


2006

Analysis Of Aircraft Arrival Delay And Airport On-time Performance

Yuqiong Bai
University of Central Florida

 Part of the [Computer Sciences Commons](#), and the [Engineering Commons](#)
Find similar works at: <https://stars.library.ucf.edu/etd>
University of Central Florida Libraries <http://library.ucf.edu>

This Doctoral Dissertation (Open Access) is brought to you for free and open access by STARS. It has been accepted for inclusion in Electronic Theses and Dissertations, 2004-2019 by an authorized administrator of STARS. For more information, please contact STARS@ucf.edu.

STARS Citation

Bai, Yuqiong, "Analysis Of Aircraft Arrival Delay And Airport On-time Performance" (2006). *Electronic Theses and Dissertations, 2004-2019*. 890.
<https://stars.library.ucf.edu/etd/890>

COORDINATION, MATCHMAKING, AND RESOURCE ALLOCATION FOR LARGE-SCALE DISTRIBUTED SYSTEMS

by

XIN BAI

B.S. Northern Jiaotong University, 1993

M.S. University of Central Florida, 2003

A dissertation submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
in the School of Electrical Engineering and Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Summer Term
2006

Major Professor:
Dan C. Marinescu

© 2006 Xin Bai

ABSTRACT

While existing grid environments cater to specific needs of a particular user community, we need to go beyond them and consider general-purpose large-scale distributed systems consisting of large collections of heterogeneous computers and communication systems shared by a large user population with very diverse requirements. Coordination, matchmaking, and resource allocation are among the essential functions of large-scale distributed systems. Although deterministic approaches for coordination, matchmaking, and resource allocation have been well studied, they are not suitable for large-scale distributed systems due to the large-scale, the autonomy, and the dynamics of the systems. We have to seek for nondeterministic solutions for large-scale distributed systems. In this dissertation we describe our work on a coordination service, a matchmaking service, and a macro-economic resource allocation model for large-scale distributed systems. The coordination service coordinates the execution of complex tasks in a dynamic environment, the matchmaking service supports finding the appropriate resources for users, and the macro-economic resource allocation model allows a broker to mediate resource providers who want to maximize their revenues and resource consumers who want to get the best resources at the lowest possible price, with some global objectives, e.g., to maximize the resource utilization of the system.

*To my wife: **Xia Liu***

*and my parents: **Banghua Bai** and **Meixian Yang***

ACKNOWLEDGMENTS

This dissertation would not have been completed without the advice, supports, and encouragement from my advisor, my research committee members, my research associates and my group colleagues.

I would like to express my sincere gratitude to **Dr. Dan C. Marinescu** for his valuable advice during my doctoral research endeavor in the past years. As my advisor, he has brought me to the field of large-scale distributed systems, and has spent countless time and efforts on sharing his research ideas with me. He has helped me to establish the ultimate goal of my research work and has constantly encouraged me to remain focused to reach the final achievement. His generous support is significant not only in the academic aspects, but also in the financial situations. I am very honored and appreciative to have such a great opportunity to work with him!

I would like to thank my committee members, **Dr. Ladislau Bölöni**, **Dr. Mostafa Bassiouni** and **Dr. Joohan Lee**, for their precious and thoughtful comments to help me improve my dissertation. I am very grateful and honored to have them in my committee. In addition, I would like to thank **Dr. Howard Jay Siegel** from the Colorado State University for his insightful suggestions on my research papers.

TABLE OF CONTENTS

LIST OF TABLES	ix
LIST OF FIGURES	x
LIST OF PUBLICATIONS	xvi
CHAPTER 1 INTRODUCTION	1
1.1 Motivation	2
1.2 Algorithms, Models, Implementation, and Performance Studies	8
1.2.1 Algorithms and Models	8
1.2.2 The Implementation	10
1.2.3 Performance Studies	23
1.3 Contributions	24
1.4 Organization	25
CHAPTER 2 COORDINATION	26
2.1 Introduction and Motivation	27

2.2	Background and Related Work	29
2.2.1	Grid Computing	29
2.2.2	Agent-Based Computing	35
2.2.3	Workflow Management	40
2.3	Coordination and Coordination Services	43
2.3.1	Process Coordination	44
2.3.2	Coordination Techniques	47
2.3.3	Process Coordination and Workflow Management	58
2.3.4	Process Description and Case Description	64
2.3.5	Coordination Services	64
2.4	A Case Study: the Coordination Service in BondGrid	67
2.4.1	Process Description and Case Description	67
2.4.2	Ontologies for BondGrid Coordination	70
2.4.3	The Coordination Service	71
2.4.4	Performance Measurements	81
CHAPTER 3	MATCHMAKING	83
3.1	Introduction and Motivation	84
3.2	Background and Related Work	87

3.3	Resource Ontologies	91
3.4	The Matchmaking Problem	92
3.5	A Case Study: the Matchmaking Service in BondGrid	101
3.5.1	The Matchmaking Service	101
3.5.2	Performance Measurements	104
 CHAPTER 4 A MACRO-ECONOMIC RESOURCE ALLOCATION MODEL		
	112
4.1	Introduction and Motivation	112
4.2	Background and Related Work	117
4.3	Basic Concepts	122
4.3.1	Price Function	123
4.3.2	Utility Function	126
4.3.3	Satisfaction Function	127
4.3.4	Resource Provider-Consumer Model	129
4.4	The Role of Brokers in the Macro-Economic Model	133
4.5	A Simulation Study	136
 CHAPTER 5 CONCLUSIONS		149
 LIST OF REFERENCES		160

LIST OF TABLES

4.1	The parameters for the simulation are uniformly distributed. The parameters and the corresponding intervals are shown.	139
-----	--	-----

LIST OF FIGURES

1.1	Core and end-user services.	16
1.2	XML specification for instances.	19
1.3	XML specification for classes.	20
1.4	The blueprint for a coordination service.	21
2.1	The coordination can be centralized, or distributed; the components may be confined to a single system, to a LAN, or to a WAN; the system may be open, or closed.	45
2.2	A shared-data space coordination model. The producer of the coordination information pushes an item into the shared data space, a consumer pulls it out. Little, or no state information needs to be maintained by the shared- data space. The model supports asynchronous communication between mobile agents. The agents may join and leave at will, the model supports open systems.	51

2.3	A broker acts as an intermediary between a client and a set of servers. The sequence of events: (i) servers register with a broker; (ii) a client sends a request; (iii) the broker forwards the request to a server; (iv) the server provides the response to the broker; (v) the broker forwards the request to the client.	55
2.4	A matchmaker helps a client select a server. Then the client communicates directly with the server selected by the matchmaker. The sequence of events: (i) servers register with the matchmaker; (ii) a client sends a request to a broker; (iii) the broker selects a server and provides its ID; (iv) the client sends a request to the server selected during the previous step; (v) the server provides the response to the client.	57
2.5	A mediator acts as a front end or a wrapper to one or more servers; it translate requests and responses into a format understood by the intended recipient. A mediator may be used in conjunction with brokers or matchmakers.	58
2.6	BNF grammar for the process description.	65
2.7	A process description for the 3D structure determination. $D1, D2, \dots D13$ are the symbolic names of the input and output data files for the programs carrying out different end-user activities.	68
2.8	Logic view of the main ontology for BondGrid coordination.	72
2.9	The components of the coordination service.	73
2.10	The task state transition diagram.	74

2.11	The activity state transition diagram.	76
2.12	The coordination engine executes iteratively the procedure.	77
2.13	The message handler executes iteratively the procedure.	78
2.14	The interactions between the coordination service and the end-user.	78
2.15	The interactions between the coordination service and other core services. . .	79
2.16	The performance of encoding, transmitting, and decoding instances between the coordination service and other components of the environment.	82
3.1	The matchmaking process in large-scale distributed systems: 1) Providers send resource descriptions to the matchmaking service; 2) A request is sent to the matchmaking service; 3) The matchmaking service executes a match- making algorithm and returns a set of ranked resources to the requester; 4) The requester chooses a resource from the set and contacts the corresponding resource provider.	86
3.2	Classads describing a laser jet printer (left) and a print job (right).	90
3.3	The hierarchical relationship among resource classes.	92
3.4	The Workstation, Cluster, MPP, and SMP classes.	93
3.5	Program, Library, and Package classes (top). Service, Data, and Storage classes (middle). CPU, Memory, Harddisk, NIC, and OS classes (bottom). .	94
3.6	The input and output of the matchmaking problem.	95

