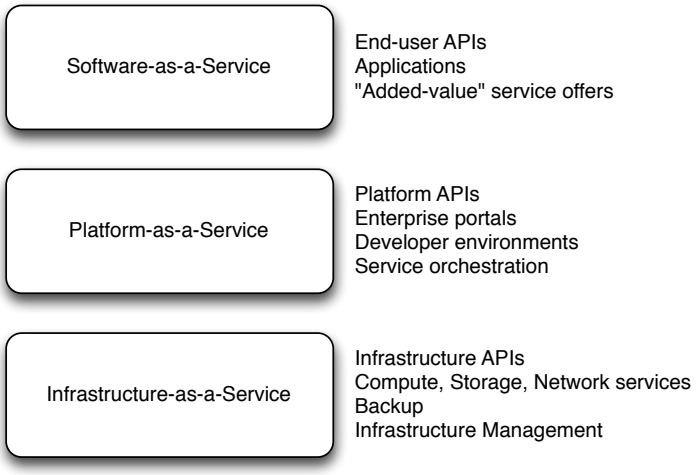

Figure 2.2.: The main XaaS service categories and their characteristics

The growing revenue created by cloud-like service offers, according to the International Data Corporation increasing from $16.5 billion in 2009 to over $55 billion in 2014 [55], is already prominently visible looking at the growing number of cloud services and users. Amazon, for example, is well known for its Amazon Web Service (AWS) suite[12], offering inter alia the Elastic Compute Cloud (EC2) and the Simple Storage Server (S3) on IaaS level. On platform level, Google's App Engine[13], Microsoft's Windows Azure[14], and Salesforce's Force.com[15] are just some examples of the broad range of services offered. Last but not least, there is an rapidly increasing offer of software-as-a-service, ranging from project management software by Basecamp[16], over office solutions to complete customer relationship management (CRM) solutions like SAP's CRM OnDemand[17].

[12] Amazon Web Service, last visited: February 17, 2011. http://aws.amazon.com/de/.
[13] Google app engine, last visited: February 17, 2010. http://appengine.google.com.
[14] Windows Azure – Microsoft's Cloud Service Platform, last visited: February 17, 2011. http://www.microsoft.com/windowsazure/.
[15] Force.com, last visited: February 17, 2011. http://www.salesforce.com/platform/.
[16] Projects Manage Themselves with Basecamp, last visited: February 17, 2011. http://basecamphq.com/.
[17] SAP CRM OnDemand Solution, last visited: February 17, 2011. http://www.sap.com/solutions/business-

But there is not only buy-in on the commercial side, there are also plenty of research projects and open source efforts targeting a variety of topics associated with clouds. The following list provides a few pointers to related efforts:

- SLA@SOI[18] is a European project that develops a generic service-level management solution. An instantiation of this solution provides management functions for IaaS.
- OpenNebula[19] is on open source toolkit that can be used to manage a cloud infrastructure.
- A similar approach as taken by OpenNebula represents Eucalyptus[20].

On the interface between commercial offers and academic research resides the request for standards; although a standard – in the cloud area as in any other – does not necessarily imply a global agreement on a certain protocol or interface[21]. However, interfaces like libvirt[22], Amazon's EC2/S3, or the Open Grid Forum's Open Cloud Computing Interface (OCCI[23]) show significant uptake and may represent future (de facto) standards.

While there are many differences in Cloud and Grid infrastructures, there are also many similarities. Both paradigms cater for simplified access to distributed services and resources, which are made available by different providers. Clouds are suitable to create a layered architecture that separates infrastructure from applications, utilising the same infrastructure services to run arbitrary applications. Similarly, applications can be decomposed into several software services (or multiple services orchestrated into an added-value application), which run on such virtual infrastructures. Grids, however, target a similar space by combining resources from different providers in a networked infrastructure. Grids also require substantial middleware efforts to provide the foundation of core services. These similarities are well described by the common research challenges, which we outline in the following section.

2.1.3. Common Challenges of Distributed Computing Infrastructure Research

Common challenges for designing, building, and operating distributed computing infrastructures exist independent of any trend or paradigm that is pushing the technology evolution forward. Some, DCI-inherent, challenges remain the same since the days of metacomputing, as there is for example the omnipresent wish of providers to maximise the load of their machines (identifying it with minimising down-time and preventing idle resources), while others, like the demand for energy-efficient operation of resources, focus attention for a specific period and are often the result of external factors. The following paragraphs

suite/crm/crmondemand/index.epx.

[18] SLA@SOI, last visited: February 17, 2011. http://sla-at-soi.eu/.

[19] OpenNebula – The Open Source Toolkit for Cloud Computing, last visited: February 17, 2011. http://www.opennebula.org/.

[20] Eucalyptus – The Open Source Cloud Platform, last visited: February 17, 2011. http://open.eucalyptus.com/.

[21] A comprehensive summary of the ongoing cloud standards development is provided in the Cloud Standards Wiki, last visited: February 17, 2011. http://cloud-standards.org/wiki/index.php?title=Main_Page.

[22] libvirt – The virtualization API, last visited: February 17, 2011. http://libvirt.org/.

[23] OCCI – Open Cloud Computing Interface, last visited: February 17, 2011. http://occi-wg.org/.

list some of those challenges to set the general scene for distributed computing infrastructures, while the specific challenges addressed by this theses are then introduced in Section 3.

Scalability

Although the efficient usage of large infrastructures with thousands of computing units is challenging, DCI operators, developers, and users understand well how to gain notable application performance in many domains. Observing the increase of processing cores per CPU and the deployment of respective infrastructures, however, we will experience a significant increase in cores provided by an average HPC system over the next years. This perspective already results in the evaluation of scalability issues regarding 'peta-scale' applications and the respective programming models [44] and leads to various approaches to cope with these issues [72]. Similarly, we also face novel challenges concerning federations of such systems, whether they form large-scale grids like DEISA or cross-cloud infrastructures. There, problems are not limited to application-programming models, but also include scalability-related issues concerning infrastructure and service management. A DCI with millions of cores, in which each may be treated like a separate resource, requires new approaches towards information management, control mechanisms, and monitoring. A multi-layered service-level management system will be most likely the solution to handle the size of such infrastructures. This then raises the question of whether solutions feasible for large industrial cloud providers can in the same way serve the needs of private cloud operators, thus introducing not only technical but also economical aspects to the scalability challenge.

Manageability

The management of DCIs remains a challenge independent of the technology used: it was a difficult task with respect to grids and remains problematic realising clouds. Although the latter came up with the promise of easy usage, this is only the case for customers and only at the expense of a reduced set of functions. Providing and managing clouds is still a challenge that even the largest industrial providers sometimes fail to meet successfully. Such a failure combined with the deliberate 'outsourcing' of management functions to the customer can lead to significant service downtimes [56], a situation which even more emphasises the necessity for research in the area of DCI manageability.

The management of service-oriented (distributed) infrastructures is, from a general viewpoint, an undertaking which spawns almost all parts of the organisation providing the respective service. One holistic approach to integrate all management tasks is the *IT Infrastructure Library* (ITIL) which tackles, in addition to service-level management (the technical realisation of which we cover in this thesis), processes like capacity management, availability management, problem management, and application management [135]. Especially in the cloud context ITIL and similar approaches are valuable input trying to cope with the increasing complexity of management tasks.

Efficiency

Research related to distributed computing infrastructures often addresses optimisation issues to provide more efficient services or operate IT infrastructures in a more efficient manner. Common challenges in this area, related especially to the topic of this thesis, are the optimisation of application response-times, the increase of a DCI's activity throughput, or, driven by the global need for energy efficiency [133] and sustainability[24], the efficient utilisation of the physical infrastructure. This challenge requires methods, tools, and algorithms that support the optimisation of multiple, often contradictory, criteria. Especially scheduling services are core to efficient IT taking application requirements into account and matching them to infrastructure capabilities.

Reliability

Distributed infrastructures grow continuously in size (see paragraph about 'Scalability') and applications increasingly exploit the dynamic orchestration capabilities of service-oriented systems. We therefore face a landscape where, dynamically based on multi-criteria optimisation, applications are dependent on multiple services and thus are the service offers which are based on these applications. Especially cloud-based business services, which are, for example, developed using vmforce[25], executed on Amazon's EC2, and making use of Microsoft's Azure Storage[26], vitally need reliable infrastructures and auxiliary services.

To ensure reliability, the management of quality-of-service requirements and constraints through electronic contracts plays an important role. The variety of services, roles, and business objectives in large-scale DCIs requires a suitable abstraction of quality-of-service parameters and their automated processing. Moreover, models are needed that not only capture job-like resource requirements and capabilities [7], but provide the foundation for redundant, fault-tolerant, and secure service provisioning. Service-level agreements, as they are for instance described by ITIL, are a potent tool to build reliable service infrastructures.

Security

Existing grid infrastructures like those mentioned in Section 2.1.1 have already built-in security features to manage public key infrastructures (PKIs) and virtual organizations, or to transfer data via encrypted and secured channels. But experience presumes that many of these solutions may not be the right choice for usage in large-scale cloud infrastructures with different business models and dynamic service provisioning. As trust and security are still prominent challenges on the cloud research agenda [8], easy-to-use and seamless, but at the same time scalable security solutions will be necessary to extend clouds beyond the current business model. Such solutions will include support for various standards which can not be found in grid-like infrastructures and they will provide more integration

[24] ICT for Sustainable Growth, last visited: January 25, 2013. http://ec.europa.eu/information_society/activities/sustainable_growth/index_en.htm.
[25] The trusted cloud for enterprise Java developers, last visited: January 25, 2013. http://www.vmforce.com/.
[26] Windows Azure Storage, last visited: January 25, 2013. http://www.microsoft.com/windowsazure/storage/.

of various approaches[27].

Complexity

Most of the aforementioned aspects relate to the overall challenge of mastering the complexity of a distributed computing infrastructure which in general should scale well, should provide services dynamically, and should integrate easily across different administrative domains. With an envisaged scale of millions of services, compute nodes, and an orders of magnitude larger number of connected devices and data objects that altogether will be part of a DCI, or even many at once, we have to handle complex system beyond what is nowadays possible. Such systems need to, on the one hand, hide some of their complexity and offer simple access mechanisms, and, on the other hand, increase the degree of automation and self-management.

2.2. Scheduling in Distributed Computing Infrastructures

The general, fundamental objective of scheduling has already been described in the paragraph 'Contribution of this Thesis' of Section 1: Pinedo defines the task of scheduling as the optimisation of a number of objectives aiming at 'the allocation of limited resources to tasks over time' [111].

Taking only a single-core CPU into account, Pinedo's definition mainly implies that the various processes, which are executed on this CPU, have to be scheduled in a way that every process gets its share of CPU-time. Such 'uniprocessor time-slicing' [27], has been extensively researched and may take different, potentially competing, objectives into account. Another scheduling dimension opens up by adding more CPUs to the equation, whether through multiprocessor machines, cluster computers, or massively-parallel processing (MPP) systems. There, it is not only the question 'when' to execute a task, but also 'where'. Although the solution might seem, at first sight, simple by just assigning tasks to CPUs until the machine is completely loaded (and keep further tasks in a waiting queue), parallel I/O [30] or system fragmentation [99], to give just two examples, advocate more elaborated solutions.

Distributed computing infrastructures raise the complexity of infrastructure management, and thus scheduling, to yet another level. The following criteria, among others, increase the difficulty of developing proper scheduling solutions:

- DCIs comprise different kind of resource types which potentially have to be co-scheduled [141].
- Especially in grids, resources belong to different administrative domains that decide autonomously who has which access rights on a specific resource [41].
- A DCI might include especially scarce resources like scientific instruments [140].
- In general, most resources are embedded into some kind of local scheduling leading to DCI scheduling hierarchies [31].

[27] Like for example provided by Microsoft Azure which supports already a variety of access control mechanisms. Last visited: June 09, 2011. http://www.microsoft.com/windowsazure/appfabric/accesscontrol/.

- Although normally a computational task is treated as the entity to be scheduled and data is just seen as related input or output, large amounts of such data imply to take specific care of its locality [33].

Taking all the above constraints into account, DCI scheduling becomes a complex process that tries to optimise multiple criteria at once. We therefore face, in cases where we not only try to optimise one criterion like machine-load, a so-called 'multi-criteria scheduling problem' [134], many of which are NP-hard[28]. Hardware, like multi-core CPUs or graphical-processing units (GPUs), application domains, including complex work-flows or real-time scenarios, and scale, considering thousands of potential resources as execution entities, are additional factors which make scheduling in distributed computing infrastructures an increasingly complex problem.

In service-oriented environments, scheduling is part of service-level management and all the criteria defining the scope of service provisioning can be subsumed under the term 'quality-of-service'. The purpose of the scheduler in such an environment is to map the user's requirements to the provider's capabilities, a task resulting in an agreement about the quality-of-service, i.e. the different service-related parameters which govern the service provisioning. Such an agreement is normally called a service-level agreement[29] (SLA). Although this is mainly a terminology-related issue, we adhere throughout this thesis to service-related terms since most state-of-the-art DCIs are service-oriented infrastructures.

2.2.1. Problem Definition

In summary, we can, returning to the introductory definition of scheduling by Pinedo, define the problem of scheduling in DCIs as follows:

Scheduling in service-oriented distributed computing infrastructures translates to the assignment of services to activities over time aiming at the optimisation of multiple, potentially competing, quality-of-service criteria[30].

The core of the problem is the generation of 'good' schedules, which ideally take all quality-of-service requirements from all involved parties into account. Every developer of a scheduler tries to fulfil this fundamental objective. In an ideal environment, where all necessary data to calculate a scheduler is accessible and processable, and where the time to generate the schedule is negligible, it would be possible to operate based on the optimal schedule at every point in time. In reality, a large number of side conditions prevent this. For distributed computing infrastructures as we consider them in this work, such conditions arise from the general challenges, as introduced in Section 2.1.3, as well as from technological restrictions. Dealing with scalability, for example, is not trivial, since potentially millions of services form the target set of a scheduler's objective function. Taking all target values into account generating a schedule may lead to hour-long calculations, an

[28] Complexity results for scheduling problems, last visited: January 25, 2013. http://www.informatik.uni-osnabrueck.de/knust/class/.

[29] Please refer to Chapter 7 for a detailed introduction to SLAs.

[30] Please note that the we deliberately omitted, in comparison to the original definition, the term 'limited' in relation to services since, especially in public clouds, services may not constitute scarce resources. The application of scheduling is nevertheless of benefit as quality-of-service parameters like cost can be optimised.

approach which is not feasible in a DCI where requests have to be processed in a timeframe of seconds.

Taking the side conditions into account, it is not enough to develop sophisticated scheduling algorithms, but the scheduling problem then translates into a larger challenge, namely the design of an automated scheduling process and scheduling architecture supporting it. Tackling this and proposing a solution beyond specialised scheduling systems and application area-specific algorithms is the core objective of this work. This approach introduces, in addition to researching scheduling algorithms, further problems that we address and for which we propose solutions. These 'inferred' problems to be solved are:

- How to describe the activity to be scheduled?
- How to describe the capabilities of a DCI system?
- How to describe the quality-of-service criteria to be considered when scheduling?
- How to optimise the interaction of the various strategies forming the scheduling process?

The approach that presents a solution to these problems and that answers the question of 'how to design an automated scheduling process and an architecture supporting it' is outlined in Chapter 4 'Concept'. There, also questions like 'how the scheduling service interacts with other DCI middleware services', 'how dispersed information related to one activity is handled', or 'how the scheduling process is actually executed' are discussed and solutions are proposed.

2.2.2. A Note on Benefits of the Application of SLAs to Scheduling in DCIs

One means to support scheduling in distributed-scheduling infrastructures is its tight integration with a service-level management framework. Our work is based on such an approach, integrating scheduler, service-level agreements, and auxiliary services into a generic scheduling architecture.

Although anticipating some of the details of the following chapters, we introduce here, in relation to the aforementioned challenges, benefits of such an approach. We do this without discussing the requirements towards scheduling beyond the previous paragraph and without presenting SLAs in detail. Therefore, the following list shows the general advantages of the solution that will evolve in the course of the upcoming chapters:

- Service-level agreements can carry a rich set of parameters that describe services and resources while linking those parameters directly with quality-of-service requirements. This allows (i) the modelling of all kinds of services and resources, (ii) the inclusion of related service-level capabilities and constraints, and (iii) the application of sophisticated algorithms based on a large pool of input parameters.
- The chosen approach minimises the implications put on scheduling algorithm development by treating them as black boxes. Therefore multiple algorithms can be implemented and applied.

- The autonomy of DCI sites is not compromised and requirements on authentication, authorisation, et cetera can be easily integrated as quality-of-service parameters.
- The reservation of services or resources can be represented through time-related QoS and can be backed up through guarantees. This allows, inter alia, the provision of third-party reservation services.
- Administrative or service provisioning hierarchies can be modelled as SLA hierarchies thus relieving the schedulers from maintaining their scheduling-specific topology information.
- SLA relations can also be used to orchestrate services thus modelling work-flows or process chains.
- Data is, in the light of SLAs, handled as any other resource. This allows the handling of data (and the related locality issues) equally to any other resource within the DCI and thus for scheduling.
- Service-level agreements can be treated as legally-valid contracts giving users proper guarantees with respect to their QoS requirements. In cases where contracts are not necessary, SLAs can be used without any legal implications.
- The extension of the QoS parameter set, for example to implement further scheduling algorithms, can be modelled through SLA extensions.

This list provides an overview of the overall advantage of applying service-level agreements to scheduling in DCIs although a number of assumptions on models and implementations have been implicitly made. The last item, for example, will only be valid if the SLA model is designed in an extensible manner. We will show later what theses assumptions are and how schedulers can be implemented to benefit from SLAs.

In this chapter we motivated the work we conducted and defined the problem we try to solve. Furthermore, we provided, especially for those readers not familiar with distributed computing infrastructures, a brief historical overview to introduce the topical area.

Part II.

Requirements Gathering and Conception

3. Requirements

The purpose of this chapter is to summarise the requirements faced by researchers as well as middleware architects and implementers who deal with the management of distributed computing infrastructures. We therefore explain a couple of application scenarios from which we derive common requirements and then we link them to the respective middleware artefacts.

In the previous chapter we listed the common challenges that have to be tackled when dealing with distributed infrastructures. As this thesis deals with one particular aspect of such infrastructures and their operation, namely scheduling, we look in this chapter at the requirements that can be derived from respective application scenarios. Therefore we describe three cases where scheduling is the central issue, namely (i) the scheduling of complex work-flows, (ii) co-allocation of activities, and (iii) the delegation of scheduling requests. In doing so, we follow a common structure to simplify the induction of general requirements to be fulfilled by middleware managing DCIs. We describe these requirements in great detail to lay the foundations for the concept outlined in the following chapter.

3.1. Selected Scheduling Scenarios

In distributed computing we find a wide field of research topics: models [78], algorithms [40], services [141], and systems [10]. Accordingly, a large number of application scenarios and related requirements exist in this area. We have already collected some of them focussing on the generic aspects of grid scheduling [149]. In this chapter, we re-visit two of these scenarios, ‘scheduling of complex work-flows’ and ‘co-allocation of computational activities’, and add a third one that reflects requirements from cloud-like infrastructures, the delegation of activities from one DCI to another.

3.1.1. Application Scenario I – Scheduling Complex Work-Flows

Many distributed applications require the coordinated processing of complex work-flows, which includes scheduling of heterogeneous resources within different administrative domains. A typical scenario is the scheduling of computational resources in conjunction with data, storage, network, and other available resources as for example software licenses or experimental devices. Such scenarios are frequently found in grids and inter-cloud set-ups where scientists execute complex work-flows as part of experiments. In such cases, the scheduler has to coordinate and plan the execution of the whole work-flow in advance. While at the same time, issues like cost, resource reservation, and data staging have to be taken into account.

Scenario

A typical scenario includes scheduling of a distributed computational job including network, data, software, and storage combined with the request for a visualisation service. The user might, in terms of concrete requirements, request (i) 48 processing nodes of a specific type, (ii) 8 GB of available memory per node, and (iii) a specific licensed software package for the time of one hour between 8am and 6pm of the following day. In addition, visualisation requires a minimum dedicated bandwidth of half a GBit/s between the visualisation device and the main system during program execution. Furthermore

- the program relies on a specific input data-set from a data repository;
- the user's budget is restricted;
- they prefers a lower price to an earlier execution.

A scheduler should be able to generate the complete schedule for the execution of this work-flow including all activities implicitly necessary for data management before and after the actual start. Fig. 3.1 shows an example of a possible scheduling output.

This example, however, is quite basic for the sake of explanation. In real applications it could easily be extended to contain additional work-flow steps. The scheduler should take the allocation of all requested resources into account and, if necessary, create advance reservations.

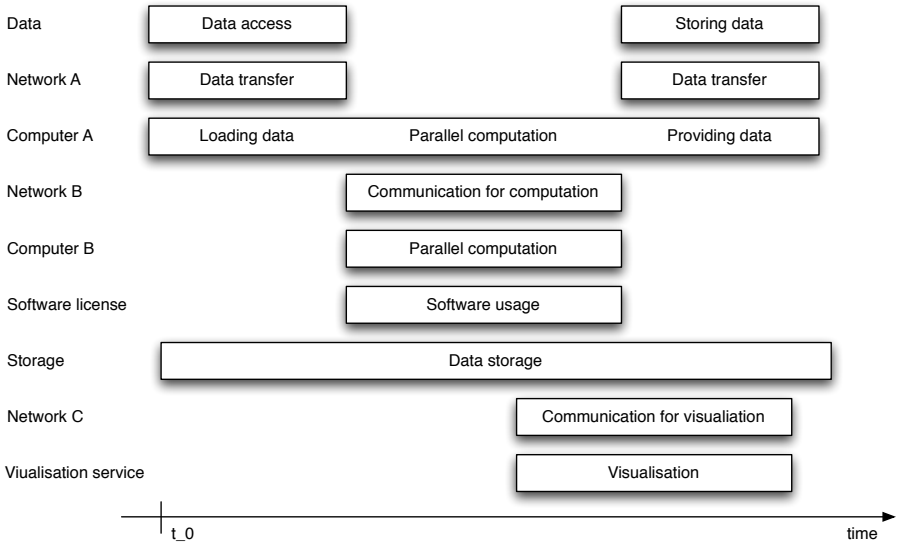

Figure 3.1.: Example schedule for work-flow scenario

This is essential to meet time or precedence requirements. The respective resources ideally should be reserved in advance in order to guarantee the proper execution of the whole work-flow. It is equally important to cope dynamically with potential changes to the schedule including failures, like the unexpected unavailability of a service or resource. Therefore, support for potential re-scheduling of work-flows is mandatory.

Sequence

The following sequence of events shows the scheduling of complex work-flows. The information in brackets refers to the services that are involved in the execution of the respective step and which are introduced later in the ‘Services’ section of this chapter.

1. *Composition and submission of the work-flow request*

   The work-flow description is generated and transferred to an entity capable of processing its contents. Following the example, a work-flow will be generated that contains the service and resource requirements and constraints. With respect to this scenario no specific language to describe the work-flow request is demanded (Services 1 and 2).

2. *Pre-processing of the work-flow request*

   The work-flow request has to be parsed and validated if possible. If the entity preprocessing the work-flow is unable to do so it may try to translate the work-flow into a suitable description (Services 2 and 5).

3. *Gathering of static resource information*

   Some service is needed which gathers static information about existing services and resources. This service may be an information service or a database. As far as the example scenario is concerned, it is assumed that this processing step identifies a pool of 800 services and resources of all requested kinds (Services 3, 4, 8 and 9).

4. *Pre-selection of resources*

   Based on the information collected in Step 3, algorithms are applied to limit the number of resources which are potentially capable of participating in the processing of the work-flow. In our example this may cut resource candidates down to 30, since some systems may not have 48 processors, may not offer the software requested, or the respective system may be maintained the following day (Service 3).

5. *Query of dynamic resource information*

   The dynamic query delivers information like whether the current load of the machine allows allocating 48 processors. This is different from Step 4, where resources are sorted out because they offer less than 48 processors. Through this, the number of potential resources, which are actually used in the next step to process the schedule, is further limited (Services 3, 4, 8 and 9).

6. *Generation of the schedule and initialization of required reservations*

   Based on the resource information gathered in the previous steps, a schedule is generated, as shown in Fig. 3.1. It is then attempted to reserve the necessary resources in advance, a process which may fail several times due to the complexity of the work-flow and the number of dependencies between the reservations needed. Failure of negotiation may lead to re-scheduling, possibly with a preceding Step 5 (Services 2 and 6).

7. *Execution of work-flow*

   Once the schedule as shown in Fig. 3.1 is confirmed, it is processed and executed. In case of the example, data is first taken from some storage system and transferred via network A to computer A. If no error occurs, the work-flow is executed until the last chunk of resulting data is written via network A to storage (Services 2 and 7).

8. *Completion of work-flow*

   This includes the finalisation of monitoring and reporting as well as the delivery of the data the work-flow produced (Services 1, 2, 8 and 9).

Functional Requirements

The evaluation of the scenario reveals certain functional requirements which the DCI middleware services have to implement.

- *Authentication, authorization, user right delegation and work-flow integrity verification*

  Authentication and authorization are essential services for a work-flow-processing DCI middleware. To enable the scheduler to act on behalf of the user, the respective rights have to be delegated from the user to the scheduler. This scenario also requires that the integrity of a work-flow, or parts of the work-flow, can be verified any time during the scheduling process.

- *Work-flow parsing and validation*

  The work-flow description has to be parsed and formally validated.

- *Retrieval of static and dynamic information*

  To map the resource requests which are contained in the work-flow description onto available resources, information about the resources and their status has to be retrieved from appropriate entities (and offered by these entities). It should be possible to gather static and dynamic resource information separately in order to restrict the time-consuming dynamic information retrieval.

- *Resource pre-selection*

  To avoid information queries on resources which do not fulfil policy constraints defined by the user or which are definitely not capable of fulfilling a resource request, a set of resources should be selected based on the so-called ‘static’ resource information.

- *Service choreography and management*

  It might be useful to have mechanisms which allow choreography and management of services representing the pre-selected resources on different levels to obtain the desired dynamic information faster and more reliable.

- *Scheduling*

  A schedule has to be generated based on the information about the work-flow, the resources, the services, the accounts, etc.

- *Agreement negotiation and advance reservation*

  It is essential to meet time or precedence requirements defined by the work-flow. Therefore resource availability has to be negotiated with schedulers or reservation services to reserve in advance the resources selected by the schedule.

- *Workflow execution/processing*

  The work-flow has to be processed. It is assumed that the local resource managers execute the atomic entities a work-flow consists of.

- *Monitoring*

  Information has to be collected about the status of a work-flow during its lifetime and the resources needed to execute it.

- *Assessment and reporting*

  The service rendered has to be assessed and reported according to the service-level negotiated and the service quality actually delivered. Cost of the service delivery, penalties in case of non-performance or -compliance, and audits are based on these functions.

- *Failure management*

  Failure management is essential not only to have an instrument to monitor and possibly re-schedule work-flows in case of failure within the system, but also to provide users with information and tools to manage such failure.

Services

The following services and functionalities are required for the scheduling[1]:

1. Submission service or agent acting on-behalf of a user (steps 1 and 8)
2. Scheduling and resource management service (steps 1, 2, 3, 6, 7 and 8)
3. Brokering service (steps 3, 4 and 5)
4. Information service (steps 3 and 5)
5. Translation service (step 2)
6. Negotiation service (step 6)
7. Work-flow execution service (step 7)
8. Monitoring service (steps 3, 5 and 8)
9. Reporting service (steps 3, 5 and 8)

[1] Related steps of the work-flow precessing sequence, as described in the ‘Sequence’ section, are listed in brackets.

3.1.2. Application Scenario II – Co-allocation of Computational Activities

A common demand for high-performance applications is the co-allocation of multiple HPC systems to obtain better performance through scaling applications onto larger DCIs. The VIOLA (Vertically Integrated Optical Testbed for Large Applications) project has hosted a number of such applications aiming at the development of a meta-scheduling framework to co-allocate distributed computers and optical networks [43].

Scenario

The applications of interest are parallel and distributed simulations from the multi-physics area the different parts of which are executed on different computational resources connected through an optical network. The user outlines her activity together with related resource requirements. In order to run such a simulation, both compute resources and the interconnecting network must be available at execution time. Furthermore, users want to have a certain service quality, specifying terms like minimal bandwidth or delay. For co-allocation a *meta-scheduling service* is necessary which interacts with both the DCI middleware and all local *resource management and scheduling systems* (RMS). The service is also responsible for negotiating a common time-slot with all local RMS to guarantee the co-allocation of their resources. Once such a common time-slot is identified, resources are reserved at all sites. Subsequently, the parallel applications can be executed, managed, and monitored. Fig. 3.2 shows an example of an application which requires two computational site and the interconnecting network.

Sequence

The following sequence of events reflects the co-allocation process [141]. The information in brackets refers to the services that are involved in the execution of the respective step and which are introduced later in the ‘Services’ section of this chapter.

1. *Composition and submission of application description*

   After the user has defined the description and resource requirements of the application, they send them to the meta-scheduling service explicitly specifying the requested HPC resources (Services 1 and 2).

2. *Resource preview and pre-selection*

   The meta-scheduling service queries the local RMS to get the earliest time the requested resources will be available. Following that, the RMS generates resource availability previews, which comprise a list of time-frames during which the requested QoS (e.g. a fixed number of nodes) can be provided. It is possible that the preview contains only one entry or even zero entries if the resource is fully booked within the requested time-frame (Services 3, 4, and 6).

3. *Generation of schedule and reservation of resources*

   Based on the preview the meta-scheduling service calculates the possible start time. If the individual start times do not allow the co-allocation of the resources, the meta-scheduling service will use the latest possible start time as the earliest start time for

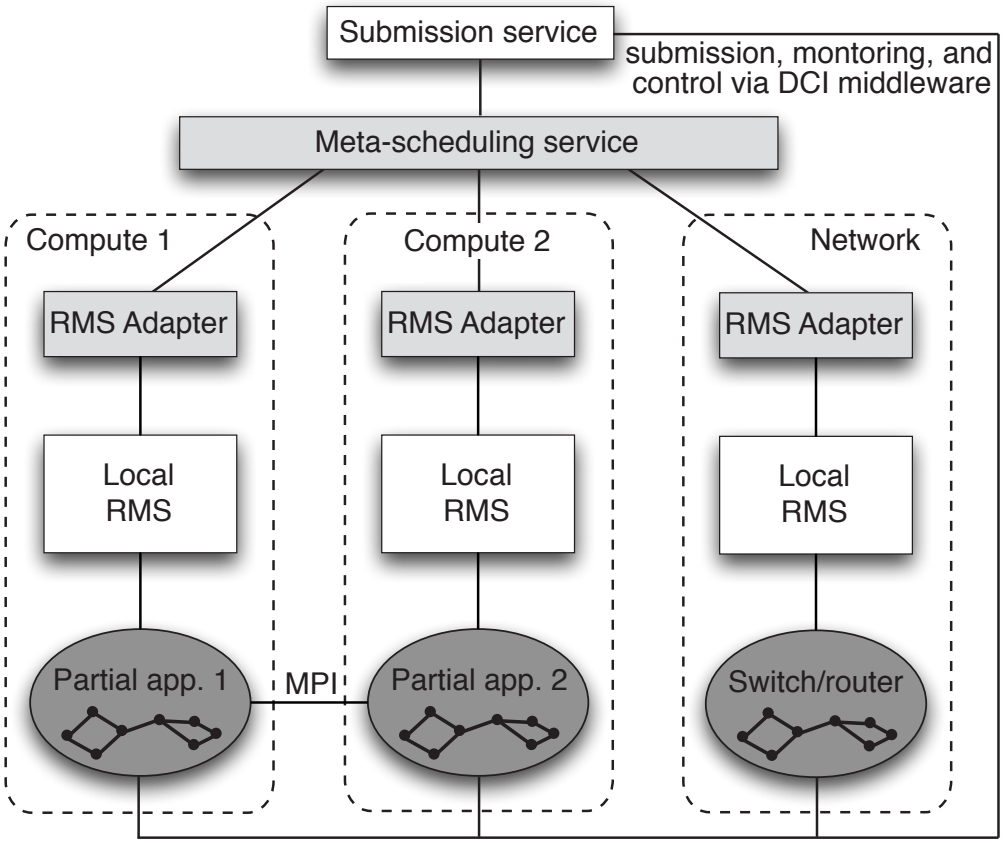

Figure 3.2.: The Co-allocation of two compute resources and a network switch (modified version based on [42])

a next scheduling iteration. The process is repeated from step 1 until a common time-frame is found or the end of the preview period for all the local systems is reached. The latter case generates an error condition (Service 2).

4. *Cross-checking of start times*

   In case the individual start times match, the meta-scheduling service checks the scheduled start times for each reservation by asking the local schedulers for the properties of the reservation. This step is necessary because new reservations may have been submitted in the meantime by other users or processes, thus preventing the reservation of resources at the requested time. If one or more reservations are invalid, all reservations will be cancelled. The latest effective start time of all reservations will be used as the earliest start time to repeat the process beginning with step 1. If all reservations are scheduled for the appropriate time, the co-allocation of the resources will be completed (Service 2 and 6).

5. *Finalisation of scheduling*

   The meta-scheduling service passes the co-allocation information back to the client

(Services 1 and 2).

6. *Co-allocation and execution of the application*

   Once the scheduled start time is due, all reserved resources (including the network) will be co-allocated and the application can be executed (Service 6).

7. *Completion of application*

   The final step includes reporting about the resources consumed and the delivery of the application output to the user (Services 5, 6, and 1).

Functional Requirements

Looking at the sequence of actions that have to be executed in this scenario, we recognise the following functional requirements:

- *Information retrieval*

  The meta-scheduling service needs information about the current and the future state of reservations for all local resource management and scheduling systems that offer their resources.

- *Negotiation*

  The meta-scheduling service must have functions to negotiate with the different RMS about the quality-of-service (i.e. the start time of the application).

- *Co-allocation*

  The scenario requires the automation of co-allocation of compute and network resources. Based on the high-level application requirements the users define, the middleware has to deal transparently with low-level scheduling issues of the co-allocation process.

- *Resource reservation*

  The requested resources must be reserved for their future allocation, a function also called *advance reservation*.

- *Support for service-level agreements*

  It is necessary that the middleware services follow an SLA model and expose interfaces that enable users to negotiate and reach an understanding on service-level agreements.

- *Monitoring and failure management*

  A monitoring infrastructure is essential to recognise failures and to gather data needed for SLA compliance assessment.

Services

The list of services that are needed to realise the above mentioned functions is quite similar to those given for the work-flow example. We also assume here that an execution environment exists to run the application. In case of IANOS [113], the most advanced implementation of this scenario, the UNICORE middleware is the respective execution environment integrated with the services listed below[2]:

1. Submission service or agent acting on-behalf of a user (steps 1 and 7)
2. Meta-scheduling service (steps 1, 3, 4, and 5)
3. Brokering service (step 2)
4. Information service (step 2)
5. Monitoring service (step 7)
6. RMS adapter (steps 2, 4, 6, and 7)

The *RMS adapter* is the interface between the meta-scheduling service and the local RMS. It hides the specifics of the RMS and enables co-allocation of virtually any kind of resource. The only potential limitation can be imposed by the local RMS or their site policy: if they do not publish at least the next possible start time or do not support advance reservation, co-allocation will not be possible.

3.1.3. Application Scenario III – Delegation of Scheduling Requests

The development of scheduling and resource management for grid infrastructures has long been influenced by meta-schedulers (also called super-schedulers or simply grid schedulers). The majority of them have one thing in common: the meta-scheduler constitutes a ‘higher-level’ scheduler which interfaces ‘lower-level’ schedulers or local RMS and passes scheduling decisions [132] to them. Another approach is the delegation of scheduling requests from one scheduler to another, inter alia proposed by the the German DGSI project[3]. Especially with clouds providing ‘pay-as-you-go’ compute resources, this scenario is an alternative worth evaluating.

Scenario

In this scenario, which is depicted in Fig. 3.3, a client sends an activity request to the *primary scheduler*. It contains all information the scheduler needs to calculate a schedule and to hand over the activity to an execution service. Upon receipt, the primary scheduler takes the activity request and, if it is willing to handle it, creates an activity instance for it, storing the initial request and, if applicable, additional information. The latter should at least include a ‘provenance record’ which denotes that the current scheduler has taken

[2] Related steps of the work-flow precessing sequence, as described in the ‘Sequence’ section, are listed in brackets.

[3] D-Grid Scheduler Interoperability, last visited March 1, 2011: http://www.d-grid-ggmbh.de/index.php?id=98&L=1.

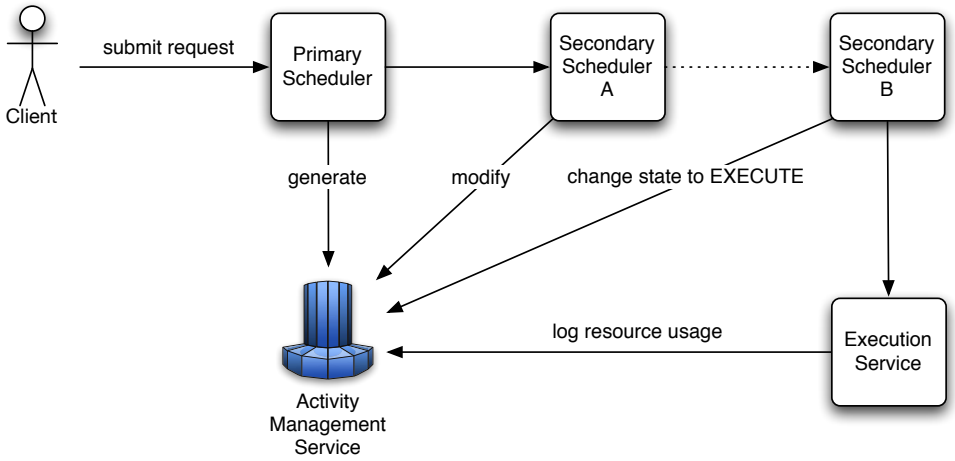

Figure 3.3.: Delegation of scheduling requests

over responsibility for the execution of the given activity. Other candidate information are scheduling attributes, dependencies on other activities, and the current state of the activity.

The scheduler decides, for reasons not relevant here, that it cannot fulfil the request and delegates it to another, *secondary scheduler*. In this case, the delegator acts like a client towards the potential delegatee. Again, if the delegatee is willing to accept the job, it takes over responsibility and the provenance records and depending information (e.g. the expected execution service) are updated accordingly. If necessary, the activity request is modified, and the manipulation history is kept. Such modifications might include, as depicted in Fig. 3.3, whether a secondary scheduler rejects or accepts a delegation request, the state transition of the activity, or the resources consumed.

Throughout the whole process, state information is constantly updated in the activity instance. After activity completion, the resource consumption is written to the activity instance. The corresponding entries and dependent parts of the activity instance could then be marked final to denote the completion of the activity.

Sequence

The sequence of messages passed between the different services in the scenario is depicted in Fig. 3.4. The information in brackets refers to the services that are involved in the execution of the respective step and which are introduced later in the ‘Services’ section of this chapte

1. *Composition and submission of the activity*

   A client submits an activity request to the primary scheduler. The request should include all information that is needed to make a scheduling decision and to execute the activity afterwards (Services 1 and 2).

2. *Activity creation*

   The primary scheduler informs the activity management service about the activity instance following the acceptance of the activity request (Services 2 and 5).

3. *Scheduling and delegation*

   The primary scheduler cannot schedule the request and therefore delegates it to the secondary scheduler A which rejects it and updates the activity instance respectively. The primary scheduler then initiates another delegation attempt addressing secondary scheduler B[4]. The secondary scheduler B informs the activity management service about the acceptance of the delegation request (Services 2, 3, 4, and 5).

4. *Execution*

   The secondary scheduler B notifies the activity management service about the handover of the activity to the execution service. This service updates the activity instance according to the state of the activity execution, informs it about the resources used, et cetera (Services 3 and 5).

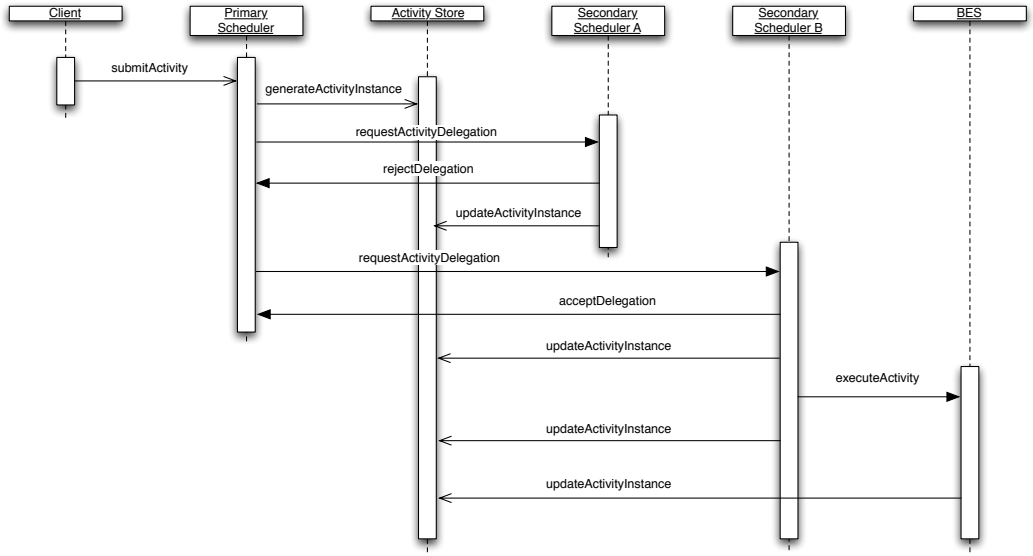

Figure 3.4.: Sequence diagram for the delegation of activities

The scenario shows, for the sake of simplicity, only the steps until the activity is executed. Further steps that occur during the processing of the activity, like feed-back of results, are not shown, nor are potential activity monitoring or assessment steps presented.

Functional Requirements

- *Information retrieval*

[4]The arrow for this delegation attempt originates only for visual reasons from secondary scheduler B.

3. Requirements

The primary scheduler needs information about the capabilities of other schedulers to prevent delegation negotiations with secondary schedulers that are not capable of accepting the request anyway.

- *Scheduling*

  Obviously this scenario features schedulers. An additional functional requirement in comparison to the other two uses cases is the demand for a common format to exchange the scheduling requests.

- *Activity management*

  A service or a framework of services is needed which handles the activity instances. They have to support at least the CRUD set of operations (cf. Section 9 for one realisation of this functional requirements).

- *Negotiation*

  The schedulers must support negotiations to get to an agreement regarding the delegation. This includes the ability to negotiate the conditions of the activity delegation like price or time. The same interfaces should be offered to the client to negotiate the initial request.

- *Monitoring*

  It is essential to have monitoring information about the underlying infrastructure to make proper scheduling decisions. Another reason for this function is the activity management, which, as we will point out later (cf. Section 5), aggregates all information related to a specific activity.

Services

For this scenario, the execution environment and other activity information sources than those depicted in Fig. 3.4 are not in our focus and are therefore not included in the following list of services[5]:

1. Submission service or agent acting on-behalf of a user (step 1)
2. Primary scheduler (steps 1, 2, and 3)
3. Secondary scheduler(s) (steps 3 and 4)
4. Information service (step 3)
5. Activity management service(s) (steps 2, 3, and 4)

[5] Related steps of the work-flow precessing sequence, as described in the 'Sequence' section, are listed in brackets.

3.2. Analysis of the Requirements of the Application Scenarios

The three scenarios include several common requirements. Obviously, scheduling is a central aspect of each case, but this was the main characteristic for their selection anyway. What we seek are common requirements that have to be considered by developers, architects as well as implementers, of service-oriented distributed infrastructures.

The following set of requirements has been identified:

- <**REQ-1**> **Discovery of services and resources**: For all three scenarios it is necessary that the schedulers can discover other entities, them being services or resources, which fulfil specific constraints or fit certain parameters.

- <**REQ-2**> **Access to service and resource information**: In addition to discovering entities, a common requirement is access to up-to-date service or resource information. As there is no clear distinction between 'static' and 'dynamic' information, we postulate that during a *pre-selection* phase all static information about a resource or service is provided and that dynamic information is requested in cases where a service has to make decisions based upon data which may have changed since the previous decision. Examples for static information are the resource category or the maximum amount of nodes a high-performance computing user can request; examples for dynamic information include the current load of a system or potential start time for an activity.

- <**REQ-3**> **Brokering**: Although brokering and scheduling are often equated with each other, we differentiate between these two requirements: a scheduling service decides where and when to execute an activity, whereas the brokering service preselects suitable services or resources that are candidates for the actual scheduling process. This feature is explicitly required by the first two scenarios and it is implicit in the 'Scheduling and delegation' step of the third one. There the primary scheduler needs a priori information about potential secondary schedulers to contact only those which are, based on their static information, capable of scheduling the request.

- <**REQ-4**> **Scheduling**: It is essential for distributed computing infrastructures that a user does not have to manually coordinate access to service. To this end, efficient resource management functions are required which automatically schedule activities, work-flows, or applications. This item also captures the demand for service orchestration, co-allocation, and delegation, all of which we see as particular forms of scheduling.

- <**REQ-5**> **Advance reservation of services and resources**: The first two scenarios mention advance reservation as an important requirement to realise work-flow scheduling as well as co-allocation. With respect to the third case, it is not relevant in what way the individual scheduler actually implements the schedule. Therefore advance reservation is not required, but it is not explicitly excluded either.

- <**REQ-6**> **Service-level agreement management**: Although this function is only required explicitly for the co-allocation scenario, all of them implicitly carry the need

for handling quality-of-service requirements. This becomes primarily evident because of the demand for negotiating the condition of either activity delegation or resource reservation. And, anticipating that almost all DCIs will be service-oriented infrastructures in the future, we foresee the necessity of SLA-driven design principles to increase. Failure management, as referred to in the first scenario, is included here.

- <**REQ-7**> **(SLA) negotiation**: All three scenarios include the requirement of negotiation. This translates to SLA negotiation under the assumption of service-level management to be a core part of future DCI management infrastructures.

- <**REQ-8**> **Activity management**: This requirement is only supported by scenario number three, we will reason later, however, why the addition of activity management to DCIs makes sense in general.

- <**REQ-9**> **Monitoring**: Prior to and during the execution of an activity it is necessary to monitor the state of services and resources. In the specific case of the third scenario, monitoring is central to the whole scenario. There, the activity is created once a user request is received and it is continuously updated with monitoring information[6].

3.2.1. Non-Requirements

Apart from the selected requirements, some of those mentioned in the scenarios do not represent core aspects of this thesis. These 'non-requirements' are:

- **Composition and submission of execution requests**: In the DCI landscape a number of clients, portals, and APIs exist which comprise functions like the specification of execution requests (also known as tasks, jobs, processes, activities, applications, work-flows, or alike) and the submission of such requests. Some are generic, other are domain-specific, maybe even tailored to one specific application. Common to all of them is that they rely on middleware interfaces and the request model prescribed by the middleware. As we focus on the middleware, its services, and interfaces, such client-side entities are of no interest to us.

- **Provisioning and execution**: Any distributed computing infrastructure, whether it is an HPC system, a grid, or a cloud, provides means to provision the services and resources requested by a user. It also needs execution services to fulfil the users' requests. Undeniably, there are plenty of requirements in this area to increase performance, manageability, or usability, but in the light of our application scenarios, provisioning and execution capabilities are taken for granted. Briefly, we are interested in the interfaces and data models but treat the services as black boxes.

- **Assessment and reporting**: The pay-as-you-go business model, which is one central pillar of cloud-like DCIs, gives prominence to proper service-level assessment, reporting, billing, and auditing. Although this is undeniably a very important feature of today's distributed infrastructures, we see these as 'added-value services' that deserve significant research effort, but are not of importance for scheduling itself.

[6] For an in-depth introduction to the notion of an activity we refer to Section 5.

We will show, however, that our contributions will help to increase service quality in this area, too.

- **Security**: Security aspects are core to almost any DCI application: it is essential, for example, to authenticate and authorise users and keep the integrity of data. To us, this is reflected in security-related service requirements and QoS demands. Such items are only part of the 'payload' of activities and service-level agreements, but not central to our research.

- **Pre-processing/validation of execution requests**: We consider pre-processing of execution as a function inherent to any processing entity in a DCI middleware. At least a syntactical check of a job or work-flow description is necessary to grant further processing. Since a multitude of today's execution requests are specified in XML, such a basic validation is executed automatically by the respective parser.

3.3. Implications for Distributed Computing Middleware

Following the previous brief summary of requirements, we now reflect on the implications of actually implementing services which fulfil them. Based on a previous evaluation which merely covers grids [122], we now execute this analysis with service-oriented infrastructures as the target environment and a maximum of generality as our main objective.

Instead of identifying a minimal subset of functions, which would probably render the resulting middleware inadequate for most application scenarios, we carefully identify the relevant set of requirements to make the services which implement them usable for multiple scenarios. On the other hand, this function set should not be too large to be implemented into a reliable and robust middleware. Doing the splits between specifics and generality is a difficult task; our respective solution is presented in Chapter 4. Whether or not it is commonly applicable should be subject of further discussion within the DCI community.

In the following paragraphs we adhere to the nine requirements the previous section and link them to middleware services which implement them.

Registry for Services and Resources

The service which lets users or other services discover entities within a DCI is called *registry*. We assume that in a first step it is necessary to identify services and resources that are principally available. The middleware must offer a fast mechanism to identify the entities which fit a given description. The actual selection or scheduling will be based on additional information retrieval or negotiation, a process which might require several steps (see also 'brokering' and 'scheduling service', as well as 'negotiator' below).

While for enterprises an index or directory information service might be feasible, it becomes a more complex problem to identify which services and resources are currently available in a large-scale infrastructure. For future DCIs, a flexible and scalable discovery service is required. There are technologies, e.g. from the peer-to-peer area, that work well for large-scale systems. For smaller environments, however, other approaches, like the mentioned index and directory services, may be more efficient. A coherent or standard interface which supports both approaches is desired.

Nevertheless, it is important to identify entities which meet the requirements of an activity in the best possible way. In general, it is not sufficient to discover all the entities of a specific type as the result set might be too large and thus not useful for processing. This implies that models for a flexibly parametrised search are required to pre-select a prioritized and small set of 'good' resources. Therefore, a flexible methodology is necessary to limit and steer the search towards the anticipated results. For resource brokerage or scheduling, it might be necessary to re-iterate and modify the search criteria in order to improve the list of suitable entities.

Information service

Besides a registry containing basic data about the entities in a DCI, it is necessary to have additional information about resources, some of which is static or at least valid for a relatively long time. Such information may be suitable for caching or storing in remote information services. Other information is highly dynamic and therefore not suitable for deferral to remote services. Independent of such characteristics, this information is provided by an *information service*.

Therefore, middleware should provide coherent access to this static and dynamic information. It also has to be considered that it may not only consist of simple key-value pairs, but that more complex and potentially time-variant data collections might as well be needed. A typical example here is data about planned or predicted future events, like resource reservations or the predicted load of a compute system.

Similarly, access to information about past events might be needed. Some systems may utilize such historic information to make predictions about future events. Nevertheless, it has to be acknowledged that access to this data is important for many application scenarios and for many scheduling and brokering services. Most of current information services do not support a rich and extensible information model. Moreover, many existing services are not suitable for a scalable scenario with many entities and corresponding information sources.

To summarise: a coherent interface to infrastructure information is essential for any DCI. It should support caching whenever possible, e.g. if the information does not change for a sufficiently long period. Furthermore, it should include mechanisms and paradigms to retrieve dynamic information. In any case, the unified information service should not be limited to specific resource types or kinds of services, but it has to be designed in a generic and extensible manner. Data, network, and software resources and their corresponding management services are, for instance, often and probably unnecessarily treated differently. Thus, a more flexible and generic information service could decrease the complexity of building DCI middleware.

Brokering service

It is essential to have flexible and automatic brokering and scheduling services. Because of the large number of potential services and resources comprising a DCI and because of the heterogeneity of access policies with different providers, the user cannot effectively execute the selection of services and resource needed for an activity in a manual fashion.

A middleware service that helps to pre-select suitable services and resources is called *broker*. Its purpose is to reduce the set of entities worth considering and hand over a

candidate selection to the scheduling service. This is best done by using only a small amount of data like service type. Typical requests sent from a broker to a registry are 'give me the EPR[7] of all services that have the capability to negotiate about the usage of compute resources' or 'list all storage resources that offer at least 300 TByte of space'.

Scheduling service

First, we have to consider that scheduling implies an activity description that includes all data needed to properly match user requirements to service and resource capabilities. While many current distributed computing infrastructures still deal with simple jobs [70], we envisage increasing demand for complex work-flow execution, business processing, and HPC applications. Therefore all services involved in the processing of such activities will have to be able to cope with them, including the scheduling service.

Due to the different access and scheduling policies that might exist with a DCI, we cannot expect that a single scheduling strategy will suit all needs. Instead, it must be taken into account that a DCI most likely has many different schedulers with varying features and strategies which are optimized for certain applications. Therefore, it is be essential for scheduling services to offer well-defined interfaces and protocols to all the functional entities and all the information a scheduler needs. Such a scheduling architecture, including the capability to exchange scheduling requests between services, will facilitate the implementation of different application-specific schedulers [132].