3.7	Examples of resource instances: a workstation and three clusters.	97
3.8	Boolean, arithmetic, and fuzzy requests.	98
3.9	BNF grammar for the resource.	99
3.10	BNF grammar for the request.	100
3.11	XML specification for a request.	102
3.12	A Boolean matching function.	103
3.13	An arithmetic matching function.	104
3.14	A fuzzy matching function.	105
3.15	The original matchmaking algorithm performs an exhaustive database search.	106
3.16	The modified matchmaking algorithm performs a restricted database search; it stops when the cardinality of a set of resources that match the request reaches $k * n$	107
3.17	Response time vs. number of resources for three requests of different com- plexities when the knowledge base is a local file.	108
3.18	Response time vs. number of resources for three requests when the knowledge base is a local file.	108
3.19	Response time vs. number of resources for three requests when the knowledge base is stored as a database.	109

3.20	Response time vs. number of resources for Request3 when the knowledge base is stored as a database. The matchmaking service runs the modified matchmaking algorithm.	110
3.21	Average matching degree vs. number of resources for Request3 when the knowledge base is stored as a database. The matchmaking service runs the modified matchmaking algorithm. The average matching degree is the average of the matching degrees of the resources returned by the service.	111
4.1	(a) Sub-linear, linear, and super-linear price functions. (b) The unit price varies with ρ , the load index of the provider.	123
4.2	(a) A sigmoid is used to model the utility function; a sigmoid includes three phases: the starting phase, the maturing phase, and the aging phase. (b) The satisfaction function for a sigmoid utility function and three linear price functions with low, medium, and high unit price.	127
4.3	The relationship between satisfaction s and the unit price ξ and amount of resources r . The satisfaction function is based on a sigmoid utility function and different price functions: (a) discourage consumption (super-linear); (b) linear; (c) encourage consumption (sub-linear); (d) a cut through the three surfaces at a constant ξ	130

4.4	The algorithm performed by the broker. The consumer request, req , is elastic. It contains the parameters describing u and s , the utility and satisfaction functions. τ is the target utility and σ is the satisficing size. The cardinality specifies the number of resource providers to be returned by the broker. . . .	135
4.5	Average hourly revenue vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).	141
4.6	Request acceptance ratio vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).	143
4.7	Average consumer satisfaction vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratio, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).	144
4.8	Average consumer utility vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies: linear (left), EDN (center), and EDL (right). . .	146
4.9	(a) The average hourly revenue, (b) the request acceptance ratio, (c) the average consumer satisfaction, and (d) the average consumer utility vs. time (in seconds) for $\sigma = 1$, $\tau = 0.9$, and $\eta = 1.0$, with different price functions. .	147

LIST OF PUBLICATIONS

Journal Articles

1. **Xin Bai**, Dan C. Marinescu, Ladislau Bölöni, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang, “**A Macro-Economic Model for Resource Allocation in Large-Scale Distributed Systems**”, (submitted).
2. Han Yu, **Xin Bai**, and Dan C. Marinescu, “**Workflow Management and Resource Discovery for an Intelligent Grid**”, in *Parallel Computing*, Volume 31, Issue 7, Pages 797-811, July 2005.
3. **Xin Bai**, Han Yu, Guoqiang Wang, Yongchang Ji, Gabriela M. Marinescu, Dan C. Marinescu, and Ladislau Bölöni, “**Coordination in Intelligent Grid Environments**”, in *Proceedings of the IEEE*, Volume 93, Issue 3, Pages 613-630, March 2005.
4. **Xin Bai**, Han Yu, Yongchang Ji, and Dan C. Marinescu, “**Resource Matching and a Matchmaking Service for an Intelligent Grid**”, in *International Journal of Computational Intelligence (IJCI)*, Volume 1, Number 3, Pages 197-205, 2004.

Book Chapters

1. **Xin Bai**, Han Yu, Guoqiang Wang, Yongchang Ji, Gabriela M. Marinescu, Dan C. Marinescu, and Ladislau Bölöni, “**Intelligent Grids**”, as a chapter of book “Grid Computing: Software Environments and Tools”, (Jose C. Cunha and Omer F. Rana, Eds.), Pages 45-74, Springer Verlag, ISBN: 1-85233-998-5, Heidelberg, 2006.
2. Ladislau Bölöni, Majid Ali Khan, **Xin Bai**, Guoqiang Wang, Yongchang Ji, and Dan C. Marinescu, “**Software Engineering Challenges for Mutable Agent Systems**”, as a chapter of book “Advances in Software Engineering for Multi-Agent Systems”, (C. Lucena, Al. Garcia, Al. Romanovsky, J. Castro, and P. Alencar, Eds.), Pages 149-167, Springer Verlag, Heidelberg, 2004.

Conference Papers

1. **Xin Bai**, Ladislau Bölöni, Dan C. Marinescu, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang, “**Pricing Strategies for Market-Oriented Grid Economics**”, (submitted).
2. **Xin Bai**, Ladislau Bölöni, Dan C. Marinescu, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang, “**Are Utility, Price, and Satisfaction Based Resource Allocation Models Suitable for Large-Scale Distributed Systems?**”, to appear

in Proceedings of the 3rd International Workshop on Grid Economics and Business Models (GECON 2006), Singapore, May 2006.

3. **Xin Bai**, Ladislau Bölöni, Dan C. Marinescu, Howard Jay Siegel, Rose A. Daley, and I-Jeng Wang, “**A Brokering Framework for Large-Scale Heterogeneous Systems**”, to appear in Proceedings of the 20th International Parallel and Distributed Processing Symposium (IPDPS 2006), Rhodes Island, Greece, April 2006.
4. **Xin Bai**, Han Yu, Yongchang Ji, and Dan C. Marinescu, “**Resource Matching and a Matchmaking Service for an Intelligent Grid**”, in Proceedings of the International Conference on Computational Intelligence (ICCI 2004), Pages 262-265, Istanbul, Turkey, December 2004.
5. Han Yu, **Xin Bai**, Guoqiang Wang, Yongchang Ji, and Dan C. Marinescu, “**Meta-information and Workflow Management for Solving Complex Problems in Grid Environments**”, in Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004), Santa Fe, New Mexico, April 2004.
6. Ladislau Bölöni, Majid Ali Khan, **Xin Bai**, Guoqiang Wang, Yongchang Ji, and Dan C. Marinescu, “**Software Engineering Challenges for Mutable Agent Systems**”, 2nd International Workshop on Software Engineering for Large-Scale Multi-Agent Systems (SELMAS 2003), Portland, Oregon, May 2003.

7. Dan C. Marinescu, Yongchang Ji, Gabriela M. Marinescu, and **Xin Bai**, “**Physical Awareness and Embedded Software Agents**”, Workshop on Ubiquitous Agents on Embedded, Wearable, and Mobile Devices, Bologna, Italy, July 2002.

CHAPTER 1

INTRODUCTION

In this dissertation, we present new models and algorithms for coordination, matchmaking, and resource allocation in large-scale distributed systems, analyze their performance, and discuss some of the services we have implemented. Coordination is necessary to automate complex applications and to shield the end-user from the complexity of the computing environment. A coordination service acts as a proxy on behalf of end-users to react to unforeseen events and to plan how to carry out complex tasks. Matchmaking is the process of selecting from a pool one or more objects that best match a set of requirements. In a large-scale distributed system, abundant computing resources provided by different entities are available. A matchmaking service allow users or agents on behalf of users to describe their needs and get a list of candidate resources ranked according to their matching degree to users' needs so that further decisions can be made. Last but not least, we are concerned with resource allocation models. In a large-scale distributed system, applications typically require resources from different domains; resource sharing requires cooperation between the administrative authorities in each domain. Resource providers and consumers are self-interested; providers wish to maximize their revenues while consumers want to obtain the maximum possible re-

sources for the minimum possible cost. Resource allocation models allows us to control the behavior of resource providers and consumers so that their interests are balanced.

The results presented in this dissertation are partially derived from several publications [9], [10], [11], [12], [13], [14], [141], and [142], which I coauthored with colleagues from UCF and elsewhere during my years as a Ph.D. student.

1.1 Motivation

In the past few years we have seen the emergence of large-scale distributed systems as a new paradigm of high performance computing. Grids are typical examples of large-scale distributed systems. Data, service, and computational grids, collectively known as information grids, are collections of autonomous computers connected to the Internet and giving to individual users the appearance of a single virtual machine [45] [82] [49].

A *data grid* allows a community of users to share content. An example of a specialized data grid supporting a relatively small user community is the one used to share data from high energy physics experiments. The World Wide Web can be viewed as a data grid populated with HTTP servers providing the content, data, audio, and video.

A *service grid* will support applications such as electronic commerce, sensor monitoring, telemedicine, distance learning, and Business-to-Business. Such applications require a wide spectrum of end services such as monitoring and tracking, remote control, maintenance and

repair, online data analysis and business support, as well as services involving some form of human intervention such as legal, accounting, and financial services. An application of a monitoring service in health care could be monitoring outpatients to ensure that they take the prescribed medication. Controlling the heating and cooling system in a home to minimize energy costs, periodically checking the critical parameters of the system, ordering parts such as air filters, and scheduling repairs is an example of control, maintenance, and repair services respectively. Data analysis services could be used when arrays of sensors monitor traffic patterns or document visitor's interest at an exhibition. There are qualitative differences between service and data grids. The requirements for a service grid are more stringent; the end result is often the product of a cooperative effort of a number of service providers, it involves a large number of sensors, and it is tailored to specific user needs. Individual users may wish to compose dynamically a subset of services. Dynamic service composition has no counterpart in the current Web where portals support static service coordination.

A *computational grid* is expected to provide transparent access to computing resources for applications requiring a substantial CPU cycle rate, very large memories, and secondary storage that cannot be provided by a single system. The **seti@home** project, set up to detect extraterrestrial intelligence, is an example of a distributed application designed to take advantage of unused cycles of PCs and workstations. Once a system joins the project, this application is activated by mechanisms similar to the ones for screen savers. The participating systems form a primitive computational grid structure; once a system is willing to accept work it contacts a load distribution service, it is assigned a specific task and starts comput-

ing. When interrupted by a local user, this task is checkpointed and migrated to the load distribution service. The requirements placed on the user access layer and societal services are even more stringent for a computational grid than for a service grid. The user access layer must support various programming models and the societal services of a computational grid must be able to handle low-level resource management.

There are many similarities between data, service and computational grids and it is highly desirable for the three to share as many standards, architectural concepts, and even components, as practical. It seems very unfortunate that for many years research in computational grids had a very loose connection with the mainstream efforts of the World Wide Web Consortium (W3C). Recently, a more rational approach is noticeable, e.g., the Globus project has embraced standards developed years ago, such as Web Services Definition Language (WSDL) and Simple Object Access Protocol (SOAP). WSDL is an XML format for describing network services as a set of endpoints operating on messages containing document- or procedure-oriented information. WSDL supports an abstract description of the operations and messages exchanged among endpoints. Both operations and messages are bound to a *concrete network protocol and message format* to define a *concrete endpoint*. One or more concrete endpoints are combined into *services* or abstract endpoints. WSDL description of the endpoints and their messages is independent of the message formats or network protocols used to communicate. SOAP is an XML-based application layer protocol developed as a standard by W3C. It is extensible, application- and platform-independent.

There are also important dissimilarities. For example, the service requests in a computational grid require a much finer granularity of resource allocation [84].

The defining characteristics of large-scale distributed systems are: (a) resource sharing among a large user population and (b) support for collaborative activities. In the context of a large-scale distributed system the term *resource* is used in a wide sense; it means hardware and software resources, services, and content. *Content* generally means some form of static or dynamic data or knowledge. *Autonomy* implies that the resources are in different domains and resource sharing requires cooperation between the administrative authorities in each domain.

Large-scale distributed systems inherit many of the traditional attributes of the Internet. Among the characteristics of large-scale distributed systems which distinguish them from the more traditional distributed systems of the past decades we note [49]:

- (i) Scale. A large-scale distributed system may consist of tens of thousands, or more nodes.
- (ii) Heterogeneity and diversity. Nodes with different processors and system architectures are expected to populate the system. The communication channels linking these nodes differ in term of latency and bandwidth. The operating systems (OS) of individual nodes may be different. The application software running on the nodes are very diverse, multiple versions of the same application software may be available.
- (iii) Autonomy of individual nodes. The nodes are in different administrative domains possibly with different access, security, and resource management policies [84].

- (iv) The dynamic and open-ended character. The system evolves in time; new resources are constantly added to the system, existing ones are modified, and others are retired.
- (v) The dominant service policy in the system is based upon a “best effort”. Enforcing end-to-end quality of service constraints is rarely possible.
- (vi) A large user population with individual and often conflicting objectives.
- (vii) User’s requirements may be dynamic, subject to change, or even cannot be known a priori.
- (viii) Complex, resource-intensive tasks submitted by individual users [83]. The complexity of a task is rather difficult to quantify. It has multiple facets: it may refer to the number and relationship of component activities, the predictability of the amount of resources needed for the completion of individual activities, the security constraints, the presence or absence of soft deadlines, the duration of individual activities, the diversity of resources used, and so on [84].

All these characteristics require us to develop new models and algorithms for coordination, matchmaking, resource discovery, resource allocation, scheduling, planning, and other functions for large-scale distributed systems. Although deterministic approaches for coordination, matchmaking, and resource allocation have been well studied, they are not suitable for large-scale distributed systems due to the large-scale, the autonomy, and the dynamics of the systems. We have to seek for nondeterministic solutions for large-scale distributed systems.

In this dissertation, we address the problem of coordination, matchmaking, and resource allocation for large-scale distributed systems. We present two services in large-scale distributed systems: the coordination service and the matchmaking service. The coordination service coordinates the execution of complex tasks in a dynamic environment and the matchmaking service supports finding the appropriate resources for users. We also present a macro-economic resource allocation model based upon utility, price, and satisfaction functions. In this model, a broker mediates resource providers who want to maximize their revenues and resource consumers who want to get the best resources at the lowest possible price, with some global objectives, e.g., to maximize the resource utilization of the system.

We choose to focus on these three aspects of large-scale distributed systems because they are critical for the future development of such systems and provide ample opportunities for new contributions. Coordination is necessary whenever we have to carry out complex tasks consisting of primitive activities related to each other. A *process description* shows the conceptual dependencies among the primitive activities and is augmented by a *case description* that reflects a particular instance of the execution of a process description. We are primarily concerned with dynamic coordination where the process description as well as the environment supporting the execution changes in time. Dynamic coordination algorithms have to respond to new user requests, to ensure fault-tolerance by providing alternative mechanisms to obtain the resources when resources are not available. Execution of a complex task can only be carried out successfully if the necessary resources are available. A large-scale distributed system is a resource-rich environment and matching the need of a consumer with

available resources is a non-trivial task. Matchmaking algorithms attempt to provide an optimal or near-optimal solution to the problem of selecting an appropriate system to carry out an atomic activity. Finally, we are concerned with high level resource management where computing and communication resources as well as consumers belong to different administrative domains. The resource providers try to maximize their revenues. The consumers want to obtain the maximum possible resources for the minimum possible price. A resource allocation model allows us to balance the interests of resource providers and consumers. The three problems we address are intimately related to each other.

1.2 Algorithms, Models, Implementation, and Performance Studies

We developed algorithms and models, implemented some of them as services, and evaluated their performance for the three problems addressed in this dissertation. Now we summarize our work in each of these three facets.

1.2.1 Algorithms and Models

Coordination of complex tasks requires algorithms capable to ensure a seamless transition from one activity to the next. Coordination algorithms need as input process and case

descriptions. A process description defines the data dependencies among the activities of a complex task and consists of end-user activities that correspond to individual computational tasks and flow control activities which ensure transition from one activity to the next. A case description provides additional information for a particular instance of the process the user wishes to perform. We designed an algorithm to coordinate the execution of a task and to supervise the execution of each activity of a task.

A matchmaking algorithm evaluates a request in the context of a particular resource. We introduced an ontology-based matchmaking solution, where resource advertisements from providers and resource requests from consumers are described through ontologies. Large numbers of resource advertisements are stored in knowledge bases. A resource request is to be evaluated in the context of a resource advertisement to determine how well they match. We designed two matchmaking algorithms: a simple algorithm that requires an exhaustive search of all resource advertisements; and a modified algorithm that only covers a portion of the knowledge base. We also introduced several types of matchmaking functions.

We proposed a macroeconomic model for resource allocation. We introduced a consumer utility function to represent the utility provided to an individual consumer. We also introduced price functions, and a consumer satisfaction function to quantify the level of satisfaction that depends on the utility and the price. In our three-party model, a broker performs a brokering algorithm. The broker is able to mediate resource providers who want to maximize their revenues and resource consumers who want to get the best resources at the

lowest possible price, with some global objectives, e.g., to maximize the resource utilization of the system.