Considering the different types of resources, it is important that all of them can also be included in the brokering and scheduling processes. Especially, data and network resources are often associated with dedicated management and information services [9, 118]. For instance, access to planned data transfers or network reservations are important to co-allocation scenarios like the one presented in Section 3.1. Current approaches, however, do usually not provide adequate means to integrate such resources into scheduling, although the management tasks are very similar to those of other resources. Here, consistent and coherent interfaces are vital to integrate any kind of resource and service into the management of a DCI. Ideally, an activity should be able to request data resources and network links from a scheduler in the same fashion as it requests software components or compute power.

It is also critical that different scheduling instances can interact with each other, a requirement supported by the 'delegation of scheduling requests' scenario. One approach is to support such interactions through the provision of negotiation interfaces which permit the creation of agreements between scheduling services. Due to the complexity of the scheduling task in a large-scale DCI, such negotiations can take a long time as many iterations of probably parallel negotiations with different other providers might be necessary. Therefore, any access to information that can support this procedure will be helpful. This can include tentative information about availability, potential quality of service, or costs.

Advance reservation service

As stated above, all three scenarios require the reservation of resources in advance. This is essential to guarantee their availability once the environment for a particular activity

[7] An *end-point reference* provides means to contact a service. A common way to format EPRs is a *Uniform Resource Identifier*: http://tools.ietf.org/html/rfc3986 (last visited March 3, 2011).

has to be deployed. In case of compute resources this feature is already included in some of the typical batch systems like LSF[8] or PBS[9], but many services that manage other kind of resources, like storage or data, do not provide this kind of function.

Clouds, which are the latest technology base for DCIs, are considered to be 'virtually inexhaustible' resources. This, however, is only true for their public representatives and only for the current business model. Private cloud-based or even public computing infrastructures, once they properly support the HPC business model and funding agencies get involved in it, will also experience resource shortage. In such cases 'best-effort' approaches will not work effectively, and scheduling based on advance reservation will be necessary.

Service-level agreement framework

While some usage scenarios can get by with simple job submission paradigms, others have more extensive requirements or need even guarantees for certain levels of service. An example for the former are Globus grid jobs, like those submitted via the globus-job-run command[10], which mainly allows to specify the executable, the files to be staged in and out, the number of processes, and the maximum runtime. In contrast to such an HPC type of activity, our three scenarios include requirements like guaranteeing the time of execution or negotiating certain QoS parameters between services.

There are other such applications [2, 12]. And the increase in service-orientated concepts also mandates the inclusion of service-level management principles into the design of DCIs. We therefore foresee that service-level agreements will become an integral part of DCIs. In such an agreement, which is created before the actual activity execution and service provisioning, all necessary quality-of-service parameters are specified. It can include both the simple job submission to a remote queuing system and actual execution time constraints.

Negotiator

A *negotiator*, or negotiation service as it is also called, is a middleware component that is needed to execute the process of SLA negotiation. One instantiation is needed on the service consumer as well as on the service provider side. Governed by their respective business objectives, they negotiate about the quality-of-service, and the rights and responsibilities of both parties. The common goal is to reach an agreement, i.e. to establish a contract that contains the specifics of service consumption and delivery.

Because of the scale of DCIs (the European Grid Initiative EGI.eu has 10.000 users, for example, and operates around 243.000 compute cores [11]) negotiation with manual intervention is not possible. DCIs therefore need negotiation services that can, potentially within the scope of a framework contract, automatically agree on the terms of service

[8] Platform LSF – The HPC Workload Management Standard, last visited: January 25, 2013. http://www.platform.com/workload-management/high-performance-computing.

[9] PBS Professional, last visited: January 25, 2013. http://www.pbsworks.com/Product.aspx?id=1.

[10] globus-job-run – Execute a job using GRAM, last visited: January 25, 2013. http://www.globus.org/toolkit/docs/5.0/5.0.3/execution/gram5/pi/#gram5-cmd-globus-job-run.

[11] Steven Newhouse, The European Grid Infrastructure, last visited: January 25, 2013. http://www.egi.eu/export/sites/egi/results/presentations/EGI-AstroPP1.pdf.

delivery. Moreover, these services need to support various business objectives leading to different negotiation strategies. Hence, support for this kind of flexibility is of great importance.

Activity management framework

The management of activities as illustrated in Section 3.1 involves a number of services comprising an *activity management framework*. It should be designed to capture, if desired, all information related to an activity, which comprises a great amount of data as well as a large variety of different data sources.

The integration of an activity management framework into a DCI middleware necessitates great care during the design process as it is central to the whole distributed computing infrastructure.

Monitoring framework

Monitoring the status of an activity or a service-level agreement is an integral part of DCIs. Due to the dynamic nature of these infrastructures, frequent and unexpected changes require immediate reactions from the management services. To this end, these changes must be reliably detected and signalled.

To improve usability, it is necessary that monitoring and event notification are provided as core services without the implication of significant additional overhead for applications. For future generation DCIs it will be important to have a generic and coherent interface for all kinds of monitoring operations. This should include, but is not limited to, the monitoring of resource conditions, service state, reservations, current schedules, activity execution, and conformance of allocations to service level agreements.

It is open to discussion whether the data captured by monitoring should be integrated into a general information service or not. Monitoring is a valuable source for dynamic data as requested by requirement <REQ-9>, but most likely it is not feasible to keep and advertise all data gathered by a monitoring framework using an information service. We will resume this discussion later in Chapter 11.

3.4. Examination of DCI Schedulers

Although it is widely acknowledged that the co-ordinated execution of activities in a DCI is vital — even in environments without competitive resource usage — we observe that the findings of research and development in the area of scheduling seldom find their way into production systems. Many of them still make use of comparatively simple scheduling algorithms like first-come-first-serve (FCFS) combined with backfilling [128]. Or they do not deploy a DCI-wide scheduler at all and leave resource allocation to local, on-site schedulers. Large DCIs like DEISA operate that way and burden users with site-selection as well as acquisition of knowledge about capabilities and restrictions of the various sites. The reasons for not deploying sophisticated scheduling solutions are manifold, reaching from costly algorithms over the lack of generic-enough scheduling solutions to simply the missing awareness of the potential of scheduling.

Table 3.1.: Schedulers for distributed computing infrastructures

	Cluster Scheduler	GridWay	Hadoop Capacity Scheduler	Haizea	IANOS	MOAB Workload Manager
Type	cloud scheduler; centralised	grid meta-scheduler; decentralised	scheduler for Hadoop DCIs	cloud scheduler; centralised	grid meta-scheduler; decentralised	cluster/grid scheduler; decentralised
Information model	proprietary	MDS2 & MDS4	proprietary	proprietary (& OpenNebula parameters[a]	GLUE	proprietary
QoS	none	time (deadline) & a few performance parameters	none	time	time & cost optimisation	multiple[b]
SLA model	none	none	none	none	WS-Agreement	none
Negotiation	no	no	no	no	accept/reject	no
Delegation	no	(migration)	no	no	no	(migration)
Reservation	no	no	no	yes	yes	yes
Middleware	Amazon EC2, Eucalyptus, Nimbus, or OpenNebula[c]	Globus	Hadoop	OpenNebula	agnostic; implemented for UNICORE	none
Dependencies[d]	Condor	Globus services	Hadoop Common services	none	information, brokering, monitoring	none
License	open source (Apache License 2.0)	open source (Apache License 2.0)	open source (Apache License 2.0)	open source (Apache License 2.0)	open source	commercial
Misc	creates HPC environment for batch processing via Condor	Virtual appliance for Amazon EC2 is supported	none	can operate in simulation mode	can be customised for cloud computing	none

[a] Haizea and OpenNebula), last visited: January 25, 2013. http://haizea.cs.uchicago.edu/manual/node30.html.
[b] Quality of Service (QoS) Facilities, last visited: January 25, 2013. http://www.adaptivecomputing.com/resources/docs/mwm/7.3qos.php.
[c] OpenNebula.org – The Open Source Toolkit for Cloud Computing, last visited: January 25, 2013. http://www.opennebula.org/.
[d] All grid schedulers interface with local RMS. They are not listed under dependencies.

A number of schedulers (which are sometimes also called 'brokers') exist for different kinds of infrastructures and for different purposes. We examine six of them, which represent the large variety of existing systems, briefly, namely the Cluster Scheduler[12], GridWay[13], the Hadoop Capacity Scheduler[14], Haizea[15], IANOS [78], and the MOAB Workload Manager[16]. The purpose of this examination is to show the characteristics of selected schedulers based on the following criteria[17]:

- The *type* criterion describes the target DCI type (cloud, grid, etc.) and whether the scheduler is centralised or de-centralised.
- The *information model* criterion captures whether a standard model is supported or a proprietary one is used.
- The *QoS* criterion lists the quality-of-service parameters supported by the schedulers.
- The *SLA model* criterion captures whether the scheduler supports service-level agreements.
- The *negotiation* criterion shows whether or not negotiation of QoS is supported.
- The *delegation* criterion expresses whether or not a scheduler allows the delegation of scheduling requests to other schedulers.
- The *reservation* criterion shows the reservation capabilities of a scheduler.
- The *middleware* criterion captures whether a scheduler has been implemented for a certain kind of DCI middleware.
- The *dependencies* criterion reflects lists other services which are needed to operate the respective scheduler.
- The *license* criterion shows under which license the scheduler is distributed.
- The *Misc* criterion lists additional valuable information.

The results, which are shown in Table 3.1, reveal that none of the schedulers support all features that are required (see also Chapter 3) for DCI schedulers, many rely on proprietary features, or support only specific target DCIs. Therefore this examination motivates our work to define a generic scheduling architecture for DCIs which, in case it is used develop a scheduler, fulfils all requirements.

[12] Cloud Scheduler, last visited: January 25, 2013. http://www.cloudscheduler.org/.
[13] GridWay.org, last visited: January 25, 2013. http://www.gridway.org/doku.php.
[14] Capacity Scheduler Guide, last visited: January 25, 2013. http://hadoop.apache.org/common/docs/r0.20.2/-capacity_scheduler.html.
[15] Haizea, last visited: January 25, 2013. http://haizea.cs.uchicago.edu/.
[16] Moab Workload Manager, last visited: January 25, 2013. http://www.clusterresources.com/pages/products/-moab-cluster-suite/workload-manager.php.
[17] We have to anticipate at this point and do not introduce the rational for selecting the criteria. It will become clear in Chapter 3 why they have been chosen.

Further scheduling systems exist, which are, for the sake of brevity, not examined here. The interested reader may also review ASKALON[18], the Cloud Scheduler[19], the Community Scheduler Framework[20], the Condor-G task broker[21], the eNANOS grid resource broker or the KOALA grid scheduler[22].

Research and development are driven by a large amount of applications which motivate the integration of scheduling services into distributed computing infrastructures. We have selected three applications and evaluated them focussing on their common issues. The resulting list of nine requirements, which constitute the framework for a generic scheduling architecture, is the first achievement of our work. To link the requirements to actual services, we provided an analysis of the implications of implementing the required functions. The section concluded with the evaluation of existing scheduling systems in the light of these functions.

[18] ASKALON – Grid Application Development and Computing Environment, last visited: January 25, 2013. http://www.dps.uibk.ac.at/projects/askalon/.
[19] Cloud Scheduler, last visited: January 25, 2013. http://www.cloudscheduler.org/.
[20] CSF, last visited: January 25, 2013. http://sourceforge.net/projects/gcsf/.
[21] Condor - High Throughput Computing, last visited: January 25, 2013. http://www.cs.wisc.edu/condor/condorg/.
[22] The KOALA Co-Allocating Grid Scheduler, last visited: January 25, 2013. http://www.st.ewi.tudelft.nl/koala/.

4. Concept

Here we outline the concept of a generic scheduling architecture for distributed computing infrastructures. It fulfils the requirements outlined in the previous chapter and links them to actual middleware services. A core part of this thesis is what connects the requirements and the services: a number of models, methodologies, and architectural designs, which are also motivated in this chapter. Finally, we introduce the scheduling process to be executed with such an architecture in place.

Distributed computing infrastructures are complex constructs. Researchers have been working on architectural blueprints for whole infrastructures as well as parts of them. An early example of a distributed middleware is CORBA, the *Common Object Request Broker Architecture*[1]. Despite a promising start, it has not gained common acceptance yet [66]. Another approach, in this case targeting grids, is the *Open Grid Services Architecture* (OGSA) [52], an by now discontinued endeavour by the Open Grid Forum. In addition to concepts for a complete architecture of a DCI, there are many that target a specific part it, for example monitoring [69] or security [32].

We acknowledge that architectural blueprints for DCIs are a two-edged sword, since architecture-compliance competes with evolving technologies, requirement-mismatch, and the ambition of researchers to push their own creations. This is why the concept we present in the following section remains on the level of services and their interactions, and does not specify classes and interfaces. Our goal is to use the architecture as a foundation for the description of the scheduling process in Section 4.2 and as a means to clarify how the different models, methodologies, and designs (cf. Section 4.3) can be realised and are linked together.

4.1. Concept of a Generic DCI Scheduling Architecture

We base our concept of a generic scheduling architecture for distributed computing environments on previous research [58, 122, 132] in which we focussed on grid infrastructures and for the most part on the architecture itself. In this thesis, we revive the outcome of this research and evaluate it with respect to the application scenarios and requirements outlined in Section 3. The result is an architecture that extends the one focussing on scheduler interoperability [58] with concise activity management, information management, and SLA management.

To begin, we re-introduce the nine requirements to be fulfilled by the scheduling service environment.

[1] CORBA, last visited: January 25, 2013. http://corba.org/.

- <REQ-1> Discovery of services and resources
- <REQ-2> Access to service and resource information
- <REQ-3> Brokering
- <REQ-4> Scheduling
- <REQ-5> Advance reservation of services and resources
- <REQ-6> Service-level agreement management
- <REQ-7> (SLA) negotiation
- <REQ-8> Activity management
- <REQ-9> Monitoring

Each of the requirements corresponds, as shown in Fig. 4.1, with a service of the architecture. The only case with no corresponding service is the requirement for service-level agreement management (<REQ-6>). The respective model and functions are integrated into a number of services. In the next two sections, the relationships between requirements and services will become clearer, as we describe the process of scheduling based on this architecture, the models, and methodologies.

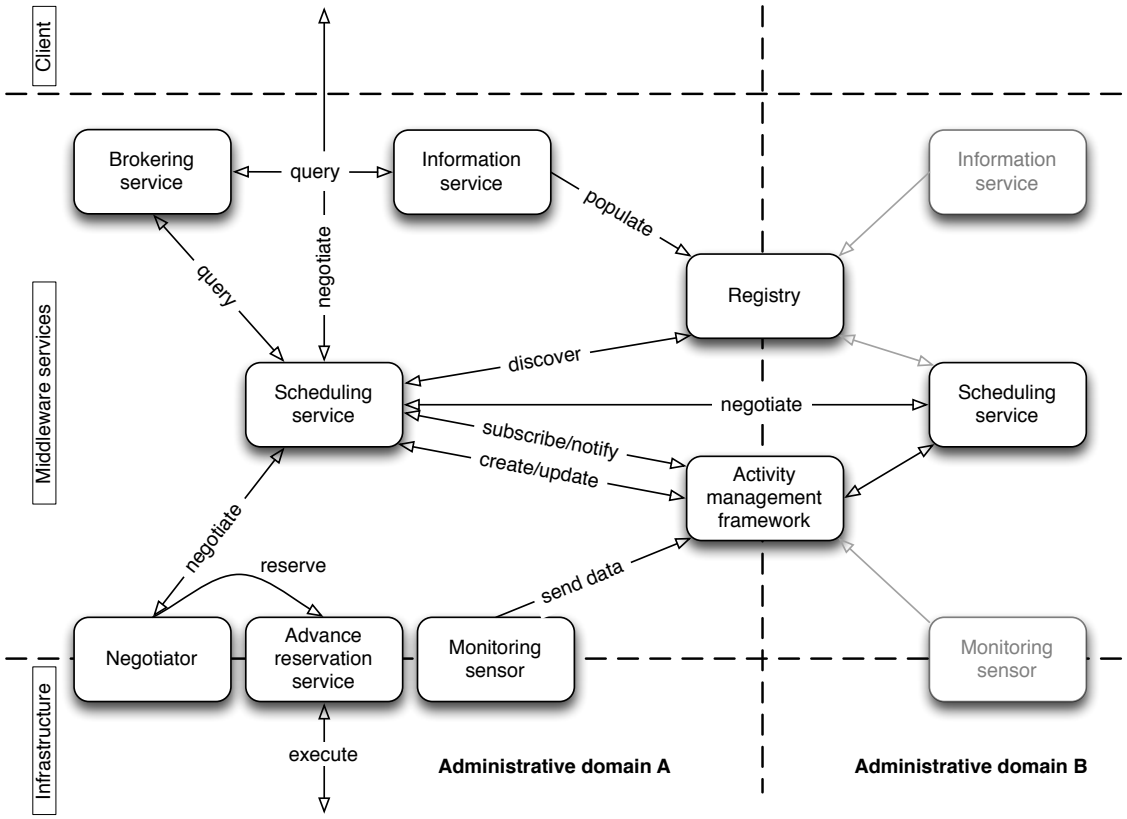

Figure 4.1.: A generic scheduling architecture for distributed computing infrastructures

We distinguish three 'levels' in this architecture: (i) client, (ii) middleware, and (iii) infrastructure. Although it is not possible to make a clear distinction between the different layers, especially in a service-oriented infrastructure where services are deployed ad-hoc to form added-value service orchestrations, the layering is valid for our application scenarios and clearly separates consumers from providers.

Every consumer-provider relationship, either between the different layers or between different administrative domains, is governed by one or many electronic contracts. A *service-level management framework* integrated in the architecture and a *negotiator* at every party[2] are the basis for fulfilling requirements <REQ-6> and <REQ-7>.

Whenever a request to execute an activity arrives at a *scheduling service*, an activity is created or its information is updated, in case it already exists. The respective data related to an activity is maintained by the *activity management framework*, which also receives data from *monitoring sensors*, residing on infrastructure level and sending data on the services and resources which have been provisioned to fulfil the activity request. Other services can, if they have the necessary authority, subscribe to an activity and receive notifications about events related to it. The activity management framework and the monitoring sensor implement the functions needed to fulfil requirements <REQ-8> and <REQ-9> respectively.

The core of the scheduling architecture is the scheduling service itself, which controls the scheduling process and makes the final decision about where and when an activity is executed[3]. To achieve this, it relies on the *brokering service* to answer queries on service and resource capabilities (as demanded by <REQ-3>). Furthermore, the scheduling service depends on resources to be allocated at the point in time determined in the SLA (required by <REQ-4>). This necessitates an *advance reservation service* for every resource in order to be able to reserve it (<REQ-5>).

The whole process of scheduling relies on data about the services and resources which form the computing infrastructure. This data is kept in an *information service*, which is accessed by brokering services and which populates the *registry* with 'basic' data needed to discover a service or a resource (as requested by <REQ-1>). The information service keeps an up-to-date view on the data relevant to make scheduling decisions (and therefore implements <REQ-2>)[4]. Such information could be generated by the same monitoring sensors that send data to the activity management framework.

Services on the boundary between two administrative domains are needed to share information between two or more sites. In case of the registry this is not imperative: each administrative domain within a DCI could host its own registries as long as all services know how to access them. In case of the activity management framework it is more complicated, since it is essential that all information related to one specific activity can be identified as such. This implies either some kind of DCI-wide name management or, as shown in Fig. 4.1, a shared activity management framework. To prevent single points of failure, all shared services should be deployed more than once within a DCI. The figure presents only one instance of the registry and the activity management framework to make clear that they maintain data shared between different administrative domains. Similarly, the

[2] For the sake of simplicity, the negotiator component on client level is not pictured and those used by the scheduling services are included in the respective boxes.

[3] The scheduling process is introduced in this chapter in Section 4.2 and the actual decision making approach is outlined in Chapter 8.

[4] To keep the figure simple, Fig. 4.1 does not picture any information sources to update this data.

architecture prescribes no strategy on replication or data maintenance policies within a DCI or within the different domains.

4.1.1. Interaction with Infrastructure Services and Resources

Each service making up a DCI middleware, and therefore also those described above, is operated by a legal entity, whether it is an academic compute centre, a *National Grid Initiative*[5], or a company. Every single service could, in theory, belong to a different legal entity, which would render communication between the services overly complex due to issues like security, trust, or simply performance. In practice, it comes handy to operate the infrastructure services per resource and most of the DCI middleware services per site.

A typical grid site[6] operates a number of infrastructure resources each of which is normally equipped with local management services. These services are employed to manage resources, retrieve information on them, and integrate them into the scheduling process. In a standard cluster or grid this is the task of a local scheduling service, also called *batch system*, *local resource manager* (LRM), or *(local) resource management system ((L)RMS)*, with *Torque*[7] being a popular open source representative. Usually, one RMS is deployed per site. It receives jobs, executes them on the compute site, and manages the output. Some RMS can also reserve resources in advance, like Torque extended by the MAUI scheduler[8], or consume SLAs, like LSF version 6[9].

Fig. 4.1 presents three services at the interface between the infrastructure and the middleware layer: (i) negotiator, (ii) advance reservation service, and (iii) monitoring sensor. In the typical grid set-up described above, the latter two services would be provided by an RMS. As of today, this would imply that SLA-based negotiation is not possible and that monitoring data is restricted to job and queue-related monitoring information. It is therefore necessary to provide a negotiator and enhanced monitoring capabilities integrated with activity management, in order to connect a DCI infrastructure to the scheduling service.

4.1.2. Interaction with Clients

As depicted Fig. 4.1, the scheduling service provides the client-side interface to distributed computing infrastructures. This modus operandi, a client interacting directly with the scheduling service, is quite common, for example, in grids, but other set-ups exist, too. This includes portals, applications, or other schedulers accessing the scheduling service on behalf of the human client and therefore being clients themselves. Any client or client agent has to implement client-side negotiation functions to come to an agreement with the scheduling service about the activity execution and its related quality-of-service parameters.

[5] EGI – Resource Providers, last visited: January 25, 2013. http://www.egi.eu/production-infrastructure/Resource-providers/.

[6] An example here is the reference installation of the German national academic grid, called D-Grid: http://dgiref.d-grid.de/wiki/Image:Architecture01.png (last visited: March 04, 2011).

[7] Torque Resource Manager, last visited: January 25, 2013. http://www.clusterresources.com/products/torque-resource-manager.php.

[8] MAUI Cluster Scheduler, last visited: January 25, 2013. http://www.clusterresources.com/products/maui-cluster-scheduler.php.

[9] Goal-Oriented SLA-Driven Scheduling, last visited: January 25, 2013. http://hpc.ilri.cgiar.org/documents/-admin_6.0/sch_sla_aware.html.

4.2. The Scheduling Process

Scheduling in distributed infrastructures is complex and, depending on the objective function and the number of entities to take into account, a lengthy process. One structured approach to describe the stages of DCI scheduling was published by Schopf in 2004 [120]. It comprises three phases, which in total include 10 different actions. As Schopf especially targets grid infrastructures and neither considers SLAs or delegation nor negotiation or activity management, we extend her approach to reflect the requirements depicted in Chapter 3.

4.2.1. 'Ten Actions When Grid Scheduling' Revisited

Schopf subtitled her contribution 'The user as a Grid Scheduler' indicating, that at the time of publication, fully automated support for her concept did not exist. Therefore many of the actions depicted in Fig. 4.2 were actually executed manually.[10]

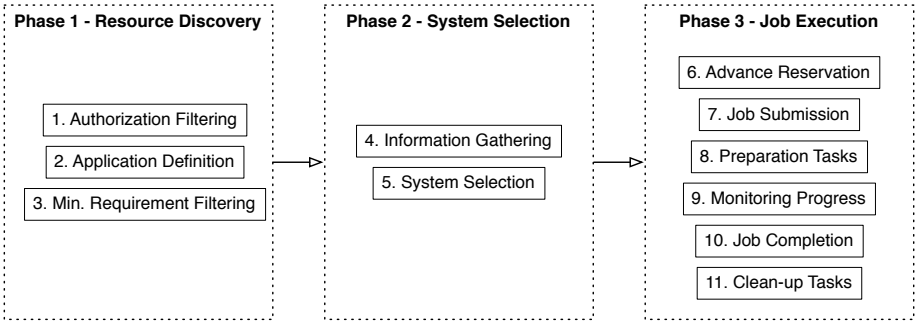

Figure 4.2.: Three phases/ten actions for grid scheduling (Step 6 'Advance Reservation' is optional) [120]

The actions are grouped in three phases: (i) resource discovery, (ii) system selection, and (iii) job execution. They contain eleven steps, one of which is optional[11]:

1. **Authorization filtering**: In this step, the scheduling service determines the resources the user has access to.

2. **Application requirement definition**: The user defines a minimal set of resource requirements to enable the scheduling service to search for candidate resources for a job.

3. **Minimal requirement filtering**: It is the purpose of this step to exclude resources from the pool which do not fulfil the minimal set of requirements.

[10] An example for the user-centric work-flow has been the usage of a UNICORE grid via a graphical client software: http://unicore.eu/documentation/manuals/unicore5/files/client_manual.pdf (last visited: March 6, 2011).

[11] Please note that we do not distinguish here between steps possibly executed manually and those carried out by a scheduling service. For the understanding of the process and with respect to the update we propose in the next paragraph this is irrelevant.

4. **Dynamic information gathering**: To calculate the best-possible schedule, the scheduling service needs up-to-date information about candidate resources.

5. **System selection**: In this step, the scheduling service selects the system(s) to run the job on.

6. **Advance reservation**: Some of the selected resources may need be reserved in advance to guarantee the execution of the job. This is an optional step.

7. **Job submission**: In this step the job is executed to the selected systems(s).

8. **Preparation tasks**: The preparatory actions ensure that the environment to execute the job is set up properly. This step includes e.g. the stage-in of data or the allocation of reservations.

9. **Monitoring progress**: For some applications it is evident to monitor the progress of the job and validate whether it adheres to certain performance demands. Negative validation may lead to re-scheduling (Step 4).

10. **Job completion**: The user needs to be informed about the completion of her job.

11. **Cleanup tasks**: This step includes actions like data retrieval or freeing of temporary storage.

We now take these steps as a basis for an updated scheduling process for distributed computing infrastructures with support for service-level management, activities, and delegation.

4.2.2. The Scheduling Process for DCIs

The first important difference between Schopf's and our approach is that we aim for a fully-automated scheduling process. The steps depicted in Fig. 4.3 can be automatically executed in a DCI environment that implements the architecture presented in Fig. 4.1. Full automation, though, is not mandatory as synchronisation points for manual intervention or checking can be added to the process. This may, for example, be necessary to authorise the negotiated price of a service offer.

Fig. 4.3 outlines the scheduling process, which comprises eleven steps. In addition, decisions taken at three of them may lead to the delegation of the scheduling request, which has been depicted accordingly. Last but not least we linked the scheduling steps to the respective phases of the SLA life-cycle, which is introduced in full detail in Chapter 7.

1. *Activity template receipt*

   Upon receipt of an activity template, a scheduling service either creates an activity instance or updates an existing one. Both is done by calling the respective function provided by the activity management framework[12]. Further content is added to the activity template, as a result of the following steps, until all necessary data is available and the activity can be executed. We do not make any assumptions about the nature of the client who submits the activity template. It can be the user herself, an

[12] We anticipate some terms and interrelations here that are introduced later in Chapter 5.

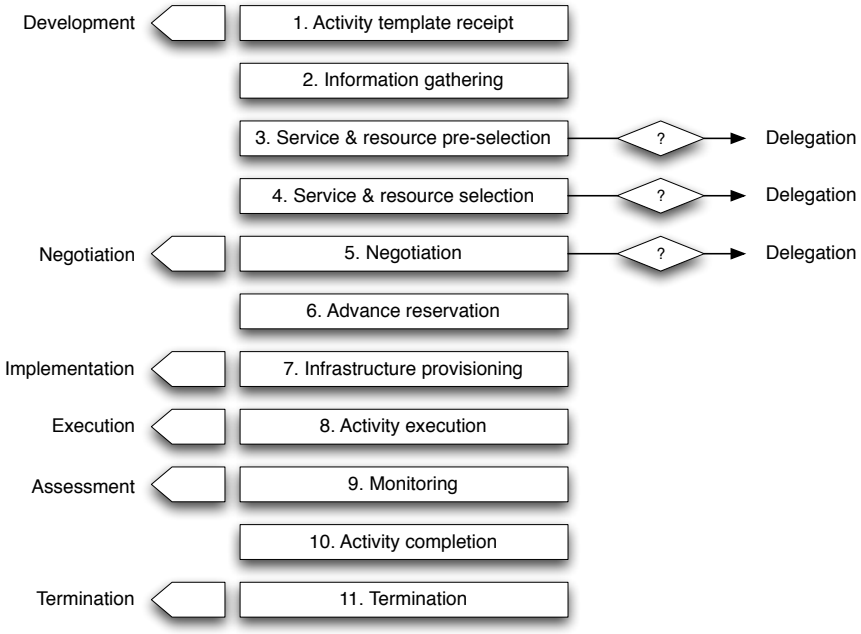

Figure 4.3.: The scheduling process for distributed computing infrastructures. The arrows to the left indicate the respective phases of the SLA life-cycle according to Fig. 7.1

agent acting on behalf of the user, or another scheduling service. As such, the activity template can be part of or embedded into an SLA and subject of negotiation itself. Regarding the scheduling process, however, the activity template merely represents the customer's requirements and we do not take chained negotiations, nested SLAs, or other interrelations into account.

2. *Information gathering*

   As a first step towards creating a schedule, information about services and resources is gathered by the scheduling service to create an up-to-date view on the resource pool. In general, the scheduler does not request all available information about all services and resources, but either uses registries to discover certain kinds of services or queries brokers for entities with certain characteristics (cf. Fig. 4.1). This step is, depending on the scheduling strategy, executed multiple times featuring a variety of query parameters. Although depicted only once in Fig. 4.3, it is suggested to gather information at least twice, first for pre-section and second for the actual selection process (see below).

   The information gathering may lead to the delegation of a scheduling request in case no suitable services or resources can be found.

3. *Service and resource pre-selection*

   As a next step, services and resources are pre-selected. In a fully automated, large-scale DCI the scheduling service most likely queries a number of brokers to retrieve

candidate services and resources based on a basic set of static parameters[13]. As a result, the scheduling service has a narrowed range of candidates and considers for scheduling, for example, only HPC systems with a maximum of 1024 cores. In case no candidates have been selected by the contacted brokers, the scheduling service may delegate the activity to another scheduler or reject the activity respectively.

Contrary to Schopf, we include user authorisation in the pre-selection step as it is just one additional static parameter.

4. *Service and resource selection*

   With a range of candidates at hand, the scheduling service selects services and resources that are needed to fulfil the request embedded in the activity template. This means, reviving the previous example, that the scheduling service calculates the best-fitting schedule taking all resources into account which offer, say, 256 cores at the requested time. Most likely, other constraints are included in the activity template, like a maximum cost, a minimum amount of memory, or the availability of a particular software license.

   If no resource(s) can be selected, it will be up to the provider of the scheduling service to define a strategy to react accordingly. This may either be re-scheduling, starting again with the 'information gathering' step, or the delegation of the request[14]. Such events can be recorded through the activity management framework, as can similar events during the previous steps of the scheduling process.

5. *Negotiation*

   The scheduling service then has to negotiate with the negotiators of all entities which are required to execute the activity. For some resources this may not be necessary, for example, if best effort network provision suffices to run an application. In all other cases, the scheduling service has to negotiate the terms of service provision with a service or a resource. In the event of one or more failed negotiations, it is again up to the provider to decide how to react (by specifying the respective policies) and the re-scheduling of the request is the likeliest strategy.

   SLA with external parties, as e.g. depicted in Fig 4.1 between two scheduling services belonging to different administrative domains, are essential contracts to agree upon quality-of-service. Especially in cloud-like environments, negotiation between a scheduler and the IaaS provider about the virtual machines and their prices are likely to become a common scenario. But also within one enterprise 'internal SLAs', or *operational-level agreements* (OLAs) as they are called by ITIL, are recommended to 'set out specific back-to-back targets for support groups that underpin the targets included in SLAs' [103].

   Negotiation itself can be a multi-step process, as described in Section 7.5. With respect to scheduling in DCIs, it is one step of the whole process and we do not make any assumptions about its specifics.

[13] For the distinction between static and dynamic service and resource information we refer to Section 3.2.

[14] We did not depict the optional delegation step here since we assume it is in the interest of the provider to find suitable resources and therefore to re-schedule.

6. *Advance reservation*

   After the negotiation of an SLA, it is, in many cases, essential to reserve a resource in advance. This is an action normally executed automatically by a provider and triggered by the event of having an SLA agreed upon. It simply implies that in case a provider guarantees a customer access to a resource, like a number of compute nodes or storage space, the provider has to reserve it in advance.

   The advance reservation step is executed by a an entity which we call advance reservation service (according to Fig. 4.1). In general, this service is integrated into a dedicated RMS for the particular resource, whether it is a storage or a network reservation system. For most compute resources in DCIs, the advance reservation service is part of the back-end batch system. It manages clusters or HPC systems, and therefore it is essential that it supports advance reservation.

7. *Infrastructure provisioning*

   At the time the activity execution has been scheduled, the infrastructure to perform this has to be in place. In state-of-the-art DCIs, many of which are grids, this step involves mainly activation of a reservation and preparative actions like staging of files or execution of pre-run scripts. With IaaS environments as the current targets for the provision of resources in DCIs, this step is more dynamic. It involves the deployment of virtual machines and services, the set-up of *VLANs*, and the creation of an ad-hoc monitoring infrastructure for the particular activity.

   Ideally, the infrastructure provisioning not only includes compute resources, but also takes issues like the time needed for data pre- and post-staging into account. Something like this would imply that the infrastructure provisioning starts hours or even days before the reservation becomes active. It may also include storage and network services which are not explicitly demanded by the client in her activity description. It is therefore the task of the scheduling service to take such problems into account.

8. *Activity execution*

   The actual execution of the activity is triggered by the advance reservation service (cf. Fig, 4.1). As it maintains the reservation, it is also responsible for starting an activity. Regarding implementations of the generic scheduling architecture, it could also be a specific execution service, which has the reservation information present and initiates the execution. It is important to note that no specific action is expected to be taken by either the client or the scheduling service as the infrastructure services will take care of the execution. All necessary information has been included in the activity template or was added to it during the negotiation step.

9. *Monitoring*

   As part of the ‘infrastructure provisioning’ step, activity-specific monitoring services are deployed or activated, and they are registered with the activity management framework. Through this, continuous monitoring of the activity during its runtime is guaranteed. Other services that assess SLA-compliance of the service offer or report the state of the activity to the client can subscribe to monitoring information. Also, the scheduling service, as presented in Fig. 4.1, is informed about the state of the activity and may act upon it.

10. *Activity completion*

    Once the execution of an activity has ended, either because all of its parts have been processed or because of an error condition, the scheduling service is informed and the activity information is updated accordingly.

11. *Termination*

    Upon termination, the environment used to execute the activity is decommissioned. Resources are de-allocated, for example if the reservation exceeds the runtime of the activity, and virtual machines are shut down. In some cases, post-execution actions are performed, like cleaning up temporary file space or conducting user-specific commands. Final information about the activity is then included in the activity record to provide data about resource usage or the cost of service delivery.

We additionally refer to Appendix A for a sequence diagram linking the scheduling process to the respective middleware services (cf. Section 4.1).

4.3. Models, Methods, and Architectures: An Overview

This work contributes models, methods, and architectural designs which serve as foundations to implement a generic scheduling architecture and to realise the aforementioned scheduling process. The respective artefacts are summarised in Fig. 4.4. To the left, the nine requirements derived from the scenarios in Section 3 are listed. The arrows in the middle column indicate the respective model, method, or architecture we have extended, contributed to, or developed to meet the various requirements. To the right, the respective implementations are listed. In case an arrow stretches more than one column, our research has revealed existing concepts that fulfil the requirements and thus do not call for any novel contribution. The color-coding indicates related concepts: blue relates to information management, red to scheduling, green to service-level agreements, and yellow to activity information management.

The scheduling process and the value of the resulting schedule rely on the quality of the information about the entities in a distributed computing environment. According to the requirements, two issues are of importance here: discovery and information access. A number of solutions that meet the former and that are applicable to DCIs already exist, inter alia UDDI[15]. We therefore focus on the modelling of information and introduce an DCI-ready adaptation of the *Common Information Model* in Chapter 6 ‘The Information Model’. Furthermore, this model is the foundation of all information management within a DCI including the activity management.

A broker can be seen as an intermediary between the information service and the scheduler fulfilling the mere task of limiting the range of scheduling candidates. Therefore, brokering is primarily used to find the best-possible mapping of queries from the scheduler to available services and resources. It is a field of ongoing research which is primarily dedicated to the development of algorithms. Advance reservation is, same as brokering, not within the focus of the methodological part of this thesis. In the context of the scheduling

[15] OASIS UDDI Specification TC, last visited: January 25, 2013. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=uddi-spec.

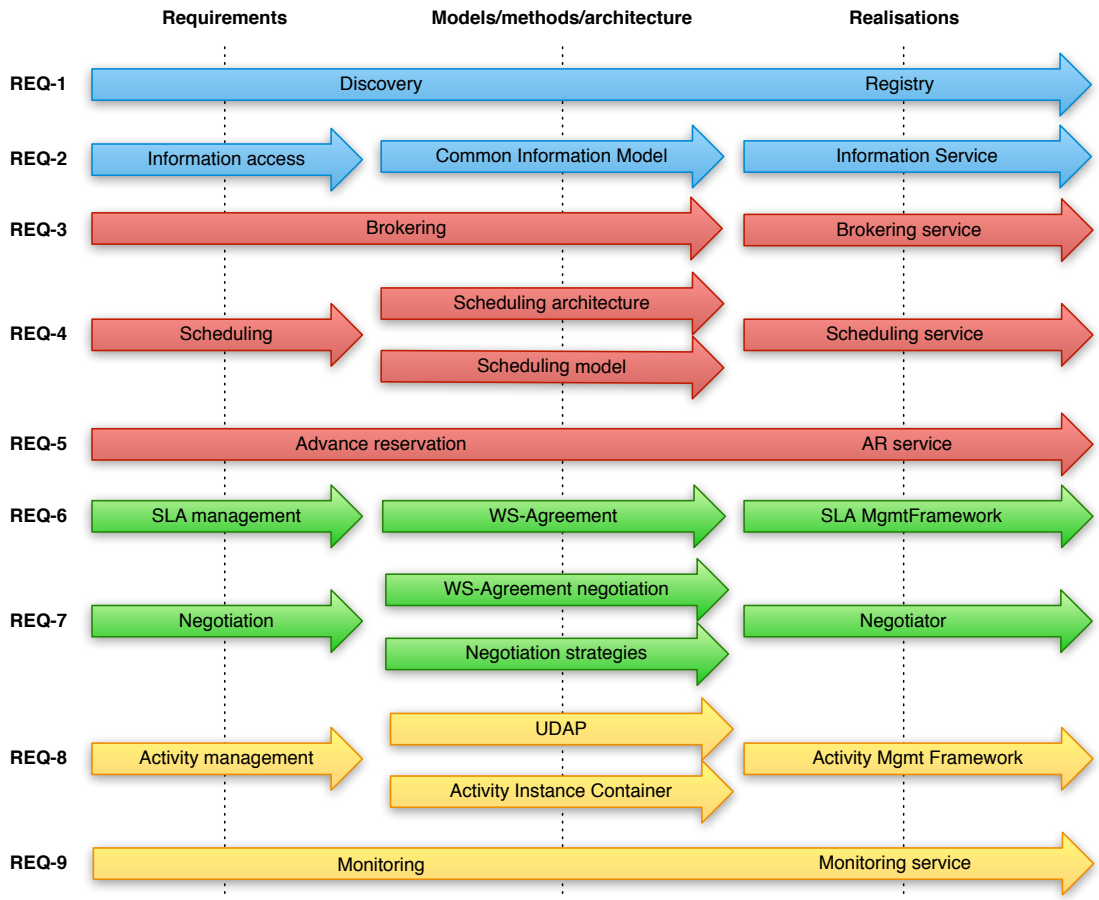

Figure 4.4.: From requirements over models, methods, and architecture to realisations ('AR service' is 'Advance reservation service'; ' UDAP' is 'Universal Dynamic Activity Package'; 'Mgmt' is 'Management')

process, advance reservation, on the one hand, is a feature to be provided by infrastructure services and, on the other hand, a function called by the scheduler. Recalling the example of a grid site from Section 4.1.1, it is usually the local RMS that implements advance reservation and offers the respective function to its clients. For the scheduling process only this is of importance, even though we extend this concept to any kind of 'reservable' entity. The main contributions with respect to the set of scheduling requirements are a generic architecture for scheduling services including the respective process and a tailored model to generate schedules for DCIs (including the delegation of scheduling requests between scheduling services). These artefacts are introduced in Section 4.1, Section 4.2, and Chapter 8 'The Scheduling Model' respectively.

In distributed computing, service-level agreements are often used as framework contracts for the delivery of services. In general, they are paper contracts containing all details about the delivery of particular services.[16] In this thesis, we refer to service-level

[16] A good example is the service-level agreement between the European Grid Initiative EGI.eu (http://www.egi.eu, last visited: February 28, 2011) and their technology providers. The respective tem-

agreement as electronic contracts which are automatically negotiated between services. To achieve this, we contributed to the the application of WS-Agreement within the scheduling domain and also to the development of WS-Agreement Negotiation.[17] The particular results of these contributions are introduced in Section 7 'The Service-Level Agreement Model'.

The concept of activity management and its tight integration into the generic scheduling architecture originates from our work in the NextGRID project. There we developed the *Universal Dynamic Activity Package*, a model to aggregate all information that is related to an activity. Chapter 5 'The Activity Model' describes this model and, in addition, presents a particular evolution of the Universal Dynamic Activity Package, namely the *Activity Instance Description*. The two models rely heavily on all monitoring information related to an activity being fed into the activity management framework. We require that the monitoring sensors which generate this information are deployed within a DCI and see the update of an activity mainly as an automated process, potentially including the transformation of monitoring data into the required format.

In the following section we briefly review the middleware services which implement the artefacts introduced in this section.

4.4. Implementation of the Concept: An Overview

Based upon the artefacts introduced in the previous section, the services which make up the scheduling architecture (cf. Fig. 4.1) have been implemented. The core of this architecture are service-level management and activity management, and its overall aim is to provide the foundation for automated negotiation of electronic contracts.

Fig. 4.4 shows the respective services. Most services correspond to those depicted in the architecture in Fig. 4.1. Others are either integrated into one of the former services or frameworks, like the brokering service, the advance reservation service, and the negotiator, or they comprise multiple services as the SLA framework and the activity management framework. The registry, however, is a special case as it has been realised as a front-end to the Common Information Service, offering tailored XQuery[18] calls for service and resource discovery. The implementations of the services are described in detail in Chapter 9.

In this chapter we introduced two major contributions of our work, as there are the concept of a generic scheduling architecture for distributed computing environments and an automated scheduling process that is based upon this architecture. Furthermore, we gave a rough idea of four fundamental models which are necessary to realise scheduling architecture and process and which present further contributions explained in the following chapters.

plate can be found here: https://documents.egi.eu/document/241, last visited: February 28, 2011.

[17] We would like to state here explicitly that we did not contribute to the initial specification of WS-Agreement [6].

[18] XQuery 1.0: An XML Query Language (Second Edition), last visited: January 25, 2013. http://www.w3.org/TR/xquery/.

Part III.

Core Models

5. The Activity Model

This chapter introduces the concept of an activity and describes two models how to realise it. The first model, called Universal Dynamic Activity Package, has been developed and implemented within the European NextGRID project. The second model, which has been derived from the first one, is the Activity Instance Container, which is maintained by the Open Grid Forum.

In state-of-the-art distributed computing infrastructures, information about an activity is fragmented and dispersed. Activity-related information, such as SLAs, activity state changes, or consumption of resources, is currently captured using a variety of schemata and services, and it is stored in different ways and by different logical components. This dispersion of activity information leads to management, security, and logistic overheads in discovering, accessing and using that information. Furthermore, the handling of information demands a lot from system administrators and results in infrastructures activity information is managed by various services. This makes it difficult for users and providers to find this information, since they have to search for and to 'keep an eye' on many sources.

One good example for a DCI where this challenge is obvious is the German D-Grid [53], since it features three different middlewares and consequently three different information systems. This led to significant development work and requires continuous operational efforts to maintain activity information.

To solve the aforementioned issues of information fragmentation, we now introduce the *Universal Dynamic Activity Package* (UDAP) model.

5.1. The UDAP Model

The Universal Dynamic Activity Package aims at bringing all of the information fragments associated with an activity into one logical package, regardless of the various schemata used to describe and capture these fragments. Using UDAP, any service requiring activity information has a single source for retrieving and updating that information. Information consumers can subscribe to the interface of an entity managing an activity, so that they may be notified of changes in activity information, thus allowing them to fulfil their role with respect to that activity. The same consumers can poll the UDAP management interface for activity information in read, update, and append operations, in addition to or instead of subscribing for notifications.

5.1.1. Clarification of the Term Activity

First of all it is essential to understand the meaning of the term activity. Those familiar with cluster computers or grids normally know the concept of a job. This is a container used to specify what to execute on the respective system. Furthermore, a job includes parameters to further define the execution environment in terms of operating systems, libraries, or storage demand. Common terms used for the same thing are task, process, or work-flow, with the latter two normally applied to interdependent tasks or jobs.

An activity can be all of the former; and it can be more: it can be a job, a task, a data processing operation, a data access operation, an application execution, or a Web service invocation. In general, it is a unit of work in a distributed computing infrastructure: it is something that a user, a service, or an application needs to do or to execute. An activity is per definition atomic. This means that, from the activity management perspective, an activity is an indivisible unit of work. Several activities, however, can be connected to form a sequence, which may be managed conditionally, sequentially, or in parallel. Therefore, activities can be atomic nodes in a work-flow.

5.1.2. Information Captured by an Activity

In general, the UDAP model allows all kind of information related to an activity to be captured. The concrete implementation of an activity therefore depends on the environment into which it is integrated, but most likely includes:

- all of the activity's requirements,
- all of its dependencies (on data and other activities) for the composition and management of activities for work-flow, scheduling and brokering processes,
- all of its contextual information, such as the topical domain (e.g. ERP or weather forecasting), security details (including the owner of the activity and who is authorised to access its information), SLAs, quality-of-service information, and other related policies, and
- all of its monitoring information, such as state, history, resource usage information, accounting data, or policy conformance.

5.1.3. UDAP Concepts

The core of the UDAP model is the *UDAP Description*. Its information is managed by a *UDAP Manager* and processed by *UDAP Clients*. The concepts are pictured in Figure 5.1 and described in the following paragraphs.

UDAP Description

The UDAP Description document is the package that holds all the information associated with an activity. It needs a schematic structure that allows it to classify activity information and to reflect the state of this activity at any point in time.

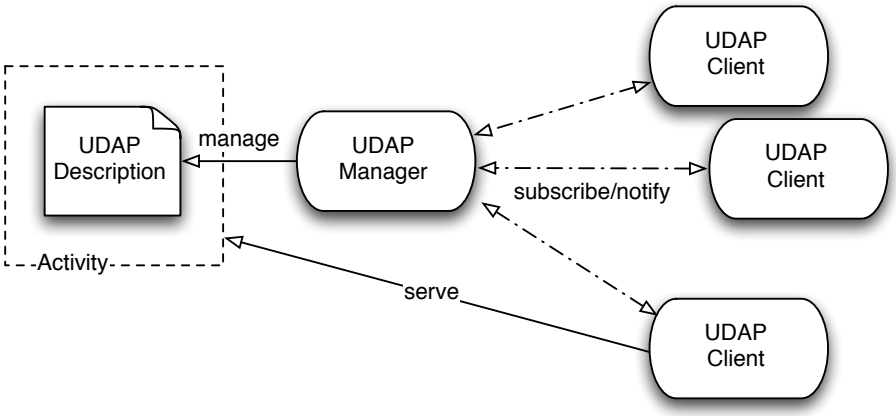

Figure 5.1.: The UDAP Manager is in charge of a UDAP Description for one specific activity (represented by the dashed box). UDAP Clients concerned with that activity may subscribe to the UDAP Manager to be notified of changes in the activity. In addition, clients that 'serve' an activity (i.e. generate information related to the activity) can add information to it.

UDAP Manager

The UDAP Manager is the entity that deals with the management-oriented aspects of all other UDAP entities. Its task includes (i) the creation and destruction of activities and (ii) the interactions with the clients, which subscribe for notifications regarding certain activities.

UDAP Client

Any service within a distributed computing infrastructure that uses and/or produces activity information is called a UDAP Client. It invokes the public management interface of the UDAP Manager for read, update, and append operations of activity information. A UDAP Client can subscribe to the interface of an activity, in order to receive notification of activity information based events. The subscription of a UDAP Client may be conditional, where the condition dictates the type of activity information-based event that the client is interested in, e.g. 'notify me if the state of the activity changes to running' or 'notify me if the resource usage of the activity has exceeded the budget of the activity owner'.

UDAP as an Intermediary

An interesting feature of UDAP is that it acts as an intermediary between activity owners and the service and resource providers that serve that activity (cf. Figure 5.2). This is a result of the activity being at the centre of the interactions between the activity owner and the service providers.

The activity owner, who initiates the activity, does this through a DCI middleware service, such as a portal or a scheduling service. This service itself is a UDAP Client, which can subscribe for activity information-based events to the activity that it has initiated.

The intermediary characteristic of UDAP leads to a symmetric, uniform interface for activity management with the initiator on one side and the service providers that serve that activity on the other, all interacting with and using the activity through the same interface.

Figure 5.2.: An activity as an intermediary between an activity owner/initiator and a service provider serving that activity. The box labelled 'Activity' represents the abstract activity entity (cf. also Figure 5.1).

UDAP Schema

The UDAP schema captures the structure of the activity information in a UDAP Description. Its purpose is to manage all information associated with an activity within a distributed computing infrastructure. This, in turn, allows any service that is concerned with an activity to access and use that information processing the UDAP schema or parts of it. The high-level structure of the schema is shown in Figure 5.3. The depicted elements contain the following information:

- The **UDAP** element is the root element of a UDAP Description. It represents a unique, atomic activity and captures all the information about that activity.
- The **ActivityID** holds the unique ID of the UDAP Description.
- The activity is described through the **ActivityDescription**.
- A **Record** captures activity information throughout its lifetime. Hence, a running history of the activity is captured by a number of Entry elements.
- An **Entry** contains (besides a time stamp indicating its creation date and time):
  - an element that describes the **state** of the activity at the time the entry is entered into the activity's record,
  - all of the **resource** information required by and associated with the activity,
  - contextual information, such as policy information or domain-specific information, which is captured by the **Context** element, and
  - **dependency** information, such as dependencies on data or other activities, state of the system.
- A **Result** element.

The XML rendering of the UDAP schema can be found in Appendix B.

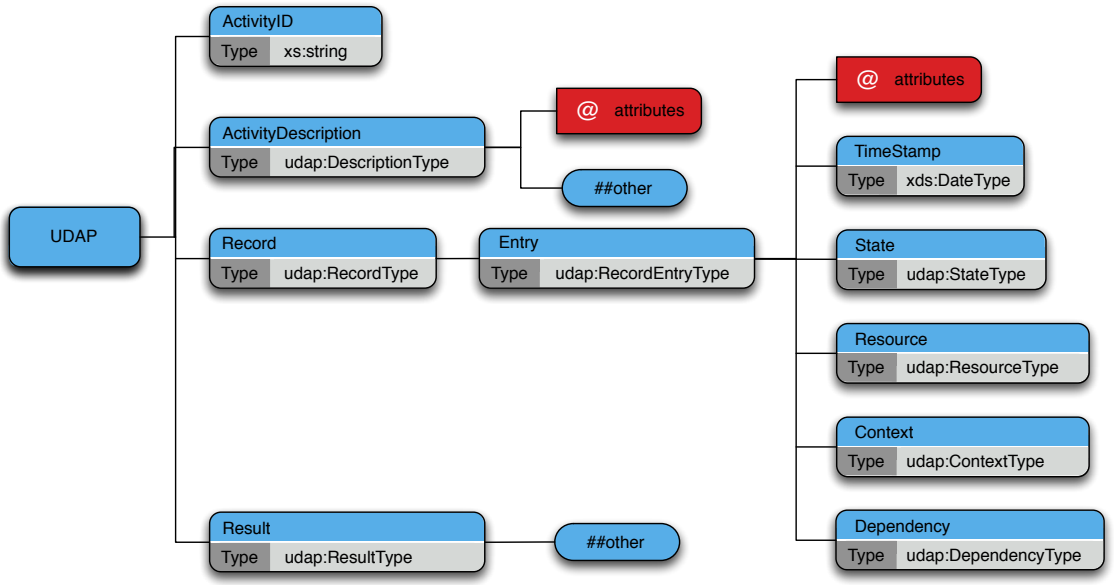

Figure 5.3.: A high-level view of the UDAP schema

5.2. The Activity Instance Container

Following the effort to define and implement UDAP within the NextGRID project, we forwarded the approach to the Open Grid Forum (OGF) for standardisation. The *Job Submission Description Language* (JSDL) working group acquired the activity concept, evaluated it in the light of its stakeholders' use cases, and specified the Activity Instance Container (AIC). Unlike UDAP, the JSDL working group pursues an approach tailored to the needs of DCIs that apply standards of the Open Grid Forum. AID integrates with concepts like the *Basic Execution Service* [49] or JSDL [7], thus keeping the model extensible for usage beyond the foundational use cases.

A thorough introduction of the Activity Instance Container is beyond the scope of this thesis. Since the activity model implemented for this thesis is the UDAP model, we refer to the OGF document for further details [106].

5.3. Assessment of the Activity Model

The application of the activity model to DCIs allows central maintenance of all information related to one activity and, furthermore, consumers are able, at every point in time, to gather up-to-date data. In a common scenario this information is currently generated by monitoring sensors, accounting components, logging services, and alike. And the resulting data is provided through different interfaces and in different formats. The ubiquitous realisation of the activity model would decrease the maintenance overhead and provide concise and up-to-date information. We did, however, not yet discuss the implications of introducing such a model. There are plenty of issues to examine, many of which are related to system design and implementation. Here, we highlight two topics which have been in-

tensively discussed in the course of developing the activity model, namely i) integration of the model into existing DCIs and ii) access control to the activity information.

Obviously, the application of the activity model requires all information sources and activity clients to follow the model and implement the respective interfaces, an undertaking difficult to achieve for an existing middleware. We implemented our prototypes from scratch, utilising grid services mainly for execution purposes, and therefore cannot provide the respective heuristics. However, with the current implementation of multiple cloud-based solutions, it might be worth considering the integration of the activity model into service-oriented DCIs. Nevertheless we are confident that such issues can be resolved and show one implementation tackling some of them in Section 9.1.

In this chapter, we introduced the activity model. It represents a possibility to capture all information related to an activity in one (logical) place and allows to design DCIs with a concise information maintenance concept. The model has been successfully applied to a number of scenarios and, furthermore, this part of our work has been taken up by the Open Grid Forum to specify the Activity Instance Container.

6. The Information Model

The Common Information Model is a standard to capture the manageable entities of an IT infrastructure. In this chapter, we describe the basics of this model, provide a rationale for its application to distributed computing infrastructures, and introduce an extension to the model that has been defined to manage resources in UNICORE-based DCIs.

By definition, an information model is a data model representing entities in an arbitrary problem domain. In the context of distributed infrastructures, an information model presents a viable solution for the management and exploitation of resources. Applied by an information service it can, for instance, assist a resource broker to discover the best resource by analysing model instances. Another example would be that the information service queries resource data on request of a job scheduler, in order to schedule and monitor activities. To successfully schedule distributed systems, it is essential that the entities involved in the processing of an activity are properly described. Applied to a service-oriented system, this means that a scheduler needs the best possible description of the characteristics of a service to find the most suitable one, while in a grid, the scheduler has to match between the resource requirements of a user and the capabilities of the DCI. More concretely, in case an application relies on a certain software version, it is of no help if an information system captures the deployment information with a boolean value, but contains no data about the actually deployed version. In the same way, many scenarios can be obtained that involve other components or services concerned with resource management or scheduling.

The Common Information Model (CIM) [83] is one possibility to describe the manageable entities of infrastructures. We show in the course of this chapter that it fulfils the requirements of an information model for distributed computing infrastructures, introduce its basic concepts, and present one specific extension and implementation of CIM for infrastructures that operate the UNICORE middleware[1] [17].

6.1. Requirements

We evaluated existing information models and conducted a requirement analysis within the NextGRID project. From this, the following requirements to be fulfilled by an information model for DCIs have originated [94][2]:

[1] Pleas note that we refer throughout this thesis to UNICORE Version 5.

[2] We refer to GLUE version 1.3 throughout this thesis since the evaluation has been conducted before the GLUE 2 standard has been published by the Open Grid Forum in 2009 [5].

- A unified model is needed which captures all resource concepts of a DCI like type (like software or hardware) and environment (like compute or data).
- It is essential that the model distinguishes clearly between abstract and concrete classes.
- The defined class hierarchy must be extensible to allow the integration of future concepts.
- The model has to be based on the object-oriented paradigm.
- An implementation of the model has to be based on interoperable and open standards such as XML Schema to maximise interoperability.

Although a number of information services for the kind of infrastructures we consider already exist, we discovered some gaps that have to be filled regarding the requirements listed above. We therefore decided to use the Common Information Model as the foundation for an information service for DCIs, which is described in Section 9.

6.2. Rationale for Choosing the Common Information Model

An information service is believed to be highly dependent on the underlying information model, since it is the base for discovering services or resources. During the design phase of the information service, there were two potential approaches for modelling information: the first was the creation of a customised model from scratch, and the second was the adaptation and evaluation of one of the existing information models well-known to the grid and e-Science community.

Each of the two has advantages and disadvantages. The approach of designing a new model from scratch would result in a proprietary solution which furnishes only our proposed information service, but would not be interoperable with other existing services and middlewares. Not only does this effect interoperability, it may be prone to some fallacies in terms of completeness and correctness catering the information discovery requirements of all potential clients of an information service, e.g. a broker or a scheduler.

The second approach would require the evaluation of existing information models which are currently used in distributed computing environments. We conducted such an evaluation which revealed [94] that these models actually do not meet all of our requirements. Therefore, extensions need to be applied, introducing additional complexity in case of varying requirements. This shows on the one hand, that by adapting an existing model we can profit from its maturity, semantic correctness and from interoperability with already existing implementations. On the other hand, if the adapted model already aggregates hidden defects, these are implicitly part of the extended model. Therefore architects should balance the arguments carefully when adapting a standard; especially when a model evolves into a new version through a standardisation community. Furthermore, in the majority of cases a standards-compliant model is customised for specific business requirements, which again raises the question of interoperability.

In our case we followed the second approach as standard-compliance and interoperability are two of the main requirements. Ab initio, two options appeared to be adequate

candidates for the information model: GLUE and CIM. While the GLUE model is widely accepted by major middlewares in the grid community, CIM is mainly used in industry.

GLUE, the *Grid Laboratory Uniform Environment*, is an information model that aims to provide interoperability among different grid stake-holders. It is a collaborative effort initiated by the European DataTAG[3] and the US iVDGL[4] projects. Later EGEE[5], Globus [48], NorduGrid[6] and LCG[7] joined the process of development and standardisation. GLUE is an object-oriented information model that represents computing and storage resources without exposing any complex hierarchies of entities. This model is concrete and handles information using aggregation and composition between entities. The GLUE information model contains simple entities without complex association and extensibility. Therefore, the major challenge in applying GLUE is to realise the essential extensions for the SLA-based scheduling-driven use cases. However, we could avoid enforcing extensions by realising workarounds, but this only provides short-lived solutions. Consequently, such a solution results in a lack of interoperable interfaces with other third party components which already use GLUE. As interoperability was one of our objectives, GLUE's shortcomings lead us to contemplate the second alternative, the CIM model.

6.3. The Common Information Model

CIM, developed by the Distributed Management Task Force (DMTF)[8], is a model that represents managed physical and logical computing entities, applications, and systems [83]. The scope of the Common Information Model is not only limited to distributed computing entities, it also accommodates the modelling of entities required to manage file systems, operating systems, or even hardware controllers. DMTF defines CIM as follows [36]:

> The DMTF Common Information Model (CIM) is a conceptual information model for describing computing and business entities in 46 enterprise and Internet environments. It provides a consistent definition and structure of data, using object-oriented techniques. The CIM Schema establishes a common conceptual framework that describes the managed environment. A fundamental taxonomy of objects is defined both with respect to classification and association, and with respect to a basic set of classes intended to establish a common framework.

Furthermore, CIM adheres to object-oriented design principles and is based upon a hierarchy of abstractions: 'CIM is an information model [...] that attempts to unify and extend the existing instrumentation and management standards [...] using object-oriented constructs and design.' [36]. From there we can exploit a number characteristics that help to design an information schema for distributed computing infrastructures. These include:

- **Abstraction and classification**: CIM supports the classification of management objects by extracting objects with common behaviour, properties, and relationships.

[3] EU-DataTAG, last visited February 23, 2011: http://datatag.web.cern.ch/datatag/.
[4] The project's web site is not online any more.
[5] Enabling Grids for E-sciencE (EGEE), last visited February 23, 2011: http://www.eu-egee.org/.
[6] NorduGrid, last visited February 23, 2011: http://www.nordugrid.org/.
[7] Worldwide LHC Computing Grid, last visited February 23, 2011: http://lcg.web.cern.ch/LCG.
[8] Distributed Management Task Force,Inc., last visited February 23, 2011: http://www.dmtf.org/.

- **Object Inheritance**: CIM helps making taxonomies of management objects by subclassing high-level objects with more domain-specific objects.

- **Dependency definition**: Various dependencies are defined with the help of associations to depict relationships between management objects. CIM takes full advantage of object paradigms by showing complex relationship with associations and labels in a standardized format.

While performing the analysis and evaluation of the GLUE and the CIM models, we found the latter to be the proper candidate fostering our requirements of representing heterogeneous, distributed, and platform-agnostic resources.

6.3.1. Basic Concepts

On the top level, CIM defines three levels of abstraction which represent the 'basic concepts': *core model*, *common model*, and *extension schema*. The core model is a set of abstract classes on which all other models are based. It introduces, among other things, the notion of model scalability and extensibility. Common models are based on the core model and represent concepts like systems, applications, or devices. The extension schema provides room for extensibility and customisation. All three are briefly discussed in next paragraphs.

6.3.2. The Core Model

The CIM core model is the root of all the CIM common models. It represents all objects and relationships that the common models share. The root core entity is the abstract *Managed Element* class. All other CIM classes inherit from it (cf. Figure 6.1), making it the basis of a large collection of associated entities that can be used to describe an IT infrastructure. Since an in-depth description of the core model (and any other CIM model) would exceed the scope of this thesis, we refer the interested reader to Westerinen and Strassner [143], who introduce the core model in detail.

6.3.3. The Common Models

The purpose of the various CIM common models is to encapsulate the characteristics of what the DMTF calls 'management domains'. They represent the DMTF's approach to subdivide the vast amount of managed objects into groups which inherit properties common to all groups from the core model. The overall goal is to define these models independent of any specific implementation or technological approach, which guarantees broad applicability and interoperation.

CIM comprises twelve common models: Database, Event, Interoperability, Metrics, Network, Physical, Policy, Support, User, System, Device, and Application. In the following paragraphs we introduce the last three, since only they are used for the specific model introduced in Section 6.4.1.

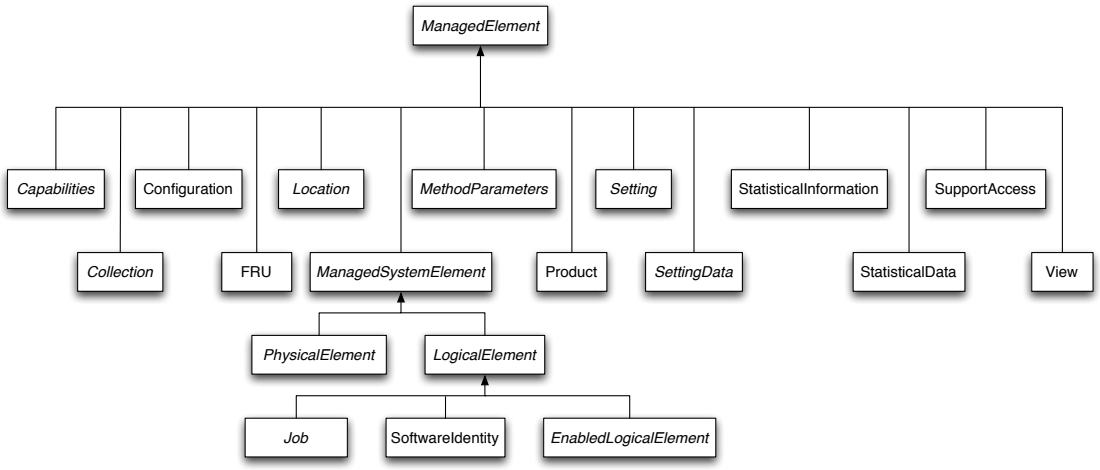

Figure 6.1.: An overview of the root entities of the CIM core model (abstract classes are italicised)

System Model

The *System Model* defines abstract concepts related to a computer system and the aggregation of its components which are meant to compose one single system [38]. In addition, the model describes the functions provided by the particular components, as there are for example file (system, operating system, job, processes, or threads). All different concepts of the System Model are derived from the CIM_System object in the core model.

Device Model

The abstraction of hardware, ranging from lower-level concepts like LEDs or batteries to higher-level concepts like storage systems, is captured by the *Device Model* [37]. In addition to the mere functions of such devices, it also conceptually defines the state and configuration of various components. One important concept to be mentioned specifically is the logical device, which itself is defined in the core model. Its subclasses do not stand for the hardware itself, but the functions provided by a specific device. A logical device is linked to systems as defined by the System Model, which can aggregate other systems the same way as logical devices can aggregate other devices.

Application Model

The CIM *Application Model* has been created to represent software and application management [39]. It allows to model software products from desktop applications to distributed computing or web-based applications. In practice, information system developers create domain-specific instances for their application and populate it with the necessary information for their software. This information, which could refer to software-related bits like memory demand, other software it depends on, or version information, is then consumed by management tools or operating systems. Figure 6.2 shows these concepts and relationships in greater detail.

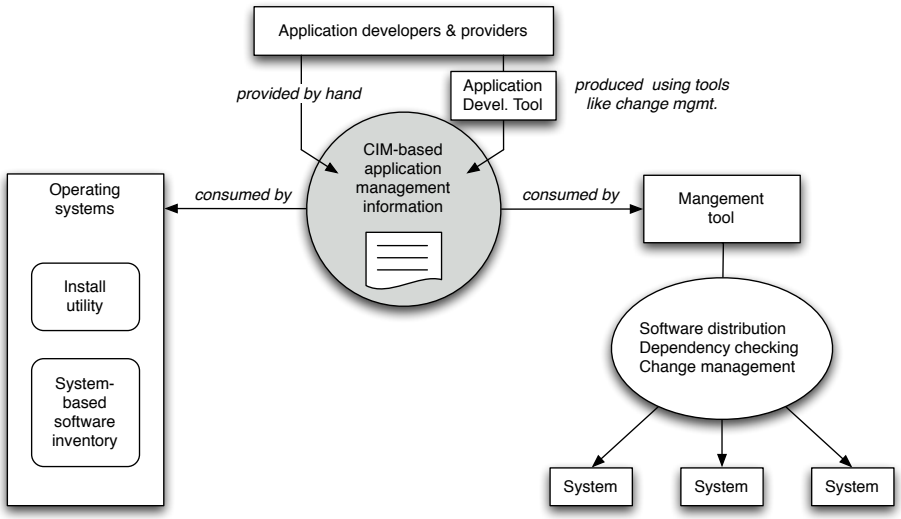

Figure 6.2.: Developing a management-ready application [39]

6.3.4. The Extension Schema

One specific object-oriented design-related characteristic that led to the selection of the Common Information Model is its extensibility [36] (also known as sub-classing or inheritance). This allows product vendors or technology providers to extend CIM in order to model their demands, products, and services.

Extensibility is also one of the requirements for an information model of distributed computing infrastructures (see Section 6.1). We used this capability extensively to realise a CIM-based information model for the UNICORE grid middleware, which we introduce in the section below. The reason to apply extensions to the core or common models is the demand for expressing the specifics related to a certain infrastructure and its management.

6.4. En Route to a Distributed Computing Information System: The UNICORE Information Model

We proposed and implemented an information service called CIS, the *Common Information Service* [94] (cf. Section 9.2), that is built upon a CIM-based information model. This information model emerged from two diploma theses [93, 95] and has been specifically modelled for the UNICORE distributed computing middleware. In addition to the core classes introduced before, additional classes have been derived to fulfil UNICORE-specific requirements.

6.4.1. The UNICORE Resource Model

In contrast to other DCIs, UNICORE uses the same resource model to describe site capabilities and to request resources for the execution of activities. This concept makes the matching of consumer demand and provider capabilities a simple task and it integrates

well with the UNICORE job model [129].

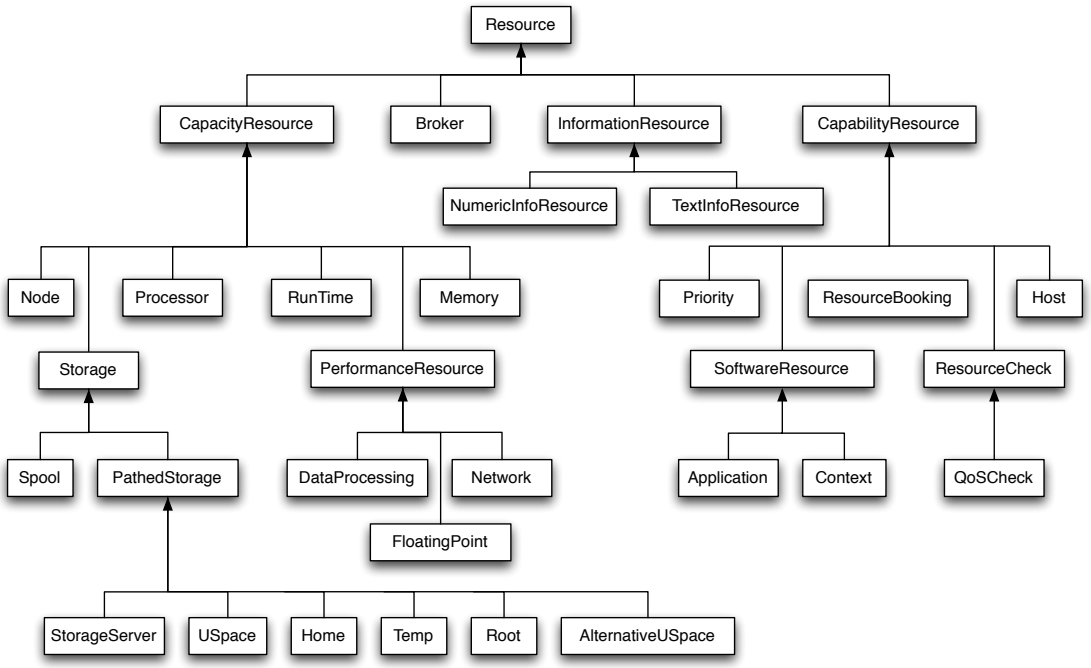

Figure 6.3.: The UNICORE resource model

Figure 6.3 illustrates the UNICORE's hierarchical resource model. It evolves from a core concept of everything being a resource, from which all other resource types are derived. The top-level concepts are:

- **Resource**: The abstract core concept of the UNICORE resource model is the Resource. It captures the kind of information that is inherited by all other resources.
- **Capability Resource**: The UNICORE Capability Resource describes a capability, including its type and its functions, that must be present at a particular site.
- **Capacity Resource**: A Capacity Resource is the a measurable entity which is provided by a site (or requested by an activity). Obvious examples are nodes, processors, or memory.
- **Information Resource**: The UNICORE information model captures the meta-information that is necessary to describe a resource through a concept called Information Resource.
- **Broker**: A UNICORE Broker is a specific site which provides resource information for end-users or schedulers.

6.4.2. Mapping the UNICORE Resource Model to CIM

The new information model which derives from the UNICORE resource model is called UNICORE-CIM model. It uses every possible CIM concept to model UNICORE resources.

Consequently, concepts not present in CIM are modelled through extensions. In addition to core model concepts, the System, Device, and Application Models are exploited. Table 6.1 gives examples of the UNICORE to CIM mapping along with the respective CIM model which is extended for UNICORE customisation. As a result, the UNICORE-CIM model reflects the same semantics as the original UNICORE resource model, but is extensible, portable, and platform-independent. Furthermore, the UNICORE-CIM model is standards-based.

Table 6.1.: Mapping UNICORE resources to CIM classes – Selected examples

UNICORE resource	**CIM class**	**CIM common model**
USpace	UCR_USpace	System Model extension
Home	UCR_Home	System Model extension
Node	UCR_NodeResource	System Model extension
PathedStorage	UCR_FileStorageResource	System Model extension
Memory	UCR_MemoryResource	Device Model extension
Processor	UCR_ProcessorResource	Device Model extension
Application	CIM_SoftwareElement	Application Model

There are further concepts that have been selected to create an information model capable of serving service-oriented infrastructures, like *Service* and *ServiceAccessPoint* (both included in the CIM core model). These additions to the UNICORE-CIM model enable providers to offer their resources as services, thus advertising type, features, and access point through an endpoint reference.

Based on the requirement to create a standards-based and interoperable information model for distributed computing infrastructures, we developed a CIM-based information model. Although it is specifically tailored to the needs of UNICORE, the outlined approach can also be applied to other DCIs. The Common Information Model has the capabilities to capture the managed entities of any such infrastructure, allowing extensions to fill potential gaps. Unmanageable entities of an IT infrastructure, however, are not covered due to the approach taken by the DMTF. From a service-level management point of view this is irrelevant, but for a general information model it is something to take into consideration.

7. The Service-Level Agreement Model

In this chapter, we describe essential foundations for the understanding of service-level agreements and the way they fit into the big picture of service-level management. We also examine the Web Services Agreement standard which is the basis for the core contribution of this thesis. Furthermore, we introduce contributions to SLA negotiation, leading to the Web Services Agreement Negotiation specification currently being standardised.

An electronic agreement provides a contractual frame in which customer relationships are managed. This evolved to the common notion of service-level agreements throughout industry during the previous two decades. SLAs are used to map business-level objectives to services and allow for proper monitoring and reporting, a capability not only limited to business cases, but also relevant to any consumer-provider relationship in which the automated management of SLA is a key requirement. This includes the initiation of an SLA between two parties, the provisioning of the service, and the monitoring of the SLA while a service is provided until the completion or termination of a service delivery.

According to [46], the TeleManagement Forum's SLA Handbook defines a service-level agreement as:

'[a] formal negotiated agreement between two parties, sometimes called a service level guarantee [...]], it is a contract (or part of one) that exists between the service provider and the customer, designed to create a common understanding about services, priorities, responsibilities, etc.'

7.1. Life-Cycle of a Service-Level Agreement

A structured approach towards the management of the different phases an individual service-level agreement passes during its lifetime is essential. This facilitates the implementation of a service-level management system and is applicable independent of the domain the SLA is actually applied to. Although there is no universal definition of an SLA life-cycle, the one defined by the TeleManagement Forum [46] is commonly applied in the distributed computing area [145] or is used as the basis for refinements and specialisations.

Figure 7.1 shows this life-cycle, which consists of the six phases *Development*, *Negotiation*, *Implementation*, *Execution*, *Assessment*, and *Termination*. These phases are passed from service development over service provisioning to the termination and decommissioning of a service governed by an SLA. Depending on the application scenario, phase

transitions may not be visible, or phases may be even omitted, but for the general case this life-cycle is applicable. In the following the different phases are described.

7.1.1. Service and SLA Template Development

In the first phase of the SLA life-cycle, the service and the respective *SLA Template(s)* are developed. In general, the provider has the service already in place, for example a compute resource offered via a DCI or some simulation software offered as a service, and constructs the SLA based on the existing service. This may lead to the notion of an SLA as an added-value feature to be included in the development process of the service becoming the common approach once SLAs will be ubiquitous in such infrastructures.

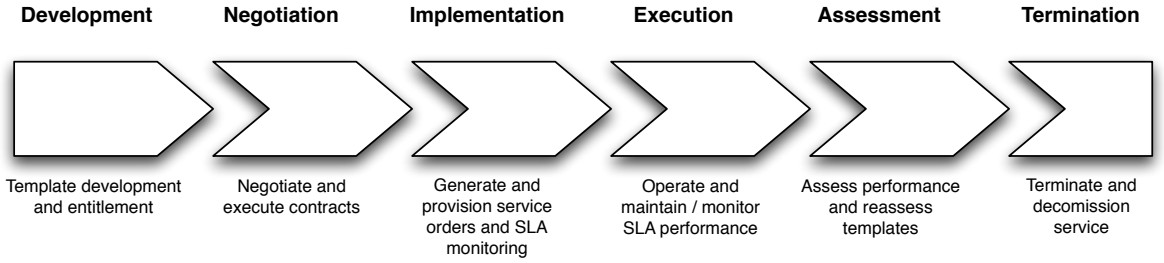

Figure 7.1.: The SLA life-cycle according to the TeleManagement Forum [46]

According to Lee and Ben-Natan [84], the SLA Template itself should contain, from a 'purely contractual standpoint', the following items: the agreement definition, i.e. parties, contract terms and conditions, and service access points, the service definition, performance/metric definitions, measurement definitions, correction definitions and reconciliation definitions. In the DCI landscape, however, the contractual requirements are sometimes relaxed (e.g. in academic environments where the actual costs are not always billed) or the focus is on other or additional service characteristics. In Section 7.2.2, we elaborate on different SLA (Template) specifications, which result from the divergent requirements of different stakeholders. The SLA Templates are used to offer the service to customers and therefore to lay the foundation for a common understanding of the service quality. Between the service and its templates there may be a one-to-many relationship as a provider may offer the same service under different conditions.

There is an ongoing discussion whether the service and SLA Template development should be part of the life-cycle since this phase may, depending on the service provided, only occur once for numerous SLA instances and therefore for multiple passes of the SLA life-cycle. On the other hand, however, there are several application scenarios within the DCI community which include this phase at every pass. One example is the dynamic generation of services through orchestration and the respective dynamic development of SLAs on a case to case basis. Although this development will most likely follow fixed patterns and make use of draft SLA Templates, it justifies the inclusion of this phase into the SLA life-cycle.

7.1.2. Contract Negotiation

Once a service customer has retrieved, through whatever means necessary, an SLA Template (i.e. a service offer) that fulfils her requirements, the customer and the provider need to reach a binding agreement regarding the service to be delivered. To get there, consumer and provider negotiate the service delivery terms governed by the SLA Template. Regarding negotiation, a number of models and specific protocols for distributed computing infrastructures exist, which are described in Section 7.5.

The result of the negotiation phase is either an agreed upon and ready-to-execute SLA, i.e. a contract, or nothing. In the latter case, the consumer may look for other providers to negotiate with, change her requirements, or wait for a later point in time where the preconditions for a negotiation may have changed.

7.1.3. Implementation

The implementation of an SLA includes the instantiation, configuration and provisioning of the service instances which are needed to fulfil the contract between the service consumer and the service provider. In contrast to the phases described before, the implementation phase is usually performed per service instance, including all means to monitor, measure, and report performance. This normally does not only include the service(s) defined in the SLA, but depending on performance metric or measurement definitions also auxiliary services that provide the necessary monitoring information for the service provider to control the service execution.

7.1.4. Service Execution

During service execution the service provider has to deliver what has been contractually agreed upon as the result of the negotiation process: the service according to the service-level agreement. This includes, as mentioned before, the operation of not only the service to be delivered, but also auxiliary services and functions necessary to fulfil the SLA. It is essential during this phase to monitor the performance of the service, measure and record it, and, in parallel, assess the performance as described in the next phase. In case the SLA is breached, depending on the kind of service provided, monitoring and assessment may result in immediate actions to provide the service agreed upon (e.g. in case of a downtime of a data streaming service during an important presentation of a visualisation software) or the data gathered is just recorded for reporting purposes (which is often the case for academic computer services which operate on a best effort policy).

7.1.5. Performance Assessment

The assessment of the service's performance is executed during its execution (although post-termination assessment is implicitly part of the decommissioning phase) and serves the service provider and the service consumer, who both want their business objectives to be met. For the service provider, this implies the control over executing services while the service consumer demands status information about the currently delivered quality of service. In both cases, this includes monitoring, evaluation and potentially the application of corrective measures. In addition, this phase may result in a re-assessment of the SLA

Templates in such cases where e.g. the quality of service cannot be provided as advertised in the template, or where the provider may see potential revenue improvements.

7.1.6. Service Termination and Decomissioning

The final phase in the SLA life-cycle includes the termination of the service (and the auxiliary services) as well as its decommission. The service termination is due to a certain condition, which is an arbitrary event that triggers it. Most likely it is either the fulfillment of the service or some SLA violation during the service runtime. In both cases, the service provider decommissions the service, which includes actions like final assessment, accounting of the service execution, or billing for the provided service.

7.2. Development of Service-Level Agreements

The first phase of the SLA life-cycle deals, as mentioned above, with formulating the service offer in terms of an SLA Template. This is then used to advertise the service to potential customers, describing all necessary details about the forthcoming electronic contract. SLA Templates are normally developed by service developers, often supported by other parties within a company, like the sales or the legal department.

7.2.1. Content of a Service-Level Agreement

According to the definition given in the introduction, an SLA must contain all information necessary for service consumers and providers to achieve a common understanding of services, priorities, responsibilities, et cetera. This definition is not very concrete when it comes to the actual content of an SLA, i.e. the specification of the terms on which provider and consumer agree during the negotiation. But since the content of a service-level agreement is highly application-domain-dependent anyway, a uniform definition is neither possible nor desired. This is confirmed by the existence of a number of different SLA specifications that are actually used within the distributed computing landscape and which are introduced in the following paragraph.

7.2.2. Existing SLA Specifications

A large variety of SLA specifications already exists analogically with the different domains they are applied to. Even in the specialised area of distributed computing we see a good dozen specifications aiming at different use cases, projects, or infrastructures.

It is not the purpose of this thesis to deliver an in-depth description of all existing specifications. We therefore refer mainly to existing literature, compare the different contributions, and come up with a rationale for the choice we made for our work: the *Web Services Agreement* (WS-Agreement) specification[1].

[1] Please note that a description of WS-Agreement is not included here as it is introduced in detail in Section 7.4.

Project Developments

We first list SLA specifications which we do not consider for this thesis as they represent specific project developments that have not been created as stand-alone models for greater communities:

- Akogrimo [87].
- AssessGRID [35].
- BEinGRID [34].
- BREIN [104].
- TrustCOM [148].

Parkin et al. describe and compare them [107] together with the NextGRID SLA model, which we consider below.

Discontinued Efforts

Discontinued efforts, which have not created significant uptake but nevertheless came up with concepts worth mentioning, are the *SLAng language*, the SLA model from the European project SLA@SOI [75], and the *NextGRID SLA model*. SLAng has been developed by the University College London to '[associate] QoS targets (e.g. performance, availability, reliability, etc.) with identifiable Application Service Provider (ASP) architectural elements' [82] using service-level agreements. The objective of SLAng is to provide a language which supports the whole SLA life-cycle for ASP business scenarios that employ component-oriented middleware. The SLA model specified by SLA@SOI introduces a syntax that can be used to formally describe SLAs and their respective templates in a machine-processable and language-independent manner.

Another such example is the the NextGRID SLA model. Wieder and Yahyapour provide a thorough description of the model [145]. Figure 7.2 depicts an overview of the basic concepts of the NextGRID SLA, as there are SLA Identifier, SLA Context, SLA Terms, and Signatures.

Interesting exemplary details of NextGRID's SLA model, most likely the result of the industrial focus of the project [96], are SLA terms like *Success Criterion* or *Service Management*, and the concept of *Signatures*.

The Success Criterion of every SLA term is defined through a value (or a range of values) and the respective unit. An example for a success criterion could be the waiting time (value) of a Grid job measured in seconds (unit) or the number (value) of available software licenses (unit). Whether a Success Criterion is met or not is defined through the *Metric* element. To achieve this, the address of a measurement service is determined, which monitors the Success Criterion's value and measures its success. Regarding the above-mentioned examples this would e.g. guarantee that the waiting time is not longer than 3600 seconds or that at least five licenses of some software are available.

Through the Service Management term it is possible to specify auxiliary services which are used to manage the service that is subject of the agreement. In the NextGRID context this implies the indication of the service address. Furthermore, it is distinguished between

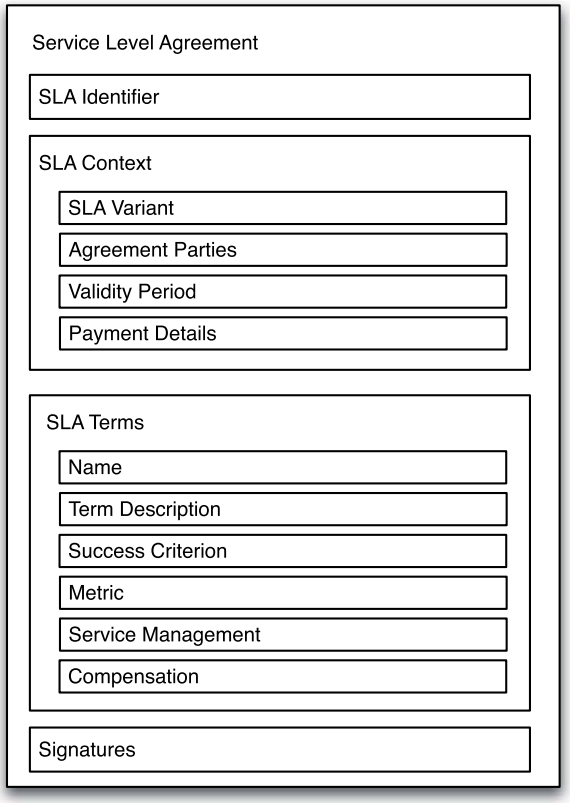

Figure 7.2.: The NextGRID SLA [145]

resource provision services, customer escalation services, and more general management services that can be used to manage SLA terms.

To prevent that a service-level agreement is forged, once it has been agreed upon, it can be signed by an arbitrary number of parties. Which party actually signs the SLA depends on the business convention between the service provider and the service consumer. The signers may not necessarily be the provider and the consumer, but typically both parties sign the SLA. Other usage scenarios may foresee to have a trusted third party, e.g. an SLA broker, which also signs the agreement.

Web Service Level Agreement Language

IBM has developed the *Web Service Level Agreements* (WSLA)[2] framework for specifying and monitoring service-level agreements for web services [77]. The initial version was published in 2001 as part of IBM's 'Emerging Technologies Toolkit'. A final revision was published in 2003. Although WSLA is not continued as far as the specification itself is concerned, the approach still attracts researchers and developers [102, 115].

The framework is able to measure and monitor the QoS parameters of a web service and to report violations to the parties specified in the SLA. In a web service environment,

[2]Web Service Level Agreements (WSLA) Project – SLA Compliance Monitoring for e-Business on demand, last visited: January 25, 2013. http://www.research.ibm.com/wsla/.

services are usually subscribed dynamically and on demand. Therefore, automatic SLA monitoring and enforcement helps to fulfil the requirements of both service providers and consumers. WSLA provides a formal language based on XML Schema to express SLAs and a runtime architecture, which is able to interpret this language. This architecture comprises several SLA monitoring services, which may be outsourced to third parties (supporting parties). The WSLA language allows service customers and providers to define SLAs and their parameters and specify how they are measured. The WSLA monitoring services are automatically configured to enforce an SLA upon receipt.

The SLA management life-cycle of WSLA consists of five distinct stages:

- **Negotiation/Establishment**: In this stage, an agreement between the provider and the consumer of a service is arranged and signed. An SLA document is generated.
- **SLA Deployment**: The SLA document of the previous stage is validated and distributed to the involved components and parties.
- **Measurement and Reporting**: In this stage, the SLA parameters are computed by retrieving resource metrics from the managed resources and the measured SLA parameters are compared against the guarantees defined in the SLA.
- **Corrective Management Actions**: If a service-level objective has been violated, corrective actions are carried out. This could be opening a ticket or automatically communicating with the management system to solve potential performance problems.
- **SLA Termination**: The parties of an SLA can negotiate the termination the same way the establishment is done. Alternatively, an expiration date can be specified in the SLA.

Although the WSLA life-cycle phases carry different names than those described in Section 7.1, and the initial SLA development phase is missing, WSLA follows the same pattern and therefore represents the same approach towards service-level management.

The SLA* Model

In 2010, Kearney et al. published the SLA* syntax for service-level agreements [76]. They explain the need for it as follows:

> Historically, SLA* was developed as a generalisation and refinement of the web-service specific XML standards: WS-Agreement, WSLA, and WSDL. Instead of Web services, however, SLA* deals with services in general, and instead of XML, it is language independent. SLA* provides a specification of SLA(T) content at a fine-grained level of detail, which is both richly expressive and inherently extensible: supporting controlled customisation to arbitrary domain-specific requirements.

SLA* is an outcome of the SLA@SOI[3] project and is an intrinsic part of its service-oriented infrastructure. Domain-independence was especially important to support the various

[3] SLA@SOI – Empowering the service industry with SLA-aware infrastructures, last visited: January 25, 2013. http://sla-at-soi.eu.

business domains SLA@SOI has as targets and to integrate with legacy software that already exists there. Language-independence and a broader view on services than taken by its competitors were also essential for the specification of SLA* to prevent semantic restrictions by using e.g. XML and to be able to include e.g human services respectively.

7.3. Rationale for choosing WS-Agreement

We chose WS-Agreement as the SLA specification for the work presented in our thesis for the following five reasons:

- domain independence [88],
- extensibility [6],
- standard compliance [6],
- wide spreading [123], and
- tool support [11].

The SLA* syntax (cf. previous paragraph) would have been the next best candidate, but, owing to the fact that it was released in 2010, it was not available when the decision about an SLA specification was taken. It may still lack wide up-take and it is not standard-compliant, but language independence and the inclusion of non-electronic services are appealing attributes, which WS-Agreement lacks.

The project developments listed in the previous section are mainly using WS-Agreement in combination with WSLA, but have not been short-listed due to their alignment to specific project requirements. Although this is not true of the SLA model developed by the NextGRID project, it was not chosen either, since it is discontinued and the model is not even accessible any more via the project web site. This is not the case with SLAng and WSLA, but in comparison to WS-Agreement these two lack standard-compliance and tool support.

Although WS-Agreement provides the best foundation for scheduling in distributed infrastructures, we have to mention an issue which could be proven disadvantageous[4]: WS-Agreement is tightly-coupled to XML Schema and uses artefacts from the following Web service standards: *WS-Addressing*[5] (WSA), *Web Services Resource Framework*[6] (WSRF), and *Web Services Description Language*[7] (WSDL). We don not regard this as a disadvantage per se; the semantics of XML Schema, however, are restrictive in some cases [76]. Furthermore, significant parts of the community do not acknowledge the notion of stateful Web services, which are core to WSRF, and propagate the 'RESTful' approach. Therefore the support for WSRF, and hence WS-Agreement, in terms of technologies and tools could be improved.

[4] It is often argued that WS-Agreement is overly complex and difficult to use. However, our experience with service-level management in general and service-level agreements in particular shown that this is no criterion to base a decision on, since all of the specifications mentioned here require significant investments using them.

[5] Web Services Addressing, last visited: January 25, 2013. http://www.w3.org/Submission/ws-addressing/.

[6] OASIS Web Services Resource Framework (WSRF) TC, last visited: January 25, 2013. https://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

[7] Web Services Description Language (WSDL) 1.1, last visited: January 25, 2013. http://www.w3.org/TR/wsdl.

7.4. Introduction to Web Services Agreement

It is the objective of the WS-Agreement specification [6], defined by the GRAAP Working Group[8] of the Open Grid Forum, to provide a domain-independent and standard way of establishing and monitoring service-level agreements. The specification comprises three major elements: (i) a description format for agreement templates and agreements; (ii) a basic protocol for establishing agreements, and (iii) an interface specification to monitor agreements at runtime.

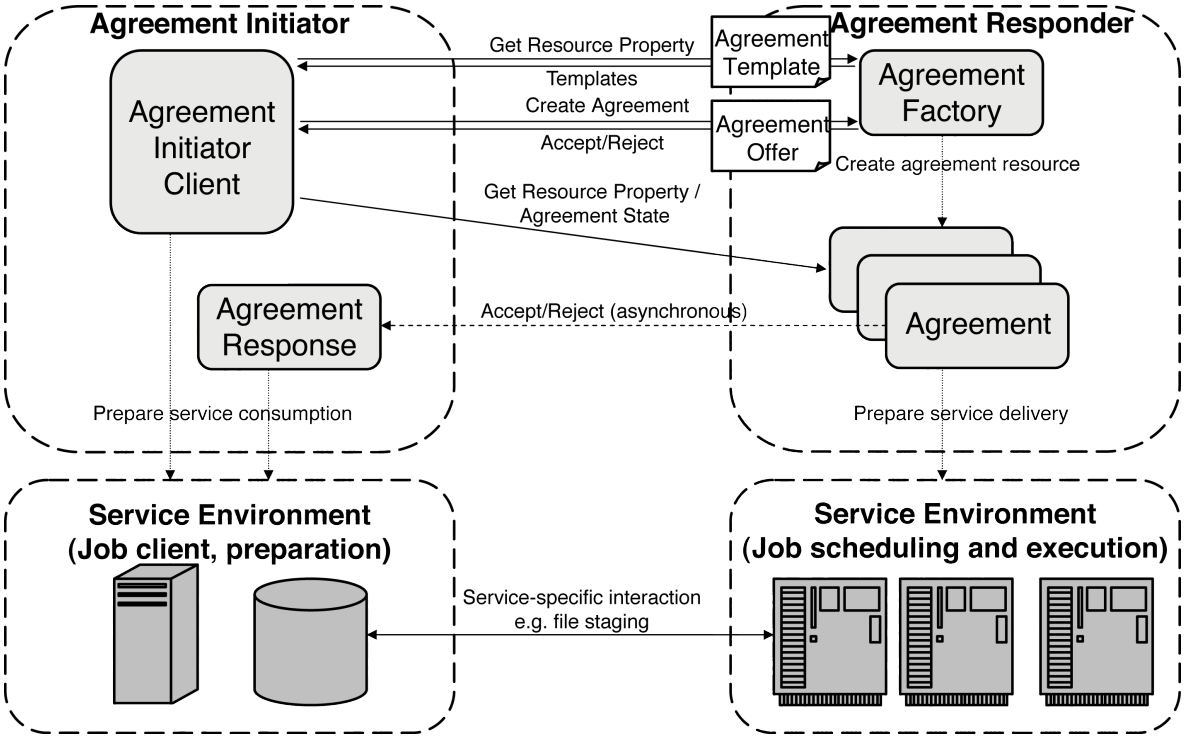

Figure 7.3.: Concepts and interfaces of WS-Agreement [88]

Agreements according to the specification are bilateral and set up between the *Agreement Initiator* and the *Agreement Responder*, as shown in Figure 7.3. These roles are independent of the roles of service providers and service consumers, and the domain in which jobs or other services are performed. An agreement defines a dynamically-established and dynamically-managed relationship between the two parties. The object of the relationship is the delivery of a service, e.g. the execution of a job or the reservation of compute resources by one of the parties within the context of the agreement. The terms of this delivery are set by negotiating the respective roles, rights and obligations of the parties. The agreement may not only specify functional properties for identification or creation of the service, but also non-functional properties of the service, such as performance or availability.

Figure 7.3 also outlines the main concepts and interfaces of the WS-Agreement specification. In the chosen example, the Agreement Responder is a service provider, the Agree-

[8] Grid Resource Allocation Agreement Protocol WG (GRAAP-WG), last visited: January 25, 2013. https://forge.gridforum.org/sf/projects/graap-wg.

ment Initiator the service consumer. An Agreement Responder exposes an interface of an *Agreement Factory*, which offers an operation to create an agreement and an operation to retrieve a set of agreement templates proposed by the agreement provider. Agreement templates are potential agreements which include fields that still need to be filled in. Templates describe the outline of a potential agreement. They help an Agreement Initiator to create agreements that the agreement provider can understand and accept. The *Create Agreement* operation returns 'accept' or 'reject' if a synchronous reply is expected. Otherwise, in case of a longer decision-making process on resource allocation, the service responder can convey the decision to an *Agreement Response* interface that the initiator exposes. If the Create Agreement operation succeeds, an agreement instance is created. The agreement instance exposes the terms of the agreement as properties that can be queried. In addition, the runtime states for the agreement as a whole and its individual terms can be inspected by the Initiator. All interfaces exposed by the parties, Agreement Factory, *Agreement*, and Agreement Response are resources according to the Web Services Resource Framework (WSRF)[9].

Upon acceptance of an agreement, both, service provider and service consumer have to prepare for the service, which typically depends on the kind of service which is subject to the agreement. For example, a service provider schedules a job that is defined in the agreement. A service consumer will make the stage-in files available as defined in the agreement. Further service specific interaction may take place between the parties governed by the agreement.

The WS-Agreement specification defines the content model of, both, agreements and agreement templates as an XML-based language (cf. also Figure 7.4). Structurally, an agreement consists of a name, a context section, and the agreement terms. The agreement context contains definitions like parties involved and their roles in the agreement. The agreement terms represent contractual obligations and include a description of the service as well as the specific guarantees given. A *Service Description Term* (SDT) can be a reference to an existing service, a domain specific description of a service (e.g., a job description, a data service, etc.), or a set of observable properties of the service. Multiple SDTs can describe different aspects of the same service. A Guarantee Term, on the other hand, specifies non-functional characteristics in service level objectives, an optional qualifying condition under which objectives are to be met, and an associated business value specifying the importance of meeting these objectives.

The WS-Agreement specification only defines the top-level structure of agreements and agreement templates. This outer structure must be complemented by means of expressions suitable for a particular domain. For example, a guarantee term is defined as comprising the scope of an element, qualifying condition, service level objective, and business value. There are no language elements defined to specify a service level objective. Parties have to choose a suitable condition language to express the logic expressions defining a service level objective. Agreement templates contain the same content structure as agreements but add a constraints section. This section defines which content of a template agreement can be changed by an agreement initiation and the constraints which must be met when filling in a template to create an *Agreement Offer*. A constraint comprises a named pointer to an XML element in the context section or the term section of the

[9] OASIS Web Services Resource Framework (WSRF) Technical Committee, last visited February 26, 2011: http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf.

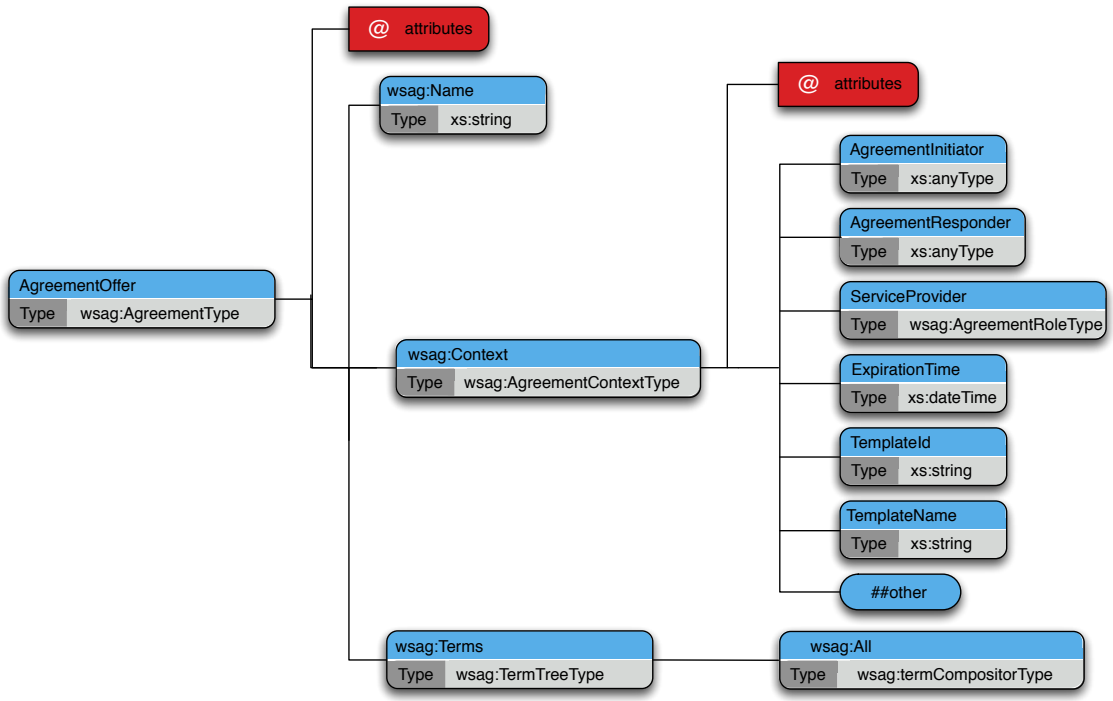

Figure 7.4.: A high-level view of the WS-Agreement schema

agreement and a constraint expression defining the set of eligible values that can be filled in at this position. For example, an Initiator can choose between several options of a job or can specify the location of a stage in file.

7.5. Negotiation of Service-Level Agreements

It is the purpose of the negotiation phase within the SLA life-cycle (as described in Section 7.1) to reach an agreement between customer and service provider. In the context of negotiation, not only technical issues are important, but also legal ones need to be considered, like responsibilities, payment details, compensations, etc. Negotiating multiple criteria like the aforementioned can be a costly process, which may turn out to be uneconomical. For this reason, McKee et al. [91] propose to balance the advantages and disadvantages of negotiation carefully and execute complex negotiation only where its cost are justifiable.

This thesis is about scheduling with electronic contracts and service-level agreements are an intrinsic part of the concept presented in Section 4. Nevertheless, contributions to the WS-Agreement specification have not been in our focus, except for the publication of an experience document about the usage of the standard [12][10]. Central to our research is the application of WS-Agreement to various scheduling-related usage scenarios, which are listed in Section 8, and advancements in SLA negotiation [65].

[10]Such a document is required to make an OGF proposed recommendation a full recommendation, i.e. an OGF standard.

Before we approach the actual contributions, we introduce a number of negotiation models and already existing SLA negotiation protocols. They serve as the foundation to introduce a multi-step negotiation approach we proposed en route to the WS-Agreement Negotiation protocol specification.

7.5.1. Negotiation Models

Negotiation models are descriptions of how a contract between one or more sellers of goods, resources or services and one or more buyers of such products can be established. The major part of establishing a contract is to reach an agreement (i.e. a common understanding) on the terms of the contract. This is accomplished by negotiations between the buyer and the seller, who have different requirements and constraints. Negotiations come in different forms and involve diverse procedures. In this section, the models most relevant to service-related research and development are introduced.

Accept/Reject

Probably the simplest negotiation protocol is the 'Accept/Reject' model. In this protocol the seller advertises his goods to potential customers. Buyers select the products they would like to purchase and announce their choice to the seller. The seller then decides on whether to accept or reject the deal. In this process, none of the parameters of the deal are negotiable. The seller can only accept or reject whatever choice the buyer has made. This implies that an offer usually is identical with the seller's advertisement, as otherwise no agreement is reached. As this is the common procedure in self-service supermarkets, this protocol is also termed the 'supermarket approach'.

Formally, the displaying of goods is considered an 'invitation to treat' in the context of contract law [19]. The buyer then makes a binding offer to the seller, who can accept or reject it. The result of this negotiation is either a contract for exchanging goods for money, or nothing.

Mapped to the SLA world, an SLA template is the equivalent of an invitation to treat, as it is a (more or less) public display of the seller's offer. The buyer then submits an SLA offer. If this is accepted, the negotiation process results in an SLA which is a binding contract agreed upon by both parties. The contract usually stipulates that the described services are provided to the customer who in turn has to provide some form of payment. As noted before, none of the parameters is negotiable. Translated to SLAs, this means that none of the terms can be changed and the SLA template, the offer, and the final SLA all contain the same non-functional properties.

Discrete Offer

The discrete offer model, as depicted in Figure 7.5, follows the so-called 'take-it-or-leave-it' approach [64]. Sometimes also referred to as 'one-phase-commit' model, it assumes that service providers have a pre-defined set of service offers, which are accessible to customers. Once a customer receives a service offer, this negotiation model foresees that he agrees to one of the offers but cannot to modify any of the terms.

This model is quite similar to the accept/reject model introduced before. The main difference is that the roles of who provides a binding offer and who finally accepts or rejects an

SLA are reversed. For the accept/reject model, the consumer makes a binding offer and the provider decides on acceptance/rejection. In the discrete offer model, the provider makes a binding offer and the consumer accepts or rejects it.

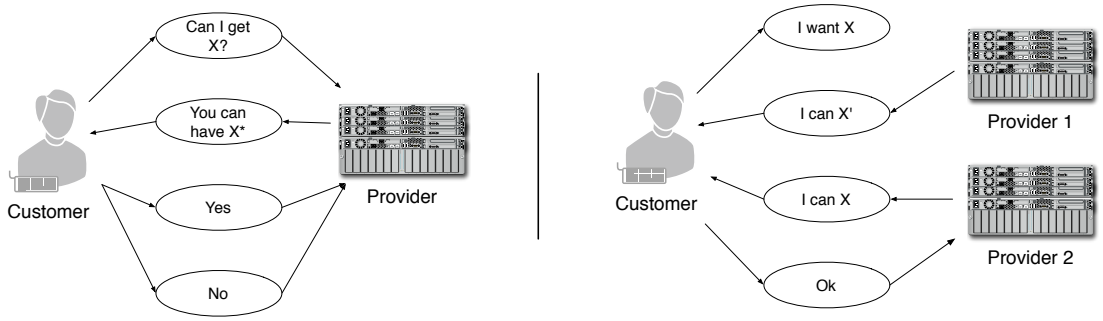

Figure 7.5.: The Discrete Offer model (left) and the Invite-Tender model (right)

Invite-Tender

The invite-tender model moves away from the accept/reject protocol, in which consumers search for appropriate service providers. Here, the service consumer specifies his requirements and publishes this specification to the 'outside world' to be found by service providers. Providers interested in providing the needed service can send tenders on this invitation. The tenders are then reviewed by the consumer, potentially adapted and sent back to the provider who can send in new tenders. In general, it follows a similar approach to the multi-phase (n-phase) negotiation and it could be realized by a symmetric protocol. Fig. 7.5 shows a customer following this model with two service providers sending in tenders. In the example, service provider 2 offers exactly the desired service so that no further negotiation needs to take place.

Multi-phase (n-phase) negotiation

As mentioned before, the discrete offer model is rather inflexible and therefore not sufficient in all business cases. More flexibility can be achieved by extending the discrete offer approach to several rounds of negotiation. In the beginning, the concept is similar to the discrete offer model with a request and a potential offer following. Now, instead of either agreeing to or rejecting the offer, the customer can adapt the offer and send it as a counter-offer to the provider, who can also modify the counter-offer and send it back (as shown in Figure 7.6).