1.2.2 The Implementation

We implemented a coordination service and a matchmaking service in BondGrid, an intelligent grid environment. In BondGrid, core services, including the coordination service and the matchmaking service, are provided by BondGrid agents.

An Intelligent Environment

Most of the research in grid computing is focused on relatively small grids (hundreds of nodes) dedicated to a rather restricted community (e.g., high energy physics), of well trained users (e.g., individuals working in computational sciences and engineering), with a rather narrow range of problems (e.g., computer aided-design for the aerospace industry).

The question we address is whether a general large-scale distributed system could respond to the needs of a more diverse user community than in the case of existing grids without having some level of intelligence built into the core services. The reasons we consider such systems are precisely the reasons computational grids were introduced in the first place: economy of scale and the ability to share expensive resources among larger groups of

users. It is not uncommon that several groups of users (e.g., researchers, product developers, individuals involved in marketing, educators, and students) need a seamless and controlled access to existing data or to the programs capable of producing data of interest. For example, the structural biology community working on the atomic structure determination of viruses, the pharmaceutical industry, and educational institutions ranging from high schools to universities, need to share information. One could easily imagine that a high school student would be more motivated to study biology if s(he) is able to replay in the virtual space successful experiments done at the top research laboratories, leading to the discovery of the structure of a virus, e.g., the common cold virus, and understand how a vaccine to prevent the common cold is engineered.

An intelligent environment is in a better position than a traditional one to match the user profile (leader of a research group, member of a research group with a well defined task, drug designer, individual involved in marketing, high school student, doctoral student) with the actions the user is allowed to perform and with the level of resources s(he) is allowed to consume. At the same time, an intelligent environment is in a better position to hide the complexity of the system and allow unsophisticated users such as a high school student without any training in computational science, to carry out a rather complex set of transformations of an input data set.

Even in the simple example discussed above we see that the *coordination service* acting as a proxy on behalf of the end-user has to deal with unexpected circumstances, or with error conditions, e.g., the failure of a node. The response to such an abnormal condition

can be very diverse, ranging from terminating the task, to restarting the entire computation from the very beginning, or from a checkpoint. Such decisions depend upon a fair number of parameters, e.g., the priority of the task, the cost of each option, the presence of a soft deadline, and so on. Even in this relatively simple case, it is non-trivial to hard code the decision making process into a procedure written in a standard programming language. Moreover, we may have in place different policies to deal with rare events, policies which take into account factors such as legal considerations, the identity of the parties involved, the time of the day, and so on. At the same time, hard coding the decision making will strip us of the option to change our actions depending upon considerations we did not originally take into account, such as the availability of a new system just connected to the grid.

Very often the computations carried out involve multiple iterations and in such a case the duration of an activity is data dependent and very difficult to predict. Scheduling a complex task whose activities have unpredictable execution times requires the ability to discover suitable resources available at the time when activities are ready to proceed. It also requires market-based scheduling algorithms which in turn require meta-information about the computational tasks and the resources necessary to carry out such tasks.

The more complex the system, the more elaborate the decision making process becomes, because we need to take into account more factors and circumstances. It seems obvious to us that under such circumstances a set of inference rules based upon facts reflecting the current status of various system components are preferable to hard coding. Oftentimes, we also need

to construct an elaborate plan to achieve our objective or to build learning algorithms into our systems.

Reluctantly as we may be to introduce AI components into a complex system such as a grid, we simply cannot ignore the benefits the AI components could bring along. Inference, planning, and learning algorithms are notoriously slow and cannot be used when faced with fast approaching deadlines. We should approach their use with caution.

The two main ingredients of an intelligent grid are software agents and ontologies. A *software agent* is a special type of reactive program. Some of the actions taken by the agent are in response to external events, other actions may be taken at the initiative of the agent. The defining attributes of a software agent are: autonomy, intelligence, and mobility. *Autonomy*, or agency, is determined by the nature of the interactions between the agent and the environment and by the interactions with other agents and/or the entities they represent. *Intelligence* is measured with the degree of reasoning, planning, and learning the agent is capable of. *Mobility* reflects the ability of an agent to migrate from one host to another in a network.

An agent may exhibit different degrees of autonomy, intelligence, and mobility. For example, an agent may have inferential abilities, but little or no learning and/or planning abilities. An agent may exhibit strong or weak mobility; in the first case, an agent may be able to migrate to any site at any time; in the second case, the migration time and sites are restricted.

Software agents have unique abilities to:

- (i) Support intelligent resource management. Peer agents can negotiate access to resources and request services based upon user intentions rather than specific implementations.
- (ii) Support intelligent user interfaces. We expect agents to be capable of composing basic actions into higher level ones, to be able to handle large search spaces, to schedule actions for future points in time, and to support abstractions and delegations. Some of the limitations of direct manipulation interfaces, namely, difficulties in handling large search spaces, rigidity, and the lack of improvement of behavior, extend to most other facets of traditional approaches to interoperability.
- (iii) Filter large amounts of information. Agents can be instructed at the level of goals and strategies to find solutions to unforeseen situations and they can use learning algorithms to improve their behavior.
- (iv) Adjust to the actual environment. Agents are network aware.
- (v) Move to the site when they are needed and thus reduce communication costs and improve performance.

Large-scale distributed systems seem ready to accept the need of metainformation to facilitate the interpretability of various components, so we believe that sooner rather than later we shall witness an effort to build intelligent environments.

The critical components of such an intelligent environment are the *ontologies*, collections of structured data shared among different agents. Core services in such systems are pro-

vided by intelligent agents, programs with the ability to perform intelligent actions such as inference, planning, and possibly learning.

The need for an intelligent infrastructure is amply justified by the complexity of both the problems we wish to solve and the characteristics of the environment [142]. As we take a closer look at the architecture of an intelligent environment we distinguish between several classes of services. System-wide services supporting coordinated and transparent access to resources of the system are called *societal* or *core* services. Specialized services accessed directly by end-users are called *end-user services*. The core services, provided by the computing infrastructure, are persistent and reliable, while end-user services could be transient in nature. The providers of end-user services may temporarily, or permanently, suspend their support. The reliability of end-user services cannot be guaranteed. The basic architecture of an intelligent environment is illustrated in Figure 1.1.

A non-exhaustive list of core services includes: authentication, brokerage, coordination, information, ontology, matchmaking, monitoring, planning, persistent storage, scheduling, event, and simulation. *Authentication* services contribute to the security of the environment. *Brokerage* services maintain information about classes of services offered by the environment, as well as past performance databases. Though the brokerage services make a best effort to maintain accurate information regarding the state of resources, such information may be obsolete. Up to date information about the status of any resource can be gathered using *monitoring* services. *Coordination* services act as proxies for the end-user. A coordination service receives a case description and controls the enactment of the workflow. *Planning* ser-