This process happens repeatedly and might run indefinitely, which is the biggest concern of the critics of this model. However, when realized in a proper way, there are mechanisms and decision points for both customer and service provider to stop this negotiation, either with or without an agreement.

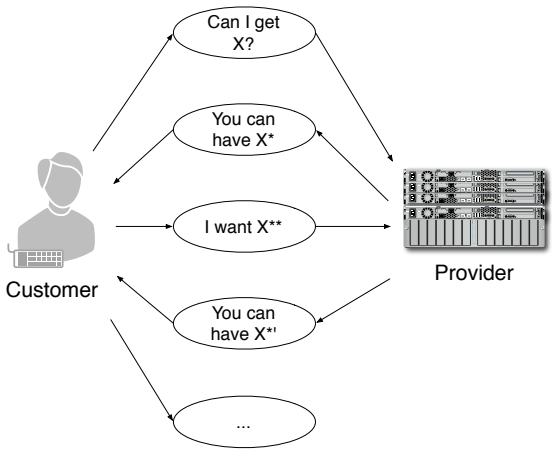

Figure 7.6.: Multi-phase negotiation

English Auction

The model of the English Auction realizes a first-price, open cry, ascending auction [100]. It starts with a price set by the auctioneer, followed by bids that constantly increase the price. If no further bid is received within a certain length of time, the highest bidder is awarded the goods. This process ensures that the price is increased until only one participant is left, who pays the highest price.

Dutch Auction

Unlike the English Auction, the Dutch Auction starts with a high price, which then is decreased subsequently [85]. Lowering the price continues until one of the participants is willing to pay the current price, or a pre-defined price is reached (the minimum acceptable price as defined by the seller in advance), in which case the item is not sold. In principle, the Dutch Auction is a very simple and quick protocol, as it needs only one bid to finish the auction.

Vickrey Auction

Vickrey proposed an adapted auction model, based on the principle of hidden offers: the Vickrey Auction model [137]. Similar to the procedure in the English Auction, the participant with the highest bid wins. However, in this adapted form, he only has to pay the second highest price. The advantage of this model is that the participants are encouraged to bid a realistic price instead of bidding above or below their real valuation.

7.5.2. SLA-related Negotiation Protocols

This section introduces specifications of computer-processable models and communication protocols that can be used to realize certain negotiation models with the help of machines. For making SLA negotiations accessible to computers, service-level agreements must be encoded in some electronic format that can be processed by a computer. Furthermore,

the communication between the negotiation partners must be defined in an unambiguous way. Depending on the desired degree of automation, other parts of the models need to be specified as well, for example computer-'understandable' languages for describing the terms inside SLAs.

The specifications introduced in this section cover the automation of the negotiation to different degrees. All of them are designed to be used generically, meaning that they are not bound to a particular application domain but can be used for any application. Using the specifications for a particular application scenario nevertheless requires additional artefacts, most commonly descriptions of properties of the targeted domain.

WS-Agreement Protocol

Although not primarily aimed at being a negotiation protocol specification, WS-Agreement is mentioned here as it offers basic SLA creation mechanisms (cf. Section 7.4) and as its SLA structure definition is used by other activities to realise more complex forms of SLA negotiation [67].

WS-Agreement defines a simple interface for establishing SLAs [6]. The interface allows one party to send an SLA offer to another party, who can then accept or reject it. Through this, WS-Agreement enables parties to use either the accept/reject or the discrete offer model as described before.

The WS-Agreement protocol supports these negotiation models through a simple protocol, which is depicted in Fig. 7.7. First, the *AgreementInititator* starts off with sending a *getResourceProperties* message, on which the *AgreementResponder* returns a set of agreement templates representing services including their descriptions, properties, guarantees, and constraints.

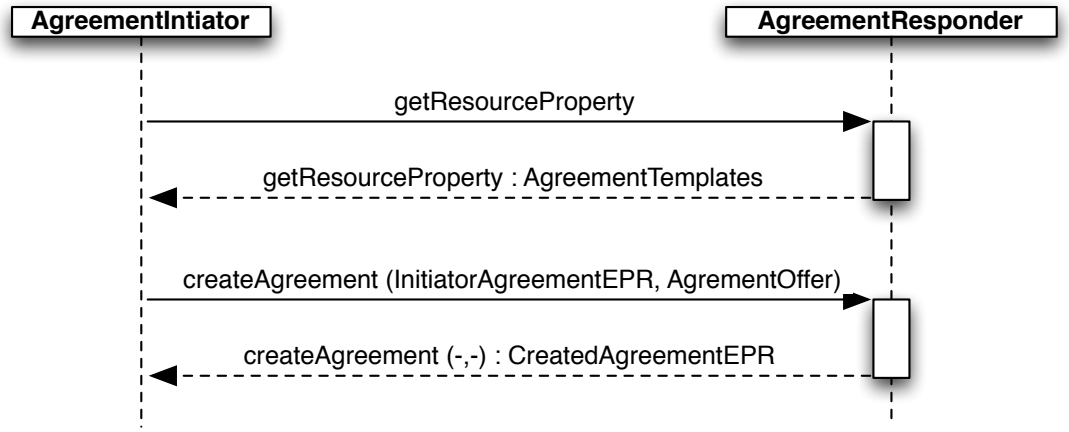

Figure 7.7.: WS-Agreement negotiation protocol (depicting the 'Accept' case; see also 7.3)

Assuming that any of the services fulfils the requirements of the AgreementInitiator, which could be for example a scheduler, it sends a *createAgreement* message to indicate the offered agreement. It is then the AgreementResponder which actually accepts the final agreement by returning its EPR or which ends the negotiation with a fault message indicating the rejection of the offered agreement.

WS-Negotiation

WS-Negotiation is an XML-based language, which was proposed by Hung et al. as a basis for agreement establishment between customers and service providers [68]. The specification is split into three parts: Negotiation Message, Negotiation Exchange, and Negotiation Decision Making. The Negotiation Message part describes the format of the exchanged information between customer and service provider. Even though planned, an enhancement of the message description with definitions of schema and semantics has not yet been performed.

Message exchanges are following the Negotiation Protocol, which covers bilateral multi-phase negotiations. The rules provided for negotiation define the protocol to be a repeated exchange of offers and counter-offers, i.e. a multi-phase negotiation model is followed.

Finally, the Negotiation Decision Making part represents decision processes that are based on the strategies of the individual users. The strategies are aligned with each negotiation participant's preferences and by that they are specific to each participant and therefore not defined in the specification.

Service Negotiation and Acquisition Protocol – SNAP

The SNAP protocol has been published by Czajkowski et al. [29] to present a protocol for remote management of service level agreements across the borders of different resource providers. The main focus of this work is on enabling resource reservation and provisioning by negotiating single service level agreements across multiple administrative domains and by that referring to different resources. Within the SNAP concept, the cross-domain issue is addressed by the introduction of three types of service level agreements:

- **TSLAs**: Task Service Level Agreements cover the execution of an activity (task). Within this construct information about service steps and resource requirements is contained.
- **RSLAs**: Resource Service Level Agreements cover the rights to consume a resource but they do not not necessarily specify what the resource will be used for. Here, the resource is characterized by its abstract service capabilities.
- **BSLAs**: Binding Service Level Agreements cover the use of a resource for a task. This task is then either defined directly by a TSLA or a unique identifier, which allows its application to a RSLA.

SNAP was published in 2002, but did not find the desired acceptance in the community. There were some basic implementations, e.g. one using a three-phase commit protocol by Haji et al. [61]. The biggest weakness of SNAP, however, is its generality, as this makes an implementation rather difficult [116].

The Contract Net Protocol

Contract Net is a protocol used in multi-agent systems [126]. It supports distributed problem solving by task sharing between several participants. Once a big task needs to be completed, the problem is broken down into different sub-tasks, which are distributed to

other participants. In this model, an invitation to tender is generated for each of the subtasks. Based on this, bids are received from potential contractors (agents) and the winning ones get the tasks to execute.

FIPA Contract Net Protocol

The Foundation for Intelligent Physical Agents (FIPA, a standards body in the agent-technology area) has proposed a specification on top of the Contract Net Protocol as depicted in Figure 7.8. The FIPA Contract Net Protocol (CNP) [23] foresees a single round of bids. Once a deadline has been reached, all bids are evaluated and the winners get the tasks.

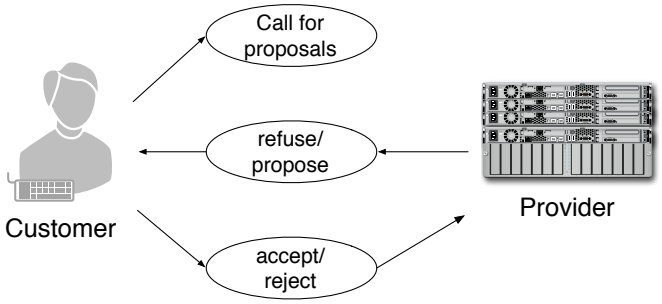

Figure 7.8.: The FIPA Contract Net Protocol

FIPA Iterated Contract Net Protocol

An adaptation of the CNP is the FIPA Iterated Contract Net Protocol (ICNP) [22], which realizes an approach allowing for several bids by one bidder. Through this, it is possible to have several bidding rounds, one after the other (depending on the pre-defined values of the auctioneer).

The Combinatorial Contract Net Interaction Protocol

Another adaptation of the Contract Net Protocol is the Combinatorial Contract Net Protocol [74]. This protocol was developed and implemented in 2009 within the context of the BREIN project. It extends the CNP to support reverse combinatorial multi-attribute auctions over multiple tiers, specifically BREIN supply chain levels. The protocol allows the evaluation of subcontracts to enable the creation of better binding proposals. The protocol is labelled ‘combinatorial’ as it provides three different combination possibilities: Combination of tasks, of attributes and of tiers. Combination of tasks allows to provide offers for not only one, but several tasks. They can be described with AND or XOR operators, the latter of which allows the customer to accept only one part of the tasks, not necessarily the complete proposal. The combination of attributes was introduced to allow for multi-attribute auctions with an enhanced winner determination. Finally, the combination of tiers enables the consideration of dependencies across different levels, especially for the purpose of subcontracting.

7.5.3. An Extension of WS-Agreement towards Dynamic Work-flow Negotiation

A number of application scenarios exist, in which the orchestration of resources is necessary to offer added-value services [15, 112]. In DCIs, scheduling services are typically deployed to execute tasks like work-flow scheduling or co-allocation (cf. also Section 3). The latter requires the meta-scheduler to co-ordinate resource management systems located in different domains. As the site autonomy has to be respected, negotiation is the only way to achieve the intended co-ordination.

In this section, we introduce a specific protocol to tackle dynamic negotiation and creation of SLAs [110]. Thereto we discuss an approach based on the concept of transactions and propose an extension to the WS-Agreement protocol which implements it.

Commit Protocols for Distributed Databases

Distributed transactional systems have been widely studied. One of their objectives is to propagate a consistent state across several systems, in a way that at any time all systems can show a consistent state to the users. This maintains and propagates a logical coherent state between systems. To provide crash recovery, several operations are logically grouped into transactions. Those transactions permit the change from one consistent view to another. For instance, you do not credit a bank account if you have not debited another bank account. These, however, are two independent operations. A bank's distributed database system must group the two operations in one transaction. Thus, it permits the change from one consistent state 'before the transfer' to another 'after the transfer'.

Database state changes are visible to other users once a transaction is committed to the system. This implies that in distributed infrastructures, each transaction can impact several other systems. To control this, commit protocols have been developed [13,79,105].

Skeen describes the purpose of a commit as follows [124]:

> The processing of a single transaction is viewed as follows. At some time during its execution, a commit point is reached where the site decides to commit or to abort the transaction. A commit is an unconditional guarantee to execute the transaction to completion, even in the event of multiple failures. Similarly, an abort is an unconditional guarantee to 'back out' the transaction so that none of its results persist. If a failure occurs before the commit point is reached, then immediately upon recovering the site will abort the transaction. Commit and abort are irreversible.

The application of transaction principles to work-flow scheduling in DCIs has led to a proposal based on the negotiation of agreement templates [110] which is outlined below.

Negotiation of Agreement Templates

To minimise the extensions to the WS-Agreement, we suggest not to negotiate SLAs but to negotiate and refine the templates that can be used to create an SLA. Here, our focus is on the bilateral negotiation of agreement templates.

In the following scenario (cf. Fig. 7.9) we describe how an agreement initiator (e.g. an activity scheduler for distributed computing infrastructures) negotiates agreement templates with two agreement providers (e.g. a network scheduler and a CPU scheduler). For

this purpose, we propose a simple n-phase negotiation model. In order to use this model with the WS-Agreement protocol, we introduce a new message to the sequence called *negotiateTemplate*. This message takes one template as input (i.e the offer) and returns zero or more templates (i.e. the counter offer).

The negotiation protocol for the scenario is divided into three steps: (i) initialisation, (ii) template negotiation, and (iii) post-processing. We now describe these steps in detail. Please note that we use the terms the agreement initiator (according to the WS-Agreement specification [6]), 'negotiation initiator', and DCI Scheduler synonymously. Accordingly, we refer to agreement providers (WS-Agreement terminology) also as 'negotiation responders'.

1. *Initialisation of the negotiation process*

   First, the negotiation initiator starts the process by querying a set of SLA templates from agreement providers. To do so, it sends a standard WS-Agreement message, getResourceProperty, to the agreement providers[11]. Then, the initiator chooses the most suitable template as a starting point for the negotiation process. This template defines the context of the subsequent iterations; therefore all subsequent offers must refer to it. This is necessary in order to enable an agreement provider to check the validity of an offer by assessing the adherence of the original creation constraints during the negotiation process.

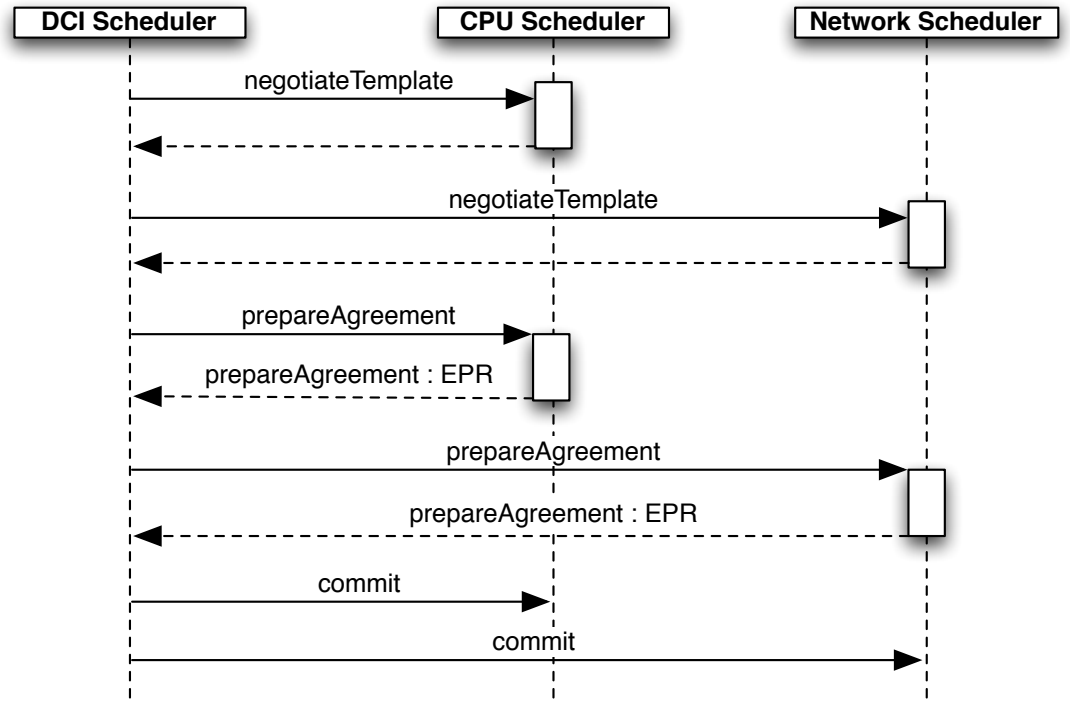

Figure 7.9.: Extension of the WS-Agreement negotiation protocol for dynamic work-flow scheduling [110]

[11]This step is not depicted in Fig. 7.9 as it is the same as already described in Section 7.5.2, Fig. 7.7.

2. *Negotiation of the agreement template*

   After the negotiation initiator has chosen an agreement template, it creates a new agreement template based on it. This step most likely includes modifications (governed by the creation constraints of the original template) of service description terms, service property terms, and guarantee terms. Hereby it is essential that any subsequent template contains a reference to the original template. In addition, the negotiation initiator may include new creation constraints to provide hints for the negotiation responder within which limits the negotiation initiator will be willing to create an agreement. For instance, the initial CPU scheduler template may contain 'any number of 2GHz x586 CPU between 5pm to 6pm', whereas the initiator may request 'at least 5 1GHz x586 CPU anytime'.

   After the initiator has created the new agreement template, it is sent to the respective responder(s) via a *negotiateTemplate* message (as shown in Fig. 7.9).

   Once the responder has received a negotiateTemplate message, it first has to check the validity of the refined template. This step includes (i) the retrieval of the original agreement template, (ii) the validation of the structure of the input document on the basis of the original, and (iii) the validation of content changes in the refined document with respect to the original creation constraints.

   Following this, the agreement provider checks whether the service defined in the request can be provided or not. In our example, it is only then that the CPU scheduler decides that 5 1GHz x586 CPUs can be provided; therefore it just returns the agreement template to the client, indicating that an offer based on that template may be accepted. Otherwise, the provider employs some strategy to create reasonable counter offers. During this process the agreement provider should take into account the constraints of the negotiation initiator. Counter offers are basically a set of new agreement templates that base on the template received from the negotiation initiator. The relationship between dynamically created templates and original ones must be reflected by updating the context of the new templates accordingly. After creating the counter offers the provider sends them back to the negotiation initiator as a response to the negotiateTemplate message.

3. *Post-processing of the templates*

   After the negotiation initiator has received the counter offers from the negotiation responder, it checks whether one or more meet its requirements. If there is no such template, the initiator can either stop the negotiation process or start afresh from step 1. In case there is a suitable template, the initiator validates whether there is demand for further negotiation steps. If so, the initiator uses the selected template and proceeds with step 2, otherwise the selected template is used to actually create a service-level agreement.

SLA Creation extending the WS-Agreement protocol

After the negotiation process has come up with an agreement template suitable for both parties, the initiator needs to create the agreement. At this point, a problem similar to the transaction problem in distributed databases systems arises.

The goal of a DCI scheduler is to create a set of SLAs with different resource providers (two in our scenario), in order to orchestrate the use of multiple resources (compute and network respectively). Therefore, the scheduler first negotiates a set of templates with the providers which identify potential resource provisioning times. However, these templates only provide hints, there is no guarantee that the providers will agree on the terms associated with them. This means that a strategy to create all SLAs or none of them is prerequisite to the execution of the related work-flow.

In principle, there are two major strategies to achieve this:

1. the usage of transactions to create the SLAs or

2. the creation of separate SLAs, applying policies to the process.

The usage of transaction mechanisms to create distributed SLAs, namely the usage of a two-phase commit protocol, has already been discussed above. Since there is no support for two-phase commit in WS-Agreement yet, we need to extend the specification to address this problem.

One potential solution is the addition of a type of agreement that has to be created in two phases: the first phase is a creation of the agreement triggered by a *prepareAgreement* message and the second involves a *Commit* message to achieve that all SLAs related to the activities of a work-flow are created.

SLA Creation using the WS-Agreement protocol

The second strategy is to create an SLA in one step using WS-Agreement functions, cancellation mechanisms, and incentives. In order to realise this, we need to investigate the content of an SLA. On the one hand, an SLA describes the service and its properties. On the other hand, it specifies the guarantees for a specific service. In a co-allocation scenario, where a scheduler uses SLAs to co-ordinate e.g. network and computational resources, it employs execution guarantees in order to assure that the different services are provided at the same time. These guarantees most likely include costs associated with the service provisioning, as well as penalties that arise when a guarantee is violated.

An SLA, however, might be prematurely terminated by the agreement initiator, before the service is actually provided. This is in fact a cancellation of an SLA. If a service provider guarantees a certain execution time for a service, this will normally be realised through resource reservations. Therefore, the resource provider wants to prevent the termination of an existing SLA. This can be achieved by including a basic payment within the SLA, a small amount that is even charged if the SLA is terminated by the agreement initiator before the service is actually provided. It is a termination penalty and represents the cost for the overhead produced by the resource reservation.

In order to provide the foundation for a scheduler to efficiently negotiate and create SLAs, there could be a certain time period in which the SLA can be terminated without penalty. The duration of this period can dynamically be specified during the negotiation process. The agreement provider could use a certain trust index in order to determine the maximum length of this period. This offers a feasible solution for the orchestration of multiple resources using the current accept/reject SLA creation of WS-Agreement.

7.5.4. WS-Agreement Negotiation

The development of a protocol to automatically negotiate SLAs that adhere to a normative format has been a long process. It started in 2007 with a set of 'Re-Negotiation Wishlists' [12] compiled by OGF's GRAAP working group, the home of WS-Agreement, and led to the *WS-Agreement Negotiation Version 1.0* proposed recommendation[13] [138].

During this time-frame, multiple negotiation protocols for multi-phase (re-)negotiation have been proposed, including

- our transaction-based protocol, which has been introduced in the previous paragraphs,
- a re-negotiation protocol we designed based on the requirements captured in the Re-Negotiation Wishlists and on requirements from contract law [108],
- SLA negotiation supported by business rules [109],
- a negotiation approach based on the alternate offers protocol [136], and
- a WS-Agreement-based framework that allows the usage of multiple negotiation protocols [67].

The purpose of our work, though, is to develop an SLA negotiation protocol that is compliant with WS-Agreement and therefore can be adopted by the community that already uses it [123]. This is why we contributed to the standardisation process of WS-Agreement Negotiation[14].

In this section, we briefly describe the main objectives of WS-Agreement Negotiation and sketch the solution. For in in-depth presentation of all the different concepts including the offer structure, the WSDL portTypes, and XML Schemata we refer to the the specification [138].

Objectives of the WS-Agreement Negotiation Specification

During the three years of development we discussed various application scenarios leading to different requirements and objectives. Finally, the following objectives have guided the design process of the WS-Agreement Negotiation specification:

- WS-Agreement Negotiation has to be built upon the WS-Agreement specification [6].
- It has to support both negotiation and re-negotiation of SLAs.
- It has to provide both binding and non-binding negotiations.
- Its protocol has to allow symmetric and asymmetric message exchange.
- All valid states and state transitions have to be well specified.
- XML Schema and WSDL have to be used to normatively describe negotiation artefacts and interfaces, respectively.

[12] Re-Negotiation Wishlists, last visited: January 25, 2013. https://forge.gridforum.org/sf/wiki/do/viewPage/projects.graap-wg/wiki/ReNegotiationWishlists.

[13] The OGF process towards standardisation is described by Catlett et al. [21]. The public comment phase is the final step before a specification becomes a proposed recommendation, i.e. a proposed standard.

[14] Please note that a standard in general cannot be attributed to a single author as it is a community effort.

The Negotiation Model

WS-Agreement Negotiation introduces a three layer negotiation model (as depicted in Fig. 7.10) by extending WS-Agreement with a negotiation layer. Briefly, its purpose is to add multi-phase (re-)negotiation capabilities to the existing specification, which only supports the basic accept/reject model. The basis of the negotiation is a series of offers and counter-offers potentially leading to an agreement. Furthermore, the layer introduces the concept of advertising negotiation offers to parties interested in negotiating SLAs. Last but not least, the termination of the negotiation process can be explicitly triggered.

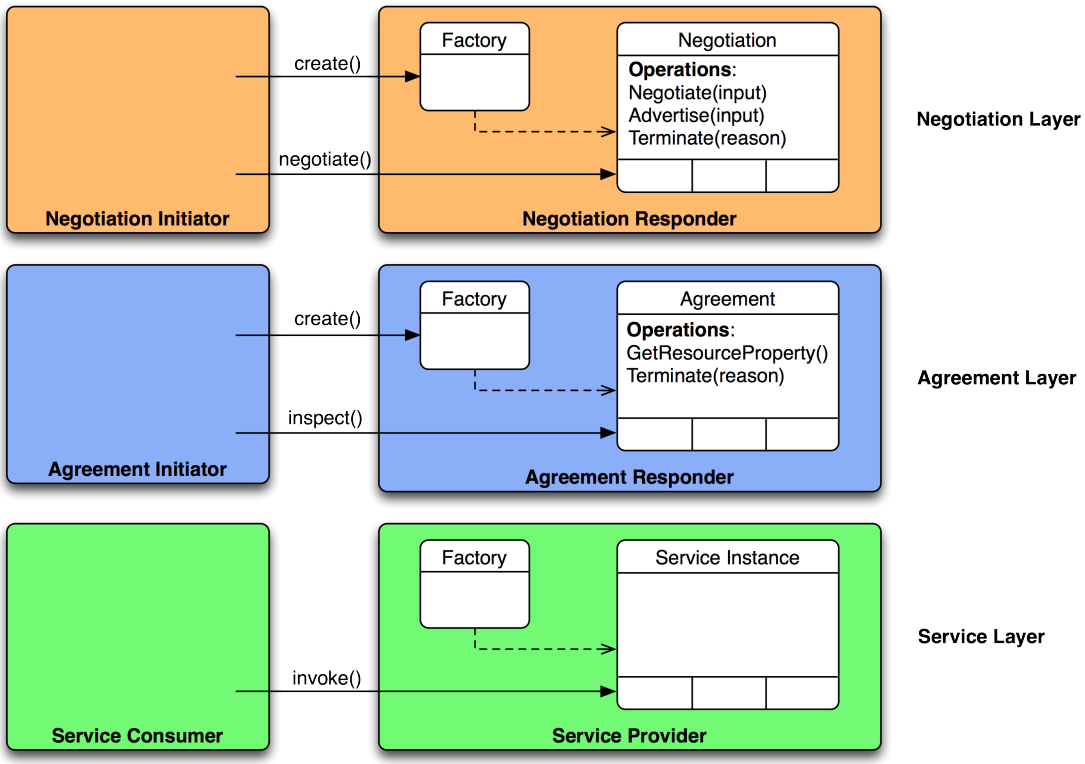

Figure 7.10.: Conceptual overview of the layered negotiation model [138]

To finally create an agreement once the negotiation process is finished, either createAgreement or *createPendingAgreement* is called on the part of the initiator's agreement layer.

Logically, the negotiation layer is decoupled from the other two, which facilitates the application of other negotiation layers on top of WS-Agreement. Through this, other negotiation models may be applied to serve requirements other than those mentioned above.

Application to the 'Delegation of Scheduling Requests' Scenario

Instead of going into greater detail, we illustrate how WS-Agreement Negotiation can be integrated with the delegation scenario set forth in Section 3.1.3. As shown in Fig. 7.11, the primary scheduler, which has received an initial service request, but does not have

access to sufficient resources to fulfil it, tries to find another scheduler to delegate the request to. In a first step, the primary scheduler contacts different secondary schedulers[15] by requesting SLA templates from them. As a result, it receives descriptions of services (i.e. templates) that can potentially fulfil the initial request. After making a scheduling decision, the primary scheduler, which in this example acts both as negotiation and agreement initiator, starts to negotiate with the secondary scheduler (*initiateNegotiation*) and subsequently receives an EPR of the negotiation service. In the following, both schedulers can exchange offers and counter offers until they reach an agreement (or none) governed by the negotiation context. If both parties agree on the delegation conditions, an SLA will be created by calling the createAgreement method. In case the secondary scheduler A cannot fulfil the demands of the primary scheduler, it will start negotiating with other schedulers as depicted in the actual application scenario in Fig. 3.3.

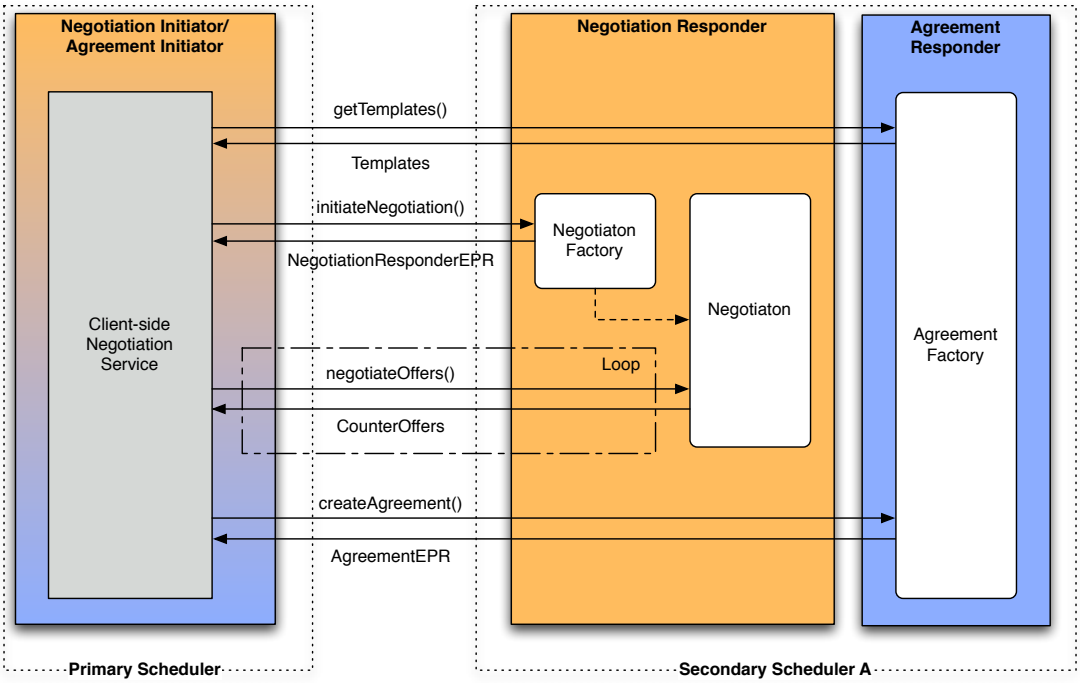

Figure 7.11.: Message exchange between two schedulers to negotiate about a scheduling request following the example in Section 3.1.3

In this part of the thesis, we introduced the life-cycle of an SLA, in order to then direct our attention to the development and negotiation of SLAs. Regarding the latter, we provided an overview of existing SLA negotiation models and protocols to motivate our research leading towards multi-phase negotiation and the contribution to the WS-Agreement Negotiation specification.

[15] We depicted only one secondary scheduler to keep the figure simple.

8. The Scheduling Model

In the previous chapters, we introduced a scheduling architecture and a fitting scheduling process. Furthermore, we presented the activity, information, and SLA models, which provide the foundation for state-of-the-art scheduling in contemporary distributed computing infrastructures. In this chapter, we link theses assets and define a scheduling model that utilises all the aforementioned concepts and entities to compute the best-suited schedule.

After introducing the models needed to develop a concise generic scheduling architecture, namely the activity, the information, and the SLA model, we now face the core scheduling model applied by scheduling services which operate within an SLA-based DCI. The model is designed to operate in an infrastructure implementing the architecture introduced in Section 4.1 and following the overall scheduling process as of Section 4.2.2. It exploits service-level agreements according to the WS-Agreement model (cf. Section 7.4) and the WS-Agreement Negotiation protocol described in Section 7.5.4.

The purpose of the scheduling model is to solve the scheduling problem presented in Section 2.2.1 and thus providing the means to realise the 'assignment of services to activities over time aiming at the optimisation of multiple, potentially competing, quality-of-service criteria' (as stated in Section 2.2). Special emphasis is put on the delegation of activities to other sites in cases where the services available to the local scheduling service are insufficient to fulfil the requirements of an activity. The respective application scenario is shown in detail in Section 3.1.3 and evaluated in Chapter 10. This tight integration of delegation integrates on the one hand well with the 'computing on demand' business model of cloud providers and offers, on the hand, more options to customers of a global service market. We envisage that this model and similar other models are essential to capture the increasing number of objective parameters and to handle the growing amount of data during decision making processes like the one considered in our work.

8.1. The Scheduling Model

Figure 8.1 depicts the proposed scheduling model. It shows the different actions that are executed by the scheduling service, the respective strategies related to each action, the middleware services (directly) involved in each action, and the relationship between the different actions and entities involved.

In the following, we first introduce the different categories of scheduling strategies, depicting their purpose and their role within the scheduling model. Then, we describe the sequence of scheduling actions, and how the scheduling service interacts with other enti-

ties and how it applies the strategies to finally compute a schedule. Last but not least we discuss the advantages, particularities, and the consequences of its application.

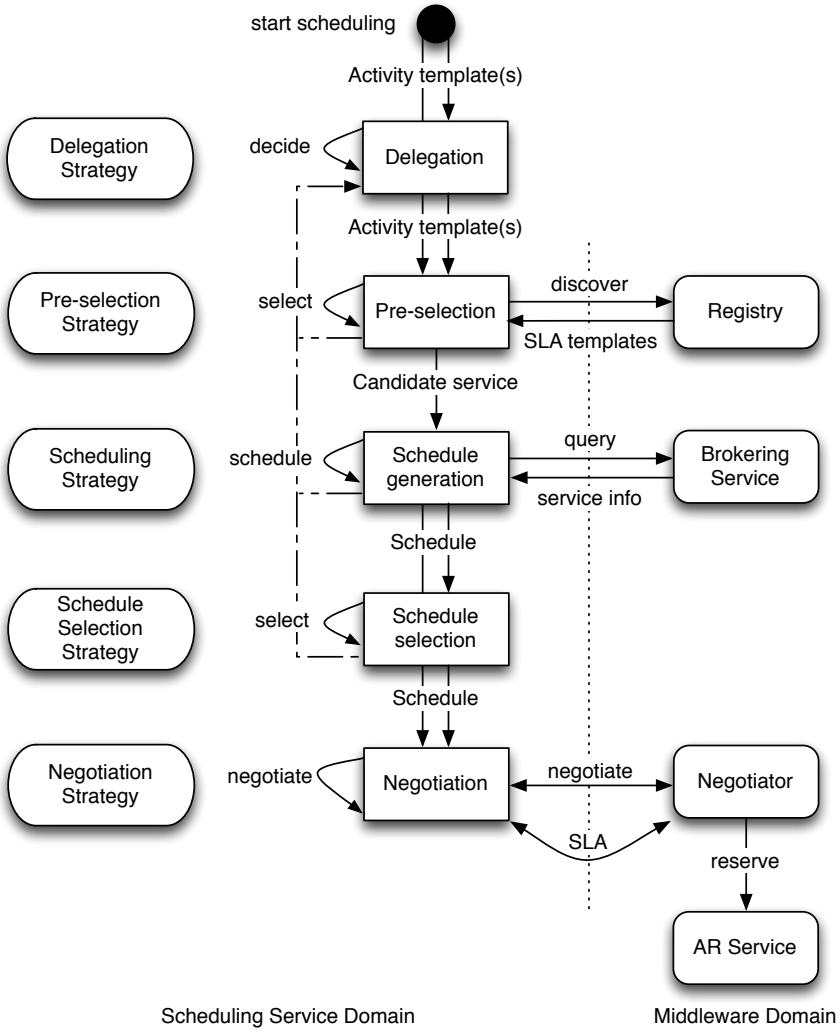

Figure 8.1.: Sequence of actions executed within the scheduling service. The respective strategies are depicted to the left. The respective actions related to other middleware services are shown to the right ('AR service' is advance reservation service)

8.1.1. Strategies

Our objective is to deal with strategies in relation to their purpose and assign roles to them in the scheduling model, respectively. The five fundamental strategic roles of the scheduling model are shown in Fig. 8.1: (i) delegation, (ii) pre-selection, (iii) scheduling, iv) service selection, and v) negotiation. They are described in the following paragraphs and set into relation in Section 8.1.2. Especially for the third role, scheduling, multiple

strategies, which are often called scheduling algorithms, exist for various application scenarios, environments, and scheduling objectives. As it is not our ambition to discuss their variety here, we refer to Bruckner [18], Pinedo [111], and T'kindt [134] for further information and as sources for advanced literature research. The specific strategies themselves, which represent the domain-specific logic used to schedule activities, are treated, as far as the model is concerned, as 'black boxes'[1].

Delegation Strategy

The delegation strategy determines which activities are delegated. This is done by evaluating all 'un-scheduled' activities under the control of a scheduling service with respect to their delegation potential. To achieve this, most likely historical data about passed scheduling cycles and the outcome of previous negotiations is considered to apply a strategy that realises, for example, reputation-based business objectives [127].

Pre-selection Strategy

The pre-selection strategy removes service and resource candidates that cannot fulfil the demands specified in the activity template. The purpose of such strategies is the reduction of choices for further scheduling actions in order to avoid costly calculations or negotiations. Pre-selection, in general, operates on data that is static at least during the execution of the scheduling cycle, like the maximum amount of cores a virtual machine can have or the maximum runtime a provider permits for job execution. Based on such information, the pre-election strategy can make binary decisions for each static parameter to eliminate service and resource candidates which are not worth to be considered for further scheduling.

Scheduling Strategy

The scheduling strategy assigns a service or resource to an activity, like a number of nodes to a parallel compute job or web service endpoints to a business process. The result provided by a scheduling strategy is zero or more scheduling scenarios that fulfil the requirements specified in the activity template. Well-known algorithms like 'earliest-due-date-first' (EDD) [134] can be applied here, but also domain- or customer-specific ones.

Schedule Selection Strategy

The schedule selection strategy has the responsibility to select the schedule which is actually executed. This is equivalent to selecting a provider (or a number of providers) and it takes different criteria into account than the scheduling strategy. Important for the final selection of a schedule can be the price to execute the schedule, site- or customer-specific policies, or provider-specific business objectives. Depending on the requirements of the scheduling service provider, schedule selection strategy and scheduling strategy may be combined into one single strategy. This is feasible in cases where the cost of applying two different strategies outweighs the benefit.

[1] Concrete strategies implemented to evaluate the scheduling approach can be found in Appendix C.

Negotiation Strategy

The negotiation strategy (called negotiation model in Section 7.5.1) describes how a contract is established between a consumer and a provider. Its purpose is to reach an agreement between both parties on the terms of the contract. Once such an agreement is reached, no further scheduling actions are carried out and the activity can be executed. Please note that negotiation is conducted between scheduling services of different DCI sites and not within a single site. In cases where the services and resources needed to fulfil a schedule are under control of one scheduling service, no negotiation is necessary.

8.1.2. Sequence of Actions

The computation of a schedule is initiated, as depicted in Fig. 8.1, once an activity template (or multiple templates) are passed to the scheduling service. In DCIs with an SLA management framework in place, this first step is part of the negotiation between a customer and the scheduling service provider (cf. Fig. 4.1). It is then decided whether the template(s) qualify for local scheduling or for delegation through the application of a delegation strategy. Candidates for delegation are all unscheduled activities, whether newly submitted or remaining from previously executed pre-selection, scheduling, or schedule selection actions. Independent of the previous decision, the same actions are executed for locally scheduled and delegated resources. The only difference is the association of services and resources to either a local or a remote site. The activity templates (representing activities to be scheduled) are then matched against SLA templates (representing service or resource offers) based on a pre-selection strategy. This action includes calls to registries, which contain the information necessary to pre-select suitable candidates for further scheduling.

The result of the pre-selection action, and the input to the schedule generation action, is a list of services (or resources) suitable to fulfil the activity template. As part of the generation of schedules, information about candidate services and resources is requested from brokering services to retrieve an up-to-date view on the service and resource landscape. Using this information, a scheduling strategy is applied to generate a list of potential schedules, which may already be ranked to indicate preferences to the subsequent schedule selection action. There, the scheduling service applies a selection strategy to decide which schedule is to be executed. In case only one candidate schedule exists or the negotiation action is used to actually select the final schedule, the schedule selection step may be omitted.

The final step is the negotiation of an SLA between the scheduling service, which is in charge of scheduling a certain activity, and the service/resource provider. After an agreement on the terms of provisioning is reached, the schedule is in place and the entity that submitted the activity template can be informed accordingly. Most likely, the provider internally reserves the respective services and resources to guarantee the fulfilment of the SLA and thus the adherence of the schedule.

Although the sequence of actions is depicted in sequential order, it is possible that actions are executed repeatedly or that the sequence is interrupted after executing a certain action and resumed at another. An example for the former case is the repeated application of a pre-selection strategy to decrease the number of candidate services; the latter may occur in cases where the negotiation strategy does not produce any result and the

scheduling service resumes with the scheduling strategy based on up-to-date resource information.

8.2. Discussion

In the introduction we stressed the fact that most existing scheduling services and scheduling algorithms are designed for specific environments or application scenarios, hence they lack general applicability. Therefore we motivated a generic scheduling architecture and a matching scheduling process. The outlined scheduling model complements both assets on a strategic level and introduces a modular and flexible approach which suits state-of-the-art service-oriented distributed infrastructures.

However, a generic approach does not imply universal applicability. Therefore we discuss in this section a number of questions related to the practicability of the scheduling model and a number of issues regarding its relation to other models introduced within the scope of this work.

8.2.1. Relation to the Scheduling Process

The scheduling process as outlined in Section 4.2 illustrates the life-cycle of an activity from the scheduling perspective. It includes all steps from the receipt of an activity template to the termination of an activity (cf. also Fig. 4.3). The scheduling model, however, defines the different strategies to be applied by a scheduling service, the sequence of actions to actually compute and realise a schedule, and the interactions between the scheduling service and other middleware services. It is included in the scheduling process[2] and provides a fine-grained view focussing on the scheduling service and taking the specifics of the scheduling architecture into account. In short, the scheduling model introduced here is one particular method to generate a schedule following the scheduling process for DCIs as specified in Section 4.2.

8.2.2. Relation to the Activity Model

The core entity, which is scheduled, is an activity according to the description in Section 5. This implies that the scheduling service interacts with an activity management framework (cf. Fig. 4.3) to create or update activities, or to receive notifications regarding its activity subscriptions. The decision on which activity-related events are considered, however, is completely domain-specific. The financial service application scenario as discussed in Section 9.1.2 [96], for example, creates an activity just after the SLA has been negotiated and the service contract is in place. Other scenarios, like those where scheduling decisions themselves are of interest, might imply the recording of any strategic decision made, the documentation of any candidate schedule, or the logging of any negotiation. As long these actions are executed according to the activity model, decisions about granularity and amount of activity recordings are incumbent on the scheduling service provider.

[2] Please note that the scheduling process can be used with other scheduling models, too.

8.2.3. Thoughts on Delegation

Delegation is an intrinsic feature of the approach we propose. Its integration is motivated by application scenarios like the one presented in Section 3.1.3, but also the increasing need to consider data center locality when using and operating DCIs. The latter factor, as of today mainly motivated by the requirement to reduce energy cost [3], is specifically addressed by the steadily growing cloud-like service offers. Furthermore, a number of European research projects, like VENUS-C[3] or BonFIRE[4], deal with this topic from a research perspective.

Our scheduling model is designed to operate in distributed computing infrastructures where users and operators require out-sourcing of activities based on policies or budgetary constraints. Delegation can be a strategic decision reflected in the negotiation strategy or based solely on events like unavailability of local resources (cf. Section 4.2.2).

It is, though, also possible to prevent delegation, either through software design decisions or through the definition of the respective strategies. One particular aspect to consider there is cost. The application of a delegation strategy may result in one negotiation per delegated activity. As negotiation can be costly [91], it is particularly important to consider the ROI of such decisions. 'Inexpensive' negotiation strategies, like Accept/Reject (cf. Section 7.5.1), or alternative decisions, for example the rejection of the initially received activity template, should be then taken into consideration.

8.2.4. Complexity of the Model and Inter-relation between Strategies

The scheduling model includes five strategies in order to compute and realise a schedule. This seems to be, at first sight, a complex approach to achieve something which, in literature, is often realised through the application of one scheduling algorithm. Since the purpose of the different strategies has already been introduced above, we discuss in the following the feasibility of the model in different situations and the inter-relation between the strategies:

- The scheduling model can be customised according to the domain-specific needs and policies of the scheduling service provider. The following options can be applied to reduce complexity in case it is not feasible to implement the complete scheduling model:
  - The delegation strategy can be omitted and all activity templates can be locally scheduled. The scheduling model then still follows the scheduling process where upon activity template receipt the pre-selection step is executed. The downside of this approach is that costly actions are executed to schedule (types of) activities although subsequent actions of the scheduling process are likely to fail.
  - Scheduling strategy and schedule selection strategy maybe, as previously discussed, combined. A simple approach, which is often used in practice, is the prioritisation of schedules and selection of the one with the highest priority.

[3] VENUS-C – Virtual multidisciplinary environments using cloud infrastructures, last visited: January 25, 2013. http://www.venus-c.eu/Pages/Home.aspx.

[4] BonFIRE, last visited: January 25, 2013. http://bonfire-project.com/.

Especially with respect to strategies applying single-criteria optimisation, this selection is implicit.

  - The schedule selection strategy may be omitted completely.

- The separation of scheduling and schedule selection strategy may exclude well-suited schedules (regarding the efficient use of a DCI) due to certain business objectives. The scheduling service provider has to be aware of the implications of applying such objectives, involving the respective business units in the definition and operation of the service. This implies additional overhead compared to solutions where such business criteria are not part of the equation.

- In case there is demand for negotiating with multiple sites over one activity, e.g. if the invite-tender negotiation model described in Section 7.5.1 is applied, the scheduling model has to be implemented in a way that allows to configure the different strategies accordingly. Either the schedule selection step is skipped or it selects schedules tailored for the chosen negotiation strategy.

- It should be noted that, although it is a general issue related to negotiation, we need to take care of that negotiations finish in finite time and that they do not block the overall scheduling process. To achieve this, it is essential to design appropriate negotiation strategies, but also to monitor the overall progress of scheduling closely.

In this section, we treat the strategies as 'black-boxes'. As the strategies are domain- or site-specific and carry the particular logic a scheduling service provider needs (or offers their clients), their realisation and customisation is a complex undertaking. Nevertheless, we think that such effort is well invested and that in the long term, the flexibility of our approach will lead to a positive ROI. To provide an initial evaluation of the scheduling model, we have chosen in Chapter 10 the delegation scenario from Section 3.1.3 as a particular example to discuss the interaction between the various strategies.

The scheduling model as introduced in this chapter specifies how to compute the best-suited schedule in an environment defined by the previously introduced scheduling architecture and scheduling process, and combined with the three core models. We therefore described how the scheduling model fits into this environment, how it interacts with the other entities, and which particular strategies are needed to create a schedule. Furthermore, we discussed the implications of the application of the model regarding aspects like domain-specifics or cost.

Part IV.

Implementation

9. Realisation

After introducing the various models that constitute the basis for DCI scheduling with electronic contracts, we now show selected implementations of these models. To this end, we introduce the UDAP activity management framework, the CIS information service, and the IANOS scheduling framework.

The full realisation of our scheduling architecture, including the essential auxiliary services, would be tantamount to creating a more or less complete new DCI middleware. We did not approach this venture as part of our work, as we aim at providing the concept of a generic scheduling architecture for state-of-the-art DCIs and not at a complete realisation of this concept.

Nevertheless, some of the models we have introduced have been implemented within German and European projects. This includes the realisation of an activity management framework based on the UDAP model (cf. Chapter 5). It has been implemented as part of the NextGRID project and is described in Section 9.1. The same project is accountable for the implementation of the Common Information Service, a software entity based on the UNICORE-specific, CIM-based information model. A derivative of the initial service, which is outlined in Section 9.2, is now part of the UNICORE distribution[1]. Last, but not least, the idea of integrating service-level agreements and scheduling mechanisms to provide reliable quality-of-service is something we have been following for quite some time. Although there is no complete implementation of the generic DCI scheduling architecture as outlined in Section 4.1, we contributed to a number of research projects working towards such an implementation. The most advanced, IANOS, is introduced in Section 9.3.

We conclude this chapter with a discussion on the implications of realising the complete generic DCI scheduling architecture. This conclusion summarises the issues which have been debated within the different projects and it states the challenges practitioners will face.

9.1. The Activity Management Framework

The most complete implementation of an activity management framework is the the *UDAP Infrastructure* [64,96]. It realises the UDAP model described in Section 5.1 and implements a number of generic services which can be customised to serve a large variety of application scenarios.

[1] UNICORE – CIS, last visited: January 25, 2013. http://www.unicore.eu/unicore/architecture/service-layer.php#anchor_cis

The definition of what an activity actually is, and therefore which state changes are to be recorded, essentially depends on the operator of the infrastructure. In case of a grid job that is executed via a *Basic Execution System* [49], the job has one of the following states: pending, running, terminated, failed, or finished. Consequently, any state transition will most likely trigger an update of the activity instance. In other cases, having no well-defined state machine, it may be more complicate to answer questions like 'Is a delegation request to another scheduler a change of the activity instance?'

The generic activity management framework introduced below is the foundation for the realisation of arbitrary activity management scenarios. Among those implemented where for example ranking of SLAs, adjustment of SLAs, and the business example described in Section 9.1.2.

9.1.1. Generic Architecture

The UDAP activity management framework consists of a set of services which realise three major functionalities: management of activities instances, registration of activities, and service discovery. Fig. 9.1 depicts the three fundamental packages of this framework, as there are *UDAP Manager*, and *Activity Registry*, and *UDAP Advertiser*. The details pertaining to each package are described in the following three subsections.

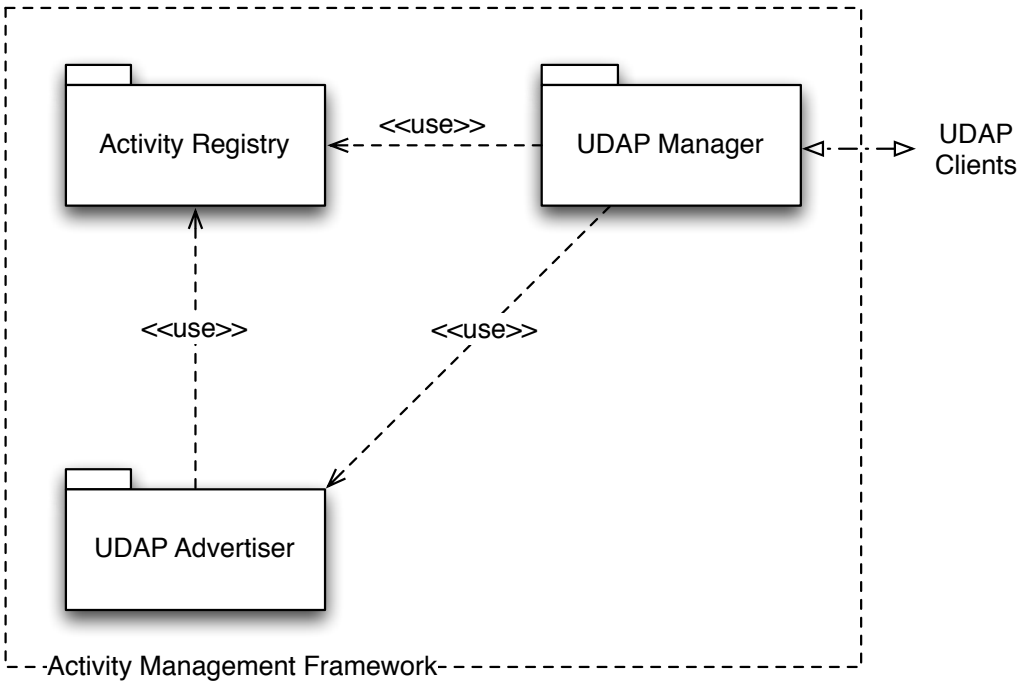

Figure 9.1.: A high-level view on the UDAP Infrastructure components (representing the activity management framework as depicted in Fig. 4.1)

UDAP Manager

The UDAP Manager is the central entity that deals with the management-oriented aspects of all other UDAP parts, including functions like creation and removal of activity instances. Besides managing activities, the UDAP Manager handles the interaction with client-side components, thus representing the interface to the other services of the generic scheduling framework (as depicted in Fig. 4.1). Client services can execute actions like the creation of an activity or the subscription to activity state changes. In the former case, the UDAP Manager acts as a factory to create the various activities.

The UDAP Activity (representing a UDAP document as introduced in Section 5.1.3) is the central component of the UDAP Manager as illustrated in Figure 9.2. It is an abstract representation of an activity; its specialisations, like for example an SLA Activity or an IaaS Activity, extend it with domain-specific properties and functions. Authorised clients can update or read an activity, and thus either adding Entries to the activity document or retrieve an activity's Record (cf. Section 5.1.3).

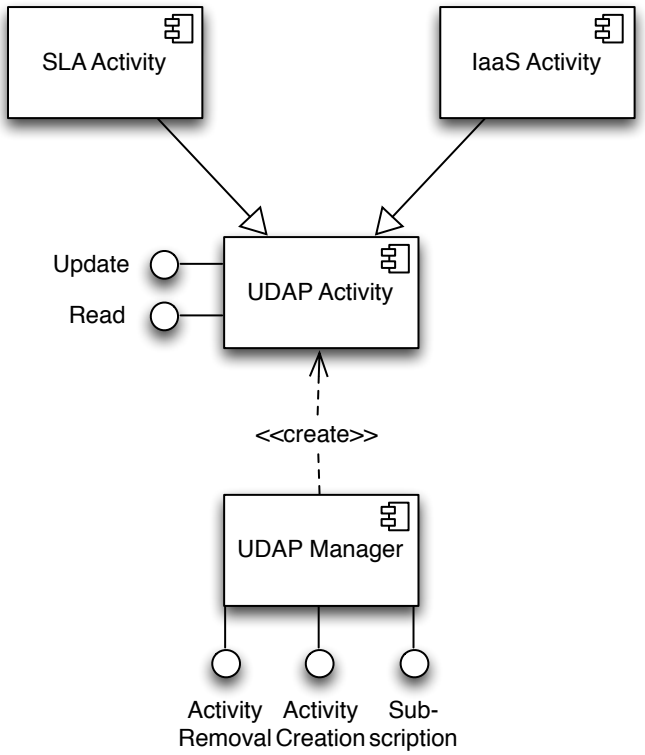

Figure 9.2.: The UDAP Manager provides the factory interface to create UDAP Activities

Activity Registry

The Activity Registry is an intrinsic part of the UDAP framework. It is the underlying component providing persistent storage of activity information. In case of the aforementioned SLA Activity, for example, all data related to one specific service-level agreement is captured within the Activity Registry. This data can then be used to record all service and

resource usage related to an SLA or to evaluate the compliance of the service delivery with the corresponding service terms.