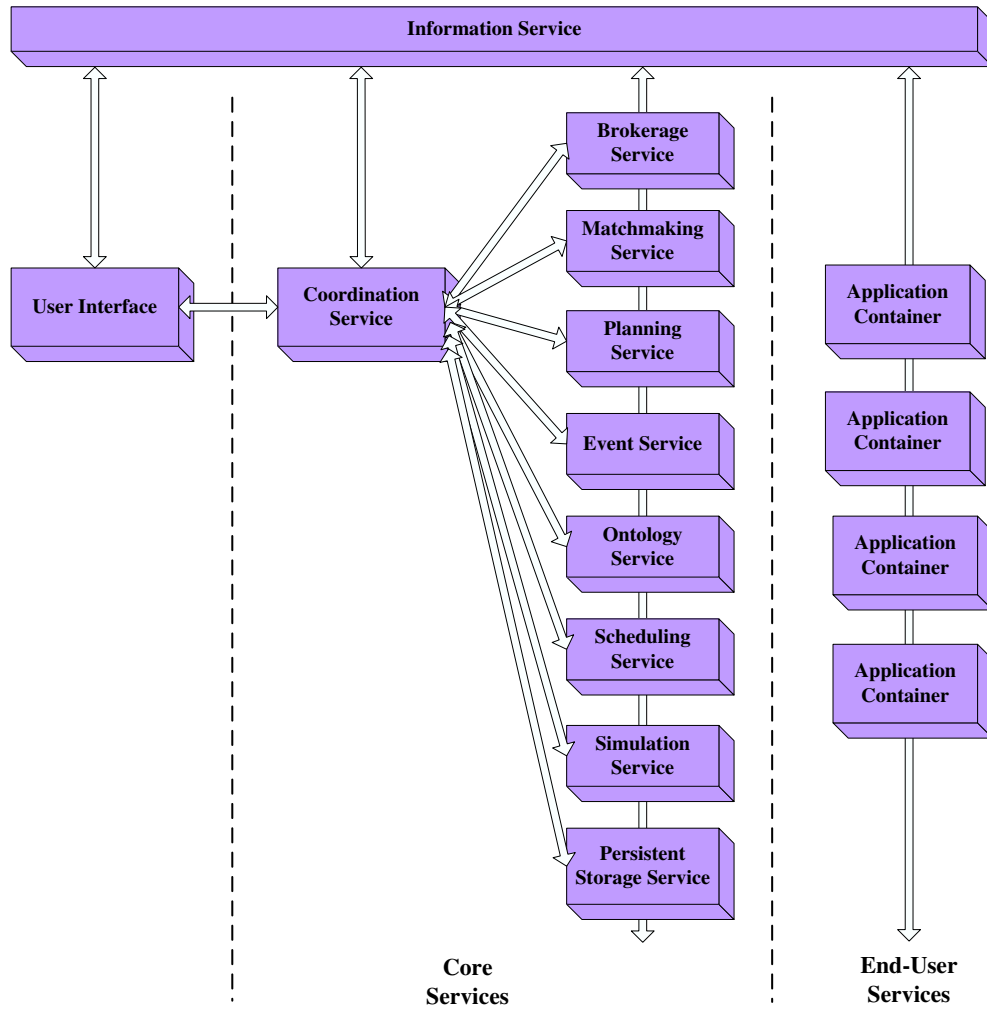


Figure 1.1: Core and end-user services. The User Interface (UI) provides access to the environment. Applications Containers (ACs) host end-user services. Shown are the following core services: Coordination Service (CS), Information Service (IS), Planning Service (PS), Matchmaking Service (MS), Brokerage Service (BS), Event Service (EvS), Ontology Service (OS), Simulation Service (SimS), Scheduling Service (SchS), and Persistent Storage Service (PSS).

vices are responsible for creating the workflow. *Scheduling* services provide optimal schedules for sites offering to host application containers for different end-user services. *Information* services play an important role; all end-user services register their offerings with the information services. *Ontology* services maintain and distribute *ontology shells*, i.e., ontologies with classes and slots but without instances, as well as ontologies populated with instances, global ontologies, and user-specific ontologies. *Matchmaking* services allow individual users represented by their proxies (coordination services) to locate resources in a spot market, subject to a wide range of conditions. Individual users may only be intermittently connected to the network. *Persistent storage* services provide access to the data needed for the execution of user tasks. *Event* services provide a method for event handling and message passing. *Simulation* services are necessary to study the scalability of the system and are also useful for end-users to simulate an experiment before actually conducting it.

Core services are replicated to ensure an adequate level of performance and reliability. Core services may be organized hierarchically in a manner similar to the DNS (Domain Name Services) in the Internet. End-user services could be transient in nature. The providers of such services may temporarily or permanently suspend their support, while most core services are guaranteed to be available at all times. Content-provider services, legal, accounting, tracking, various application software are examples of end-user services.

The BondGrid

BondGrid is an intelligent environment we are currently building. In the following we describe the structure of the ontologies used in BondGrid and the BondGrid agents.

a) Ontologies

An *ontology* is an explicit specification of a conceptualization and a conceptualization is an abstract and simplified view of the world that we wish to represent for some purpose. An ontology defines a common structure that facilitates the sharing of information. It includes machine-interpretable definition of the basic concepts in a domain and their relations. Ontologies are the cornerstone of interoperability. They represent the “glue” that allows different applications to use various grid resources. Creating ontologies in the context of grid computing represents a monumental task.

The structure of a category of entities is described as a *class* in Protégé knowledge base [54] [101]. Protégé is an open-source, Java-based tool that provides an extensible architecture for the creation of customized knowledge-based applications. Protégé uses *classes* to define the structure of entities. A class consists of one or more slots. A *slot* describes one attribute of the class and consists of a name and a value. An instantiation of a class is called an *instance* of that class. A class may have one or multiple instances. The type of a slot value may be simple types, such as integer, float, Boolean, and string. The type of a slot value may also be an instance of a class. A class may be extended from another class and inherit all the slots of that class.

Ontologies are stored in a knowledge base which can be accessed by applications. Applications also need to exchange ontologies with each other. BondGrid uses an XML format to describe instances of classes for exchange. Figure 1.2 shows an informal description of the XML format. Each instance has a unique ID. The **slot-value** can be a value, or an instance.

```
<?xml version="1.0" encoding="UTF-8"?>

<project project-name="projectname">
  <instances>
    <instance class-name="classname" ID="id">
      <slot slot-name="slotname">
        <value> slot-value </value>
        <value> ..... </value>
      </slot>

      <slot slot-name="slotname">
        .....
      </slot>
    </instance>

    <instance class-name="classname" ID="id">
      .....
    </instance>
  </instances>
</project>
```

Figure 1.2: XML specification for instances.

BondGrid also uses an XML format to describe various classes. The **slot-type** can be: string, boolean, float, integer, or an instance. The cardinality-value can be a nonnegative integer or a wild card '*'. Figure 1.3 shows an informal description of the XML format.

```

<?xml version="1.0" encoding="UTF-8"?>

<project project-name="projectname">
  <classes>
    <class class-name="classname">
      <slot slot-name="slotname">
        <type> slot-type </type>
        <cardinality> cardinality-value </cardinality>
      </slot>

      <slot slot-name="slotname">
        .....
      </slot>
    </class>

    <class class-name="classname">
      .....
    </class>
  </classes>
</project>

```

Figure 1.3: XML specification for classes.

b) BondGrid Agents

Services in BondGrid are provided by **BondGrid** agents based on **JADE** [63] and **Protégé** [54] [101], two free software packages distributed by Telecom Italy and the Stanford Medical Institute, respectively.

JADE (Java Agent DEvelopment Framework) is a FIPA compliant agent system fully implemented in Java and using FIPA ACL as an agent communication language. The **JADE** agent platform can be distributed across machines which may not run under the same OS. Each agent has a unique identifier obtained by the concatenation (+) of several strings:

$$AID \leftarrow agentname + @ + IPaddress/domainname + portnumber + /JADE$$

BondGrid uses a multi-plane state machine agent model similar to the Bond agent system [18]. Each plane represents an individual running thread and consists of a finite state machine. Each state of a finite state machine is associated with a strategy that defines a behavior. The agent structure is described using a Python-based agent description language called *blueprint*. A **BondGrid** agent is able to recognize a blueprint, to create planes and finite state machines accordingly, and to control the execution of different planes automatically. Figure 1.4 shows the blueprint for a coordination service.

```

openKnowledgeBase("kb/BondGrid.pprj","CS")

addPlane("Service Manager")
s=bondgrid.cs.ServiceManagerStrategy(agent)
addState(s,"Service Manager");

addPlane("Message Handler")
s=bondgrid.cs.MessageHandlerStrategy(agent)
addState(s,"Message Handler");

addPlane("Coordination Engine")
s=bondgrid.cs.CoordinationEngineStrategy(agent)
addState(s,"Coordination Engine");

```

Figure 1.4: The blueprint for a coordination service.

The knowledge bases are shared by multiple planes of an agent. The **BondGrid** agents provide a standard API to support concurrent access to the knowledge bases. Messages are constructed using the Agent Communication Language (ACL). A message has several fields: sender, receivers, keyword, and message content. The keyword enables the receiver of a message to understand the intention of the sender. A message may have one or more

user-defined parameters. Agents use XML formatted messages to exchange an instance of a class or the structure of a class.

The Coordination Service

We implemented a coordination service based on BondGrid agent. The coordination service consists of a message handler, a service manager, and a coordination engine. The message handler is responsible for the communication between the coordination service and other entities in the system. The service manager provides a GUI for monitoring the execution of tasks and the interactions between coordination service and other services. The coordination engine manages the execution of tasks submitted to the coordination service.

The Matchmaking Service

We implemented a matchmaking service based on BondGrid agent. The matchmaking service has a knowledge base that holds the resource advertisements from providers. Requests from consumers are evaluated with resource advertisements in the knowledge base. The matchmaking service consists of a message handler, a service manager, and a matchmaking engine. The message handler is responsible for the communication between the matchmaking service and other entities in the system. The service manager provides a GUI for monitoring the matchmaking engine and the interactions between matchmaking service and other services.

The matchmaking engine handles the matchmaking requests submitted to the matchmaking service.

1.2.3 Performance Studies

We studied the performance of a coordination service implemented in BondGrid. We investigated the response time the coordination service needs to encode, transmit, and decode an ontology (in XML format) as the size of the ontology increases. We also studied the maximum request handling rate of the coordination service. We tested the coordination service for an important application of biology computation, the 3D atomic structure determination of macromolecules based upon electron microscopy.

We investigated the performance of a matchmaking service implemented in BondGrid. We studied the response time of the matchmaking service versus the number of resource advertisements when the knowledge base is stored in a local file or a database. We also tested two matchmaking algorithms: a simple algorithm that requires an exhaustive search of all resource advertisements; and a modified algorithm that only covers a portion of the knowledge base.

We carried out intensive simulation studies of the macro-economic resource allocation model. We studied the effect of different pricing strategies upon measures of performance

important for the consumers (satisfaction and utility) and providers (revenue and acceptance ratios) with different parameters over time.

1.3 Contributions

In this dissertation, we address the problems of coordination, matchmaking, and resource allocation for large-scale distributed systems. We describe our work on a coordination service, a matchmaking service, and a macro-economic resource allocation model for large-scale distributed systems. Compared to traditional approaches, we have made the following contributions:

1. We developed a dynamic coordination algorithm to coordinate the execution of complex tasks in large-scale distributed systems; and we developed, implemented, and evaluated a coordination service based on the concept of process and case description.
2. We proposed an ontology-based resource matching scheme for large-scale distributed systems; and we developed, implemented, and evaluated an ontology-based matchmaking service.
3. We introduced the concept of consumer satisfaction that is based on the utility provided to the consumer and the price paid for the resources; we proposed a macro-economic resource allocation model based upon utility, price, and satisfaction functions for large-

scale distributed systems; and we evaluated the performance of the model with different pricing strategies through a set of simulations.

4. We developed and implemented BondGrid, an intelligent environment for large-scale distributed systems in which we implemented and evaluated a coordination service and a matchmaking service.

1.4 Organization

The rest of this dissertation is organized as follows. Chapter 2 discusses the existing coordination techniques, demonstrates a coordination service for large-scale distributed systems, and presents some performance evaluation results. Chapter 3 proposes an ontology-based resource matchmaking solution for large-scale distributed systems, demonstrates a matchmaking service, and shows its performance with different knowledge base storing methods and matchmaking algorithms. Chapter 4 demonstrates a macro-economic resource allocation model based upon utility, price, and satisfaction functions for large-scale distributed systems and shows its performance with different parameters and strategies through a set of simulations. Finally, Chapter 5 presents the summary of our work and the conclusions.

CHAPTER 2

COORDINATION

In this chapter, we study the problem of coordination for large-scale distributed systems. A large-scale distributed system is a complex system. The state space of a complex system is very large and it is unfeasible to create a rigid infrastructure implementing optimal policies and strategies that take into account the current state of the system. We need a coordination service to hide the complexity of the system from the end-user. The coordination service acts as a proxy on behalf of end-users to react to unforeseen events and to plan how to carry out complex tasks. It should be reliable and able to match user policies and constraints (e.g., cost, security, deadlines, quality of solution) with the corresponding system policies and constraints. We implemented a coordination service in BondGrid. The coordination service interacts with other core and end-user services for the execution of computing tasks. It implements an abstract machine that understands a description of the complex task, we call it a process description, and a description of a particular instance of the task, we call it a case description.

2.1 Introduction and Motivation

Whenever there is a contention for a limited set of resources among a group of entities or individuals, we need control mechanisms to mitigate access to system resources. These control mechanisms enable a number of desirable properties of the system, e.g., fairness, provide guarantees that tasks are eventually completed, and ensure timeliness when timing constraints are involved. Security is a major concern in such an environment. We want to ensure confidentiality of information and prevent denial of service attacks, while allowing controlled information sharing for cooperative activities. Considerably simpler versions of some of the problems mentioned above are encountered at the level of a single system, or in the case of small-scale distributed systems (systems with a relatively small number of nodes in a single administrative domain). In case of a single system such questions are addressed by the operating system which transforms the “bare hardware” into a user-machine and controls access to system resources. The question of how to address these problems in the context of a grid has been the main focus of research in grid environments, and, at the same time, the main stumbling block in the actual development of computational grids.

Some research in grid computing proposes to transfer to grid computing concepts, services, and mechanisms from traditional operating systems, or from parallel and distributed systems without taking into account the effect on system reliability and dependability of the attributes of computational grids. This is a clearly inadequate approach. For example, there is a proposal to extend the Message Passing Interface (MPI) to a grid environment. In its

current implementation, the MPI does not have any mechanism to deal with a node failure during a barrier synchronization operation. In such a case all the nodes involved other than the defective one wait indefinitely, and it is the responsibility of the user to detect the failure and take corrective actions. It may be acceptable to expect the programmer to monitor a cluster with a few hundred nodes housed in the next room, but it is not reasonable to expect someone to monitor tens of thousands of nodes scattered over a large geographic area. Thus, we cannot allow MPI to work across system boundaries without any fault detection mechanism.

Coordination allows individual components of a system to work together and create an ensemble exhibiting a new behavior without introducing a new state at the level of individual components. Scripting languages provide a “glue” to support composition of existing applications. The problem of coordinating concurrent tasks was generally left to the developers of the parallel scientific and engineering applications. Coordination models such as the coordinator-worker, or the widely used Single Program Multiple Data (SPMD) were developed in that context.

The problem of coordination of complex tasks has new twists in the context of large-scale distributed systems. First, it is more complex and it involves additional activities such as resource discovery and planning. Second, it has a much broader scope due to the scale of the system. Third, the complexity of the computational tasks and the fact that the end-user may only be intermittently connected to the network force us to delegate this function to a proxy capable of creating the conditions for the completion of the task with or without user

intervention. It is abundantly clear that such a proxy is faced with very delicate decisions regarding resource allocation or exception handling. For example, should we use a more expensive resource and pay more to have guarantees that a task completes in time, or should we take our chances with a less expensive resource? In the case of the MPI example should we kill all the processes in all the nodes and restart the entire computation, should we roll back the computation to a previous checkpoint if one exists, or should we simply restart the process at the failing node on a different node.

There is little doubt that the development of large-scale distributed systems poses formidable problems. We concentrate on problems related to resource management, exception handling, and coordination of complex tasks.

2.2 Background and Related Work

2.2.1 Grid Computing

Grid computing has several names along with its evolution, such as metacomputing, scalable computing, global computing, Internet computing, and now we call it grid computing. The earliest effort to grid computing could be traced back to the CASA project that was one of the US gigabit test beds deployed in 1989. Since then the evolution of grid computing could be identified as three stages [106]: the first generation, the second generation, and the third

generation, although there are not always clear boundaries among them. In the following we survey some typical grid computing projects based on [106].

The projects of the first generation of grid computing focused on providing computational resources to high performance applications. Among them, FAFNER (Factoring via Network-Enabled Recursion) [42] and I-WAY (the Information Wide Area Year) [43] are two representative ones. The initiative of FAFNER project is based on the fact that factoring extremely large numbers for the widely used RSA (Rivest, Shamir, and Adelman) public encryption algorithm is computationally very expensive. FAFNER uses parallel factoring algorithms so that the computation of factoring can be distributed among computers. A person who would like to contribute to this project can simply invoke a CGI script anonymously through a set of Web pages. His Web client uses HTTP protocol to GET parameters and POST results back to the server. His machine is not necessarily a high performance computer. I-WAY links lots of high performance computer and devices located in 17 US sites through high bandwidth networks. Key sites install I-POP (point-of-presence) servers to be a gateway of I-WAY. An I-POP server supports authentication, resource reservation, process creation, and communication functions. CRB (Computational Resource Broker) works as a resource scheduler. There is a single central CRB and each I-POP server has a local CRB. The central CRB maintains the job queues and information of local machines. Both FAFNER and I-Way work as a metacomputing environment by integrating computational resources although FAFNER could use a slow computer and I-WAY needs high performance

computers. Both of them lacks scalability. For example, FAFNER needs lots of human intervention and I-WAY was limited by the components of I-POPs.