The Activity Registry provides, as depicted in Figure. 9.3, several interfaces in order to manipulate the activity state. These interfaces implicitly provide methods like database CRUD (Create, Read, Update and Delete) operations. To realise its functions, the Activity registry uses two sub-components: the *DB Manager* and the *Lifetime Manager*. The DB Manager offers core data access capabilities and performs low-level storage operations, whereas the Lifetime Manager manages the life-time of the individual activities in order to ensure the data integrity of activities.

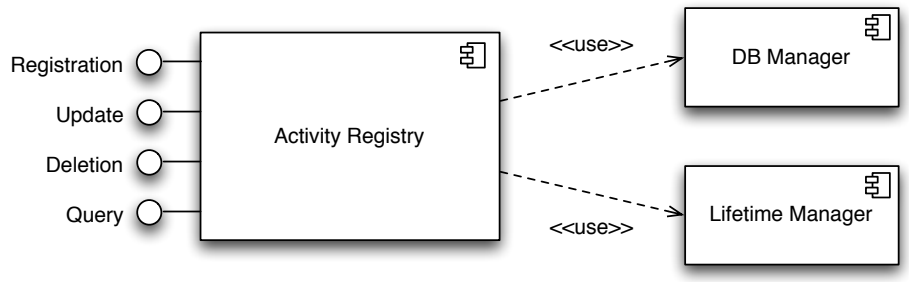

Figure 9.3.: The Activity Registry used in the UDAP framework

UDAP Advertiser

The UDAP Advertiser matches service capabilities with service requests. Services (or resources depending on the environment UDAP is operated in) are published using the *Publish* interface of the UDAP Advertiser while queries are executed via the *Query* interface. In the back-end, the UDAP Advertiser uses the Activity Registry to register services and search for them.

The *Model Transformer* is an abstract component which is used by the UDAP Advertiser to handle services seamlessly. It provides means to dynamically add specialised transformers thus extending the capabilities of the UDAP Advertiser. In case of the UDAP framework, which uses CIM as the model to capture information (see also Section 6), a CIM instance document is constructed by a language-specific *CIM Transformer* whenever a service is published. This allows domain-specific service description languages to be used with the UDAP framework without changing the client-side model. The same adheres to the query interface which transforms domain-specific requests to XQuery FLOWR expressions implementing *XQuery Transformer*s.

This construct a) provides a consistent instrument to internally model services and activities in the same way and b) re-uses the Activity Registry for services, too. The UDAP implementation actually utilises the Common Information Service (see Section 9.2) to implement part of the capabilities described here.

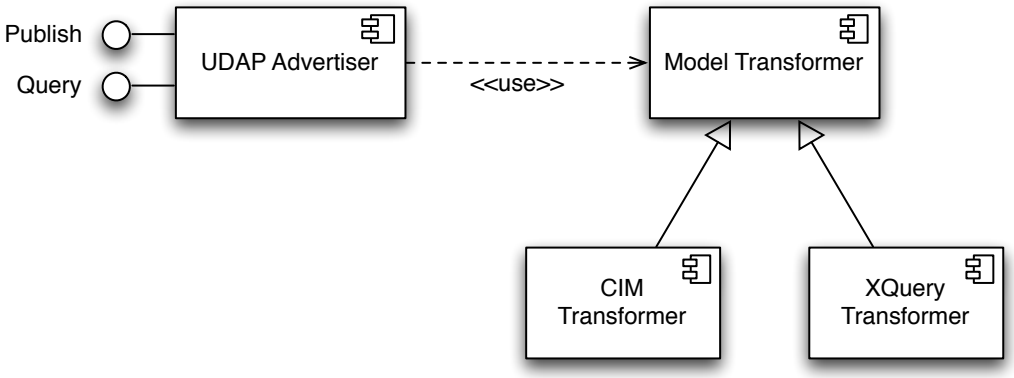

Figure 9.4.: The UDAP Advertiser and its sub-components (the CIM Transformer and XQuery Transformer are place-holders for concrete implementations of transformers between service/resource descriptions and the CIM model and between service/resource queries and XQuery expressions)

9.1.2. UDAP Application Scenarios

The Activity Model, as introduced in Section 5, has been specified in a way that does not restrict activities to certain domains or application areas. In general, this should also apply to any implementation of the model. In practice, we designed the activity framework described above, which fulfils the requirements of a number of application scenarios (inter alia the one described by Hasselmeyer et al. [64] and the one introduced by Mersch et al. [96]), but may not meet those of other applications outside the scope of our developments. Among these scenarios some are commonly applicable to distributed service-oriented infrastructures. Serving as an example, we first detail a common scenario for which the UDAP framework has been customised: the integration of activities with an SLA management framework. Second, we show a particular application scenario which benefits from an activity management framework and which has been implemented as part of the NextGRID project, the implied volatility calculation.

Integration with SLA Management

UDAP provides a unified interface to manage activities of any kind. The term ‘activity’ is, as pointed out in Chapter 5, abstract in notion. Therefore, a service-level agreement with all its life-cycle-related actions (see Section 7.1) can also be treated as an activity and the respective SLA management can exploit the activity management framework as the central entity for SLA-related information. This UDAP application scenario has been implemented within the NextGRID project and was applied in a customised form to the implied volatility scenario introduced below.

Generally, the SLA management service is deployed by providers to monitor the compliance of the service provision with the service terms that have been agreed between service providers and consumers. In this process, it needs up-to-date information about all services, processes, and resources which are linked to the respective SLA. This information is delivered by the activity management framework.

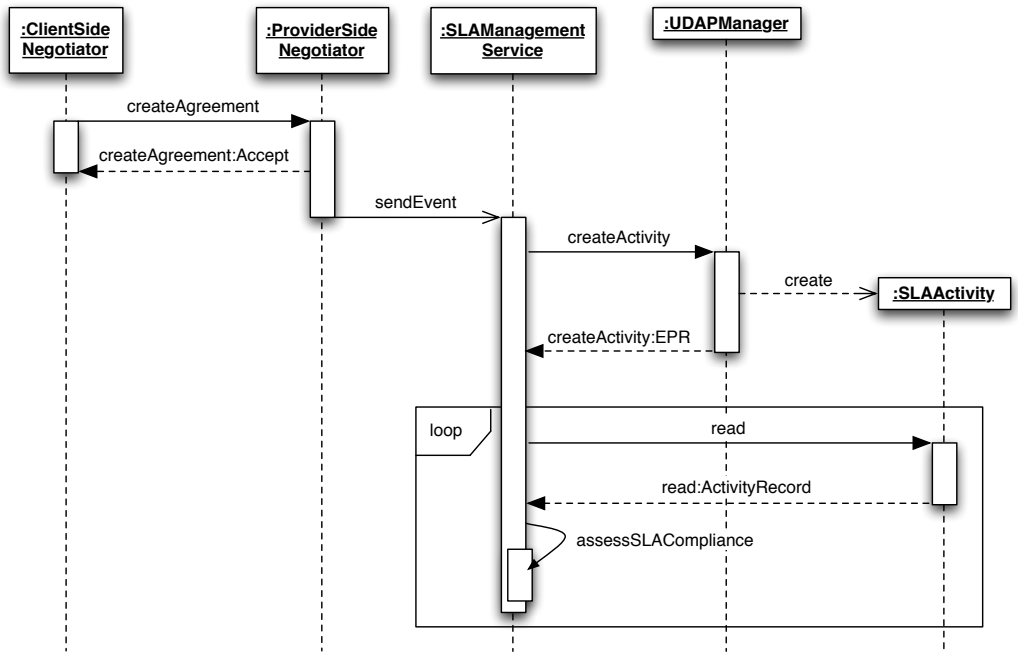

Figure 9.5.: The sequence diagram depicting UDAP creating an SLA-related activity

The sequence of events is initiated by the negotiation of an SLA. Once the client is satisfied with the result of the negotiation, the following steps are executed as shown in Figure 9.5 (the pre-`createActivity` negotiation messages are not depicted here):

1. The client-side negotiator[2] sends the final offer via an `createAgreement` message to the provider-side negotiator which sends the respective accept message back (in case of an reject message, no activity instance is created).
2. Then, the provider-side negotiator sends an event to the SLA management service indicating that an SLA has been accepted. In case of the UDAP implementation, this event contains the necessary information showing the type of event and the SLA itself.
3. Triggered by the event, the SLA management service sends a `createAgreement` message to the UDAP Manager (thus acting, accordant with Figure 5.1, as a UDAP Client).
4. As a result, the SLA Activity is created.
5. Then, the SLA management service (and other services which are not considered here) retrieves the EPR of the SLA Activity. Once the activity has been created, any authorised service can read or update it (see also Figure 9.2).
6. As an example, the continuous monitoring of the SLA compliance is shown in a loop which comprises reading the activity record and assessing the SLA compliance based on the agreed terms and the content of the activity record.

[2]For enhanced readability class names are not written in their concatenated form (as done in Figure 9.5 following the UML notation).

The update and read of the SLA activity is not, as shown here, restricted to the assessment through the SLA management service, but other applications like billing, which exploits the final activity record containing the resources and services used to fulfil the SLA, or client-side monitoring, which accesses only particular parts of the activity record intended for consumption by a customer, are possible, too. Depending on the application domain, rather any service can act as a UDAP client and, depending on the authorisation settings, read or update the information on the SLA Activity.

The Implied Volatility Application

The calculation of the implied volatility of stock options is a computationally expensive process which in general exceeds the resources available at a customer's site. Financial service providers therefore offer the required implied volatility services, adapting dynamically their own resource consumption to the customer's demands. The success of such a business model relies on carefully negotiated and observed service-level agreements. The NextGRID project has designed and implemented an implied volatility framework [96] which is based on the NextGRID SLA framework [64] and the UDAP framework.

Introducing Implied Volatility Within the stock market, stock and stock options can be purchased. Stock signifies an ownership position within a corporation. Options represent an option to buy (in the case of a call option) or sell (in the case of a put option) a set amount of stock from/to a third party at a set price (the strike price) in the future (the maturity date of the option). An option is purchased from the third party and if it is profitable on the maturity date (for example, the strike price of a call option is less than the current value of the stock, allowing the holder of the option to buy the stock more cheaply than would otherwise be possible) it will be exercised; otherwise it will be left to expire.

When the stock market is open, stocks and option prices are constantly being updated. Stock options are normally priced using the Black Scholes model [14]. This equation contains a volatility parameter which can not be observed in practice. There is a one-to-one relationship between the theoretical price of a stock option and its volatility. Unfortunately there is no closed form solution for implying the volatility from the stock option price. If the volatility is known, trades can be executed to take advantage of volatility spikes. The implied volatility must be calculated using a numerical method; a Newton-Raphson [150] iterative process is normally used, which is computationally expensive as the peak rate of the option market is 120,000 prices per second.

The Financial Service Scenario The implied volatility application features four actors: the *Financial Customer*, the *Financial Provider*, the *Compute Provider*, and the *Data Provider*.

Instead of each Financial Customer having the implied volatility software running on a local machine or on one supplied by the Financial Provider, the customers have access to a portal where they can search for financial services (implied volatility being only one such service). They may also browse what data feeds are available to supply these services. Once a customer has selected a service, a choice of available Financial Providers (together with the respective service-level agreement templates) is presented for the customer to choose the most suitable. This choice then stimulates the Financial Provider to discover

compute and data services and establish the corresponding business with Compute and Data Providers. The respective relationships between the different actors and the services under negotiation are pictured in Figure 9.6.

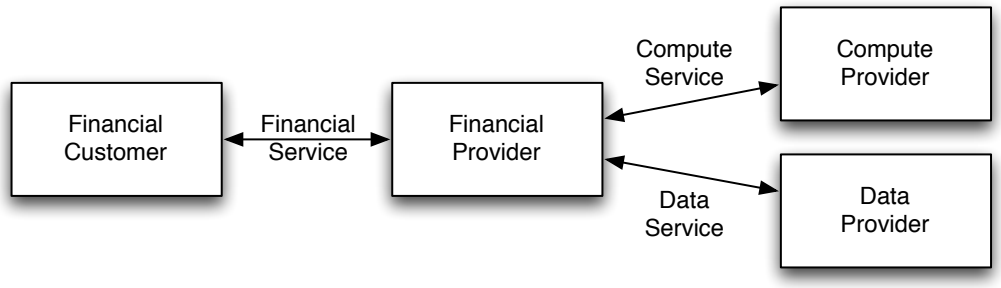

Figure 9.6.: Relationships between the different parties indicating the three SLAs needed for the financial service scenario

One essential customer requirement in the given scenario is high reliability, which implies the fast and correct response of the system to failure. As an example, one could imagine the partial or complete failure of the service provided by the Compute Provider, which results in breaching the SLA between the Financial Provider and the Compute Provider. To avoid significant down time, the Financial Provider will react automatically to the situation, halt the supply of the data feed, discover a new suitable compute resource, agree a new SLA with the new Compute Provider, and deploy the software. The data feed may then be re-started pointing at the new compute resource which will then begin supplying the Financial Customer with the output data stream. From the point of view of the customer, the SLA breach will therefore only affect her in terms of a short delay while the Financial Provider switches Compute Providers.

Integrating UDAP with the Implied Volatility Application The implementation of the financial application scenario integrates a number of services: application-related, infrastructure-related, SLA management-related, and activity management-related. They are in greater detail described by Mersch et al. [96], while here we focus on the interrelation between service delivery, SLA management, and activity management.

The whole scenario is driven by a Financial Customer's interest in retrieving a continuous data feed containing relevant implied volatility information. To provide this service, a number of other services are needed (as shown in Figure 9.7). As a first step, which is not shown here, the customer needs to find suitable providers who are able to deliver the requested service. Afterwards, once the customer has retrieved a list of Financial Providers and has chosen one, they need to negotiate the terms of service provision with the respective provider. If an SLA between the Financial Customer and the Financial Provider is established, additional compute and data services are needed to fulfil the requirements of the customer. Consequently, the provider will search for and negotiate with Compute and Data Providers about the respective terms and conditions. In case of success, three SLAs are established according to Figure 9.6 and subsequently, the Financial Provider can deploy the the implied volatility software at the Compute Provider, the data feed from the

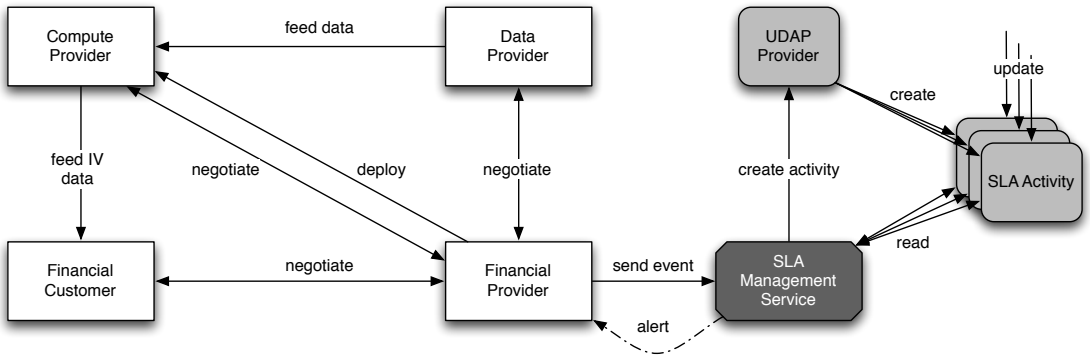

Figure 9.7.: The implied volatility application ('IV' at the leftmost label is the abbreviation for implied volatility)

Data Provider can be established and the customer retrieves the desired implied volatility data.

With three SLAs being established consequently three activities are created following step 1 to 5 of Figure 9.5. The three activities are continuously updated by different information sources, like a Ganglia monitoring system for the compute service or some specific data feed control service. With an activity management framework in place, the SLA management service can retrieve all data related to a specific SLA and thus can assess whether the current service provision is still compliant with the terms set in the SLA. In cases where an SLA is breeched (or is tended to be breeched), the SLA management service can take appropriate actions based on pre-set policies. One such action, which has been implemented as part of the implied volatility application, is sending an alert to the Financial Provider in cases where failure of the financial service puts the Financial Providers reputation at risk. Such an alert may, for example, result in finding another provider or in adding more compute resources to fulfil the high demands of the implied volatility calculation.

This automated failure management based on the UDAP framework and the NextGRID SLA management means improvement in comparison to the manual intervention which has been executed before. Through the application described here, the required reliability of the financial service has therefore been improved.

9.2. The Common Information Service

The demand for an information service based on the Common Information Model and its UNICORE-specific customisation (see Chapter 6 for details) led to the implementation of the *Common Information Service* (CIS) [94]. Initially developed within the NextGRID project and customised to work with UDAP and the implied volatility application, CIS is now part of the UNICORE distribution[3] and is deployed in DCIs world-wide.

[3]CIS: Current Status and Roadmap, last visited: January 25, 2013. http://unicore.eu/community/development/-CIS/cis.php.

9.2.1. Requirements

A analysis of the requirements to be fulfilled by CIS, based on potential usage scenarios, the service discovery methodology [63] of the NextGRID project, and the resource information handling of the UNICORE middleware [130], led to the following functional requirements the information service should comply with:

- The service (or resource) information registration has to be compliant with the information model.
- An information update capability has to be provided.
- Interfaces to perform user-specific and template-based search are required.
- Operations have to be exposed through a standardised interface, most likely WSDL.
- The communication with remote clients has to be realised via XML-based protocols.

Furthermore, a number of design decisions have been made, mainly to be able to adapt CIS to various domains (using various information models) and to ease integration into service-based infrastructures. This included the possibility to extend or exchange the underlying information model to adopt to the different requirements of DCI operators.

The entirety of requirements was not fulfilled by other informations services, like the Monitoring and Discovery Service (MDS4) [121] the Relational Grid Monitoring Architecture (R-GMA) [25], which have been evaluated prior to the development of CIS [94].

9.2.2. Architecture

A holistic view on the architecture of the Common Information Service is depicted in Figure 9.8. It adopts the standardised service-oriented architecture [89] and is based on three intrinsic steps:

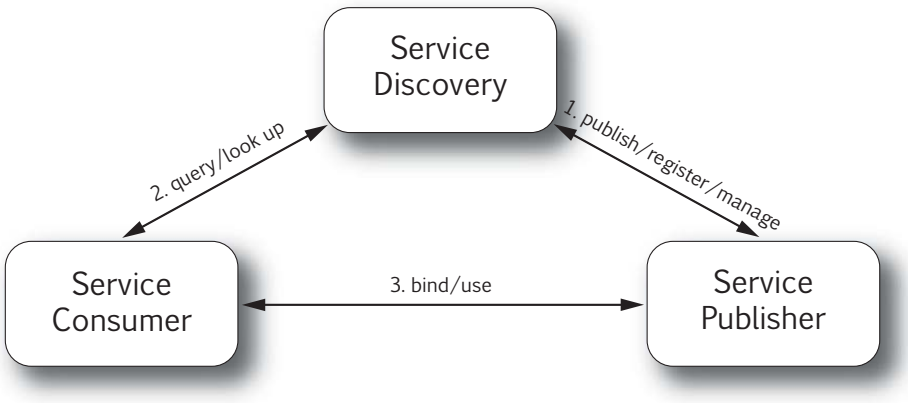

Figure 9.8.: A generic view on the service-oriented architecture of an information service

1. **Publish service information**: The *Service Publisher* publishes detailed information about the services and resources it provides (like CPU, memory, or software). This information can be managed through operations like update or remove.

2. **Look-up** of desired services: The *Service Consumer* sends a query to the *Service Discovery* requesting published information provided by different publishers. In return, a list of EPRs and other meta-data of Service Publishers is provided.

3. **Bind/Use**: Once the Service Consumer has received the EPR of the desired services, a session is established to use the services offered by the Service Publisher.

The detailed architecture and the design of each component are described in the following subsections.

The Service Publisher and the Information Provider

The *Service Publisher* is responsible for publishing the description and capabilities of a service to a registry. This is achieved via a web service request that registers the resource information at the Service Publisher in the form of an XML document. Up-to-date resource or service information is usually gathered via a wrapper service which acts as a gateway to different types of existing information providers, for example Hawkweye[4] or Ganglia [90].

The Service Publisher sub-component of CIS comprises, as depicted in Figure 9.9, the following set of entities to accomplish the processes of service registration and information publishing:

- Information Provider.
- Information Detector.
- Information Translator.
- Information Service Delegate.

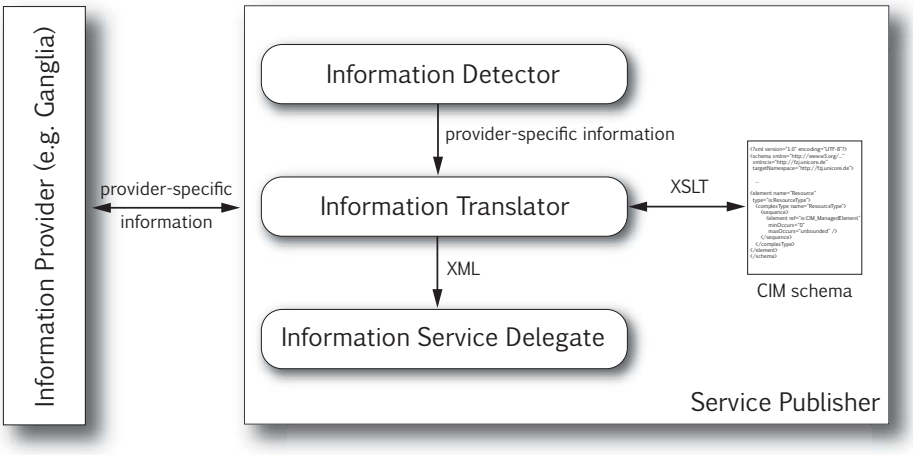

Figure 9.9.: The structure of the Service Publisher (and its relation to the Information Provider)

[4]Hawkeye – A Monitoring and Management Tool for Distributed Systems , last visited: January 25, 2013. http://www.cs.wisc.edu/condor/hawkeye/.

The *Information Provider* is an entity external to CIS, like the aforementioned systems Hawkeye or Ganglia, which serves as a source of information. It is responsible for supplying up-to-date data about services and resource to the core of the information service. This data can be used by the consumers of CIS (see below) and it is essential for a variety of services to, for example, make scheduling decisions or assess the compliance of service-level agreements.

The *Information Detector* is based upon the Adapter design pattern [54]; it is used to integrate two systems with incompatible interfaces. It specifically detects the up-to-date dynamic information from the Information Providers introduced before and passes this information on to the *Information Translator* (see next paragraph). The Information Detector plays an important role to guarantee consistency and integrity of information by observing Information Providers constantly through a notification mechanism. The Information Detector and Translator work in a coordinated way; in case updated information is detected, it is directly translated and stored in the information service.

The responsibility of the Information Translator is to map the resource or service information provided by Information Providers via the Information Detector to the underlying information model which, in case of CIS, is an customised version of the Common Information Model as described in Chapter 6. Such a task is necessary since, in general, the models used by the Information Providers differ from CIM. This component requires a sophisticated logic, since correct matching between two systems in the context of an information service is a complex task. Therefore, the translator converts the data delivered by the Information Providers to the CIM schema. This requires the mapping of semantics as well as checking whether the resulting XML documents are valid and well-formed. From the implementation perspective, the conversion mechanism makes use of XSLT transformations, which are performed to create CIM schema-compliant information in the form of an XML document.

The last entity comprising the Service Publisher, the *Information Service Delegate*, is based on the J2EE design pattern called Business Delegate [4]. Since the Common Information Service is distributed over different tiers, it hides the complexity of locating an information service, i.e. the function call to register or to update service and resource descriptions. In other words, it provides access to the business logic layer, the core of CIS. An Information Delegate is being used at the Service Consumer side, too, so as to hide complexity of looking up the querying service.

The Service Consumer

The information published through Service Providers is accessed using an entity called *Service Consumer*. Such a consumer provides transparent access to the data kept in the information service via its query interface thus hiding the user interaction and information service location. To achieve this, a Service Consumer looks up and queries the information service for desired services on behalf of its users. As soon as a set of service EPRs is retrieved, the Service Consumer tries to communicate with the individual services using their respective endpoints.

In Figure 9.10 a detailed view of the Service Consumer is given focussing on higher-level entities. It is based on the MVC (Model View Controller) architecture and hence provides extensibility of the individual model, view, and controller components.

The Service Consumer comprises the following core components:

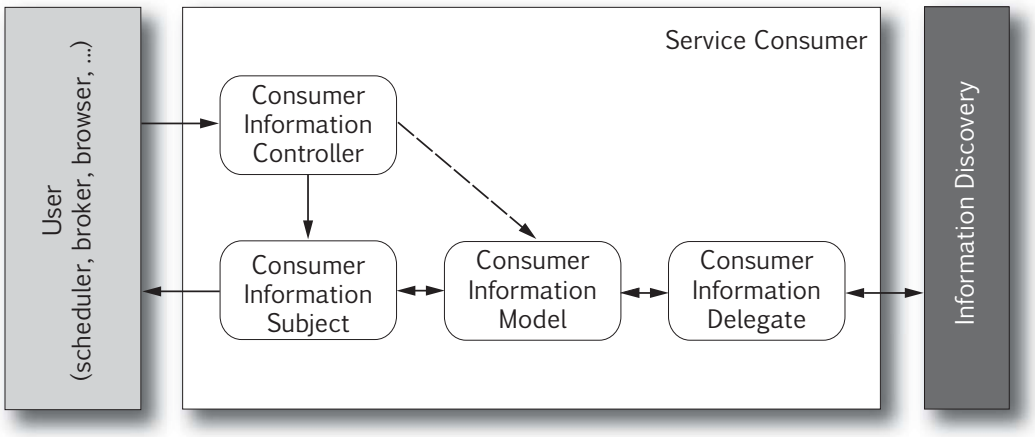

Figure 9.10.: The Service Consumer and its connections to external components

- The *Consumer Information Delegate* plays the same role as the Information Service Delegate plays in the Service Publisher context. Precisely, it encapsulates the information of the connection to Service Discovery, de-couples the Service Consumer from the information service and delegates the function calls to the business logic layer.
- The *Consumer Information Model* knows about the information that is needed by the customers (the User in Figure 9.10). Transformations of transfer objects are being carried out here, too, to suppress the load of network calls.
- The *Consumer Information Controller* is responsible for accepting query requests from different kinds of users as there can be web browsers, schedulers, or brokers. These requests are processed and then delegated to the information service. On the event of response, the controller decides which response to send to which client; it, for example, sends a query result to a particular JSP page if the client is a web browser.
- The *Consumer Information Subject* is a 'view' in terms of the MVC pattern. Regarding the Service Consumer, a view can be considered the representation of information according to a particular client. A dynamic user interface, for example, is generated for a web client using JSPs or customised DTOs (Data Transfer Object) are constructed for information brokers and schedulers.
- Potential *User*s can, as mentioned before, be web clients, information brokers, and schedulers. A web client provides means for the end users to visually publish, query, and manage lifetime for resource descriptions (through HTML based interface), while schedulers and brokers use these services transparently through API function calls.

Service Discovery

Service Discovery is the third base component of CIS and it provides two main functions: (i) service registration and (ii) service look-up. It consists of three layers as presented in

Figure 9.11.

The core layer is the *Business Logic Layer* which encapsulates the mechanisms for the aforementioned registration and look-up/querying functions and which includes, apart from the *Façade*, the main components *Service registration* and *Service look-up* as illustrated in Figure 9.11. Service registration is responsible for providing registration-related functions as there are inter alia register resource, update resource description, and destruction of registration. Service look-up, on the other hand, is implemented to provide the look-up services to users. They are customised utility functions that include queries featuring the most frequent criteria, e.g. a query by memory size or a query for specific application software installed. Furthermore, the service look-up is complemented by query interfaces that support plain XPath or XQuery expressions.

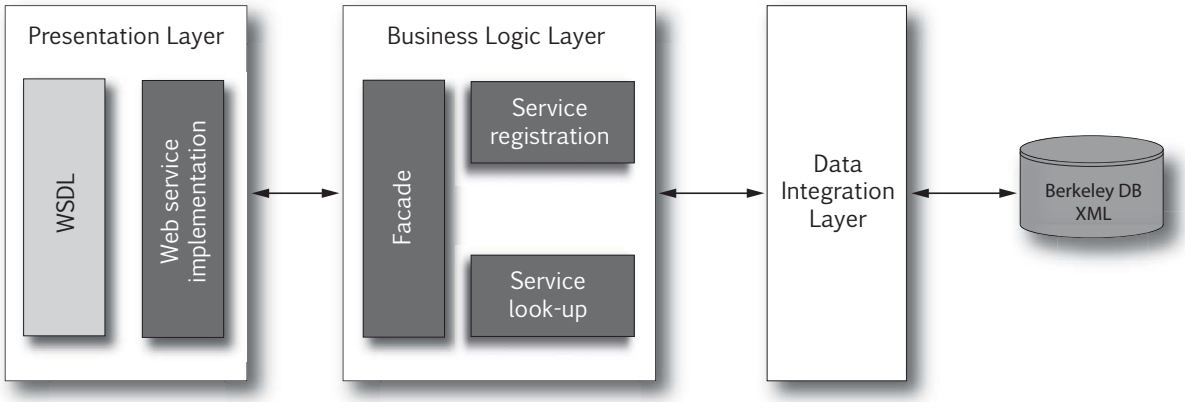

Figure 9.11.: The different layers of Service Discovery

The second layer, called *Data Integration Layer*, performs registration and querying transactions. Moreover, it functions as a bridge to the information repository (in case of the CIS implementation, a *Berkeley XML database* is used) and provides operations to manipulate the data sets. The Data Integration Layer encapsulates all the functionality related to the handling of the core database functions like connection pooling, creation, and destruction. The purpose of separating this layer from the business logic layer is to de-couple both layers thus supporting several databases from different vendors.

The *Presentation Layer* encapsulates the functionality required to expose service and resource information using web service interfaces. The Presentation Layer usually communicates with the Business Logic Layer via a Façade. This is schematically shown in Figure 9.11. A WSDL document describes the interface of this layer and hence represents the access mechanism for Service Consumers.

9.2.3. Replication

Distributed computing infrastructures, like e-Science environments or grids, often handle massive amounts of client requests and have to deliver responses without substantial delays or errors. Consequently, they need redundant software and hardware resources to provide the required quality-of-service. Since replication is one basic principle to provide fault-tolerant, highly available, and scalable applications, we designed an auxiliary layer to

complement the core Common Information Service. This layer handles redundancy issues in the context of fault tolerance and availability.

In this section, we therefore first briefly sketch the requirements that a replication architecture for an information service has to meet and second introduce a specific replication framework for CIS[5].

Replicating an information service has one obvious requirement, the consistency of information. This has to be guaranteed independent of the software or hardware entity actually providing the required data. Regarding of the different entities within a replication framework, another requirement emerges: the clients of the information service do not have to deal with the complexity and topology of the framework; replication has to be transparent for them. In case an error is detected, either software or hardware-related, the replication framework should recover automatically by removing the respective replica from the compound. As a final requirement, we see transactional control as an essential task to be met by an information service for DCIs. Due to the large amount of transactions which influence the overall performance of the system, we realise an "exactly-one" transaction policy, preventing the system from dealing with redundant transactions.

An evaluation of existing replication techniques [95] revealed that it is essential to realise a combination of web services layer and database layer replication to fulfil the requirements outlined above. As the most promising and tangible approach to multi-tier replication we see, according to the evaluation, the ADAPT framework developed by a European IST project of the same name[6]. Following the ADAPT approach, CIS exposes its web services tier through a load balancer which chooses the best-suited server to process a request. The web services tier itself is deployed as a set of replicas interacting with the database tier, which again provides its own replicas.

Through the independent replication at both tiers CIS is able to handle different fail-over scenarios. The web service tier manages its own replication algorithm and relies on a centralised database. Therefore, if any of the currently serving web service replicas crashes, the next server will take over and the database tier will not be effected being considered as centralised. In the same way the database replica tier is connected to a reliable and centralised web services container. This implies that the failure of any database replica will be transparent to the web services tier.

Technically, the replication on the web services tier is based on WS-Replication [117], a specification for web services with high availability demands which ensures reliability and scalability. The database tier, however, is replicated through the introduction of an additional middleware layer, the XML Replication Middleware (XRM) [86].

9.3. The IANOS Scheduling Framework

Exploiting the benefits of SLAs in distributed computing environments is on the research agenda for a decade now [29] and in the focus of our work almost since [114]. Our first work integrating SLAs into a scheduling framework was in the course of the German VIOLA project [26], an effort which led to the creation of the *meta-scheduling service* (MSS) [42].

[5] An in-depth discussion of replication concepts and models is beyond the scope of this work, we therefore refer to [71] as one source of further information.

[6] ADAPT Middleware Technologies for Adaptive and Composable Distributed Components, last visited: January 25, 2013. http://adapt.ls.fi.upm.es/adapt.

The latest development continuing these efforts, an integration of the MSS and the *Intelligent Scheduling System* (ISS) [60], is IANOS, the so-called *Intelligent Application Oriented Scheduling Framework* [78, 113, 147]. Although it does not fully adhere to the scheduling architecture, process, and model proposed in our work, it is an intermediate step towards their realisation. With respect to the models we propose, it uses the SLA model and the scheduling model (although not to its full extend), whereas the activity and the information model are not exploited.

The overall idea behind IANOS is the allocation of computing and network resources in a coordinated fashion governed by service level agreements. Its implementation is based on state-of-the-art web service technologies and DCI-related standards (like WS-Agreement, JSDL, or GLUE), combined to provide scheduling services within a DCI.

IANOS is, by design, agnostic to specific middleware except for an adaption layer that has to be adopted depending on the target infrastructure. IANOS, furthermore, delivers a monitoring framework to collect resource and application data on past activities that can be used to detect overloaded resources and to pin-point inefficient applications that could be further optimized. The following sections introduce the IANOS architecture and the scheduling process, and, in addition, compare them to the concept proposed in Chapter 4.

9.3.1. Architecture

The IANOS architecture is presented in Figure 9.12, depicting two different DCI sites, which offer infrastructure resources and services to clients. All activity processing in IANOS is governed by SLAs, i.e. the MSS negotiates contracts with clients as well as DCI sites to secure the QoS of activity execution. To achieve this, a negotiator as introduced in Section 4.1 is integrated into the MSS.

The various services and components comprising the MSS framework are introduced in the following paragraphs.

The Meta-Scheduling Service

The *meta-scheduling service* is the core entity of the IANOS framework and is one particular instantiation of a scheduling service as depicted in Fig. 4.1. It represents providers in negotiations with clients exploiting WS-Agreement as the foundation for an SLA framework (cf. Section 7.4). The MSS validates client requests, queries the underlying DCI sites for available services, and offers suitable SLA templates to clients. Furthermore, it negotiates with clients about the actual service level, queries the broker for candidate resources and services, negotiates with potential sites about service provision, and schedules the actual activity execution.

Middleware Adapter

The *middleware adapter* mediates access to a DCI sites through a generic set of modules. It provides information about resources and handles the submission of scheduled activities. This includes the provision of site-specific SLA templates (to be offered to clients via the meta-scheduling service) and the reservation of resources and services according to the active schedule. The middleware adapter is the only module of IANOS that is con-

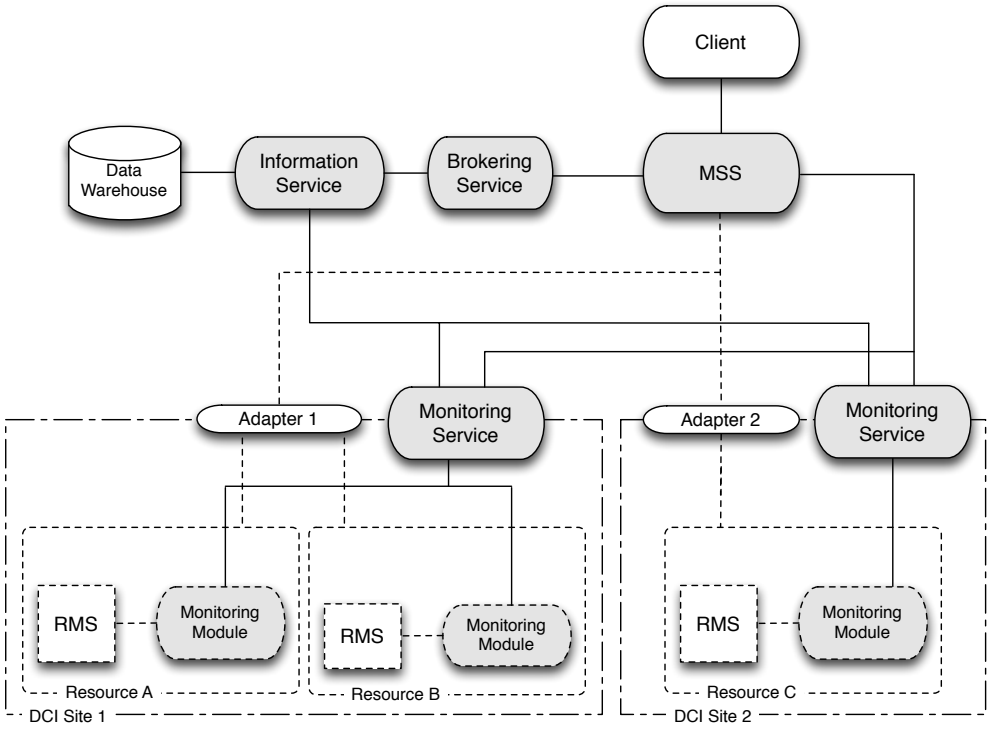

Figure 9.12.: IANOS Architecture ('MSS' is the meta-scheduling service; 'RMS' resolves to resource management service)

nected to a DCI site. Therefore, interfaces are defined for each adapter module to easily integrate with site-specific DCI middleware using a plug-in mechanism.

Brokering Service

The *brokering service* selects the candidate resources and services suitable for a particular query from the meta-scheduling service. To compute the selection, it uses two objective functions: the *execution time model* and the *cost model* [78]. Exploiting these models and data about the infrastructure, QoS requirements, and activities, the brokering service computes an ordered list of suitable configurations (including start-time and deadlines), and sends it to the MSS. The brokering service retrieves all relevant information about the DCI sites from the information service.

Information Service

The *information service* of the IANOS framework has the purpose to collect historic data about previous application runs (via an interface to the data warehouse) and data about the current state of resources and services (exploiting the interface to the monitoring service). This data is, on request, presented to the brokering service formatted according to the GLUE standard [5].

Data Warehouse

The *data warehouse* is a repository that stores all information related to services, resources, and historic application runs. In particular, the data warehouse stores the following information:

- Resource data: Application-independent infrastructure capabilities and IANOS-related parameters
- Application data: Application characteristics and requirements regarding software, libraries, memory, and performance
- Execution data: Execution-dependent infrastructure usage records, application-intrinsic parameters, and proprietary information

Monitoring Service & Module

The *monitoring service* receives activity information from the MSS and passes them to the *monitoring module* [59] to monitor the execution of applications. The monitoring module measures and collects execution information relevant to IANOS (like the MFLOPS/s rate, memory needs, cache misses, or communication and network information) during application execution. It, in addition, performs a mapping between hardware monitoring data using Ganglia[7] and application data using the site-specific RMS. At the end of the execution, it prepares and sends the respective data to information service for further processing.

9.3.2. IANOS Scheduling Process

The IANOS scheduling framework adheres to the scheduling process as outlined in Section 4.2. All eleven steps shown in Fig. 4.3 are realised, although not linked to all concepts we propose (cf. the evaluation of the realised models in Section 9.4). The instantiation of the process as implemented in IANOS is depicted in Fig. 9.13 featuring the following steps:

1. The user submits the activity request, which is called a job following IANOS terminology, in the form of a WS-Agreement SLA offer thus initiating a negotiation. Here, IANOS implements the Accept/Reject model (cf. Section 7.5.1). Prior to this step, the client can already select between the SLA templates offered by the IANOS provider(s).
2. The meta-scheduling service pre-selects suitable candidate resources based on parameters like the client's access rights or the availability of requested applications.

3a. Based on the pre-selection, the MSS queries candidate resources (represented by the respective DCI sites in Fig. 9.13) to obtain up-to-date information about the available resources.

3b. This information is the passed to the brokering service as part of a *requestCandidates* request. In addition, quality-of-service requirements derived from the initial SLA offer are sent.

[7] Ganglia Monitoring System, last visited: January 25, 2013. http://ganglia.sourceforge.net/.

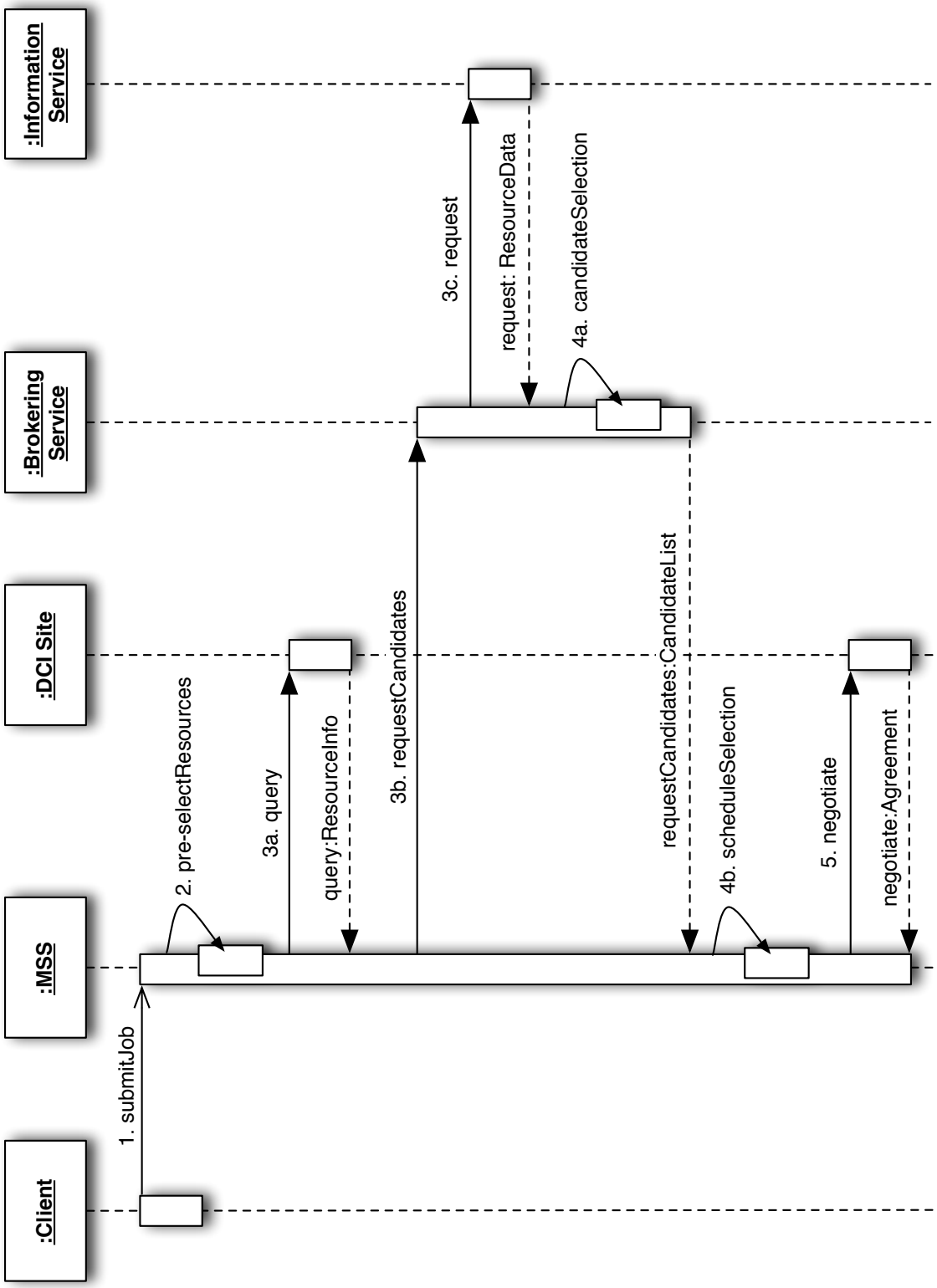

Figure 9.13.: The IANOS scheduling process ('MSS' is the meta-scheduling service)

3c. The previous step triggers a request from the brokering service to the information service that results in the delivery of data about eligible resources. This data includes inter alia historic information about past job executions.

4a. The brokering service then selects, taking all information about the job request and the resource capabilities into account, a list of candidate resources. To achieve this, the brokering service takes the IANOS *cost function model* [59] into account and provides the MSS with an ordered list of candidates.

4b. The meta-scheduling service takes the first candidate and schedules it accordingly.

5. Based on the schedule, the MSS negotiates with the respective distributed computing infrastructure sites to agree upon the quality-of-service requested by the client. For this step, again, WS-Agreement and the Accept/Reject negotiation model are used.

In case the negotiations with the DCI sites are successful, all prerequisites are fulfilled for the IANOS provider to accept the initial SLA offer and finalise the agreement with the client (the respective and all following steps are not depicted in Fig. 9.13). Then, according to the scheduling process we propagate (cf. Fig. 4.3), the implementation phase starts and the infrastructure can be provisioned, the job can be executed, monitored, and assessed governed by the SLA. We do not consider these steps in more detail here, since they do not show any particularities in comparison to the general process.

9.4. Discussion of the Realisation of the Models

All four core models have been realised: the activity model (cf. Chapter 5) through the activity management framework (cf. Section 9.1), the information model (cf. Chapter 6) through the Common Information Service (cf. Section 9.2), and both the service-level agreement model and the scheduling model (cf. Chapter 7 and Chapter 8) through the IANOS scheduling framework (cf. Section 9.3). All implementations have been tested within projects and the results have been published in conference proceedings and journals. The development of the models and their realisations, however, progressed and we therefore briefly assess their status based on the following criteria:

- Compliance with the model
- Sustainability
- Potential for integration into a DCI

The results of the assessment are summarised in Table 9.1.

In this chapter we introduced the realisations of our four core models. Furthermore, we presented the application of the models to scenarios like implied volatility calculations, information provision, and SLA-governed scheduling. A complete realisation of the generic scheduling architecture applying our scheduling process has not been realised. The rationale for this is discussed in Chapter 11.

Table 9.1.: Assessment of the models and their realisations

	Activity model	Information model	SLA model	Scheduling model
Realisation	Activity management framework (NextGRID project)	Common Information Service (NextGRID project)	IANOS scheduling framework (IANOS consortium)	IANOS scheduling framework (IANOS consortium)
Compliance with model	Full	Full	Full	Partly (no delegation)
Sustainability of the model	The Activity Instance Container at the Open Grid Forum	CIM is further developed through DMTF	WS-Agreement (Negotiation) is an OGF recommendation	Not assessable (full model has only been simulated)
Sustainability of the realisation	The NextGRID implementation is deprecated, but the German project DGSI (http://dgsi.d-grid.de/) has implemented the activity instance description	The successor of the NextGRID realisation, also called Common Information Service, is part of the UNICORE framework and will be further maintained by the UNICORE consortium; it is based on GLUE	The IANOS framework is maintained by Fraunhofer institute SCAI (www.ianos.org) and the inherent WS-Agreement implementation WSAG4J (http://packcs-e0.scai.fraunhofer.de/wsag4j/) is used in a multitude of projects	The IANOS framework is maintained by Fraunhofer institute SCAI
Potential for integration	The existing realisation is not maintained any more and should therefore not be used. The DGSI implementation of the activity instance description could be an option	If using UNICORE for setting up an DCI, CIS is the right choice; for other middleware a thorough assessment is mandatory	WSAG4J is configurable and extensible to serve as the core SLA framework for any DCI	It is recommend to evaluate concepts and design of the current implementation, but the integration of the current implementation is not recommended as it is not integrating with all the other models

Part V.

Evaluation

10. Simulation and Evaluation

To evaluate our SLA-based scheduling model, we simulate a multi-site scheduling environment according to one of the application scenarios introduced in Section 3.1. The evaluation is a two-stage process featuring artificial workload traces as well as realistic ones. Through this, we show the feasibility of our model and provide an initial idea of how the different scheduling strategies interact.

Our scheduling architecture, process, and model, including the necessary foundations, i.e. the core models, have been specified and partly implemented. An extensive evaluation of all these assets would imply the implementation of a more or less complete DCI middleware, an undertaking way beyond our work[1]. We therefore decided to take the activity delegation application scenario from Section 3.1.3 and to apply the scheduling model from Chapter 8 to it. Through this, we show the feasibility of our scheduling model mapped to a scenario of growing importance in cloud-like environments: scaling-out to external providers or, to express it in our terms, the delegation of activities.

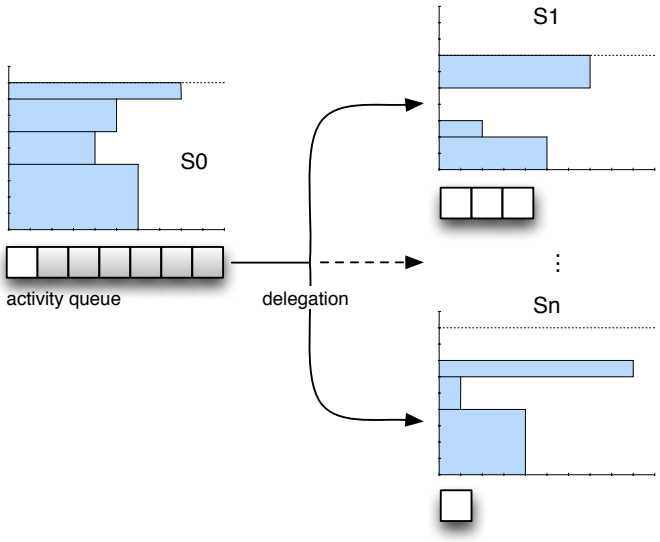

Figure 10.1.: The simulated application scenario: scheduling service S_1 delegates activities to services S_1, ..., S_n

[1]Please refer to Chapter 11 to retrieve more information regarding this issue.

Fig. 10.1 shows the inherent scheduling question of the delegation scenario, which is a distributed off-line scheduling problem. Each site S_n has an active schedule, depicted with the time on the horizontal axis and the allocated number of resources, which are CPUs in our simulation, on the vertical axis. The intersection of both axes represents the current point in time t_0. Underutilised sites, with utilisation of $u < 1$, have free slots ($CPUs_{free}(t_0)$) and are potential targets for delegation.

The local scheduling strategy tries to associate one activity to one specific machine that is under the control of the respective site. In addition to the local scheduling, there are consequently potentially $N - 1$ available sites to delegate activities to, each with a limited set of resources and potentially different parameters like processing cost or CPU speed. In Fig. 10.1, for instance, site S_0 utilises all its resources and has seven activities in its queue. Six of these activities have already been chosen to be delegated to other sites (depicted as grey boxes). Sites S_1 and S_n, however, have spare resources and at the same time activities in their queue. This implies that the queued activities have resource requirements that exceed the locally available resources.

The purpose of this simulation is, in short, to evaluate our scheduling model through the application of various scheduling strategies to answer the question which activities are scheduled locally and which are delegated to remote sites.

10.1. Simulation Environment

To solve the scheduling problem, we apply our scheduling model introduced in Section 8. To achieve this, we extended an existing scheduling simulator, called teikoku [57], to execute SLA-based scheduling strategies using the WSAG4J framework[2], and define the following pre-requisites:

- Activities are either scheduled locally, queued locally, or delegated.
- Delegation is governed by service-level agreements.
- Only one-step delegations are considered. In case a negotiation fails, no other site is contacted during the present scheduling cycle.
- A site can delegate multiple activities at once.
- SLAs and their templates are expressed using WS-Agreement (see Section 7.4).
- The negotiation protocol is the WS-Agreement protocol implementing an accept/reject negotiation model (see Section 7.5.2).
- There is exactly one scheduling service per site.
- The simulation environment does not support advance reservation.
- Workload traces to be fed into the simulation system follow the Standard Workload Format (SWF)[3].

[2] Welcome to WSAG4J, last visited: January 25, 2013. http://packcs- e0.scai.fraunhofer.de/wsag4j/.

[3] The Standard Workload Format, last visited: January 25, 2013. http://www.cs.huji.ac.il/labs/parallel/workload/swf.html.

Within the simulation environment, we set up a number of sites, as described in Section 10.5 and Section 10.6, respectively. Activity-specific information is either generated or taken from SWF traces and integrated into service-level agreement templates, both of which are introduced below.