The second generation of grid computing tries to solve three main issues [106]: 1) Heterogeneity in that computing resources could be heterogenous in nature and could belong to different domains; 2) Scalability in that the size of the grid could be millions; 3) Adaptability in that due to the size of the grid, the probability of failure could be high. The features of the second generation of grid includes: 1) Administrative hierarchy in that it decides how components of the grid deal with each other; 2) Communication services in that diverse communication modes exist and QoS should be considered; 3) Information services in that a grid is dynamic and grid should provide mechanism for registering, unregistering, and obtaining grid information; 4) Naming services in that it provides a uniform name space across the whole grid; 5) Distributed file systems and caching in that data could be stored distributedly; 6) Security and authentication in that confidentiality, integrity, authentication, and accountability are necessary for any distributed system; 7) System status and fault tolerance in that it is important to monitor the status of resources and application so that fault can be detected and handled; 8) Resource management and scheduling in that these should be designed carefully so that resource can be utilized efficiently and effectively; 9) User and administrative GUI in that they are necessary for controlling the grid resource and user tasks. The second generation grid computing involves a lot of projects dealing with areas such as infrastructure, key services, coordinations, specific applications, and domain portals. Globus [44] is US multi-institutional computational grid project that aims to provide a software in-

frastructure making distributed heterogeneous resource a single virtual machine. It evolved from the I-Way [43] project. It provides a Globus Toolkit that consists of basic components such as security, resource location, resource management, and communication. Globus infrastructure is a layered architecture. The Globus Toolkit includes: 1) An HTTP-based GRAM (Globus Toolkit Resource Allocation Manager) protocol to allocate, monitor, and control resources; 2) An GridFTP file transfer protocol, which is supports security, partial files access, and parallel transferring; 3) GSI (Grid Security Infrastructure) for authentication and authorization; 4) LDAP (Lightweight Directory Access Protocol) used to access distributed structure and state information; 5) GASS (Global Access to Secondary Storage) used to access data via sequential and parallel interface remotely; 6) GEM (Globus Executable Management) to locate and cache executables; 7) GARA (Globus Advanced Reservation and Allocation) used for resource reservation and allocation. Legion [53] is an object-based software infrastructure which makes heterogeneous and geographically distributed high performance computers interact seamlessly. It encapsulates all its components as objects and has the advantages of object mechanism: data abstraction, encapsulation, inheritance, and polymorphism. The core object types of Legion include classes representing managers or policy makers, metaclasses that is classes of classes, host objects representing computational resources, vault objects representing storages, binding agents mapping object IDs to physical addresses, etc. Jini [67] aims to provide a software infrastructure supporting network plug and play. Services and users use Jini protocols to constitute a Jini community that is generally a workgroup. Applications are written in Java or wrapped in Java if non-Java code is

necessary. Jini users can find a community to join, to look up for the service he wants, lease the service for a period, receive and handle remote events, and use transactions to make sure the system state is consistent. Condor [34] [47] aims to provide a software package supporting batch jobs on UNIX computers. In Condor, resource location and allocation are automatic. Jobs can be check-pointed and migrated to new site for execution. Idle resources enter a pool waiting for allocation and leave the pool when they are busy. Nimrod-G [90] [21] is a grid resource manager and scheduler. It uses Globus to deploy Nimrod-G agents on remote resources. It is capable of allocating resources based on the capability, cost, and availability of resources and the time deadline of users. Different algorithms can be used to achieve cost optimization, time optimization, and cost-time optimization. UNICORE [1] provides an open architecture and a uniform user GUI through Java and Web. Applications in UNICORE have multiple parts so that different parts can be executed at different sites, either parallelly or sequentially. Through web pages, users can create, submit, and control job execution from anywhere. Users are also authenticated through Web pages. P2P computing [33] aims to making computers on Internet to share resources, such as spare CPU cycles and storage space. In P2P computing, there is no central server and computers communicate with each other directly. P2P computing scales much more effectively than the traditional client-server model. The most important obstacle in P2P computing is the security. Since P2P computing allows to use resources of other machines, computer systems are vulnerable to viruses and malicious activities. NMI (NSF Middleware Initiative) [92] aims to define, develop, deploy, and support an integrated and stable middleware infrastructure

created from a number of open source grid framework, including Globus, Condor-G, and so on. The second generation Grid computing projects focus on developing middleware to support large-scale data and computation. The major drawback of the second generation grid computing is the lack of transparency among different middleware, i.e., different grid environments use noninteroperable solutions.

The third generation of grid computing aims to reuse existing standards in a flexible manner. Two key characteristics of the third generation grid computing include the adoptions of a service-oriented model and metadata. The Web service standards from W3C (World Wide Web Consortium) [128] have gained popularity. The established standards include: 1) SOAP (Simple Object Access Protocol) that provides an envelope encapsulating XML data; 2) WSDL (Web Service Description Language) that describes a service in XML; and 3) UDDI (Universal Description Discovery and Integration) that is a specification for Web services registries. The OGSA (Open Grid Services Architecture) [95] from GGF (Global Grid Forum) [49] combine the Web Services standards and grid computing. OGSA supports the creation and maintenance of services maintained by a virtual organization. Services could be computers, storages, networks, data, and executable programs. Standard interfaces defined in OGSA include: 1)Discovery so that service can be found; 2)Dynamic service creation so that service can be created; 3)Lifetime management so that state information of services can be monitored and failure can be handled; 4)Notification so that events can be captured; 5)Manageability so that services can be managed; and 6)Simple hosting environment so that local resources and native facilities can be managed. The Semantic Web [109] of W3C defines

a standard of defining Web matadata. The most important technologies of Semantic Web include XML and RDF (Resource Description Framework). XML allows us to structure documents without saying anything about the meaning of documents. Meaning of documents are expressed in RDF. The Semantic Grid [108] is a natural evolution of the Semantic Web and grid computing and is an effort to utilize the Semantic Web technologies in grid computing. In Sematic Grid, computing is knowledge-centric and metadata-driven. With the increasing complexity of dealing with grid computing issues such as task coordination and resource matchmaking, grids need a significant amount of autonomic functions, known as Autonomic Computing [7]. The key autonomic functions of a grid include: 1)Self-configuring so that grid components are able to adapt to the changes in the system; 2) Self-optimizing so that grid components are able to achieve optimal performance; 3)Self-healing so that grid components are able to recover from failure; and 4)Self-protecting so that grid components are able to detect and defense intrusions.

2.2.2 Agent-Based Computing

An agent is defined as “an encapsulated computer system that is situated in some environment, and that is capable of flexible, autonomous action in that environment in order to meet its design objectives” in [137]. An agent has the following characteristics [64]:

1. Its goal is to solve a clearly identifiable problem and it has well-defined interfaces and boundaries.
2. It is situated in an environment. It can sense the change of the environment and act on the environment accordingly.
3. It is designed to fulfill a specific role with its particular problem solving capability.
4. It has control over its internal states and own behaviors.
5. It is both reactive, i.e., able to respond in a timely fashion to the change of the environment, and proactive, i.e., able to act on its own initiative to reach its goal.

Multiple agents are suitable for representing a decentralized system with multiple loci of control, multiple perspectives, or competing interests [46]. A system may contain both cooperative agents to maximize the social welfare of the system and selfish agents to maximize their own individual return. The interactions among agents are based on semantic understanding and sophisticated social activities to cooperate, coordinate, and negotiate. Intelligent agents are with those autonomic functions required by the third generation grid computing. Agent technologies have been used for the development of distributed software systems for several years [65]. The agent-based system allows agents to control themselves with their own internal logic rather than a fixed function or external intervention. The aforementioned autonomic functions are close to the weak agency [138] property of agents. The weak agency is composed of the following: 1) Autonomy so that agents can work without human intervention; 2) Social ability so that agents communicate with each other through an

ACL (agent communication language); 3)Reactivity so that agents can perceive and respond to the change of their environment; and 4)Pro-activeness so that agents exhibit goal-directed behavior. Agent-based computing [64] opened a new window to grid computing. The research about agent-based grid computing involves a lot of areas including infrastructure, key services, algorithms, simulations, etc.