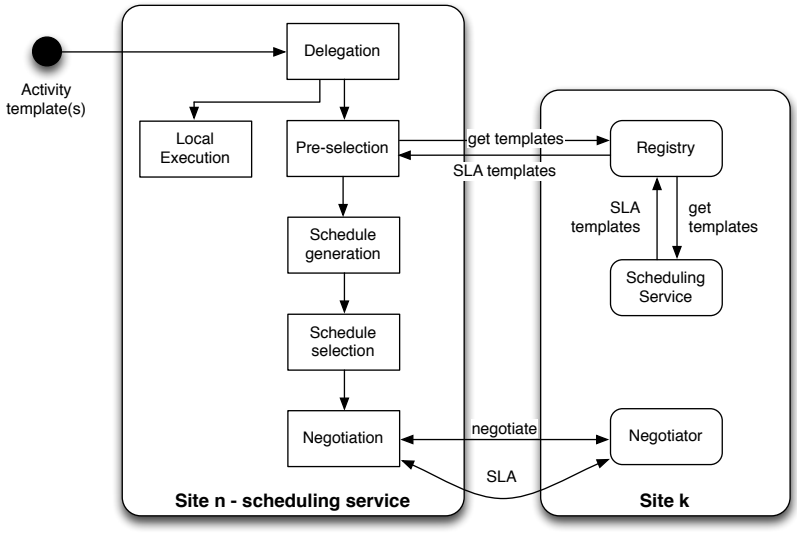

Figure 10.2.: The simulation environment exemplified depicting two sites

The simulation set-up is depicted in Fig. 10.2. It shows two sites, Site n receives activity templates and executes actions according to the scheduling model, whereas Site k is a potential destination for delegation. In a first step, Site n executes a delegation strategy to determine which activities are candidates for delegation and which are scheduled locally. The strategy is executed upon all queued and newly submitted activities. In case of the simulation, locally scheduled activities are executed immediately, a procedure equivalent to the execution a first-come-first-serve scheduling algorithm, until no local resources are available any more. This is done to 'de-couple' local and remote jobs, hence making the criteria for delegation candidates clear and the evaluation results reliable. In a production DCI, the scheduling of local jobs will most likely also follow our scheduling model and the scheduling service will apply the different strategies to compute the best-suited schedule.

In a next step, Site n requests templates from the registry of Site k. In a DCI implementing our generic architecture according to Fig. 4.1, this information would be provided by the information service, in case of the simulation, the SLA templates are generated by the scheduling service (cf. Section 10.1.2 later in this chapter). The SLA templates are then fed into the pre-selection process and the respective scheduling actions are executed as outlined in Section 8.1.2. The final step is the application of the negotiation strategy which is, as the simulation uses WS-Agreement (not WS-Agreement Negotiation), a simple accept/reject model applied using the WS-Agreement basic negotiation protocol. If the offered agreement is accepted, the respective activity is delegated to Site k. Not all initiated delegation requests lead to an offer or even to an agreement. A template may contain constraints which are not acceptable for the initiator. An offer, however, may be not accepted due to status changes of other activities, which e.g. have been started in the

mean time and consume resources necessary to accept a delegation request.

10.1.1. Activities

The activities used in the simulations are described using the following parameters[4]

- ActivityID — a unique identifier for each activity in the trace.
- Requested runtime — runtime requested by the client (wall-clock execution time of the activity).
- Number of requested CPUs — number of CPUs requested by the client.
- Submission time — submission time relative to the beginning of the trace.
- Average CPU time used — average time consumed (summed over all allocated CPUs).
- Number of allocated CPUs — actually allocated CPUs.

For our simulations, these parameters are either derived from workload traces following the SWF (see also Section 10.6) or are artificially generated (as described in Section 10.5).

10.1.2. Service Level Agreements

Fig. 10.3 depicts the various WS-Agreement-related documents that are created during a simulation cycle. The first document is a pre-defined draft template for each activity to be scheduled. This template is processed and filled with simulation-specific parameters. The resource-related activity data from the workload traces (see previous section) is then added to the template as Service Description Terms (cf. Section 7.4). After that, the modified template is transformed into an agreement offer document that, upon acceptance, becomes an agreement.

In the first step, data about the specifics of the simulation environment is added to the draft template, including information about the agreement initiator and responder roles, template ID, state information, the price of the service offer, maximum number of CPUs offered, maximum runtime offered, and source as well as destination site of a potential delegation.

Before the actual WS-Agreement offer is created, the template is modified by adding descriptions of the activities that should be delegated an thus negotiated. The descriptions are added to the template as Service Description Terms using Job Submission Description Language (JSDL) [7]. Each activity is described separately by using its Universally Unique Identifier (UUID) in an element called *JobIdentification*, its number of requested CPUs in an element called *TotalCPUs*, and its requested runtime in an element called *TotalCPUTime*. After adding all activities to the template, another Service Description Term (SDT) is entered which contains sums of the totally requested runtime and number of CPUs. This is required to compare the requirements with the maximum runtime and the maximum number of CPUs offer. After these modifications have been accomplished, the template is converted into an agreement offer compliant with the WS-Agreement specification. This

[4] These parameters are a mixture of client-side resource requirements and resource usage information logged in the traces. All this data is coded into a UDAP description (cf. Section 5.1.3).

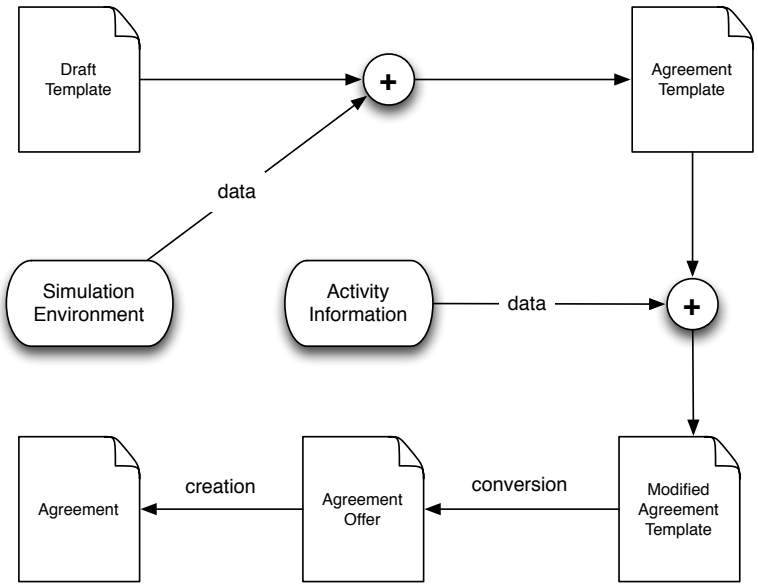

Figure 10.3.: The different states an SLA passes during a simulated scheduling cycle

offer is then the staring point for negotiations and the (potential) creation of an agreement about activity delegation.

10.2. Optimisation Criteria

The selection of one or more optimisation criteria for the simulation (and also for production DCIs) is a difficult and far-reaching decision as the criteria reflect the, in general conflicting, business objectives of the different stakeholders involved in service provisioning and consumption. We therefore decided to apply three different criteria and simulate the effects. Furthermore, we observe theses effects from a local and a global perspective, with one simulated site and, respectively, the whole simulation environment as reference systems.

10.2.1. Local Optimisation Criteria

We use the following local optimisation criteria to for a DCI site providing a scheduling service: Average Weighted Response Time (AWRT), utilisation, and cost.

The *AWRT* is defined according to Ernemann et al. [45]:

$$AWRT = \frac{\sum_{j \in activities} (ResourceConsumption_j \cdot (endTime_j - startTime_j))}{TotalResourceConsumption} \text{ with}$$

$$TotalResourceConsumption = \sum_{j \in activities} ResourceConsumption_j \text{ and}$$

$$ResourceConsumption_j = (ReqResources_j \cdot (endTime_j - startTime_j)).$$

The metric *utilisation* states how many processors of all available processors are in use:
$\mathfrak{u} = \frac{\mathrm{CPUs_{used}(t)}}{\mathrm{CPUs_{total}}}$.

It helps DCI providers to assess how well their computing resource are utilised. To prevent idle systems, negotiation strategies (like those introduced in Section 7.5) help to optimise load by delegating activities at peak times and receiving remote activities during times with low utilisation.

The *cost* to execute all locally submitted activities is another metric that is applied in our simulations. As an optimisation criterion it is interesting for providers to exploit it to minimize the cost for computing incoming activities.

10.2.2. Global Optimisation Criteria

The system-wide optimisation criteria include the local ones described before. From a global perspective, it is also profitable to decrease AWRT and price, and to increase the utilisation at the same time. The global metrics will be provided as average values spanning all sites at any given moment. On a global scale, the overall simulation time can also be used as a metric. It is the simulated time, no the wall-clock time, that it takes to execute all available activities. The wall clock time is expected to increase for simulations which execute a large number of negotiations, as these consume a considerably long time. In such cases, the simulation's wall-clock time rises whereas at the same time e.g. the AWRT ideally shrinks with each negotiation.

10.3. Strategies and Policies

The scheduling model introduces a variety of strategies that can be used in combination. One purpose of the simulation is to evaluate a number of them and the effect of different optimisation criteria. This section introduces the specific strategies we have implemented including the provider-side policies to reflect business objectives.

10.3.1. Strategies related to the Scheduling Model

The implemented and simulated strategies are introduced in detail in Appendix C. They are ordered according to the role they take in the scheduling model:

- Delegation Strategies:
  - CPU Threshold Delegation Strategy (*cpu_thresh_del*)
  - Runtime Threshold Delegation Strategy (*runtime_thresh_del*)
  - Activity Threshold Delegation Strategy (*activity_thresh_del*)
  - Global Average Weighted Response Time Delegation Strategy (*global_awrt_del*)
  - No Delegation Strategy (*no_del*)
  - Random Delegation Strategy (*random_del*)
  - Wave All Activities Through Delegation Strategy (*wave-all_del*)

  - Wave All New Activities Through Delegation Strategy (*wave-all-new_del*)

- Pre-selection Strategies
  - Random Pre-selection Strategy (*random_pre-sel*)
  - Wave Through Pre-selection Strategy (*wave_pre-sel*)
- Scheduling Strategies
  - Random Scheduling Strategy (*random_sched*)
  - Price Greedy Scheduling Strategy (*price-greedy_sched*)
  - Lowest Cost First Lowest CPU First Scheduling Strategy (*low-cost_low-cpu_first_sched*)
  - Lowest Cost First Shortest Runtime First Scheduling Strategy (*low-cost_low-rt_first_sched*)
  - Lowest Cost First Minimal CPU Tardiness Scheduling Strategy (*low-cost_first_min_cpu-tard_sched*)
- Schedule Selection Strategies
  - Random Schedule Selection Strategy (*random_sched-sel*)
  - Wave Through Schedule Selection Strategy (*wave_sched-sel*)
- Negotiation Strategies
  - WS-Agreement Negotiation Strategy (*wsag_basic_neg*)

10.3.2. Provider-related Policies

We introduce a set of policies that providers can use to implement their business objectives. These following policies have been considered (see Appendix D for further details):

- CPU Time Policies
  - Global AWRT CPU Time Policy (*global-awrt_cpu-time_policy*)
  - Random CPU Time Policy (*random_cpu-time_policy*)
  - Static CPU Time Policy (*static_cpu-time_policy*)
  - No CPU Time Policy (*no_cpu-time_policy*)
- Pricing Policies
  - (Global) Utilisation-Based Pricing Policy ((global_)*util_price_policy*)
  - Static Pricing Policy (*static_price_policy*)

10.4. Evaluation Metrics

This section covers the metrics used for evaluation. It first introduces the various metrics and describes the relations between them. Then, these metrics are discussed from the perspective of the consumer and the the provider.

10.4.1. Average Weighted Response Time

The response time for each activity is determined measuring the time between the submission of an activity and its end time. The result is weighted using the requested resources, i.e. the requested number of CPUs, and the average per site is computed. This metric is also used on a global, i.e. simulation-wide, basis calculating the mean value over all sites.

10.4.2. Resource-related Metrics

The *LocallyAvailableCPUTime* metric reflects the amount of CPU time offered to other sites. The *RequestedCPUTime* is the accumulated value of CPU time requested for activities in the current delegation cycle. *LocallyAvailableCPUs* is the amount of free CPUs at a specific site. *RequestedCPUs* is the accumulated number over all activities' CPUs in the current negotiation. *ActivitiesToDelegate* reflects the number of activities that have been chosen by the delegation strategy, whereas *RemoteSites* is the number of remote sites that are available for delegation. *Utilisation* captures the current utilisation of a site and is defined as

$$u = 1 - \frac{\mathrm{freeCPUs}}{\mathrm{totalCPUs}}.$$

QueuedCPU is the accumulated amount of CPU requests currently queued. *WaitingActivities* and *WaitingTime* are the respective number of activities currently queued and the accumulated amount of requested CPU time of activities currently in queue. The latter three metrics are useful to asses the load of a queue.

10.4.3. Price-related Metrics

The *ActualPrice* metric represents the actually paid price for the execution of an activity. It is the price paid for delegating an activity (i.e. remote execution) or the cost to execute it locally. The *AveragePricePerHour* is the mean value calculated taking the prices of all templates in the current delegation cycle into account. The *OwnCost* metric represents the cost which accrue to execute an activity locally. The *OwnCostPerHour* is the previous metric weighted with time (and the price offered to other sites). The *SavedThroughDelegation* metric captures the savings or cost arising from delegation. It is calculated according to the following formula:

$$\mathrm{SavedThroughDelegation} = \mathrm{OwnCost} - \mathrm{ActualPrice}$$

The result is 0 if the activity is executed locally; if delegated SavedThroughDelegation is either positive or even negative depending on the price for delegation.

10.4.4. Negotiation-related Metrics

SuccessfulNegotiations is the number of successfully established negotiations. The number of established agreements is likely to be higher since one negotiation often implies the creation of a number of agreements with various sites. *NegotiationDuration* is the duration of the whole negotiation process. It is a value measured in real time, not simulation time, and provides valuable information about the cost to carry out negotiations. *Outcome* notes the outcome of a negotiation, its value is 0 in case the current negotiation fails and

1 otherwise. The metric *StartedNegotiations* keeps track of the number of attempted negotiations. This value is useful in relation to SuccessfulNegotiations since it delivers data regarding the rate of successful negotiations.

10.4.5. Stakeholders' Perspective

In our application scenario, we observe two different perspectives on the evaluation metrics: consumer and provider (of resources and services). Both have a different view on the importance of a certain metric, a fact we briefly discuss in this section.

Consumer

The AWRT is in general important for consumers. It is a reference value indicating how long it might take for a submitted activity before it is actually executed. Especially in cases where a timely processing of an activity is business-critical this value becomes a key indicator to choose a resource provider. A consumer might also be interested in the savings that arise from a delegation. If a provider is less expensive than another or delegation is cheaper than local execution of an activity, this metric becomes relevant. Last, but not least, the number of CPUs available at the delegation target is of relevant for a consumer. It describes a resource capability and is, in our simulation, used exemplary for the large variety of resource capabilities normally offered to consumers through an SLA.

We therefore use the following metrics to evaluate the simulation of the scheduling model form the consumer's perspective: AWRT, SavedThroughDelegation, and LocallyAvailableCPUs. More metrics could have been included in the evaluation, but are either redundant or in general not relevant for consumers.

Provider

A resource provider is, compared to the consumer, more interested to gather information on idle resources and on the utilisation of their systems. Therefore, utilisation $\mathfrak{u}$ (as defined in Section 10.4.2) is an important, if not the most important, metric for a resource provider. Further metrics are WaitingTime, WaitingActivities, and QueuedCPUs, indicating whether whether a site is overloaded by submissions. Regarding negotiations, a provider is interested to know how long these negotiations may take and how many succeed. This is allows to estimate the overhead related to negotiations. Last, but not least, resource providers are interested to keep their costs under control and therefore how much they can save through delegation.

We therefore exploit the following metrics to evaluate our simulations from the providers perspective:

- Utilisation,
- WaitingTime,
- WaitingActivities,
- QueuedCPUs,
- NegotiationDuration,

- Outcome, and
- SavedThroughDelegation.

10.5. Evaluation of Artificial Traces

In a first pass, a trace generator has been written to create custom workload traces. Using this tool, traces can be modelled by changing the activity characteristics. Both runtime and CPU requirements can be configured for activities. The generated traces have a simple structure and allow the generation of a 'laboratory environment' for simulations. The advantage of using artificial traces is the possibility to quickly assess key features for a later production set-up. Disadvantages evolve around the fact that theses traces do not reflect user behaviour, which differs significantly depending on user grouping, application mixture, and other factors. We, therefore, consider realistic workloads in Section 10.6 and try to reveal additional data complementary to the method used in this section.

10.5.1. Simulation Set-Up

The six sites in our simulation are configured half as sinks and half as sources. Sinks have more resources than necessary to execute the assigned workload. Thus, they operate with a low utilisation and a low AWRT (in case delegation is not applied). Sources, in turn, have more submissions than they can handle, resulting in general in an AWRT higher than at sinks. The utilisation of sources can vary depending on the size of the submitted activities.

Six machines with the following characteristics are used to evaluate artificial traces:

- Site 1 – 3 (sinks):
  - Maximal number of CPUs: 384.
  - Initial price per hour: 100 units.
  - Initial runtime: $0.5 \cdot 10^4$ seconds.
- Site 4 – 6 (sources):
  - Maximal number of CPUs: 128.
  - Initial price per hour: 100 units.
  - Initial runtime: $5 \cdot 10^4$ seconds.

Fig. 10.4 shows an assessment for the artificial traces, which are generated according to the algorithms shown in Appendix E. The requested number of CPUs increases from site 1 to 3 and from site 4 to 6 accordingly. The maximally requested runtime is the same for all sites, however, the average again shows the distribution of short and long activities. This distribution becomes also evident looking at the maximal activity size, which is the product of requested CPUs and requested runtime. The total simulation time is the same for all traces.

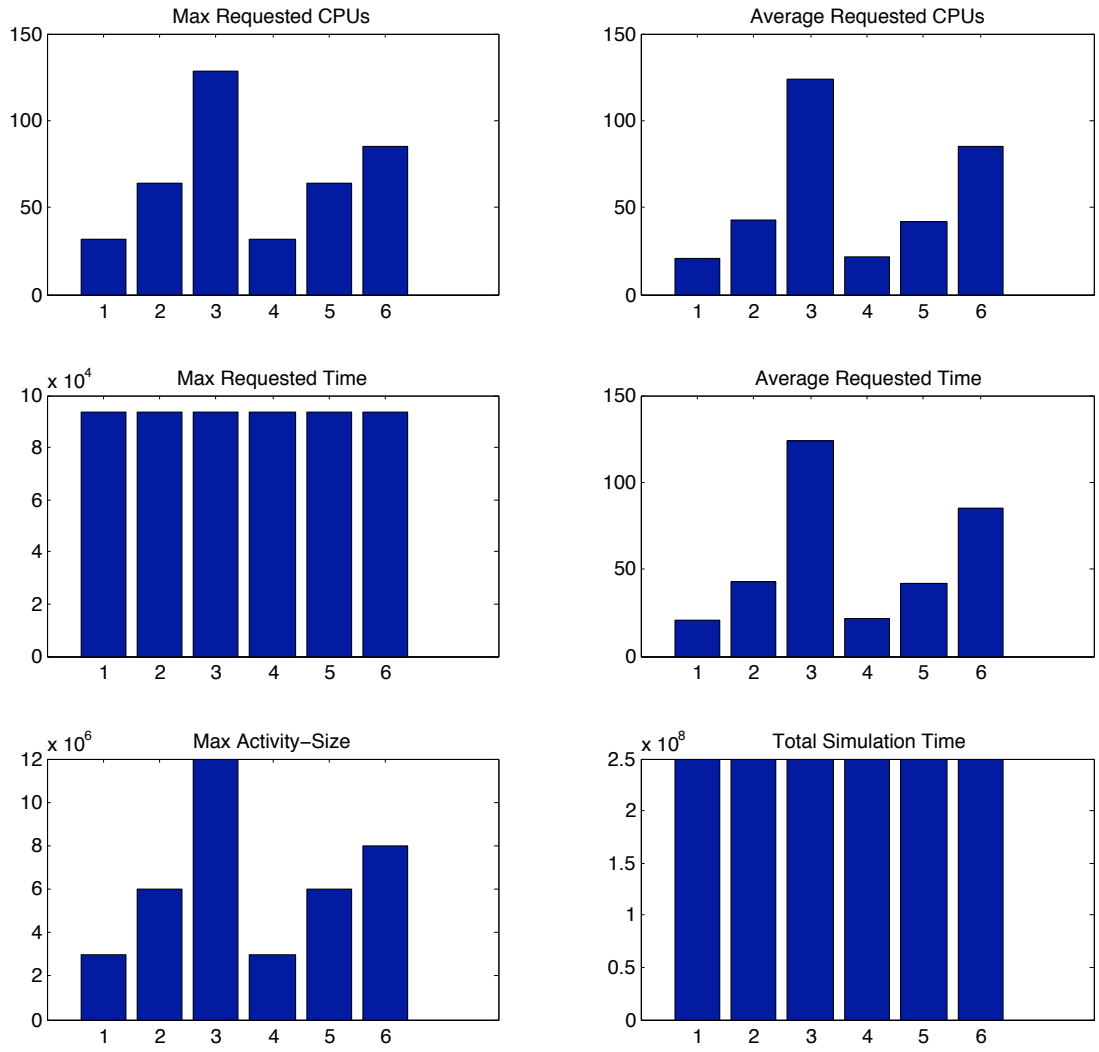

Figure 10.4.: Characteristics of the artificial traces (The abscissa shows the different sites, the ordinate the following values: number of requested CPUs (first row), requested runtime in seconds (second row), and the product of the number of requested CPUs and the requested runtime in seconds as well as the total simulation time in milliseconds (third row)).

10.5.2. Reference Run

A reference run with the *no_del* delegation strategy has been executed to provide a basis for the evaluations of subsequent simulation set-ups. This strategy enforces that no delegation takes place implying that all simulation sites have to be configured to use the *no_del* strategy. As a result, source sites keep being overloaded with activities and sinks are under-utilised.

The metrics AWRT and utilisation for the reference run are depicted in Fig. 10.5 and Fig. 10.6, respectively. Each upper diagram shows the current value with the simulation time on the abscissa. Each lower diagram shows the mean values for the corresponding

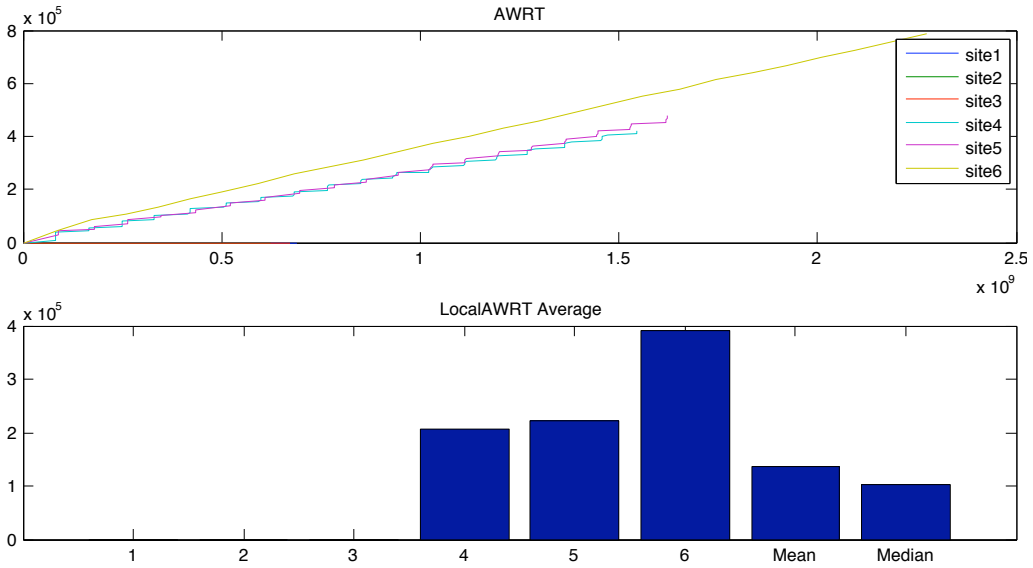

Figure 10.5.: The AWRT for the No Delegation Strategy (*no_del*) reference run (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including the mean value and the median).

sites and, in addition, the mean and median values.

Diagram 10.5 shows the AWRT without any delegation strategy. Sites 1, 2, and 3 show a negligible average weighted response time in contrast to sites 4, 5, and 6. The diagram, too, reflects the dependence of the average AWRT on the average activity size with site 6 showing the steepest slop. Furthermore, the evaluation reveals that the first three sites finish activity processing around $0.7 \cdot 10^9$ milliseconds. This point in time is shortly after the last activity submission in the trace. Contrary to the under-utilised sites, sites 4 and 5 need almost twice as long to process activities and site 6 needs around $2.25 \cdot 10^9$ milliseconds.

Diagram 10.6 depicts the utilisation for same simulation run. Sites 1, 2, and 3 have a low utilisation, as they provide, at any time, more resources than necessary to process the simulated load. Sites 4 and 5 are initially almost a 100% utilised, but utilisation decreases due to fragmentation. Site 6 has a utilisation of circa 66% which may appear low. The reason for that is the trace design which assigns a number of $256/3 = 85.\bar{3}$ CPUs requested CPUs to each activity. As site 6 provides 128 CPUs, just one activity can be executed at a time. This leads to the utilization of $85.\bar{3}/128 = 0.\bar{6}$.

The average key metrics for the *no_del* reference run are listed in 10.1 to provide reference values for subsequent simulations.

10.5.3. Best Set-ups for Artificial Traces

We simulated a large variety of strategy combinations to find the best set-up for delegation. For these set-ups, every source site applies a delegation strategy $s \in \mathrm{DelegationStrategies} \neq \mathrm{no\_del}$. As CPU Time Policy it is mandatory to use *no_cputime_policy* to prevent delegation to a source site. Other strategies can be chosen arbi-

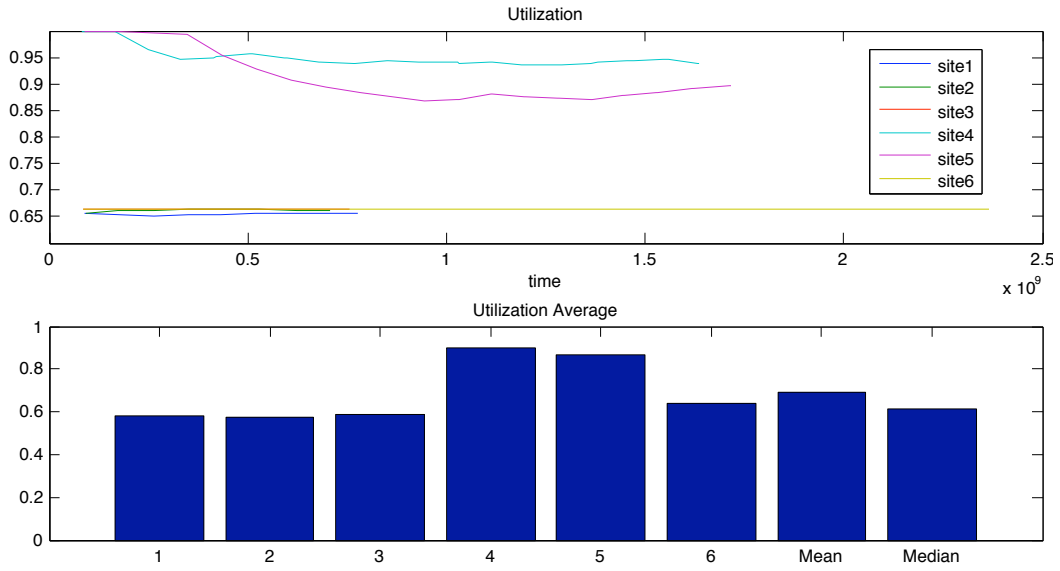

Figure 10.6.: The utilisation for the no-delegation reference run (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean and median).

Metric	Value
AWRT	$1.36 \cdot 10^5$
SavedThroughDelegation	N/A
NumberLocallyAvailableCPUs	88.3
Utilisation	0.692
WaitingActivities	999
WaitingTime	$3.92 \cdot 10^6$
QueuedCPUs	16725
NegotiationDuration	N/A
Outcome	N/A

Table 10.1.: Average values for the key metrics of the reference run

trarily and, thus, influence to the simulation result.

A sink site applies the *no_del* strategy to prevent delegation of activities to a source site. As CPU time policy any $s \in \mathrm{CPUTimePolicies} \neq \mathrm{no\_cpu\text{–}time\_policy}$ can be used. This ensures that a CPU time value is offered to a source and consequently the delegation to a sink is possible.

In addition to the aforementioned pre-requisites, i.e. sinks do not delegate activities and sources do not accept activities for delegation, we restricted the usage of strategies and policies applied within our scheduling model:

- The pre-selection strategy is always *wave_pre-sel*. This implies that all SLA templates are taken into consideration.
- The schedule selection strategy is *wave_sched-sel*. Consequently, all schedules are

Strategy / Policy	Source	Sink
Delegation Strategy	*activity_thresh_del*	*no_del*
Pre-selection Strategy	*wave_pre-sel*	*wave_pre-sel*
Scheduling Strategy	*low-cost_low-cpu_first_sched*	*low-cost_low-cpu_first_sched*
Sched. Select. Strategy	*wave_sched-sel*	*wave_sched-sel*
Negotiation Strategy	*wsag_basic_policy*	*wsag_basic_policy*
CPU Time Policy	*no_cpu-time_policy*	*static_cpu-time_policy*
Pricing Policy	*global_util_price_policy*	*global_util_price_policy*

Table 10.2.: The best set-up for AWRT optimisation

Metric	Value
AWRT	$0.523 \cdot 10^5$
SavedThroughDelegation	$1.23 \cdot 10^8$
NumberLocallyAvailableCPUs	46.72
Utilisation	0.734
WaitingActivities	243
WaitingTime	$4.09 \cdot 10^6$
QueuedCPUs	8170
NegotiationDuration	3
Outcome	0.93

Table 10.3.: Average values for key metrics of the best AWRT set-up

candidates for negotiation.

This set-up is feasible for configurations like we used for our simulation. In cases with more sites and more delegation targets it is sensible to restrict candidate services and schedules to make computation and negotiation less costly.

Best Set-up for AWRT Optimisation

Table 10.2 lists the strategies and policies used to achieve best results regarding the global mean of the per-site AWRT values. Table 10.3, however, shows the key metrics for this simulation. In contrast to the reference run, the AWRT decreases from $1.36 \cdot 10^5$ to $0.523 \cdot 10^5$ milliseconds, which is a considerable improvement. Furthermore, this set-up provides an acceptable negotiation duration and a good success rate for negotiations (the mean Outcome is 0.93 with 1 being a successful negotiation and 0 otherwise.

Figure 10.7 depicts the AWRT per site, which has improved significantly for sites 4 and 5 compared to Fig. 10.5. Site 6 shows only minor improvements because the mean requested number of CPUs of its activities is so big that they can hardly be executed on other sites. A small decrease of the AWRT, though, can be observed compared to the reference run.

In Fig. 10.8 a slight positive effect on the utilisation can be perceived. In particular the utilisation of sites 1 to 3 increases from approximately 0.6 in the reference run to a values

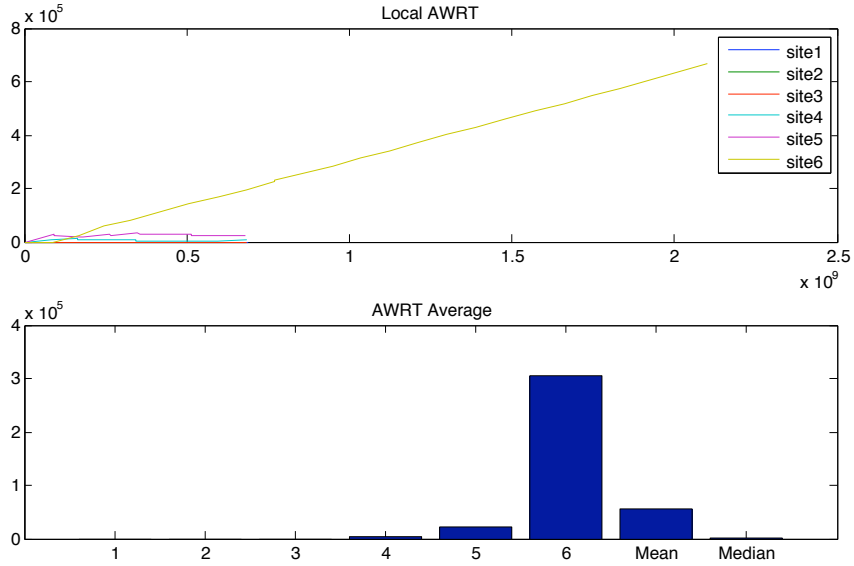

Figure 10.7.: The AWRT for the AWRT-optimised set-up according to Table 10.2 (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including mean value and median).

near 0.7. The global gain in utilisation, however, is small with a change from 0.692 to 0.734. Moreover, sites 1 to 5 finish the processing of activities at a similar time, an effect due to delegation. Site 6, however, in general fails to delegate its activities as the sinks do not offer big enough slots (in terms of LocallyAvailableCPUs). A more adaptive approach towards CPU Time Strategies might solve this issue.

Fig. 10.9 displays the negotiation-related evaluation results. They reveal that only sites 4 and 5 have completed negotiations, site 6 is in general unable to pass on any of its large activities, and sites 1 to 3 do not apply delegation strategies. In this set-up, sites 4 and 5 gain by delegating activities according to the *util_price_policy*.

Best Set-up for Utilisation Optimisation

Table 10.4 lists the strategies and policies used to achieve best results regarding global utilisation. The respective mean values for the key metrics are shown in Table 10.5, revealing that the optimal mean utilization is 0.768 which is a gain of 0.076 compared to the reference run (cf. Table 10.1). Even though this is an improvement of 10,98%, there is potential for further enhancement as an analysis of queued activities exposes, showing that the majority of waiting activities is small or medium sized and consequently suitable for delegation.

The AWRT according to Fig. 10.10 also shows an improvement compared to the reference set-up. Site 6, however, is only marginally participating in the negotiation process due to reasons mentioned in the previous section. As a results, its AWRT again rises to the level of the reference run. Fig. 10.11 visualises the utilisation for all sites separately. It is worth

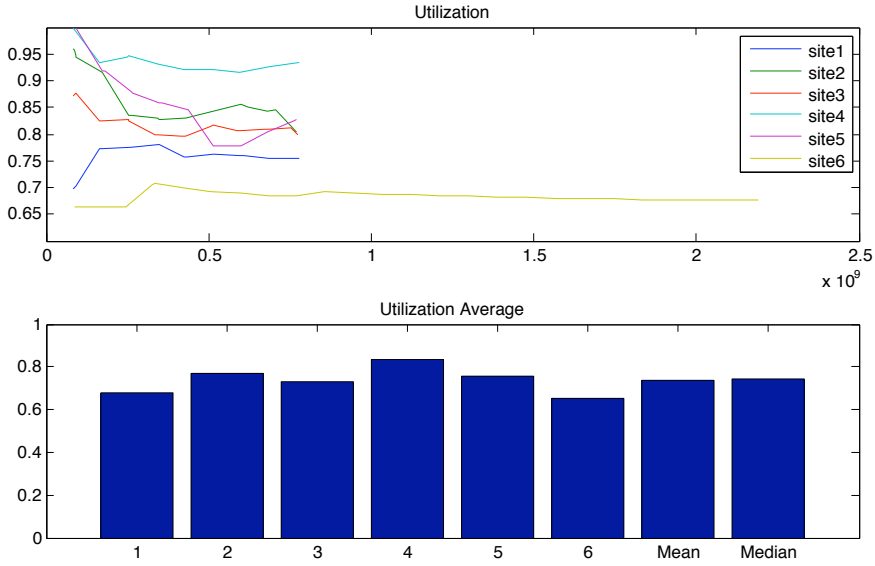

Figure 10.8.: The utilisation for the AWRT-optimised set-up according to Table 10.2 (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean value and median).

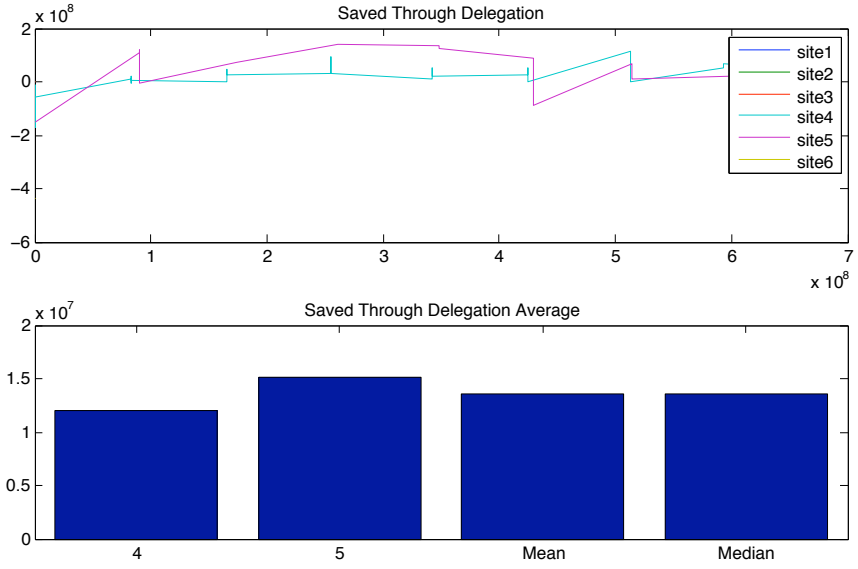

Figure 10.9.: The cost savings through delegation for the AWRT-optimised set-up according to Table 10.2 (The upper diagram shows the savings as a function of the simulation time (in milliseconds). The lower diagram shows the average savings per site including mean value and median).

noting that the utilisation of sites 1, 2, and 3 improves significantly compared to the refer-

Strategy / Policy	Source	Sink
Delegation Strategy	*runtime_thresh_del*	*no_del*
Pre-selection Strategy	*wave_pre-sel*	*wave_pre-sel*
Scheduling Strategy	*low-cost_low-cpu_first_sched*	*low-cost_low-cpu_first_sched*
Sched. Select. Strategy	*wave_sched-sel*	*wave_sched-sel*
Negotiation Strategy	*wsag_basic_policy*	*wsag_basic_policy*
CPU Time Policy	*no_cpu-time_policy*	*global-awrt_cpu-time_policy*
Pricing Policy	*static_price_policy*	*static_price_policy*

Table 10.4.: The best set-up for utilisation optimisation

Metric	Value
AWRT	$0.7 \cdot 10^5$
SavedThroughDelegation	0
NumberLocallyAvailableCPUs	86.9
Utilisation	0.768
WaitingActivities	225.9
WaitingTime	$4.03 \cdot 10^6$
QueuedCPUs	1050
NegotiationDuration	2.94
Outcome	0.89

Table 10.5.: Average values for key metrics of the best utilisation set-up

ence run, site 2, for instance, shows an improvement from 0.6 to over 0.8. The cost saving through delegation is zero for all sites due to the application of the *static_price_policy*, which implies that all sites execute activities at the same price.

Best Set-up for Cost Optimisation

Table 10.6 groups the various strategies and policies used to optimise cost for the simulation of artificial traces. Although mean value of cost saved through delegation is optimal compared to the other set-ups, most other key metrics show significantly worse values in contrast to set-ups optimising AWRT and utilisation.

Strategy / Policy	Source	Sink
Delegation Strategy	*cpu_thresh_del*	*no_del*
Pre-selection Strategy	*wave_pre-sel*	*wave_pre-sel*
Scheduling Strategy	*low-cost_low-rt_first_sched*	*low-cost_low-rt_first_sched*
Sched. Select. Strategy	*wave_sched-sel*	*wave_sched-sel*
Negotiation Strategy	*wsag_basic_policy*	*wsag_basic_policy*
CPU Time Policy	*no_cpu-time_policy*	*static_cpu-time_policy*
Pricing Policy	*global_util_price_policy*	*global_util_price_policy*

Table 10.6.: The best set-up for cost optimisation

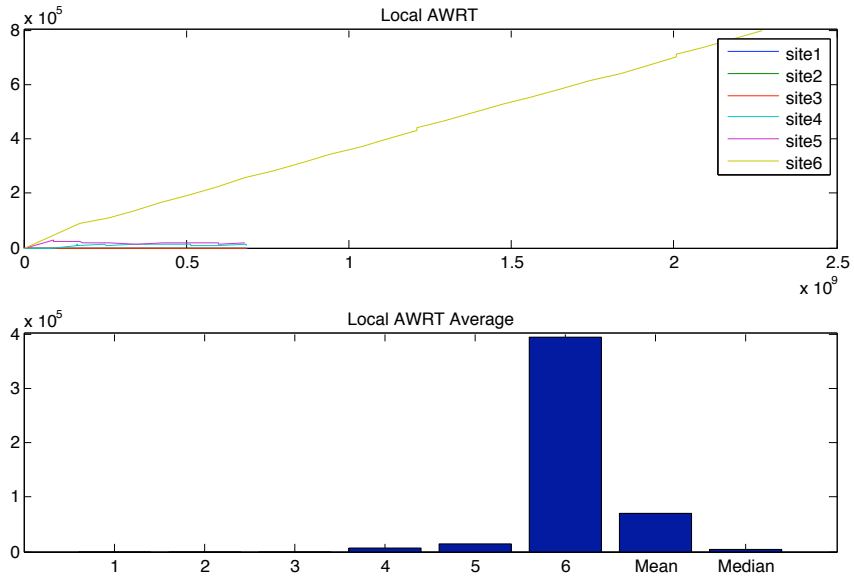

Figure 10.10.: The AWRT for the utilisation-optimised set-up according to Table 10.4 (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including mean value and median).

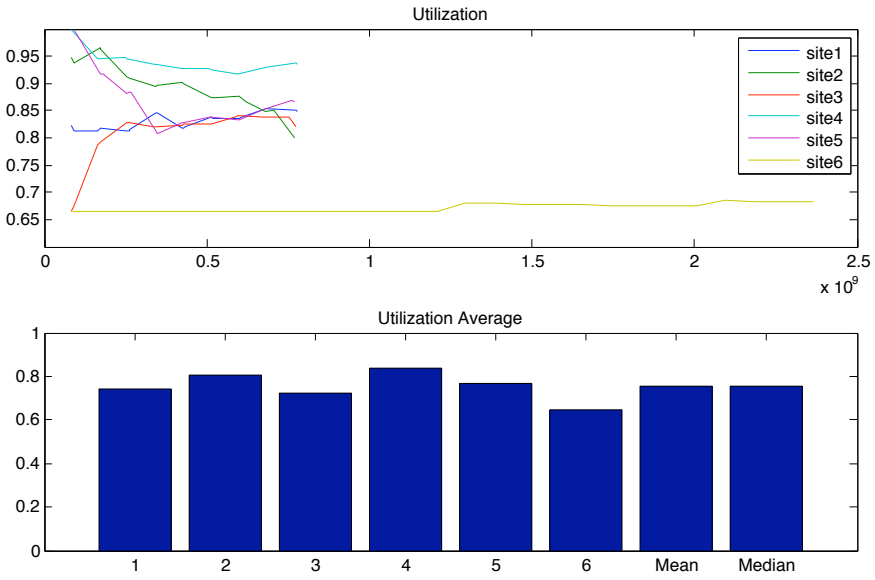

Figure 10.11.: The utilisation for the utilisation-optimised set-up according to Table 10.4 (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean value and median).

Metric	Value
AWRT	$0.93 \cdot 10^5$
SavedThroughDelegation	$1.373 \cdot 10^{10}$
NumberLocallyAvailableCPUs	51.84
Utilisation	0.701
WaitingActivities	41.9
WaitingTime	$4.31 \cdot 10^6$
QueuedCPUs	1420
NegotiationDuration	2.35
Outcome	0.93

Table 10.7.: Average values for key metrics of the best cost set-up

The AWRT for this set-up shows a slight improvement in comparison to reference run: $0.93 \cdot 10^5$ versus $1.36 \cdot 10^5$ milliseconds (see Fig. 10.12). Fig. 10.13, however, shows only a minimal improvement regarding mean utilisation compared to the reference run: some sites show small improvements whereas others experience a small decrease. AWRT optimisation and utilisation optimisation show significantly better values for this metric.

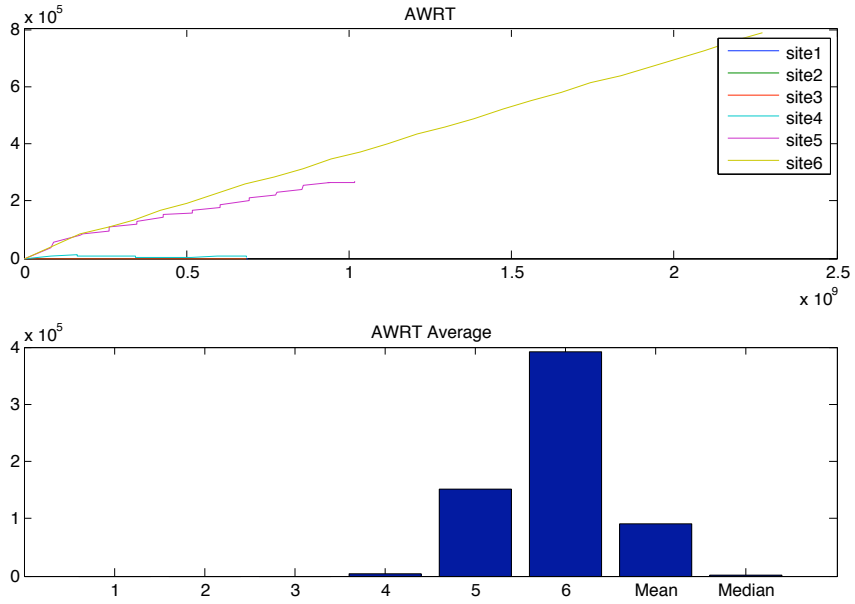

Figure 10.12.: The AWRT for the cost-optimised set-up according to Table 10.6 (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including mean value and median).

Fig. 10.13 shows a rapid drop in utilisation for site 3 around $1 \cdot 10^8$ milliseconds. This decrease in utilisation results in a price drop for the respective site (cf. the upper diagram of Fig. 10.14) as the current utilisation fell below the average utilisation. This effect is

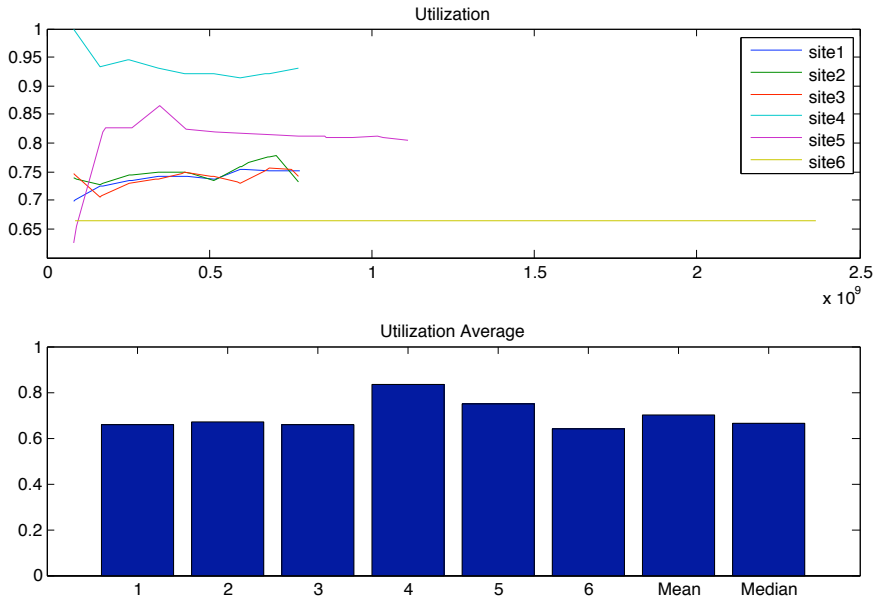

Figure 10.13.: The utilisation for the cost-optimised set-up according to Table 10.6 (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean value and median).

driven by the *global_util_price_policy* pricing policy and let, according to the simulation log files, to an increase of delegations of large activities from site 6 to site 3. The respective cost savings become evident in Fig. 10.14.

10.5.4. Discussion of Results

The results from the different set-ups show that an improvement of the global AWRT is possible, as all simulations result in better values compared to the reference scenario. Looking at the utilisation of the whole simulation environment, we experience advancements on a global scale (i.e. the mean utilisation), although always below 11% compared to the no-delegation scenario. However, on a per-site basis, no clear trend towards higher utilisation is visible as some sites show even worse utilisation values compared to the reference run. Especially the acceptance of large activities delegated from a remote site can lead to blocking behaviour, thus preventing smaller incoming activities from being executed. Here, another negotiation model than Accept/Reject as integrated into WS-Agreement is essential to a priori assess the impact of delegations for the local site. The cost saving due to the chosen pricing strategies leads to no benefit or even to a loss except for the cost optimisation case. Even then, the other key metrics are not satisfying. Further pricing policies and simulations are necessary to investigate this issue.

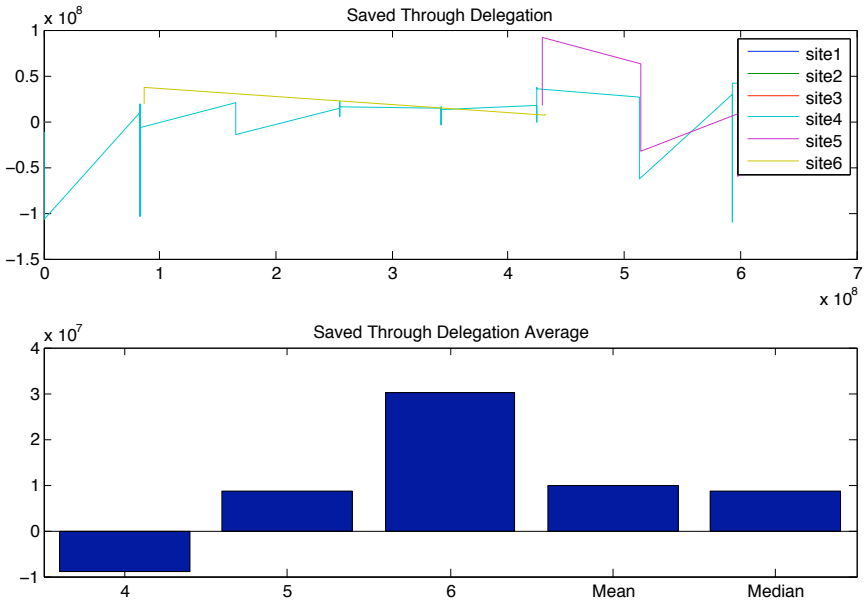

Figure 10.14.: The cost savings through delegation for the cost-optimised set-up according to Table 10.6 (The upper diagram shows the savings as a function of the simulation time (in milliseconds). The lower diagram shows the average savings per site including mean value and median)

10.6. Evaluation of Traces from Realistic Workloads

To be able to judge the conclusiveness of the results from the previous evaluations on realistic application scenarios, we compare them with realistic workloads and run the same tests for the realistic as we did for the artificial ones to find the optimal set-ups for the three core metrics.

10.6.1. Simulation Set-Up

The realistic traces for these evaluations have been taken from the workload archive[5]. They are formatted according to the *Standard Workload Format* (SWF). There are a number of noteworthy characteristics of the archive:

- The majority of traces is taken from systems commonly referred to as 'HPC systems'.
- Most traces are not annotated with information about activity types or user classes.
- The characteristics of the individual traces vary heavily in terms of the logged period, number of available CPUs, submission frequency, or mean/maximal activity size.
- Only a small absolute number of traces is available.[6]

[5] Logs of Real Parallel Workloads from Production Systems, last visited: January 25, 2013. http://www.cs.huji.ac.il/labs/parallel/workload/logs.html.

[6] The archive contains 30 traces as of May 3, 2012.

These facts makes reliable statements about the general applicability of the results of the simulation difficult. Nevertheless, we have chosen three traces with similar values regarding the maximal available number off CPU size and the average requested number of CPUs to have an environment suitable to measure the impact of our scheduling model on realistic workloads. Fig. 10.15 shows the characteristics of the three traces, each of which is assigned to one particular site. Sites 1 and 2 have similar maximal requested runtimes, whereas site 3 has a substantially larger value here. Regarding the average requested runtime, however, site 2 has the largest value. The total simulation time has been adopted to fit the shortest-running trace.

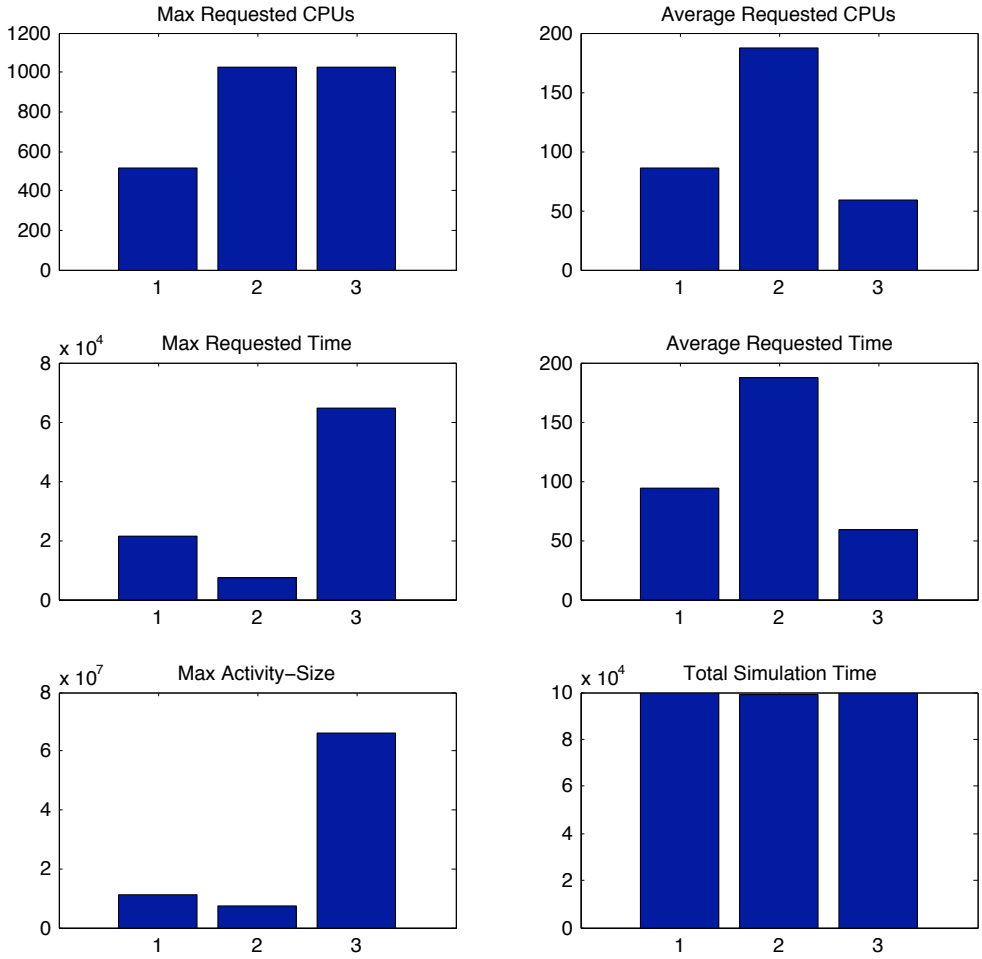

Figure 10.15.: Characteristics of the artificial traces (The abscissa shows the different sites, the ordinate the following values: number of requested CPUs (first row), requested runtime in seconds (second row), and the product of the number of requested CPUs and the requested runtime as well as the total simulation time units, both in seconds (third row)).

Site #	$\mathrm{CPU}_{\max}$	Price per CPUh
1	256	100
2	512	100
3	256	100

Table 10.8.: Site configurations for the realistic workload traces

Metric	Value
AWRT	$2.79 \cdot 10^4$
SavedThroughDelegation	N/A
NumberLocallyAvailableCPUs	66
Utilisation	0.752
WaitingActivities	67.6
WaitingTime	$1.49 \cdot 10^6$
QueuedCPUs	$4.29 \cdot 10^5$
NegotiationDuration	N/A
Outcome	N/A

Table 10.9.: Average values for the key metrics of the reference run

In a first attempt, the three sites have been configured offering the maximum number of CPUs corresponding to the characteristics of the traces. This resulted in mean utilisations which did not make delegation necessary. As this set-up is not worthwhile with our application scenario in mind, the sites have been configured according to Table 10.8, resulting in overloaded and under-utilised sites. Activities with an amount of requested CPUs larger than the respective site could handle have therefore been dismissed.

10.6.2. Reference Run

The set-up for the reference run with SWF traces is the same as for the artificial simulations without delegation (see Section 10.5.2). Table 10.9 lists the achieved average values for all key metrics. Again, no values are available for the duration of negotiations, outcome, and cost savings since the *no_del* strategy is applied.

Fig. 10.16 shows a continuously rising AWRT for all three site, however, the incline is less steep for sites 1 and 2. The simulation ends for these sites at approximately $1.5 \cdot 10^8$ milliseconds (the last activities for all sites have been submitted before $1 \cdot 10^8$ milliseconds), whereas site 3 needs a longer period to execute all accumulated activities.

Fig. 10.17 depicts that site 3 has the lowest utilisation of all sites despite its large (and steadily increasing) mean AWRT and the growing amount of queued activities. This verifies that site 3 is often blocked by queued activities which are larger than the remaining number of available CPUs, a situation which is not resolved due to the local application of the first-come-first-serve scheduling strategy. Site 1, however, has an average utilisation just below 80% and site 1 has the highest average utilisation greater than 95%. Assessing the diagrams it is expected that site 2 will benefit from the usage of delegation strategies and that the utilisation will improve.

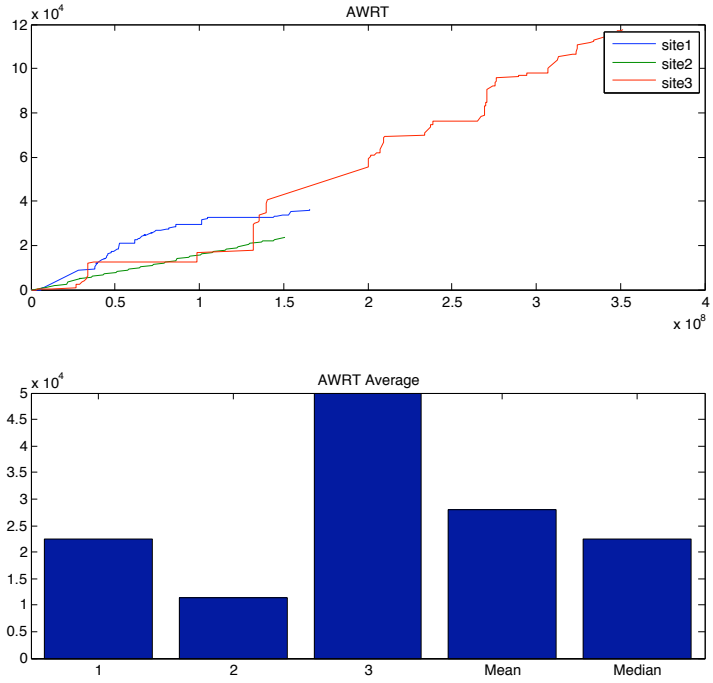

Figure 10.16.: The AWRT for the No Delegation Strategy (*no_del*) reference run (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including the mean value and the median).

The diagrams in Fig. 10.18 show, for the sake of completeness, the cost for the three sites. Since all three sites have the same price per CPUh, site 3 has, owing to the trace characteristics, the largest costs per activity.

10.6.3. Optimal Set-ups

The simulations with realistic traces have been conducted analogically to the artificial cases to determine the optimal set-up for the scheduling model. The results are described and evaluated in the following sections.

Best Set-up for AWRT Optimisation

The set-up with the best AWRT is listed in Table 10.10. It is worth mentioning that it is nearly identical to the cost optimisation set-up for the artificial runs (cf. Table 10.6). The only difference is the *static_price_policy* used in this set-up.

The results for the simulations with this configuration are listed in Table 10.11, providing the same key metrics used in the previous evaluations. A clear benefit of this set-up is the significant improvement of the mean AWRT from $2.79 \cdot 10^4$ milliseconds in case of the reference run to $0.653 \cdot 10^4$ milliseconds. As a downside, we see a decrease of the mean utilisation below the value of the reference simulation.

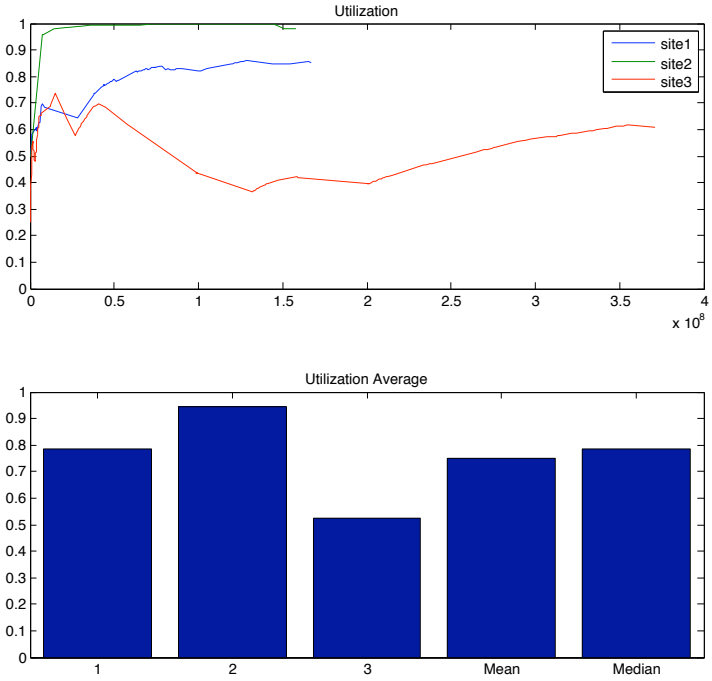

Figure 10.17.: The utilisation for the no-delegation reference run (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean and median).

Strategy	Instance
Delegation Strategy	*cpu_thresh_del*
Pre-selection Strategy	*wave_pre-sel*
Scheduling Strategy	*low-cost_low-cpu_first_sched*
Sched. Select. Strategy	*wave_sched-sel*
Negotiation Strategy	*wsag_basic_policy*
CPU Time Policy	*static_cpu-time_policy*
Pricing Policy	*static_price_policy*

Table 10.10.: The best set-up for AWRT optimisation

Fig. 10.19 and 10.20 depict the diagrams for AWRT and utilisation, respectively. The improvement regarding the AWRT is weighty for site 1 and especially for site 3, whereas the value for site 2 remains on a similar level compared to the reference simulation. The mean utilisation, in contrast, decreases, although further potential to optimise the delegation strategy can be deduced from the fact that site 1 finishes its activity processing considerably earlier than the other two sites.

The cost metric is not depicted, as no savings can be achieved due to the application of the static pricing strategy.

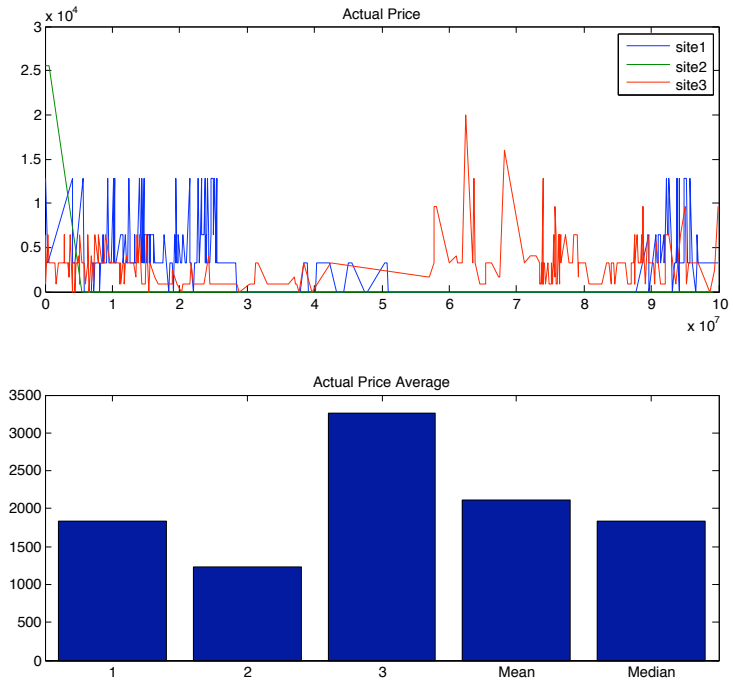

Figure 10.18.: The cost for the no-delegation reference run

Best Set-up for Utilisation Optimisation

Table 10.12 characterises the best utilisation set-up by its strategies. The mean values resulting from the simulation are given in Table 10.13. This set-up realises the best value for the mean utilisation and, at the same time, delivers the same mean AWRT as the previous, AWRT-optimised, set-up. The gain with respect to utilisation is, compared to the reference run, a marginal 2,8%, a result which shows potential for improvement. Further studies based on the results reported here may reveal under which conditions this is possible.

Fig. 10.21 exposes the same characteristic as the corresponding diagram for AWRT-optimising set-up (cf. Fig. 10.19) mainly due to the usage of the same pricing policy and the same delegation strategy (*static_price_policy* and *cpu_thresh_del*, respectively). Fig. 10.22 shows no significant increase in utilisation for site 1 and site 3 in contrast to site 2, which has a mean utilisation of nearly 95%. Again, the same effect as for the AWRT case can be observed revealing that more delegations to site 1 could improve global utilisation.

Best Set-up for Cost Optimisation

The set-up with the greatest average cost savings is the same as the best AWRT-optimising set-up (cf. Table 10.10). The only difference is the use of the utilisation-based pricing policy (*util_price_policy*), which does not affect the delegation behaviour[7]. Hence, the diagrams for AWRT and utilisation correspond to those in Fig. 10.19 and Fig. 10.20. The

[7]The pricing policy only affects delegation decisions if combined with a schedule selection strategy that rewards cheaper prices. For the current set-ups, this case is not considered.

Metric	Value
AWRT	$0.653 \cdot 10^4$
SavedThroughDelegation	0
NumberLocallyAvailableCPUs	117.9
Utilisation	0.68
WaitingActivities	5.78
WaitingTime	$1.00 \cdot 10^6$
QueuedCPUs	$1.86 \cdot 10^5$
NegotiationDuration	1.49
Outcome	0.927

Table 10.11.: Average values for key metrics of the best AWRT set-up

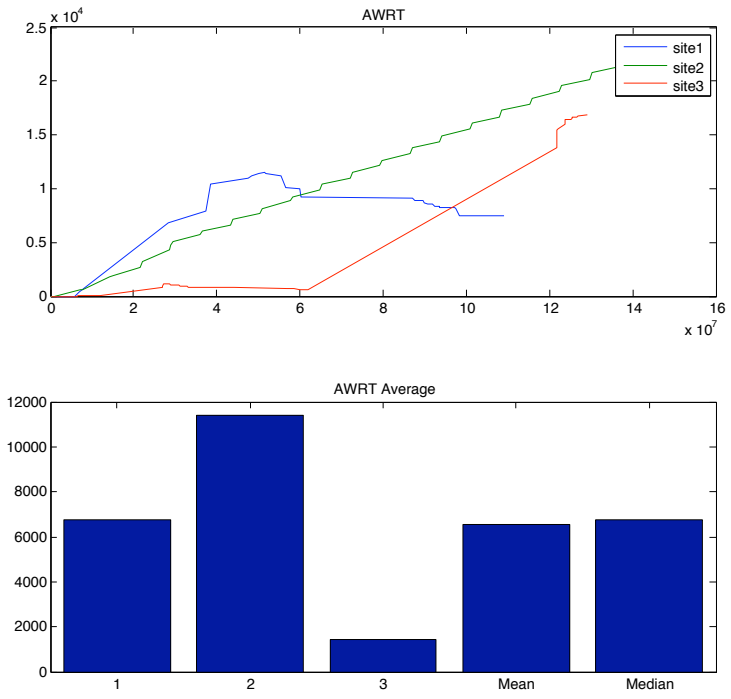

Figure 10.19.: The AWRT for the AWRT-optimised set-up according to Table 10.10 (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including mean value and median).

global average savings are $4.02 \cdot 10^6$ as shown in Fig. 10.23.

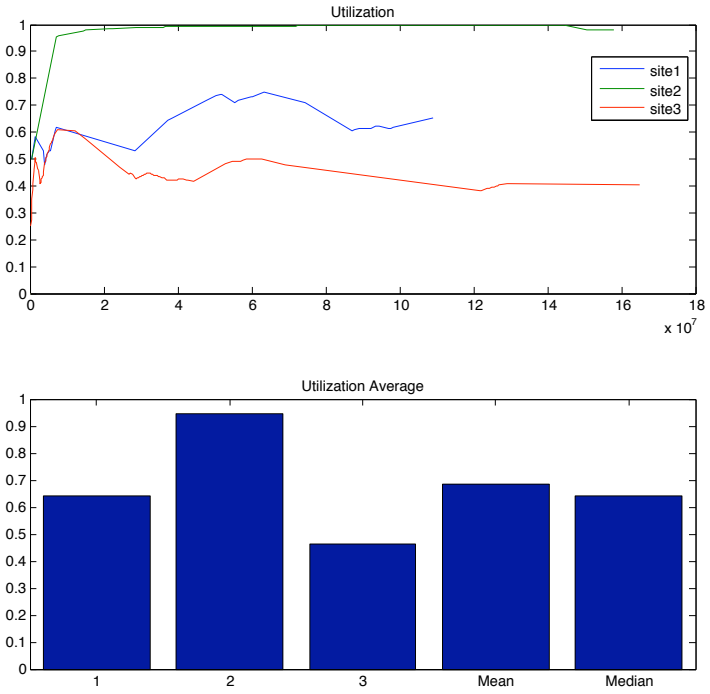

Figure 10.20.: The utilisation for the AWRT-optimised set-up according to Table 10.10 (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean value and median).

Strategy	Instance
Delegation Strategy	*cpu_thresh_del*
Pre-selection Strategy	*wave_pre-sel*
Scheduling Strategy	*low-cost_low-rt_first_sched*
Sched. Select. Strategy	*wave_sched-sel*
Negotiation Strategy	*wsag_basic_policy*
CPU Time Policy	*static_cpu-time_policy*
Pricing Policy	*static_price_policy*

Table 10.12.: The best set-up for utilization optimisation

10.6.4. Discussion of Results

The simulations of realistic traces revealed that an improvement regarding utilisation and AWRT is achievable through the application of our scheduling model. Furthermore, we see improvements regarding cost optimisation. It therefore should be possible to enhance a scheduler, implemented according to our scheduling model, with pricing policies to approximate or even calculate when to accept activities and when to delegate them to maximise profit.

It has to be noted, though, that SWF traces of suitable size have been selected and the maximal available number of CPUs has been changed compared to the original machine to

Metric	Value
AWRT	$0.653 \cdot 10^4$
SavedThroughDelegation	0
NumberLocallyAvailableCPUs	56.3
Utilisation	0.78
WaitingActivities	5.78
WaitingTime	$1.00 \cdot 10^6$
QueuedCPUs	$1.86 \cdot 10^5$
NegotiationDuration	1.52
Outcome	0.93

Table 10.13.: Average values for key metrics of the best utilisation set-up

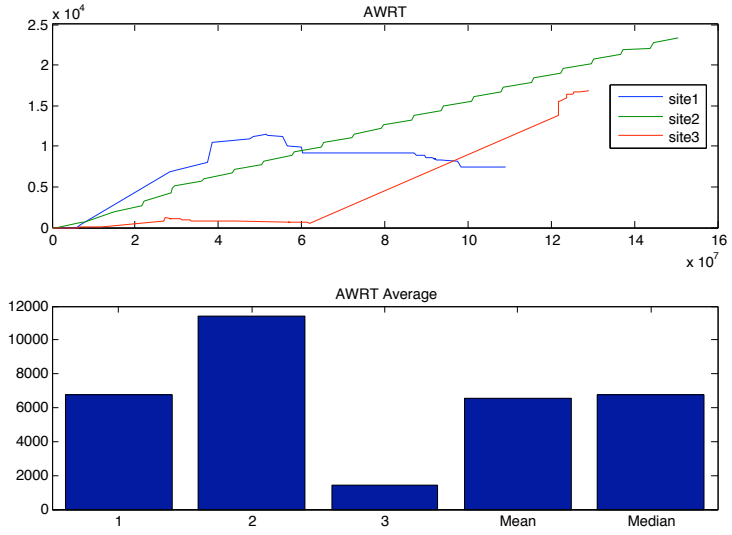

Figure 10.21.: The AWRT for the utilisation-optimised set-up according to Table 10.12 (The upper diagram shows the AWRT in milliseconds as a function of the simulation time (in milliseconds). The lower diagram shows the average AWRT per site (in milliseconds) including mean value and median).

artificially create a situation of resource scarcity. Furthermore, the effect of our scheduling model was only simulated with three distinct SWF traces. These pre-requisites restrict the general applicability of the findings and call for addition evaluations.

Despite these restrictions, it has been demonstrated that the design of our scheduling model is applicable and that it is possible to improve certain metrics in simulations with artificial and realistic workloads. Especially the impact of delegation and improvements compared to the reference runs are demonstrated and show potential for applying the model to out-sourcing or overflow scenarios.

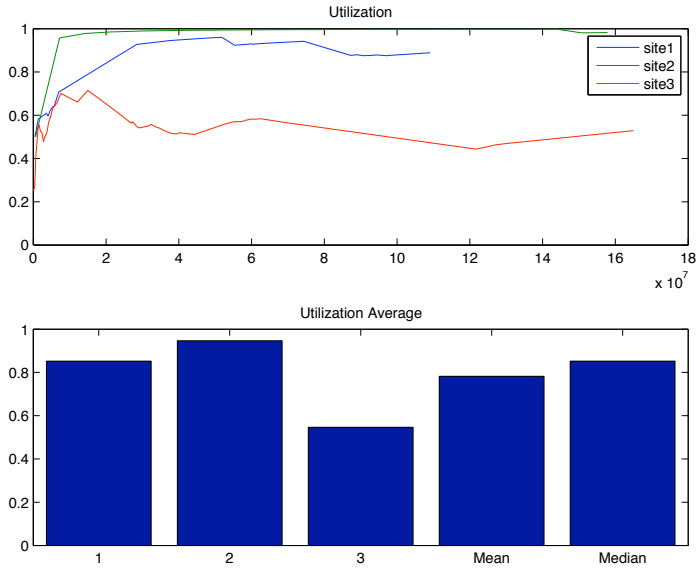

Figure 10.22.: The utilisation for the utilisation-optimised set-up according to Table 10.12 (The upper diagram shows the utilisation as a function of the simulation time (in milliseconds). The lower diagram shows the average utilisation per site including mean value and median).

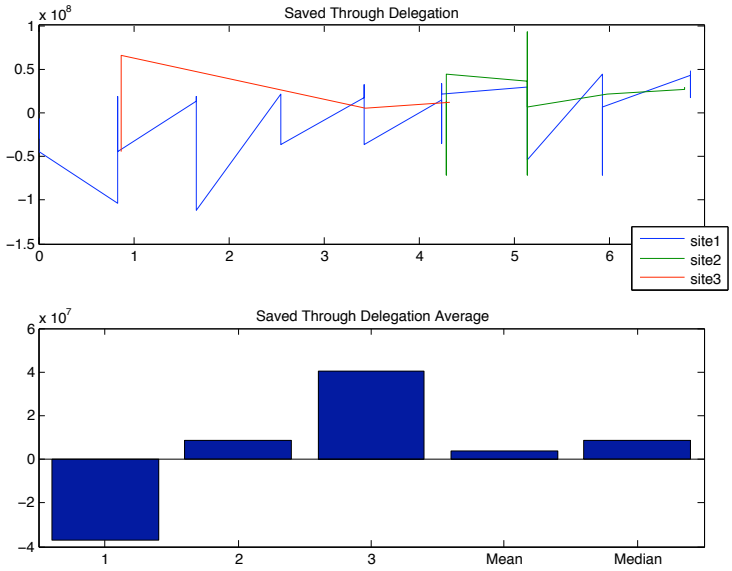

Figure 10.23.: The cost savings through delegation for the cost-optimised set-up according to Table 10.10 (The upper diagram shows the savings as a function of the simulation time (in milliseconds). The lower diagram shows the average savings per site including mean value and median).

In this chapter, we demonstrated the applicability of our scheduling model by implementing a simulation framework that integrates the model with the one of our application scenarios. Focussing on the impact of different strategies and policies on the computed schedule, we evaluated artificial and SWF workload traces. These evaluations revealed that the scheduling model is, in general, applicable to improve AWRT, utilisation, and cost. The general applicability of the model, though, has to be further tested to deduce best practices for production use.

Part VI.

Discussion

11. Conclusion

Looking at existing academic efforts in the area of scheduling in distributed computing infrastructures, we see two major strands: (i) research on algorithms and (ii) development of schedulers for specific infrastructures. As of today, with the increase of service-oriented solutions, the growing variety of application scenarios, the evolution of production grids, and the convergence of mid-tier HPC infrastructures into cloud-based DCIs, a complementary aspect has to be researched: How can we provide generic scheduling solutions for service-oriented DCIs?

Although this question subsumes our motivation in a bold and simple way, it captures the two main aspects of our work, (i) the provision of a generic scheduling architecture, process, and model and (ii) the integration of the aforementioned assets with the IT service management of state-of-the-art DCIs. In this chapter, we explore how far we have gotten realising these aspects by summarising our results, by discussing the implications of actually integrating our approach into a DCI, and by presenting perspectives for future research.

11.1. Results

The results of our work can be grouped into four categories: (i) generic concept, (ii) core models, (iii) implementations, and (iv) concept verification. The first category contains the generic scheduling architecture and the scheduling process as introduced in Chapter 4. These conceptual entities describe the services and interactions necessary to realise scheduling in state-of-the-art distributed computing infrastructures. To achieve that, however, it is necessary to implement our four core models, namely the activity, information, service-level agreement, and the scheduling model. These entities are described in Chapters 5 to 8. All four models have been fully implemented within the European projects NextGRID and IANOS (except for the scheduling model, which was lacking delegation). Furthermore, some models and implementations are sustainably maintained by the community as standards or products, as described in Chapter 9. The fourth result is the verification of the concept. We have chosen the ‘delegation of scheduling requests’ application scenario (cf. Section 3.1.3) and evaluated the effects of applying our scheduling model to it. The respective simulations demonstrated the feasibility of our approach and, at the same time, revealed potential for improvements.

One asset that was not on the roadmap is the full integration of all concepts and models into one implementation. The implication of not having a complete implementation is that it was not possible to verify the integrated approach fully, but just certain parts separately or in combination. The rationale behind this decision is given in the following section.

11.2. Implications of Integrating the Proposed Concept into State-of-the-Art DCIs

The introduction of electronic contracts to scheduling in distributed computing infrastructures is a complex undertaking, especially if the target is a generic solution that fulfils such a broad range of requirements as those gathered in Section 3. The development of the IANOS scheduler, for example, which also implements WS-Agreement and parts of the scheduling model, was a multi-year venture.

We assess the complete realisation of our concept as the implementation of a complete DCI middleware, an undertaking way beyond our work. To fulfil all requirements, it is essential that the fundamental models are considered at service design time. This is a prerequisite not given as far as state-of-the-art DCI middleware is concerned. And although some of the implementations of our models and concepts are mature and available as open source software (cf. Table 9.1), we would not advise to combine the various parts to form a DCI middleware. We rather see our work as a feasibility study for future scheduling solutions. Furthermore, it presents a modular approach applicable to a multitude of problems associated with building DCI solutions. Of course, it also serves as blueprint for a the core of a distributed computing infrastructure.

11.3. Outlook

We have thoroughly discussed the concept and models in the respective chapters and we think that the essential issues have been raised to enable readers to judge the feasibility of the contribution. But certainly issues and questions remain to complement what has been described here. We could discuss extensively further matter that was not in the scope of our work, but focus on two points that seem to be especially important: (i) the relation of our solution to IT service management in general and (ii) issues with information management.

11.3.1. IT Service Management

Although we do not propagate clouds as the ultimate solution, it seems that providers and users of distributed computing infrastructures will benefit from ubiquitous service-orientation. Although many DCIs are actually service-oriented infrastructures, the full potential is not yet exploited. Looking, for example, at the life-cycle of a service-level agreement (cf. Fig. 7.1), most DCIs support mainly the first three phases development, negotiation, implementation. The execution phase is also implemented, but most often detached from the service-level management with the activity description passed to middleware services for execution and monitoring.

The purpose of SLAs in our concept is to offer technical means for a coherent view on a DCI as a single system. SLAs are the instrument to codify the objectives and responsibilities of users and service providers and constitute, transparent for the user, the single entry point to the system. The users get their contracts and the service to be delivered, including orchestration of other services, is controlled by the DCI middleware. Whenever there is an agreement on a service-level (i.e. an electronic contract has been negotiated and agreed upon), the services necessary to fulfil the contract are provisioned. Prior to

the provisioning, an activity instance is created to capture all further information related to the contract.

Service-level agreements, however, are just one part of IT service management. One of the most comprehensive frameworks to be used for managing IT services, the *Information Technology Infrastructure Library* (ITIL) [135], consists of publications and specifications created by the UK's Office of Government Commerce (OGC). Initially, ITIL has been developed to reduce the UK government's growing cost for IT services. It defines a large variety of ITIL processes like capacity management, incident management, and service-level management, which, inter alia, deals with the application of SLAs.

Compared to the objectives fulfilled by our work, ITIL follows a much broader approach. Examples for additional objectives are [1]:

- 'Identification of process improvement opportunities [...]'.
- 'Improvements to system/service reliability and stability'.
- 'To generate operational data to be analysed as part of preventive action initiatives'.

Evidently, the fulfilment of such objectives is not solely achieved through SLAs, but involves various processes. Here, ITIL delivers the necessary best practices and recipes to integrate SLAs into IT service management solutions.

With our solution, which integrates service-level agreements as an intrinsic part of a scheduling concept, we provide the foundations for a pervasive IT service management of distributed computing infrastructures and pave the way towards automated service provisioning. It is, though, just an initial step. The ITIL-compliant service delivery is much more than the implementation of our solution. It involves things like service portfolio management, service validation management or release management, to name just a few. And it is not sufficient to create technical solutions, but mandates the realisation of the respective processes within the data center, including documentation, training, and knowledge management.

11.3.2. Information Management

We observe that in existing DCIs information about infrastructure, services, and activities is dispersed. Various data sources and consumers exist in such environments making it difficult to manage and process the respective information. We therefore introduced the activity model to have a central entity for managing activity-related information. As a result, we use the activity instance to capture the activity description, the SLA(s) related to it, data about resource and service consumption, and alike. Although we now have a model and services to centrally process and consume the bits of information needed for scheduling, there is still room for optimisation.

Here, we want to highlight especially two issues, i.e. the redundancy in modelling and keeping information and the difficulty of mapping between different models. In general, DCIs combine different languages to request activities, describe resource capabilities, search for services, formalise SLAs, or monitor their progress. Common examples are JSDL, GLUE, XQuery, WS-Agreement, or parameter–value pairs, respectively. Even though we introduced the activity model, we rely on a variety of existing standards like CIM and, as a result, have to deal with redundant information. Examples include the activity description, which is included in the SLA and the activity instance, and resource capabilities,

parts of which are kept in the registry and also the information service. Another stumbling block is the mapping between different information-carrying formats, which often offer different semantics and richness. As a result, performance-limiting technologies like XSLT have to be used to transform the different formats, an approach that might fail for certain concepts in case they are not supported by one of the languages.

Approaches to overcome such limitations are manifold, as e.g. requesting and searching for services using SLA templates that follow the format used by the service provider, or using the same language to describe resource capabilities and activity requirements. The obvious solution, the development of an integrated information model processable by all services within a DCI, is, although tempting, a less promising approach as many unsuccessful efforts show.

11.4. Future Perspective

We have experienced the difficult path from 'DCI silos' over 'Grid Services' towards virtualised service infrastructures, including various interoperability efforts and an effort called the Semantic Grid. Although this is what we call technological evolution, some of the endeavours that brought us here may have benefited from a broader view and from less 'not invented here' attitude.

For future developments, we consider it indispensable that DCIs are not only designed with a certain standard base set of functions in mind (as currently done by the European Middleware Initiative[1]), but also based on common models and comprehending an intrinsic service-level management concept. Following that approach, we will experience the convergence of academic and industrial efforts, something we already see in the cloud landscape. Especially the infrastructure-as-a-service management solutions, with CloudStack[2], Eucalyptus, OpenNebula, and OpenStack[3] as the most prominent representatives, are used by companies providing cloud services as well as academic data centres, which operate private or public clouds. Here, we will face a growing demand for elaborated service management solutions that are generic enough to be adapted to changing business objectives and technological choices.

With our generic scheduling architecture and the associated models and processes we provide one building block to satisfy this demand and to offer automated service-level management solutions for distributed computing infrastructures.

[1] European Middleware Initiative, last visited: January 25, 2013. http://www.eu-emi.eu/.

[2] CloudStack – Open Source Cloud Computing, last visited: January 25, 2013. http://www.cloudstack.org/.

[3] OpenStack – Open source software for building private and public clouds, last visited: January 25, 2013. http://openstack.org/.

About the Author

Philipp Wieder is working as a scientist at the Gesellschaft fuer wissenschaftliche Datenverarbeitung mbH Goettingen (GWDG). He graduated in 2000 from RWTH Aachen University with a diploma in electrical engineering. In his diploma thesis he already addressed the topic of scheduling and resource management targeting massively-parallel processing systems. After his studies, he entered the area of high performance computing and distributed systems, contributing intensively to research topics like MPP systems, grids, SOA, virtualization, and currently cloud computing. From 2000 until 2007 Philipp Wieder was affiliated with the Central Institute for Applied Mathematics (ZAM), now called Juelich Supercomputing Centre (JSC), of the Research Centre Juelich. From 2007 to 2011 he was with the Service Computing Group at the TU Dortmund University. At GWDG, he currently holds the position of leader of the eScience Group and deputy head of the data centre.

Bibliography

[1] R. Addy. *Service Level Management. Effective IT Service Management – To ITIL and Beyond!*, pages 275–296. Springer Berlin – Heidelberg, 1st edition, 2007.

[2] M. Ahronovitz, D. Amrhein, et al. Cloud Computing Use Cases White Paper. Technical report, The Cloud Computing Use Cases Group, 2010.

[3] D. Alger. *Grow a Greener Data Center*. Cisco Press, Indianapolis, 2010. ISBN: 978-1-58705-813-4.

[4] D. Alur, D. Malks, and J. Crupi. *Core J2EE Patterns: Best Practices and Design Strategies*. Prentice Hall, 2nd edition, 2003.

[5] S. Andreozzi, S. Burke, L. Fiel, G. Galang, B. Konya, M. Litmaath, P. Millar, and J. P. Navarro. GLUE Specification v. 2.0. Grid Forum Document GFD.147, The Open Grid Forum, Joliet, Illinois, United States, 2009.

[6] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Grid Forum Document GFD.107, The Open Grid Forum, Joliet, Illinois, United States, 2007.

[7] A. Anjomshoaa, F. Brisard, M. Drescher, D. Fellows, A. Ly, S. McGough, D. Pulsipher, and A. Savva. Job Submission Description Language (JSDL) Specification v1.0. Grid Forum Document GFD.56, The Open Grid Forum, Joliet, Illinois, United States, November 2005.

[8] G. Anthes. Security in the Cloud. *Communications of the ACM*, 53(11), Nov 2010.

[9] M. Antonioletti, M. Atkinson, R. Baxter, A. Borley, N. Chue Hong, et al. OGSA-DAI Status Report and Future Directions. In *Proc. of the UK All Hands Meeting 2004*. EPSRC, September 2004.

[10] P. Armstrong, A. Agarwal, A. Bishop, A. Charbonneau, R. J. Desmarais, K. Fransham, N. Hill, I. Gable, S. Gaudet, S. Goliath, R. Impey, C. Leavett-Brown, J. Ouellete, M. Paterson, C. Pritchet, D. Penfold-Brown, W. Podaima, D. Schade, and R. J. Sobie. Cloud scheduler: a resource manager for distributed compute clouds. *CoRR*, 2010. http://arxiv.org/abs/1007.0050.

[11] D. Battré, O. Kao, and K. Voss. Implementing WS-Agreement in a Globus Toolkit 4.0 Environment. In D. Talia, R. Yahyapour, and W. Ziegler, editors, *Grid Middleware and Services*, pages 409–418. Springer US, 2008.

[12] D. Battré, W. Ziegler, and Ph. Wieder. WS-Agreement Specification Version 1.0 Experience Document. Grid Forum Document GFD.167, The Open Grid Forum, Joliet, Illinois, United States, 2010.

[13] B. Bhargava. *Concurrency and Reliability in Distributed Database Systems*. Van Nostrand Reinhold, 1987.

[14] F. Black and M. Scholes. The Pricing of Options and Corporate Liabilities. *Journal of Political Economy*, 81(3):637–654, 1973. DOI: 10.1086/260062.

[15] M. B. Blake and D. J. Cummings. Workflow Composition of Service Level Agreements. In *Proc. of the IEEE International Conference on Services Computing (SCC 2007)*, pages 138–145, July 2007.

[16] A. Born. Get off of my Cloud – Software as a Service im Cloud Computing. *iX Spezial*, 2, 2010.

[17] D. Breuer, D. Erwin, D. Mallmann, R. Menday, M. Romberg, V. Sander, B. Schuller, and Ph. Wieder. Scientific Computing with UNICORE. In G. Münster D. Wolf and M. Kremer, editors, *Proc. of the NIC Symposium 2004*, volume 20 of *NIC Series*, pages 429–440. Forschungszentrum Juelich GmbH, 2004.

[18] P. Brucker. *Scheduling Algorithms*. Springer Berlin – Heidelberg, 2007. ISBN: 978-3-540-69515-8.

[19] A. Burrows. *A Casebook on Contract*. Hart Publishing, Oxford, UK, 2nd edition, 2009.

[20] R. Buyya, D. Abramson, J. Giddy, and H. Stockinger. Economics Paradigm for Resource Management and Scheduling in Grid Computing. In *Grid Computing, Concurrency and Computation: Practice and Experience Vol. 14, Grid Computing environments Special Issue 13-15*, pages 1507–1542, 2002.

[21] C. Catlett, C. de Laat, D. Martin, G. Newby, and D. Skow. Open Grid Forum Document Process and Requirements. Grid Forum Document GFD.152, The Open Grid Forum, Joliet, Illinois, United States, 2009.

[22] FIPA Technical Committee Communication. FIPA Contract Net Interaction Protocol Specification. FIPA Specification SC00029H, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[23] FIPA Technical Committee Communication. FIPA Iterated Contract Net Interaction Protocol Specification. FIPA Specification SC00030H, Foundation for Intelligent Physical Agents, Geneva, Switzerland, 2002.

[24] J.P. Conti. The Internet of things. *Communications Engineer*, 4(6):20 –25, 2006.

[25] A. W. Cooke, A. J. G. Gray, W. Nutt, J. Magowan, M. Oevers, P. Taylor, R. Cordenonsi, R. Byrom, L. Cornwall, A. Djaoui, L. Field, S. Fisher, S. Hicks, J. Leake, R. Middleton, A. J. Wilson, X. Zhu, N. Podhorszki, B. A. Coghlan, S. Kenny, D. O'Callaghan, and J. Ryan. The Relational Grid Monitoring Architecture: Mediating Information about the Grid. *Journal of Grid Computing*, 2(4):323–339, 2004. http://dx.doi.org/10.1007/s10723-005-0151-6.

[26] K. Cristiano, R. Gruber, V. Keller, P. Kuonen, S. Maffioletti, N. Nellari, M.-C. Sawley, M. Spada, T.-M. Tran, O. Wäldrich, Ph. Wieder, and W. Ziegler. Integration of ISS

into the VIOLA Meta-scheduling Environment. In S. Gorlatch and M. Danelutto, editors, *Proc. of the Integrated Research in Grid Computing Workshop*, pages 357–366. Università di Pisa, 2005.

[27] M. Crovella, P. Das, C. Dubnicki, T. LeBlanc, and E. Markatos. Multiprogramming on multiprocessors. In *Proc, of the Third IEEE Symposium on Parallel and Distributed Processing*, pages 590–597, 1991. DOI: 10.1109/SPDP.1991.218246.

[28] K. Czajkowski, I. Foster, N. Karonis, C. Kesselman, S. Martin, W. Smith, and S. Tuecke. A Resource Management Architecture for Metacomputing Systems. In *Proc. of the IPPS/SPDP '98 Workshop on Job Scheduling Strategies for Parallel Processing*, 1998.

[29] K. Czajkowski, I. Foster, C. Kesselman, V. Sander, and S. Tuecke. SNAP: A Protocol for Negotiating Service Level Agreements and Coordinating Resource Management in Distributed Systems. In *Proc. of the 8th Workshop on Job Scheduling Strategies for Parallel Processing*, pages 153–183, 2002.

[30] F. A. B. da Silva and I. D. Scherson. Improving Parallel Job Scheduling Using Runtime Measurements. In *Proc. of the Workshop on Job Scheduling Strategies for Parallel Processing (JSSPP '00/IPDPS '00)*, pages 18–38. Springer Verlag, 2000. ISBN: 3-540-41120-8.

[31] H. Dail and F. Desprez. Adaptive window scheduling for a hierarchical agent system. In *Proc. of the 4th International Symposium on Parallel and Distributed Computing (ISPDC 2005)*, pages 58–65, 2005. DOI: 10.1109/ISPDC.2005.12.

[32] Y. Demchenko, C. de Laat, O. Koeroo, and D. Groep. Re-thinking Grid Security Architecture. *Proc. of the IEEE Fourth International Conference on eScience (eScience '08)*, pages 79–86, 2008.

[33] F. Desprez and A. Vernois. Simultaneous Scheduling of Replication and Computation for Data-Intensive Applications on the Grid. *Journal of Grid Computing*, 4:19–31, 2006. DOI: 10.1007/s10723-005-9016-2.

[34] T. Dimitrakos, J. Martrat, and S. Wesner, editors. *Service Oriented Infrastructures and Cloud Service Platforms for the Enterprise – A selection of common capabilities validated in real-life business trials by the BEinGRID consortium*. Springer, 2010. ISBN: 978-3-642-04085-6.

[35] K. Djemame, J. Padgett, I. Gourlay, K. Voss, D. Battré, and O. Kao. Economically Enhanced Risk-aware Grid SLA Management. In *Proc. of the eChallenges Conference 2008 (e-2008)*, 2008.

[36] DMTF. CIM Concepts White Paper CIM Versions 2.4+. Technical Report DSP0110, Distributed Management Task Force, June 2003.

[37] DMTF. CIM Device Model White Paper CIM Version 2.7. Technical Report DSP0144, Distributed Management Task Force, June 2003.

[38] DMTF. CIM System Model White Paper CIM Version 2.7. Technical Report DSP0150, Distributed Management Task Force, 2003.

[39] DMTF. Understanding the Application Management Model CIM Version 2.7. Technical Report DSP0140, Distributed Management Task Force, 2003.

[40] F. Dong and S. G. Akl. Scheduling algorithms for grid computing: State of the art and open problems. Technical Report 2006-504, Queen's University, Kingston, Ontario, 2006.

[41] C. Dumitrescu and I. Foster. Usage policy-based CPU sharing in virtual organizations. In *Proc. of the Fifth IEEE/ACM International Workshop on Grid Computing*, pages 53–60, November 2004. DOI: 10.1109/GRID.2004.62.

[42] T. Eickermann, W. Frings, O. Wäldrich, Ph. Wieder, and W. Ziegler. Co-allocation of MPI Jobs with the VIOLA Grid MetaScheduling Framework. In *Proc. of the German e-Science Conference 2007*, Baden-Baden, Germany, 2007. Max Planck Digital Library. ID: 316560.0, Max Planck Digital Library / German e-Science Conference.

[43] T. Eickermann, L. Westphal, O. Wäldrich, W. Ziegler, C. Barz, and M. Pilz. Co-Allocating Compute and Network Resources. In T. Priol and M. Vanneschi, editors, *Towards Next Generation Grids*, pages 193–202. Springer US, 2007.

[44] G. Erbacci, C. Cavazzoni, F. Spiga, and I. Christadler. Report on petascale software libraries and programming models. Technical report, PRACE – Partnership for Advanced Computing in Europe, 2009.

[45] Carsten Ernemann, Volker Hamscher, Achim Streit, and Ramin Yahyapour. Enhanced algorithms for multi-site scheduling. In *Proceedings of the Third International Workshop on Grid Computing*, GRID '02, pages 219–231, London, UK, UK, 2002. Springer-Verlag.

[46] The TeleManagement Forum. *SLA Management Handbook, Volume 2, Concepts and Principles, Release 2.5*. The TeleManagement Forum, Morristown, New Jersey, United States, 2005.

[47] I. Foster. What is the Grid? A Three Point Checklist. Technical report, Argonne National Laboratory, 2002.

[48] I. Foster. Globus Toolkit Version 4: Software for Service-Oriented Systems. In *Proc. of the IFIP International Conference on Network and Parallel Computing*, volume 3779 of *Lecture Notes in Computer Science*, pages 2–13, 2006.

[49] I. Foster., A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service Version 1.0. Grid Forum Document GFD.108, Open Grid Forum, Joliet, Illinois, U.S.A., August 2007.

[50] I. Foster and C. Kesselman, editors. *The GRID: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 1998.

[51] I. Foster, C. Kesselman, and S. Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Journal of Supercomputer Applications*, 15, 2001.

[52] I. Foster, H. Kishimoto, A. Savva, D. Berry, A. Grimshaw, B. Horn, F. Maciel, F. Siebenlist, R. Subramaniam, J. Treadwell, and J. Von Reich. The Open Grid Services Architecture, Version 1.5 . Grid Forum Document GFD.80, The Open Grid Forum, Joliet, Illinois, United States, 2006.

[53] S. Freitag and Ph. Wieder. *Guide to e-Science: Next Generation Scientific Research and Discovery*, chapter The German Grid Initiative D-Grid – Status Quo and Future Perspectives. Springer, 2010.

[54] E. Gamma, R. Helm, R. Johnson, and J. Vlissdes. *Design Patterns: Elements of Reusable Object Oriented Software*, pages 139–148. Addison Wesley, 2005.

[55] F. Gens and M. Shirer. Through 2014 Public IT Cloud Services Will Grow at More Than Five Times the Rate of Traditional IT Products, New IDC Research Finds. Technical report, International Data Corporation, 2010. Last visited: May 01, 2011: http://www.idc.com/about/viewpressrelease.jsp?containerId=prUS22393210.

[56] S. Gilbertson. Lessons From a Cloud Failure: It's Not Amazon, It's You. WIRED, April 2011. URL: http://www.wired.com/epicenter/2011/04/lessons-amazon-cloud-failure/.

[57] C. Grimme, J. Lepping, and A. Papaspyrou. Prospects of Collaboration between Compute Providers by means of Job Interchange. In *Proc. of the 13th Job Scheduling Strategies for Parallel Processing*, volume 4942 of *Lecture Notes in Computer Science*, pages 132–151. Springer, 2007.

[58] C. Grimme, J. Lepping, A. Papaspyrou, Ph. Wieder, R. Yahyapour, A. Oleksiak, O. Wäldrich, and W. Ziegler. Towards a standards-based Grid Scheduling Architecture. In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Grid Computing: Achievements and Prospects*, pages 147–158. Springer Verlag, April 2008. ISBN: 978-0-387-09456-4.

[59] R. Gruber and V. Keller. *HPC@Green IT – Green High Performance Computing Methods*. Springer, 2010. ISBN: 978-3-642-01788-9.

[60] R. Gruber, V. Keller, P. Kuonen, M.-C. Sawley, B. Schaeli, A. Tolou, M. Torruella, and T.-M. Tran. Towards an Intelligent Grid Scheduling System. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Parallel Processing and Applied Mathematics*, volume 3911 of *Lecture Notes in Computer Science*, pages 751–757. Springer Berlin – Heidelberg, 2006.

[61] M. H. Haji, I. Gourlay, K. Djemame, and P. M. Dew. A SNAP-Based Community Resource Broker using a Three-Phase Commit Protocol: A Performance Study. *Computer Journal*, 48(3):333–346, 2005.

[62] V. Hamscher, U. Schwiegelshohn, A. Streit, and R. Yahyapour. Evaluation of job-scheduling strategies for grid computing. *Lecture Notes in Computer Science*, 1971:191–202, 2000.

[63] P. Hasselmeyer. On Service Discovery Process Types. In *Proc. of the 3rd International Conference On Service Oriented Computing (ICSOC '05)*, volume 3826 of *LNCS*, pages 144–157, Amsterdam, The Netherlands, December 2005.

[64] P. Hasselmeyer, H. Mersch, H.-N. Quyen, B. Koller, L. Schubert, and Ph. Wieder. Implementing an SLA Negotiation Framework. In P. Cunningham and M. Cunningham, editors, *Proceedings of the eChallenges Conference (e-2007)*, pages 154–161. IOS Press, October 2007. ISBN: 978-1-58603-801-4.

[65] P. Hasselmeyer, Ph. Wieder, and B. Koller. *Handbook of Research on Non-Functional Properties for Service-oriented Systems: Future Directions*, chapter Negotiation of Service Level Agreements, pages 442–469. IGI Global, 2011. DOI: 10.4018/978-1-61350-432-1.

[66] M. Henning. The rise and fall of CORBA. *Communications of the ACM*, 51(8):52–57, 2008.

[67] S. Hudert, H. Ludwig, and G. Wirtz. A Negotiation Protocol Framework for WS-Agreement. In *15. ITG/GI-Fachtagung Kommunikation in Verteilten Systemen (KiVS 2007)*. VDE Verlag, 2007.

[68] P. C. K. Hung, H. Li, and J.-J. Jeng. WS-Negotiation: An Overview of Research Issues. In *Hawaii International Conference on System Sciences*, page 10033b. IEEE Computer Society, 2004.

[69] E. Imamagic, B. Radic, and D. Dobrenic. CRO-GRID Grid Monitoring Architecture. In *Proc. of the 27th International Conference on Information Technology Interfaces*, pages 65–72, 2005.

[70] A. Iosup, C. Dumitrescu, D. Epema, H. Li, and L. Wolters. How Are Real Grids Used? The Analysis of Four Grid Traces and its Implications. In *Proc. of the 7th IEEE/ACM International Conference on Grid Computing (Grid2006)*, pages 262–269, 2006. DOI: 10.1109/ICGRID.2006.311024.

[71] J. D. G. Coulouris and T. Kindberg. *Distributed Systems, Concepts and Design, Third Edition*. Addison Wesley, 2001.

[72] M. Jowkar, C. Cavazzoni, G. Goumas, and X. Guo. Report on Approaches to Petascaling. Technical Report D6.4, PRACE – Partnership for Advanced Computing in Europe, 2009.

[73] K.-C. Huang. On Effects of Resource Fragmentation on Job Scheduling Performance in Computing Grids. *International Symposium on Parallel Architectures, Algorithms, and Networks*, 0:701–705, 2009. ISBN: 978-0-7695-3908-9.

[74] P. Karaenke and S. Kirn. A Multi-tier Negotiation Protocol for Logistics Service Chains. In *Proc. of the 18th European Conference on Information Systems (ECIS 2010)*, 2010.

[75] KevenT. Kearney and Francesco Torelli. The sla model. In Philipp Wieder, Joe M. Butler, Wolfgang Theilmann, and Ramin Yahyapour, editors, *Service Level Agreements for Cloud Computing*, pages 43–67. Springer New York, 2011.

[76] K.T. Kearney, F. Torelli, and C. Kotsokalis. SLA*: An abstract syntax for Service Level Agreements. In *Proc. of the 11th IEEE/ACM International Conference on Grid Computing (GRID)*, pages 217–224, 2010.

[77] A. Keller and H. Ludwig. The WSLA Framework: Specifying and Monitoring Service Level Agreements for Web Services. *Journal of Network and Systems Management*, 11(1):57–81, 2003.

[78] V. Keller, H. Rasheed, O. Wäldrich, W. Ziegler, R. Gruber, M.-C. Sawley, and Ph. Wieder. Models and internals of the IANOS resource broker. *Computer Science - Research and Development*, 23(3):259–266, 2009.

[79] W. H. Kohler. A Survey of Techniques for Synchronization and Recovery in Decentralized Computer Systems. *ACM Comput. Surv.*, 13(2):149–183, 1981.

[80] S. Krakowiak. What is Middleware. ObjectWeb – Open Source Middleware, 2003. Last visited: February 18, 2011: http://middleware.objectweb.org/.

[81] J. Krallmann, U. Schwiegelshohn, and R. Yahyapour. On the design and evaluation of job scheduling algorithms. In *Proc. of the IEEE Workshop on Job Scheduling Strategies for Parallel Processing*, pages 17–30, April 1999.

[82] D. D. Lamanna, J. Skene, and W. Emmerich. SLAng: A Language for Defining Service Level Agreements. In *Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems (FTDCS 2003)*, pages 100–106. IEEE Computer Society Press: United States, May 2003.

[83] L. Lamers, J. Piazza, A. Maier, G. Ericson, J. Davis, and K. Schopmeyer. Common Information Model (CIM) Infrastructure. Technical Report DSP0004, Version 2.6.0, Distributed Management Task Force, 2010.

[84] J. J. Lee and R. Ben-Natan. *What are Service Level Agreements? Integrating Service Level Agreements Optimizing Your OSS for SLA Delivery*, pages 3–25. Wiley Publishing, Inc., Indianapolis, Indiana, United States, 1st edition, 2002.

[85] E. Lie and J. J McConnell. Earnings signals in fixed-price and Dutch auction self-tender offers. *Journal of Financial Economics*, 49(2), 1998.

[86] Y. Lin, B. Kemme, M. Patino-Martínez, and R. Jiménez-Peris. Middleware based data replication providing snapshot isolation. In *Proc. of the 2005 ACM SIGMOD international conference on Management of data (SIGMOD '05)*, pages 419–x430. ACM Press, 2005.

[87] A. Litke, K. Konstanteli, V. Andronikou, S. Chatzis, and T. Varvarigou. Execution Management and SLA Enforcement in Akogrimo. In M. Bubak, M. Tural, and K. Wiatr, editors, *Proc. of the Cracow Grid Workshop 2006 (CGW'06)*, pages 154–165. Academic Computer Centre CYFRONET AGH, 2006. ISBN: ISBN 83-915141-7-X.

[88] H. Ludwig, T. Nakata, O. Wäldrich, Ph. Wieder, and W. Ziegler. Reliable Orchestration of Resources using WS-Agreement. In *Proc. of the 2006 International Conference on High Performance Computing and Communications (HPCC06)*, volume 4208 of *LNCS*, pages 753–762. Springer, 2006.

[89] Q. H. Mahmoud. Service-Oriented Architecture (SOA) and Web Services: The Road to Enterprise Application Integration (EAI). Technical report, Sun Developer Network, April 2005. Last visited May 03, 2012: http://www.oracle.com/technetwork/articles/javase/soa-142870.html.

[90] M. L. Massie, B. N. Chun, and D. E. Culler. The Ganglia Distributed Monitoring System: Design, Implementation and Experience. *Parallel Computing*, 30(7):817–840, July 2004.

[91] P. McKee, S. Taylor, M. Surridge, R. Lowe, and C. Ragusa. Strategies for the Service Market Place. In J. Altmann and D. J. Veit, editors, *Proc. of the 4th International Workshop on Grid Economics and Business Models (GECON 2007)*, volume 4685 of *Lecture Notes in Computer Science*, pages 58–70. Springer Verlag: Berlin – Heidelberg, 2007.

[92] D. McLaughlin, S. Sardesai, and P. Dusgupta. Preemptive scheduling for distributed systems. In *Proc. of the 11th International Conference on Parallel and Distributed Computing Systems*, 1998.

[93] A. S. Memon. Designing an Information Service for Grid Environments. Master's thesis, RWTH Aachen, 2008.

[94] A. S. Memon, M. S. Memon, Ph. Wieder, and B. Schuller. CIS: An Information Service based on the Common Information Model. In *Proc. of the e-Science and Grid Computing Conference (e-Science 2007)*, pages 465–473. IEEE Computer Society, 2007. ISBN: 978-0-7695-3064-2.

[95] M S. Memon. A Reliable Grid Information Service Using a Unified Information Model. Berichte des Forschungszentrums Jülich, JUEL-4263, RWTH Aachen, 2008.

[96] H. Mersch, Ph. Wieder, B. Koller, G. Murphy, R. Perrot, P. Donachy, and A. Anjomshoaa. Improving Business Opportunities of Financial Service Providers through Service Level Agreements. In D. Talia, R. Yahyapour, and W. Ziegler, editors, *Grid Middleware and Services – Challenges and Solutions (Proc. of the Usage of Service Level Agreements in Grids Workshop in conjunction with the 8th IEEE International Conference on Grid Computing (Grid 2007))*, pages 397–407. Springer US, 2008. ISBN: 978-0-387-78445-8.

[97] H. W. Meuer. The TOP500 Project: Looking Back over 15 Years of Supercomputing Experience. Technical report, University of Mannheim & Prometeus GmbH, 2008.

[98] P. Missier, Ph. Wieder, and W. Ziegler. *Semantic support for Meta-Scheduling in Grids*, volume 3 of *CoreGRID Series*, pages 169–183. Springer, 2005. ISBN: 978-0-387-37830-5.

[99] A.W. Mu'alem and D.G. Feitelson. Utilization, predictability, workloads, and user runtime estimates in scheduling the IBM SP2 with backfilling. *Parallel and Distributed Systems, IEEE Transactions on*, 12(6):529–543, June 2001. DOI: 10.1109/71.932708.

[100] M. G. Nabi and A. Nadeem. A Formal Model for English Auction Protocol. In *7th ACIS International Conference on Software Engineering Research, Management and Applications, 2009 (SERA '09)*, pages 119–126. IEEE Computer Society, 2009.

[101] J. Nabrzyski, J. Schopf, and J. Weglarz, editors. *Grid Resource Management – State of the Art and Future Trends*. Kluwer Academic Publishers, 2003.

[102] S. Nepal, J. Zic, and C. Shiping. WSLA+: Web Service Level Agreement Language for Collaborations. In *Proc. of the IEEE International Conference on Services Computing 2008 (SCC '08)*, pages 485–488, 2008.

[103] Office of Government Commerce. *ITIL – Best Practice for Service Delivery*. The Stationary Office, 2001.

[104] E. Oliveros, H. Muñoz, D. Cantelar, and S. Taylor. BREIN, Towards an Intelligent Grid for Business. In J. Altmann, D. Neumann, and T. Fahringer, editors, *Proc. of the 5th International Workshop on Grid Economics and Business Models (GECON 2008)*, volume 5206 of *Lecture Notes in Computer Science*, pages 163–172, 2008. DOI: 10.1007/978-3-540-85485-2_13.

[105] M. Oszu and P. Valduriez. *Principles of Distributed Database Systems*. Prentice Hall, 1991.

[106] A. Papaspyrou and Ph. Wieder. Activity Instance Document Version 1.0. Grid forum document, The Open Grid Forum, Joliet, Illinois, United States, 2012.

[107] M. Parkin, R. M. Badia, and J. Matrat. A Comparison of SLA Use in Six of the European Commissions FP6 Projects. CoreGRID Technical Report TR-0129, Institute on Resource Management and Scheduling, CoreGRID Network of Excellence, 2008.

[108] M. Parkin, P. Hasselmeyer, B. Koller, and Ph. Wieder. An SLA Re-negotation Protocol. In *Proc. of the 2nd Non Functional Properties and Service Level Agreements in Service Oriented Computing Workshop (NFPSLA-SOC'08) in conjunction with the 6th IEEE European Conference on Web Services*, volume 411 of *CEUR Worskshop Proceedings*. CEUR, 2008. ISSN: 1613-0073.

[109] M. Perry and H. Kaminski. SLA Negotiation System Design Based on Business Rules. In *Proc. of the IEEE International Conference on Services Computing 2008 (SCC '08)*, volume 2, pages 609–612.

[110] A. Pichot, O. Wäldrich, W. Ziegler, and Ph. Wieder. *Towards Dynamic Service Level Agreement Negotiation: An Approach Based on WS-Agreement*, volume 18 of *Lecture Notes in Business Information Processing*, pages 107–119. Springer Berlin – Heidelberg, 2009. DOI: 10.1007/978-3-642-01344-7_9.

[111] M. L. Pinedo. *Scheduling – Theory, Algorithms, and Systems*. Springer New York, 2008. ISBN: 978-0-387-78934-7.

[112] D. M. Quan and O. Kao. SLA Negotiation Protocol for Grid-Based Workflows. In Laurence T. Yang, Omer F. Rana, Beniamino Di Martino, and Jack Dongarra, editors, *High Performance Computing and Communcations*, volume 3726 of *Lecture Notes in Computer Science*, pages 505–510. Springer Berlin / Heidelberg, 2005.

[113] H. Rasheed, R. Gruber, V. Keller, W. Ziegler, O. Wäldrich, Ph. Wieder, and P. Kuonen. IANOS: An Intelligent Application Oriented Scheduling Framework For An HPCN Grid.

In S. Gorlatch, P. Fragopoulou, and T. Priol, editors, *Grid Computing: Achievements and Prospects*, pages 237–248. Springer Verlag, 2008. ISBN: 978-0-387-09456-4.

[114] M. Riedel, V. Sander, Ph. Wieder, and J. Shan. Web Services Agreement based Resource Negotiation in UNICORE. In H. R. Arabnia, editor, *Proc. of the 2005 International Conference on Parallel and Distributed Processing Techniques and Applications (PDPTA05)*, pages 31–37. CSREA Press, 2005.

[115] M. Risch, I. Brandic, and J. Altmann. Using SLA Mapping to Increase Market Liquidity. In A. Dan, F. Gittler, and F. Toumani, editors, *Service-Oriented Computing. ICSOC/ServiceWave 2009 Workshops*, volume 6275 of *Lecture Notes in Computer Science*, pages 238–247. Springer Berlin – Heidelberg, 2010.

[116] A. Sahai, S. Graupner, V. Machiraju, and A. van Moorsel. Specifying and monitoring guarantees in commercial grids through SLA. In *3rd IEEE/ACM International Symposium on Cluster Computing and the Grid (CCGrid 2003)*, pages 292 – 299, May 2003.

[117] J. Salas, F. Perez-Sorrosal, M. Patiño Martínez, and R. Jiménez-Peris. WS-Replication: A Framework for Highly Available Web Services. In *Proc. of the 15th international conference on World Wide Web (WWW '06)*, pages 357–366. ACM, 2006. DOI: 10.1145/1135777.1135831.

[118] V. Sander, B. Allcock, et al. Networking Issues for Grid Infrastructure. Grid Forum Document GFD.37, The Open Grid Forum, Joliet, Illinois, United States, 2004.

[119] B. Schnor, S. Petri, R. Oleyniczak, and H. Langendörfer. Scheduling of Parallel Applications on Heterogeneous Workstation Clusters. In *Proc. of the 9th Int'l Conf. on Parallel and Distributed Computing Systems (PDCS-96)*, 1996.

[120] J. M. Schopf. *Grid Resource Management – State of the Art and Future Trends*, chapter Ten Actions When Grid Scheduling – The User as a Grid Scheduler, pages 15–23. Kluwer Academic Publishers, 2004.

[121] J. M. Schopf, I. Raicu, L. Pearlman, N. Miller, C. Kesselman, I. Foster, and M. D'Arcy. Monitoring and Discovery in Web Services Framework: Functionality and Performance of Globus Toolkit MDS4. Technical Report ANL/MCS-P1248-0405, Argonne National Laboratory, April 2005.

[122] U. Schwiegelshohn, Ph. Wieder, and R. Yahyapour. Resource Management for Future Generation Grids. In V. Getov, D. Laforenza, and A. Reinefeld, editors, *Future Generation Grids, Proceedings of the Dagstuhl Workshop on Future Generation Grids*, pages 99–112. Springer, 2006. DOI: 10.1007/978-0-387-29445-2_6.

[123] J. Seidel, O. Wäldrich, W. Ziegler, Ph. Wieder, and R. Yahyapour. Using SLA for Resource Management and Scheduling – A Survey. In D. Talia, R. Yahyapour, and W. Ziegler, editors, *Grid Middleware and Services – Challenges and Solutions (Proc. of the Usage of Service Level Agreements in Grids Workshop in conjunction with the 8th IEEE International Conference on Grid Computing (Grid 2007))*, pages 335–347. Springer US, 2008. DOI: 10.1007/978-0-387-78446-5_22.

[124] D. Skeen. Nonblocking commit protocols. In *Proc. of the 1981 ACM SIGMOD international conference on Management of Data (SIGMOD '81)*, pages 133–142. ACM, 1981. ISBN: 0-89791-040-0.

[125] L. Smarr and C.E. Catlett. Metacomputing. *Communications of the ACM*, 35(6):44–52, June 1992.

[126] R.G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29(12):1104–1113, 1980.

[127] J. Sonnek, A. Chandra, and J. Weissman. Adaptive Reputation-Based Scheduling on Unreliable Distributed Infrastructures. *IEEE Transactions on Parallel and Distributed Systems*, 18:1551–1564, 2007.

[128] S. Srinivasan, R. Kettimuthu, V. Subramani, and P. Sadayappan. Characterization of backfilling strategies for parallel job scheduling. In *Proc. of the International Conference on Parallel Processing Workshops*, pages 514–519, 2002. DOI: 10.1109/ICPPW.2002.1039773.

[129] A. Streit, D. Erwin, T. Lippert, D. Mallmann, R. Menday, M. Rambadt, M. Riedel, M. Romberg, B. Schuller, and Ph. Wieder. UNICORE – From Project Results to Production Grids. In L. Grandinetti, editor, *Grid Computing: The New Frontier of High Performance Computing (14)*, pages 357–376. Elsevier, 2005.

[130] A. Streit, O. Wäldrich, Ph. Wieder, and W. Ziegler. On Scheduling in UNICORE - Extending the Web Services Agreement based Resource Management Framework. In G.R. Joubert, W.E. Nagel, F.J. Peters, O. Plata, P. Tirado, and E. Zapata, editors, *Parallel Computing: Current & Future Issues of High-End Computing*, pages 57–64. John von Neumann Institute for Computing (NIC), 2006.

[131] A. S. Tanenbaum and M. Van Steen. *Distributed Systems – Principles and Paradigms*. Pearson Prentice Hall, New Jersey, US, 2007.

[132] N. Tonellotto, Ph. Wieder, and R. Yahyapour. A Proposal for a Generic Grid Scheduling Architecture. In S. Gorlatch and M. Danelutto, editors, *Proc. of the Integrated Research in Grid Computing Workshop*, pages 337–346. Università di Pisa, 2005.

[133] David J. Brown und Charles Reams. Toward energy-efficient computing. *Communications of the ACM*, 53(3), 2010.

[134] V. T'kindt und J.-C. Billaut. *Multicriteria Scheduling – Theory, Models and Algorithms*. Springer Berlin – Heidelberg, 2nd edition, 2006. ISBN: 978-3-540-28230-3.

[135] J. van Bon, M. Pieper, A. van der Veen, and I. Verheijen, editors. *Introduction to ITIL*. The Stationary Office, 2005. ISBN: 978-0.11.330973-3.

[136] S. Venugopal, Xingchen Chu, and R. Buyya. A Negotiation Mechanism for Advance Resource Reservations Using the Alternate Offers Protocol. In *16th International Workshop on Quality of Service, 2008 (IWQoS 2008*, pages 40 –49, 2008.

[137] W. Vickrey. Counterspeculation and Competitive Sealed Tenders. *Journal of Finance*, 16(1):8–37, 1961.

[138] O. Wäldrich, D. Battré, F. Brazier, K. Clark, M. Oey, A. Papaspyrou, P. Wieder, and W. Ziegler. WS-Agreement Negotiation Version 1.0. Grid Forum Document GFD.193, Open Grid Forum, Lemont, Illinois, U.S.A., October 2011.

[139] O. Wäldrich, Ph. Wieder, R. Yahyapour, and W. Ziegler. Improving Workflow execution through SLA-based Advance Reservation. In S. Gorlatch, M. Bubak, and T. Priol, editors, *Achievements in European Research on Grid Systems (CoreGRID Integration Workshop 2006)*, pages 207–221. Springer, 2007. DOI: 10.1007/978-0-387-72812-4_16.

[140] O. Wäldrich, Ph. Wieder, and W. Ziegler. Advanced Techniques for Scheduling, Reservation, and Access Management for Remote Laboratories. In *Proc. of the 2nd International Conference on e-Science and Grid Computing (e-Science 2006)*, pages 128–134. IEEE Computer Society, 2006. DOI: 10.1109/E-SCIENCE.2006.261061.

[141] O. Wäldrich, W. Ziegler, and Ph. Wieder. A Meta-Scheduling Service for Co-allocating Arbitrary Types of Resources. In R. Wyrzykowski, J. Dongarra, N. Meyer, and J. Wasniewski, editors, *Proc. of the 2nd Grid Resource Management Workshop (GRMWS05) in conjunction with Parallel Processing and Applied Mathematics: 6th International Conference (PPAM 2005)*, volume 3911 of *LNCS*, pages 782–791. Springer, 2005.

[142] W. Walsh, M. Wellman, P. Wurman, and J. MacKieMason. Some economics of market-based distributed scheduling. In *Proc. of the 18th International Conference on Distributed Computing Systems*, pages 612–621, 1998.

[143] A. Westerinen and J Strassner. Common Information Model (CIM) Core Model Version 2.4. White Paper DSP0111, Distributed Management Task Force, August 2000.

[144] P. Wieder, O. Wäldrich, and W. Ziegler. Advanced Techniques for Scheduling, Reservation, and Access Management for Remote Laboratories. In *Proc. of the 2nd IEEE International Conference on e-Science and Grid Computing e-Science '06*, pages 128–128, Dec. 2006.

[145] Ph. Wieder and R. Yahyapour. *Encyclopedia of Software Engineering*, chapter Service Level Agreements in Grids. Auerbach Publishers Inc., 2010. ISBN: 978-1-4200597-7-9.

[146] Ph. Wieder and W. Ziegler. Bringing Knowledge to Middleware - Grid Scheduling Ontology. In V. Getov, D. Laforenza, and A. Reinefeld, editors, *Future Generation Grids, Proc. of the Dahstuhl Workshop on Future Generation Grids*, pages 47–59. Springer, 2006. DOI: 10.1007/978-0-387-29445-2_3.

[147] Ph. Wieder, W. Ziegler, and V. Keller. IANOS – Efficient Use of HPC Grid Resources. *ERCIM News*, (74):27–29, July 2008. ISSN: 0926-4981.

[148] M. Wilson, A. Arenas, D. Chadwick, T. Dimitrakos, J. Doser, P. Giambiagi, D. Golby, C. Geuer-Pollman, J. Haller, S. Ketil, T. Mahler, L. Martino, X. Parent, S. Ristol,

J. Sairamesh, L. Schubert, and N. Tuptuk. The TrustCoM Approach to Enforcing Agreements between Interoperating Enterprises. In G. Doumeingts, J. Müller, G. Morel, and B. Vallespir, editors, *Enterprise Interoperability*, pages 365–375. Springer London, 2007. DOI: 10.1007/978-1-84628-714-5_34.

[149] R. Yahyapour and Ph. Wieder. Grid Scheduling Use Cases. Grid Forum Document GFD.64, Open Grid Forum, Joliet, Illinois, U.S.A., March 2006.

[150] T. J. Ypma. Historical development of the Newton-Raphson method. *SIAM Review*, 37(4):531–551, 1995.

Part VII.

Appendices

A. Sequence Diagram of the Scheduling Process

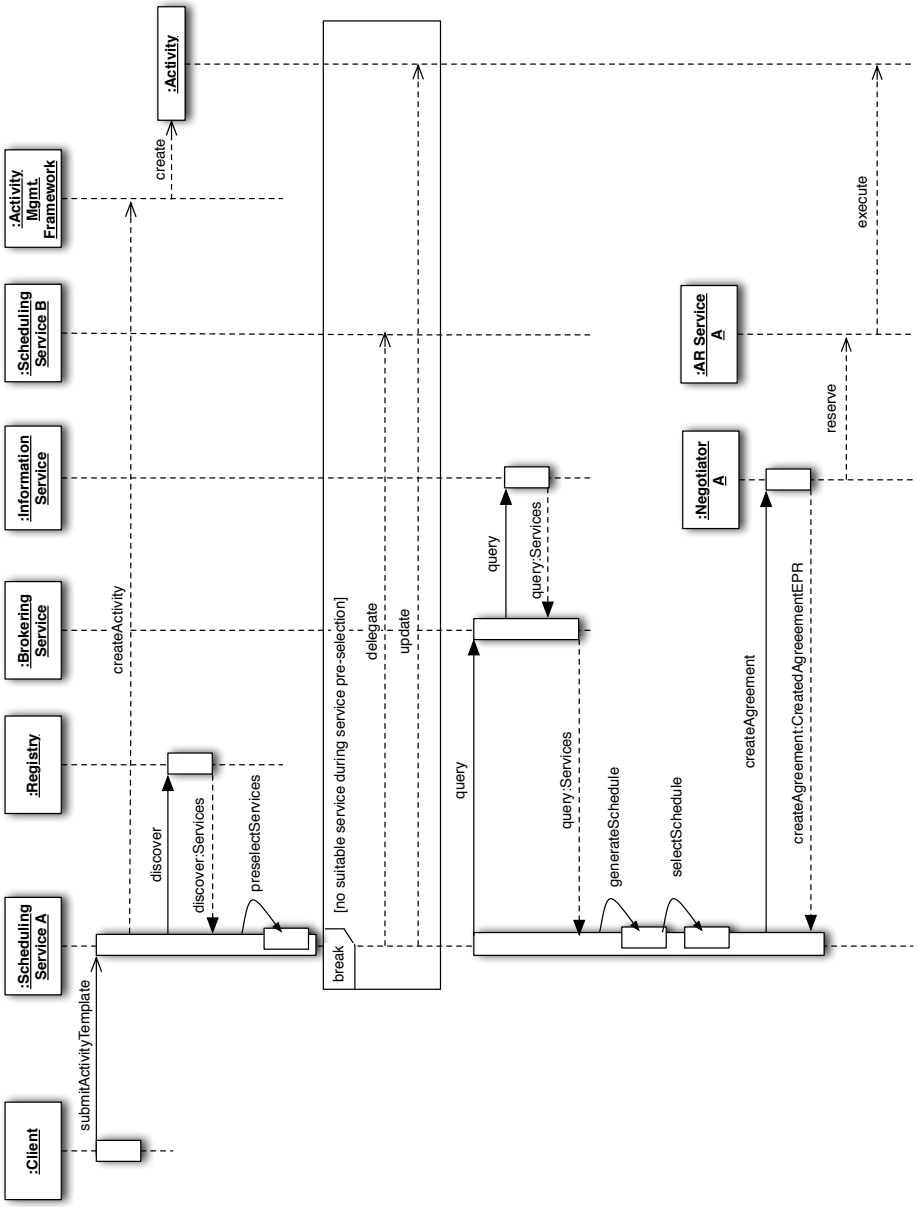

Figure A.1.: The sequence of the scheduling process as described in Section 4

A. Sequence Diagram of the Scheduling Process

Fig. A.1 depicts the sequence of messages exchanged between the services that establish the generic scheduling architecture, as shown in Fig. 4.1. The diagram thereby follows the overall scheduling process specified in Section 4.2 (cf. Fig. 4.3), thus omitting step 9 to 11, i.e. 'monitoring', 'activity completion', and 'termination', to keep the diagram clear.

Further issues to be noted:

- The creation of an activity through the activity management framework results in an EPR, which propagated back to the management framework and to the scheduling service A. The respective messages are not depicted.
- The delegation of an activity, which is depicted in the diagram as a result of preselection step that delivers no results, is actually executed at a well-defined point in time according to the scheduling model to prevent continuous delegations (see Fig. 8.1). Here, the process of flagging an activity as a candidate for delegation, which would include the respective activity in the delegation pool, is explicitly picture to show the messages sent. Again, the propagation of the EPR is not shown explicitly, only the update of the activity, reflecting the delegation, is depicted.
- The steps *generateSchedule* and *selectSchedule*, which represent two strategies of the scheduling model (cf. Fig. 8.1), are together step 3 'Service and resource selection' of the scheduling process.
- The update of the activity following the *reserve* message reflects the status change caused by the execution of the activity (which is also not explicitly depicted).
- No communication with the client is shown as it does add essential information.

B. The UDAP Schema

The following XML Schema captures the UDAP schema as described in Section 5.1 and implemented in Section 9.1.

```
<?xml version="1.0"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://nextgrid.org/2007/01/udap"
xmlns:udap="http://nextgrid.org/2007/01/udap"
xmlns:wsa="http://www.w3.org/2005/08/addressing"
attributeFormDefault="unqualified" elementFormDefault="qualified">

<!-- ****** The UDAP document ****** -->
<xsd:element name="UDAP">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="ActivityID" type="xsd:string" minOccurs="1" maxOccurs="1"/>
<xsd:element name="ActivityDescription" type="udap:DescriptionType"
minOccurs="1" maxOccurs="1"/>
<xsd:element name="Record" type="udap:RecordType" minOccurs="1" maxOccurs="1"/>
<xsd:element name="Result" type="udap:ResultType" minOccurs="0"/>
</xsd:sequence>
</xsd:complexType>
</xsd:element>

<!-- The DescriptionType describes the activity. This is
 basically xsd:any, allowing to embed any activity
 description according to the "Dialect" -->
<xsd:complexType name="DescriptionType">
<xsd:sequence>
<xsd:any namespace="##other" minOccurs="0" processContents="lax"/>
</xsd:sequence>
<xsd:attribute name="Dialect" type="xsd:string" use="required"/>
</xsd:complexType>

<!-- The RecordType -->
<xsd:complexType name="RecordType">
<xsd:sequence>
<xsd:element name="Entry" type="udap:RecordEntryType" minOccurs="0"
maxOccurs="unbounded"/>
</xsd:sequence>
</xsd:complexType>
```

```
<!-- The RecordEntryType -->
<xsd:complexType name="RecordEntryType">
<xsd:sequence>
<xsd:element name="TimeStamp" type="xsd:dateTime" minOccurs="1" maxOccurs="1"/>
<xsd:element name="State" type="udap:StateType" minOccurs="1" maxOccurs="1"/>
<xsd:element name="Resource" type="udap:ResourceType" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element name="Context" type="udap:ContextType" minOccurs="0"
maxOccurs="unbounded"/>
<xsd:element name="Dependency" type="udap:DependencyType" minOccurs="0"
maxOccurs="unbounded"/>
</xsd:sequence>
<xsd:attribute name="Category" type="udap:RecordEntryCategoryType"/>
</xsd:complexType>

<!-- Activity state -->
<xsd:simpleType name="StateType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="pending"/>
<xsd:enumeration value="running"/>
<xsd:enumeration value="runningPreempted"/>
<xsd:enumeration value="runningMigrating"/>
<xsd:enumeration value="runningPaused"/>
<xsd:enumeration value="cancelled"/>
<xsd:enumeration value="failed"/>
<xsd:enumeration value="finished"/>
</xsd:restriction>
</xsd:simpleType>

<!-- The ResourceType defines the resource-related requirements of a UDAP document -->
<xsd:complexType name="ResourceType">
<xsd:sequence>
<xsd:any namespace="##other" minOccurs="0" processContents="lax"/>
</xsd:sequence>
</xsd:complexType>

<!-- The ContextType describes policy constraints, QoS, security characteristics
 related to an activity -->
<xsd:complexType name="ContextType">
<xsd:sequence>
<xsd:any namespace="##other" minOccurs="0" processContents="lax"/>
</xsd:sequence>
</xsd:complexType>

<!-- Activity dependency -->
<xsd:complexType name="DependencyType">
```

```
<xsd:sequence>
<xsd:any namespace="##other" minOccurs="0" processContents="lax"/>
</xsd:sequence>
</xsd:complexType>

<!-- An entry in the record can be:
- the original one, i.e. the one submitted to the UDAP manager,
- the current one, which represents the current valid document, and
- an archive one, provided for monitoring and persistence. -->
<xsd:simpleType name="RecordEntryCategoryType">
<xsd:restriction base="xsd:string">
<xsd:enumeration value="original"/>
<xsd:enumeration value="current"/>
<xsd:enumeration value="archive"/>
</xsd:restriction>
</xsd:simpleType>

<!-- The ResultType -->
<xsd:complexType name="ResultType">
<xsd:sequence>
<xsd:any namespace="##other" minOccurs="0" maxOccurs="unbounded"
processContents="lax"/>
</xsd:sequence>
</xsd:complexType>

</xsd:schema>
```

C. Strategies related to the Scheduling Model

This chapter outlines the details of the various strategies that are evaluated in Chapter 10. The strategies are ordered according to the classification used in Section 10.3.

C.1. Delegation Strategies

CPU Threshold Delegation Strategy (cpu_thresh_del)

Three strategies have been realised based on a passing window approach. The first one is the *CPU Threshold Delegation Strategy*. In the beginning, it puts all queued activities and newly arrived activities in a common pool, which is ordered according the number of CPUs requested by every activity (in ascending order). The strategy then iterates over the pool, adding the current activity's number of requested CPUs to a variable. If this variable reaches an upper threshold, defining the maximum number CPUs, the iteration is terminated. If the variable stays below the threshold, the current activity is added to a collection of activities to be delegated (the delegation list), and the iteration is continued until all activities in the pool have been processed. The number of CPUs in the delegation list, in addition, needs to satisfy a lower threshold, too, which in combination with the upper threshold, defines the passing window. If this threshold is satisfied the delegation list is processed further, otherwise it is deleted and all activities remain in the queue. The simulation set-up as described in Section 10 uses a minimal passing window in the range $w \in [1, 32]$.

Ordering activities previous to the application of the strategy prevents huge activities from blocking delegations as negotiations that do not succeed for a very long time due to site fragementation [73]. Since all activities marked for delegation must be delegated, the upper threshold constrains the maximal size of a collection. The lower threshold, however, reduces the communication overhead related to negotiations. It prevents frequent negotiations with a small amount of activities. In addition, it keeps a buffer of activities in the local queue to be executed locally.

Minimal values for the lower and the upper thresholds are pre-defined. Depending on the outcome of a negotiation, the window is adapted. In the case of a successful delegation both values are doubled, in case of an unsuccessful negotiation, they are halved. Several threshold values between 0.1 and 10 have been evaluated, the value 2, though, delivered best results. Further investigations to tailor this factor seem to be promising, e.g. dynamic adaptation of the change rate rate are an option.

A challenge for the window-based approach is rapid changes of the available resources. Then this strategy may not deliver the desired results. The thresholds also have to be carefully set as the delegation strategy will not work properly if the smallest activity in the queue requests more CPUs than are allowed by the upper threshold. In such a case the

window size can lead to locking delegation until an activity is submitted that fits into the window. A following successful delegation than probably increases the window size again.

Runtime Threshold Delegation Strategy (runtime_thresh_del)

The *runtime_thresh_del* strategy is similar to the *cpu_thresh_del* strategy with the only difference of using the total runtime instead of the number of CPUs. The total runtime is the runtime assigned to an activity times the requested number of CPUs. The minimal window size is

$w \in [\frac{1}{2} \cdot \mathrm{runtime}_{\mathrm{static}}, 200 \cdot \mathrm{runtime}_{\mathrm{static}}]$.

analogue to the passing window size introduced in the previous section. $\mathrm{runtime}_{\mathrm{static}}$ is the site-specific static value for the total runtime.

Activity Threshold Delegation Strategy (activity_tresh_del)

The *activity_tresh_del* belongs to the class of passing window strategies. It uses solely the number of activities in a queue as the basis for delegation. The minimal window is $w \in [1, 2]$.

Global Average Weighted Response Time Delegation Strategy (global_awrt_del)

The *global_awrt_del* strategy polls a universal metric service, which is implemented as part of the simulation environment, for the global average weighted response time $AWRT_{\mathrm{global}}$ and compares it to the local value $AWRT_{\mathrm{local}}$. The decision to delegate either all activities in the queue or none is calculated as follows:

$AWRT_{\mathrm{local}} > AWRT_{\mathrm{global}} \Rightarrow \mathrm{DL} = \mathrm{Q}$ (delegate all activities),
else $\mathrm{DL} = \emptyset$ (delegate none)

The simulations revealed that in cases where only the *global_awrt_del* strategy is used, the delegation list DL gets large. Such a large number of activities normally cannot be delegated since its requested resources most likely exceed the available resources. Therefore, a combination with, for example, the *cpu_thresh_del* strategy can be used to adjust the size of DL respectively.

No Delegation Strategy (no_del)

The *no_del* strategy always returns an empty DL and therefore prevents all delegations from a particular site. Delegation to this site is still possible. Using the *no_del* strategy, so-called *sinks* are created, which do not delegate their activities but are able to receive activities from other sites (see also Section 10.5.1).

Random Delegation Strategy (random_del)

The *random_del* strategy assigns every activity in the queue a chance of 50% ($\mathrm{p} = 0.5$) to be delegated. Depending on the outcome of the assignment, the chance for delegation of subsequent activities is either increased by $\mathrm{p} = \mathrm{p} \cdot 1.1$ in case the previous activity has been delegated or decreased by $\mathrm{p} = \mathrm{p} \cdot 0.9$ in case of no previous delegation.

Wave All Activities Through Delegation Strategy (wave-all_del)

The *wave-all_del* strategy always delegates all activities in the queue and all newly submitted activities.

Wave All New Activities Through Delegation Strategy (wave-all-new_del)

The *wave-all-new_del* strategy always delegates the activities that have been submitted since the previous pass of the scheduling process.

C.2. Pre-selection Strategies

Two *Pre-selection Strategies* have been implemented:

- Random Pre-selection Strategy (*random_pre-sel*) — each candidate service is given a chance of 50% to pass
- Wave Through Pre-selection Strategy (*wave_pre-sel*) — all candidate services pass

No further pre-section strategies have been implemented since the number of SLA templates representing candidate services is rather low in the simulation set-up. As such, these strategies do not have a great effect on the overall performance of the scheduling process. In an environment that features a large number of sites and which offers multiple templates per site, further pre-selection will be necessary. A simple pre-selection might for example check for candidate services that do not offer enough resources to be used in the scheduling process and dismiss them.

C.3. Scheduling Strategies

Random Scheduling Strategy (random_sched)

The *random_sched* strategy takes a random SLA template and iterates over all unscheduled activities in the queue. If an activity fits, it is assigned to the template. In case all activities have been checked without any assignment to the template, the next random template is chosen. This process is repeated until no more templates are available or until all queued activities have been assigned to templates. The former case implies that no schedule could be created, the latter finishes with a valid schedule. This strategy is repeated five times resulting in 0 to 5 schedules as input to any schedule selection strategy.

Price Greedy Scheduling Strategy (price-greedy_sched)

The *price-greedy_sched* scheduling strategy chooses the cheapest SLA template, selects the largest queued activity, and tries to match both. This process is shown in Fig. C.1. The 'size' of a template is determined by the product $\mathrm{CPUs} \cdot \mathrm{runtime}$. This strategy, which is listed as Algorithm 1, has a complexity of $\mathrm{O}(\mathrm{n} \cdot \mathrm{k})$, with $|\mathrm{activities}| = \mathrm{n}$ and $|\mathrm{templates}| = \mathrm{k}$. It takes a maximum of n iterations with each of the k templates. This is a relatively low complexity compared to the non-polynomial complexities of more elaborated

strategies. Therefore, *price-greedy_sched* is used as a back-up strategy in cases where the computation of a schedule using a different strategy takes too long (the default threshold here is 30 seconds).

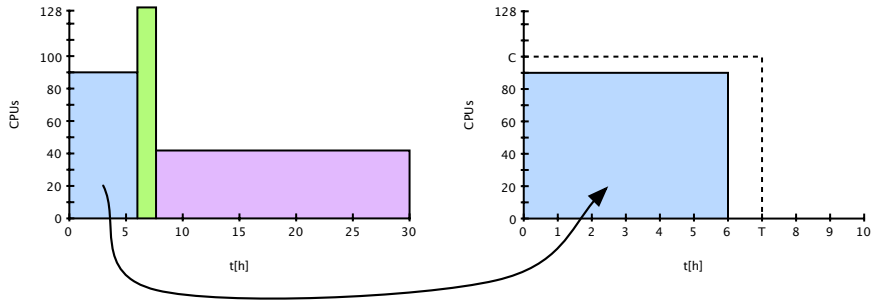

Figure C.1.: The largest matching activity from the queue (left) is chosen for an SLA template (right)

Algorithm 1 Price Greedy Scheduling Strategy

```
activities = getSortedActivities(activities)
template = retrieveCheapestTemplate(templates)
while template ≠ null ∧ |activities| > 0 do
  for i = 0 to |activities| do
    activityCPUs = activity[i].cpus
    activityRuntime = activity[i].runtime
    if activityCPUs ⩾ template.cpus ∧ activityRuntime ⩾ (activityRuntime ·
    activityCPUs) then
      scenario.addLink(template, activity)
      template.cpus = template.cpus − activityCPUs
      template.runtime = template.runtime − activityCPUs · activityRuntime
      activities.remove(activity)
      i − −
    end if
  end for
  template = retrieveCheapestTemplate(templates)
end while
```

Lowest Cost First Lowest CPU First Scheduling Strategy (low-cost_low-cpu_first_sched)

The *low-cost_low-cpu_first_sched* strategy iterates, like the *price-greedy_sched* one, over all SLA templates starting with the cheapest price. For each template, it iterates over all activities in the queue, which are sorted in this case by ascending number of requested CPUs. Each activity is assigned to the selected template if it still fits regarding the requested runtime and requested CPUs. This strategy, which is listed as Algorithm 2, op-

erates along the same lines as the price greedy scheduling strategy. The only exception is the initial activity order that is based, in case of *low-cost_low-cpu_first_sched*, on the requested number of CPUs. As such, the strategy also has the complexity of $O(n \cdot k)$.

Algorithm 2 Lowest Cost First Lowest CPU First Scheduling Strategy

```
activities = sortActivitiesByCPUsAscending(activities)
template = pollCheapestTemplate()
while template ≠ null ∧ |activities| > 0 do
  for all activity in activities do
    if activity.cpus ⩽ template.cpus then
      if (activity.time · activity.cpus) ⩽ template.time then
        scenario.addLink(template, activity)
        template.time = template.time − (activity.time · activity.cpus)
        template.cpus = template.cpus − activity.cpus
        activityToRemove = activitiesToRemove + activity
      end if
    end if
  end for
  activities.remove(activitiesToRemove)
end while
```

Lowest Cost First Shortest Runtime First Scheduling Strategy (low-cost_low-rt_first_sched)

The *low-cost_low-rt_first_sched* strategy follows the same approach as the *low-cost_low-cpu_first_sched* strategy. This time, however, the requested runtime is used to sort the activities. The complexity again is $O(n \cdot k)$.

Lowest Cost First Minimal CPU Tardiness Scheduling Strategy (low-cost_first_min_cpu-tard_sched)

The *low-cost_first_min_cpu-tard_sched* strategy iterates also over all SLA templates, starting with the cheapest. To match the selected template, it uses a strategy to minimise tardiness on a particular machine ($1||\sum T_j$, with T_j being the tardiness of activity j, see also [111]).

The pre-requisites of our evaluation set-up (cf. Section 10.5.1) for the application of the strategy are the following: activities have no due dates, all activities have already been released, and there is a common deadline for each template (i.e. total runtime). Since activities cannot be scheduled in sequence this strategy cannot be used as described by Pinedo [111]. Instead, it will treat the amount of free CPUs as time. The actual time for each slot will be used to filter out activities that do not fit. Fig. C.2 and C.3 show an activity being scheduled into a slot and the resulting adaptation of the slot, respectively. The next activity will be scheduled in sequence regarding to the abscissa, representing the free CPUs. Here, the amount of available CPUs is treated as common due date for all activities $cpu_{total} \equiv d_j, j \in activities$ and the CPUs as processing times $cpus_j \equiv p_j$. The tardiness is then the amount of CPUs that are needed in addition to the available ones.

This algorithm thus computes a schedule for each SLA template minimising the tardiness. All “tardy” activities are then collected and used for the next template.

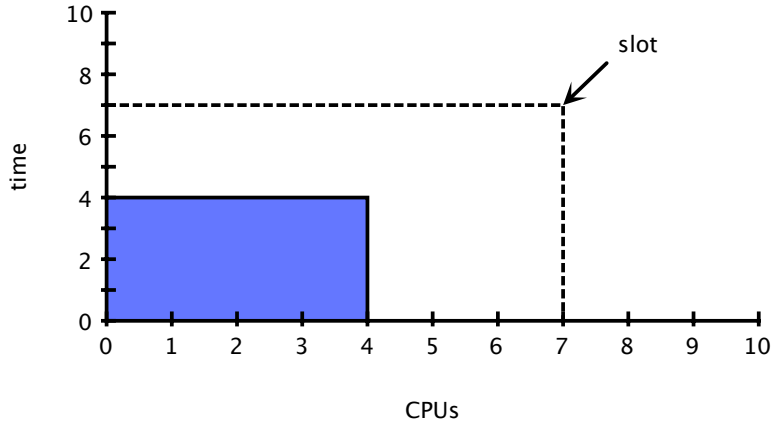

Figure C.2.: An activity is scheduled into an empty slot

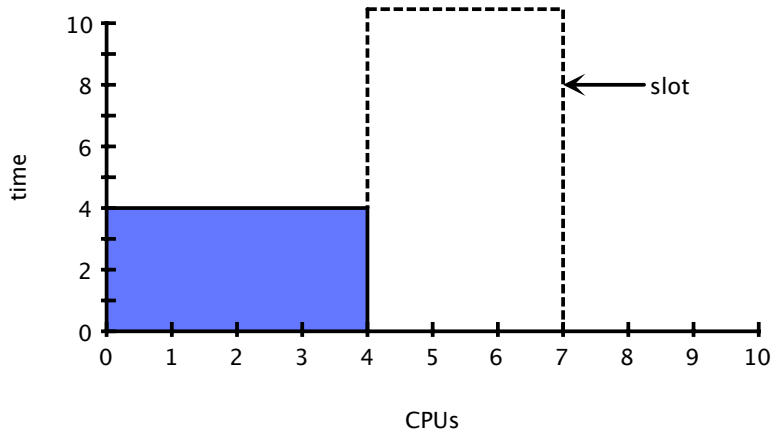

Figure C.3.: The slot size changes due to the scheduled activity

Since we do not want to produce schedules with tardy activities, an extension of the algorithm has been implemented to remove all tardy activities. A further extension is used to iterate over all templates to receive a complete scheduling scenario using as many templates as needed. For each schedule that has been calculated with the previous approach a number of sub-schedules using the remaining templates is created. These sub-schedules are then combined with the present schedule to a number of complete schedules. A tree representing the various created schedules is illustrated in Fig. C.4. The vertical axis goes deeper into the recursions and uses each time a new template. The horizontal axis shows alternate solutions using the same set of activities and templates. Fig. C.5 shows the creation of a schedule in one path down the recursion. At the beginning there are four activities and four templates. Two activities can be scheduled into the first template. Then,

in the next recursion, the remaining two activities are scheduled into the second template.

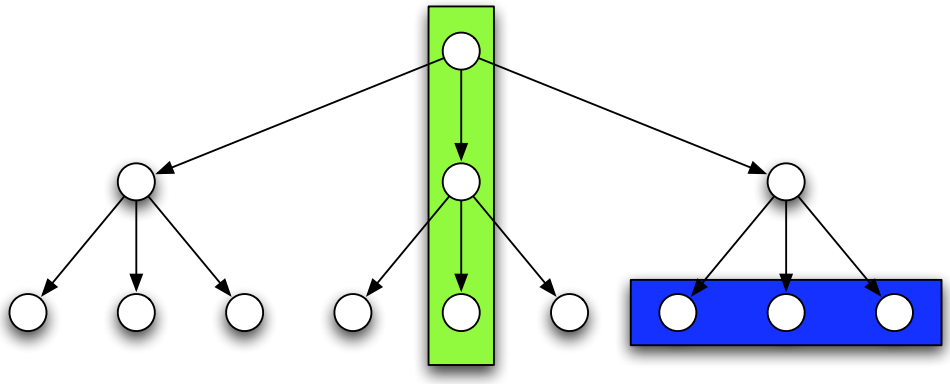

Figure C.4.: Choices encountered by the extended minimal tardiness strategy

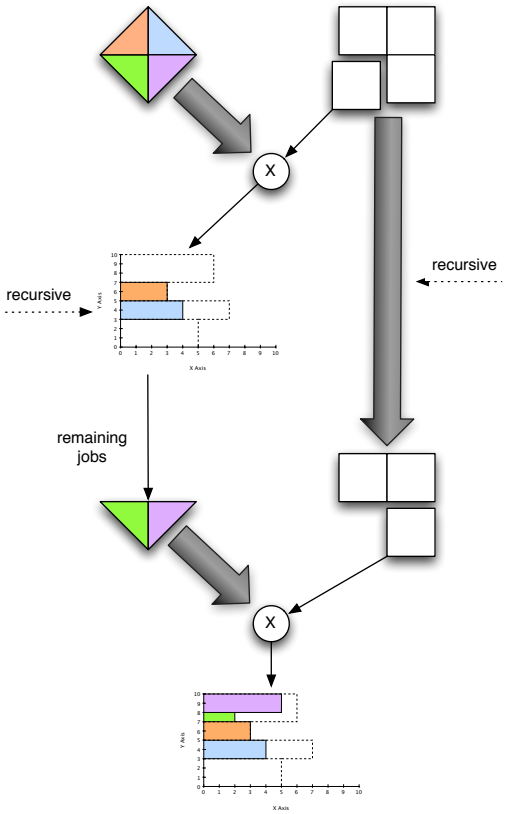

Figure C.5.: Extended minimal tardiness strategy with two recursions

Algorithm 3 first polls the cheapest templates and then creates all optimal scenarios us-

Algorithm 3 Extended version of the minimal tardiness strategy

```
template = pollCheapestTemplate()
if template ≠ null ∧ |activities| > 0 then
  optimal = calcOptSchedulesWithMinTardiness(template, activities, 0);
  for all scenario in optimal do
    remainingActivities = 0
    linksToRemove = 0
    usedCPUs = 0
    usedTime = 0
    for all link in scenario.links do
      usedCPUs = usedCPUs + link.activity.cpus
      usedTime = usedTime + link.activity.time · link.activity.cpus
      if usedCPUs > template.cpus ∨ usedTime > template.time then
        remainingActivities = remainingActivities + link.activity
        linksToRemove = linksToRemove + link
      end if
    end for
    scenario.removeLinks(linksToRemove)
    subSchedules = createSchedules(templates, remainingActivities)
    for all scenario in subSchedules do
      combinedScenario = combineScenarios(scenario, subSchedule)
      finalScenarios.add(combinedScenario)
    end for
  end for
else
  if |activities| = 0 then
    finalScenario.add(∅)
  end if
end if
```

ing the minimal tardiness algorithm. For each schedule activities remain. Those activities are separated from the schedule. These activities and the remaining templates are used to create optimal schedules for the sub-problem. The sub-schedules are then combined with the previous schedule to an equal number of complete schedules. A leaf in the schedule tree contains an empty set of optimal schedules if all activities have been scheduled. If no more templates are available a leaf containing a null value which is the end of a failed path is returned. This algorithm can be further optimized by only selecting a number of sub-schedules from all available ones.

C.4. Schedule Selection Strategies

Two schedule selection strategies have been implemented:

- Random Schedule Selection Strategy (*random_sched-sel*) — each site is given a chance of 50% to pass,
- Wave Through Schedule Selection Strategy (*wave_sched-sel)* — all sites pass.

No further schedule selection strategies have been implemented as the simulation set-up does not feature more than six sites.

C.5. Negotiation Strategies

Following the WS-Agreement specification to implement an SLA-framework, we only implemented its basic negotiation strategy (*wsag_basic_neg*), which realises an accept/reject or discrete offer negotiation model.

D. Provider Policies

Providers can use the following policies to implement their business objectives. The following policies have been implemented and evaluated throughout the simulation.

D.1. CPU Time Policies

Global AWRT CPU Time Policy (global-awrt_cpu-time_policy)

The total CPU time ($\mathrm{TotalCPUTime}$) offered at a point in time t by a particular site through an SLA is a function of local and global AWRT and of the previous $\mathrm{TotalCPUTime}$ value:

$$\mathrm{TotalCPUTime}(\mathrm{t}) = \mathrm{f}(\mathrm{TotalCPUTime}(\mathrm{t}-1), AWRT_{\mathrm{global}}(\mathrm{t}), AWRT_{\mathrm{local}}(\mathrm{t})).$$

The current value changes depending on following conditions:
$AWRT_{\mathrm{local}} > AWRT_{\mathrm{global}} \Rightarrow \mathrm{CPUTime} \uparrow$,
$AWRT_{\mathrm{local}} < AWRT_{\mathrm{global}} \Rightarrow \mathrm{CPUTime} \downarrow$.
The $\mathrm{TotalCPUTime}$ is increased or decreased, depending on the aforementioned conditions, by an interval $\Delta\mathrm{CPUTime} = \frac{1}{4} \cdot \mathrm{CPUTime}_{\mathrm{static}}$. The result is limited by a lower boundary $\mathrm{TotalCPUTime}_{\mathrm{min}} = \frac{1}{4} \cdot \mathrm{CPUTime}_{\mathrm{static}}$. The site-specific value is updated every time an event related to the local execution of an activity is received. This may be either the start or the completion of an execution.

Other CPU Time Policies

Another total CPU time strategy is the Random CPU Time Policy (*random_cpu-time_policy*). It returns a random value between $\frac{1}{2} \cdot \mathrm{CPUTime}_{\mathrm{static}}$ and $1.5 \cdot \mathrm{CPUTime}_{\mathrm{static}}$. The No CPU Time Policy (*no_cpu-time_policy*) returns no total CPU time and as such can be used to create a source node. A source node does not accept any remotely incoming activities, but it delegates its own activities to other sites. Last, but not least, the Static CPU Time Policy (*static_cpu-time_policy*) represent a provider's simple policy to always allow a fixed CPU time independent of the amount of free processors.

D.2. Pricing Policies

(Global) Utilization-Based Pricing Policy ((global_)*util_price_policy*)

The Utilisation-Based Pricing Policy (*util_price_policy*) adjusts the price attached to an SLA template according to the costs of a particular site. Metrics used for this calculation are (with global values for the respective global policy):

- u_{cur} — the current utilization of the site
- $\bar{u}$ — the average utilization of the site
- x — x determines the price according to $price = x \cdot price_{static}$.

Fig. D.1 shows the values for x, which have been implemented in the simulation, as a solid line. There are three sections in the diagram. The first one $0 \leqslant u \leqslant 0.5 \cdot \bar{u}$ (with $\bar{u}$ termed u_{av} in the figure) has constantly $x = 0.5$. The next section $0.5 \cdot \bar{u} < u \leqslant \bar{u}$ with x being linearly interpolated between 0.5 and 1. The third section $\bar{u} < u \leqslant 1$ interpolates a value for x in $]1, \ldots 2]$.

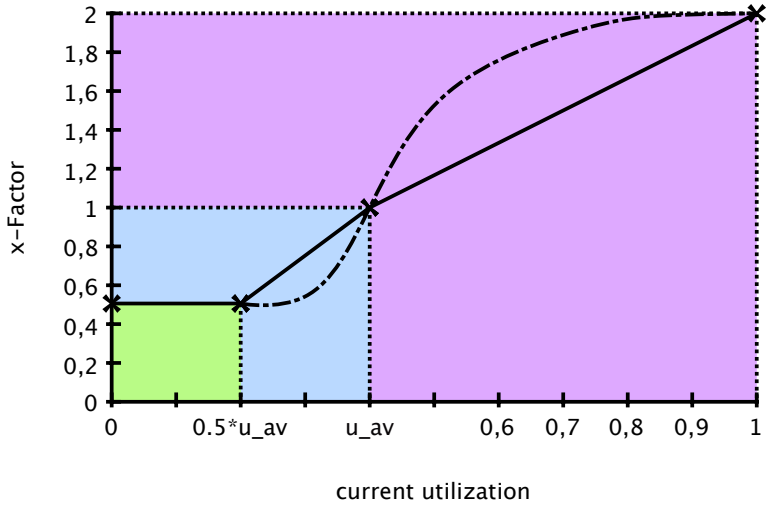

Figure D.1.: Two possible pricing-functions

The respective decision to calculate x can be described as follows:

$u_{cur} > \bar{u} \Rightarrow \frac{x-1}{u_{cur}-\bar{u}} = \frac{2-1}{1-\bar{u}} \Leftrightarrow x = \frac{u_{cur}-\bar{u}}{1-\bar{u}} + 1$

$u_{cur} < \bar{u} \Rightarrow \frac{x-\frac{1}{2}}{u_{cur}-\frac{1}{2}\bar{u}} = \frac{1-\frac{1}{2}}{\bar{u}-\frac{1}{2}\bar{u}} \Leftrightarrow x = \frac{u_{cur}-\frac{1}{2}\bar{u}}{\bar{u}} + \frac{1}{2}$

$u_{cur} = \bar{u} \Rightarrow x = 1$

This way of determining the price has the effect that if a site currently runs below its long time average utilization, it provides a cheaper price to attract consumer. In case the site operates above its average utilization it, consequently, offers a higher price.

The implemented pricing scheme with linear interpolation is one basic approach of determining the price based on the average utilization. A function similar to the dotted curve in Fig. D.1 could also be used. The determination of an optimal curve for a certain provider would exceed our work and should be evaluated separately, preferably through an economic application scenario.

Static Pricing Policy (static_price_policy)

The Static Pricing Policy (*static_price_policy*) keeps the same price for each SLA template for the whole duration of the simulation.

E. Trace Generation of Artifical Traces

To generate a trace, the maximum amount of available CPUs ($maxCPUs$), the total simulation time ($totalSimTime$), and the intended activity size have to be known. Using this information, chunks for time intervals are created as described in Algorithm 4. The chunks contain the same activity sizes throughout the interval. Fig. E.1 shows a chunk. First, the total CPU time is fragmented in accordance to the intended CPU size of the activities. Then, for each region of CPUs the whole duration is fragmented. The values for those fragmentations are chosen randomly from a pre-configured interval. All chunks are generated in sequence and are appended into one trace file as described by Algorithm 5.

Algorithm 4 The generation of a trace-chunk

$cpuFragmentationIndex = random(cpuFragmentation)$
$cpus = \frac{maxCPUs}{cpuFragmentationIndex}$
for all i in $cpuFragmentationIndex$ **do**
  $timeFragmentationIndex$ = random($timeFragmentation$)
  $duration = \frac{\Delta t}{timeFragmentationIndex}$
  for all i in $timeFragmentationIndex$ **do**
    $activities$.add(new Activity($startTime$, $cpus$, $duration$))
  end for
end for

Algorithm 5 The generation of a complete SWF trace

$\Delta t = chunkSize$
$t_{current} = 0$
while $t_{current} <= totalSimTime$ **do**
  generateChunk($t_{current}$, Δt, $cpuFragentation$, $timeFragentation$, $maxCPUs$)
end while

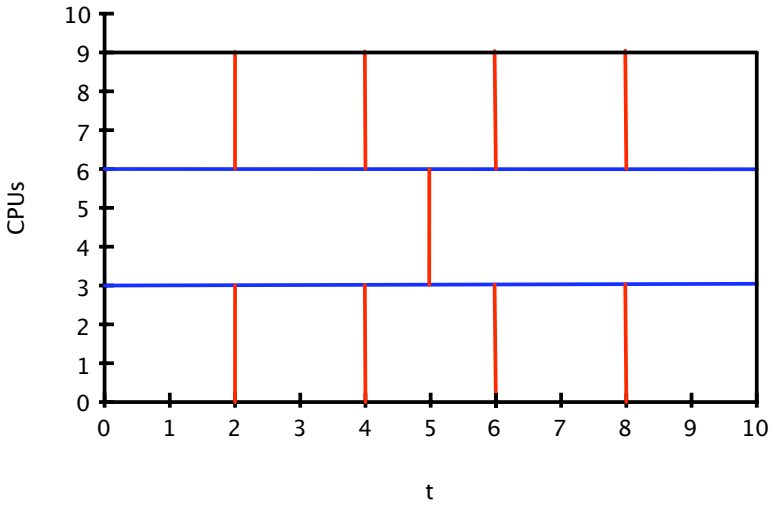

Figure E.1.: Fragmentation of CPUs and runtime