In [65], the authors point out that software agents are recognized as a powerful high-level abstraction for the modeling of complex software systems. In [81], different perspectives of grid computing was presented and the concept of Agent Grid was proposed with the DARPA ISO's CoABS (Control of Agent-Based Systems) agent grid. They proposed AgentScape, a scalable multi-agent infrastructure to support large-scale agent systems. In [103], an agent-based grid computing project is presented. It integrates services and resources for establishing multi-disciplinary problem solving environment. Autonomous agents are able to adjust their behavior according to their interactions with other agents and the changes of the environment. In [69], a distributed resource discovery method used in a wide-area distributed system made up of autonomous systems is presented. This method allows individual nodes to gather information about resources in the system. When an agent needs to find the detailed information of a component in another system, a description of a monitoring agent is sent to the remote site and a monitoring agent is created accordingly. In [122], an agent-based architecture enabling routing and handling of FIPA ACL messages was proposed. The architecture is pluggable with respect to routing algorithms. In [111], different adaptive negotiation strategies for agent-based load balancing and grid computing are listed: Contract Net Proto-

col, Auction Model, Game Theory Based Model, and Discrete Optimal Control Model. The authors propose a system in which different negotiation strategies can be selected automatically to adapt to computation needs and changes of the resource environment. The authors of [28] proposed to use a hierarchy of homogeneous agents for the resource management in a metacomputing environment. At metacomputing level, the resource management, scheduling, and allocation are abstracted to service advertisement and service discovery fulfilled by agents. Then In [25], the authors proposed ARMS, an agent-based resource management system in grid computing. In ARMS, a hierarchy of homogeneous agents is used to provide a scalable and adaptable abstraction of the system architecture at meta-level. Each agent can cooperate with others to advertise itself and discovery resources so that the tasks can be scheduled to utilize grid resources. In [24], ARMSim, an ARMS performance modeling and simulation environment, was proposed to investigate the performance of ARMS. The resource discovery performance of the grid with different strategies is presented in [27]. In [71], a monitoring and discovery service in an agent-based grid environment was presented. The monitoring and discovery service implements GRIP (GRid Information Protocol) and GRRP (GRid Registration Protocol). In [29], a performance-driven task scheduler used in an agent-based grid environment for local grid load balancing was proposed. Agents uses predictive performance data and cooperate with each other to balance the load of the grid. In [97], an agent-based approach is proposed to manage vast amount of resources in a grid. It also proposed an agent infrastructure to be integrated within the grid middleware layer. This infrastructure supports scalability of number of agents and resources. In [130], a dy-

namic model of agent-based load balancing on grids was proposed. The quality of the load balancing of the grid with different strategies was also defined and measured. In [75], an architecture for computational grids based on a multi-agent system paradigm in a cellular network was proposed. In this grid environment, agents are able to use each other's resource to complete computational tasks cooperatively. The authors also presented a distributed version of IDA* search algorithm used in the aforementioned computational grids in [74]. In [66], the authors proposed an agent-based architecture to solve the problem of software maintenance in grids. Their architecture uses a tool called Remote Maintenance Shell and exploits mobile agents to perform software upgrading without jeopardizing the running applications. In [100], the authors presented an extension of the JADE agent framework to support the development of agent-based grid systems. In [32], ACG, an Agent-based Computational Grid environment, was presented to provide a uniform high-level management of computing resources and services in the grid. Users use a consistent and transparent interface to access grid resources and services. All entities in the grid, including resources, services, and users are represented as agents. In [121], Unity, a decentralized architecture for autonomic computing based on multiple interacting agents, was presented. Agents in Unity have the property of goal-driven, self-assembly, self-healing, and real-time self-optimization. An agent within each application environment computes a resource utility according to a utility function. All utility functions are sent to a Resource Arbiter agent to compute a globally optimal resource allocation of the grid.

2.2.3 Workflow Management

Workflow management plays a very important role in task coordination. A workflow management system (WFMS) is defined as a generic software tool which allows for the definition, execution, registration, and control of workflows in [77]. Some popular commercial workflow systems include Staffware [115], COSA [36], Inconcert [62], Eastman Software [39], Domino Workflow [38], Websphere MQ Workflow [131], Visual Workflow [127], and I-Flow [60]. They are proprietary softwares and it is infeasible to use them for open source research projects.

Workflow have been a research topic in different fields for many years. A method to evaluate the capabilities of different workflow management systems through a meta model approach is introduced in [143] so that the most appropriate system for users can be selected. An agent-enhanced workflow [68] is introduced to improve control of a workflow management system through a community of intelligent, distributed, and autonomous software agents. These agents cooperate with each other to perform realtime exception handling and dynamic distribution of tasks. An open architecture for adaptive workflow management systems is introduced in [112]. This architecture uses a flexible workflow model including advanced control structures and allows users to modify workflow instances during runtime. The issue of time constraints in workflow systems is addressed in [40]. A framework is presented to plan the workflow process in time, estimate the workflow execution duration, avoid deadline violations, and satisfy external time constraints such as fixed date and upper/lower bound of activity execution time. The idea of integrating the concept of workflow transaction into

workflow management systems is addressed in [40] to ensure the consistent and reliable execution of workflows. Different failure sources and failure classes that influence and determine recovery concepts are listed. WebFlow, an environment that supports distributed coordination services on the World Wide Web, is introduced in [52]. WebFlow contains a distributed workflow component. WebFlow leverages the HTTP protocol and consists of a number of tools for the development of applications that require the coordination of multiple distributed servers. An event-based distributed execution of workflow is introduced in [48]. A layered event-based architecture is used for workflow systems. Its functionality includes event registration, detection, management, and notification to distributed, autonomous, and reactive software components. An agent-based process management system is introduced in [94] to combine the autonomous agent technology and distributed computing platforms. It allows software agents to represent the interests of autonomous departments or business units to adapt to the change of environment. It extends workflow with the abilities to anticipate the process requirements, to process resources automatically, and to adapt to exceptions. A rule-based approach for coordination in workflow management systems is introduced in [70]. This approach uses a mechanism in terms of Event/Condition/Action (ECA) rules used in the active object-oriented database systems to map various coordination policies required within different areas of workflow management systems to ECA rules.

Some research has been done in the grid community to provide a platform independent, robust, and convenient mechanism for workflow systems. Pioneering work includes WebFlow [16] that aims to the development of high performance distributed computing applications.

Symphony, a Java-based composition and manipulation framework for computational grids, was presented in [80]. It aims to combine existing codes to meta-programs without changing codes. Grid workflow system (Grid-WFS) is introduced in [59] to support flexible failure handling of workflows in a generic, heterogeneous, and dynamic grid environment. Available handling techniques for a task-level failure include retrying, replication, and checkpointing. Available handling techniques for a workflow-level failure include alternative task, workflow-level redundancy, and user-defined exception handling. The problem of generating job workflows for a grid automatically is addressed in [37]. AI planning technologies are used to develop two workflow generators: 1) Concrete Workflow Generator (CWG) that maps an abstract workflow to the available resources of the grid; and 2) Abstract and Concrete Workflow Generator (ACWG) that not only maps an abstract workflow to the available resources of the grid but also constructs the abstract workflow based on available resources. GridFlow, a workflow management system for grid computing, is presented in [26]. GridFlow is based on the prediction capabilities provided by the PACE system [93]. It includes a user portal and services of both global grid workflow management and local grid sub-workflow scheduling through applying a fuzzy timing method. GridAnt, a client-controllable grid workflow system, is introduced in [2]. GridAnt reuses Ant [6], a popular build tool in the Java community. GridAnt assists grid users in orchestrating a set of grid activities and expressing complex dependencies between them in XML. The problem of dynamic change within workflow systems is addressed in [41]. A Petri net formalism is used to analyze the structural

change of workflows. As an example, this Petri net formalism is used to prove that a class of synthetic cut-over change maintains correctness when downsizing occurs.

A lot of work has been done with defining and implementing standards for workflow management systems [77]. An XML based grid workflow specification is documented in [17] and used in the Accelerated Strategic Computing Initiative (ASCI). Workflow Management Coalition (WfMC) standard [132] and Workflow Process Definition Language (WPDL) [88] are sophisticated and too generalized for grid computing. Condor [34] and UNICORE [1] provide similar functionalities but require specific infrastructure. Since grid architecture began to shift to service-oriented framework [95], new specifications are being researched: Business Process Execution Language for Web Services (BPEL4WS) [20] and Grid Services Flow Language (GSFL) [72]. BPEL4WS represents the merger of two rival standards, WSFL (Web Services Flow Language) [139] and XLang, a language used to model business processes as autonomous agents [140]. GSFL is proposed to support a number of desirable features: peer-to-peer service interaction and complicated lifecycle management for the services.

2.3 Coordination and Coordination Services

A cursory examination of recent research in grid computing services reveals limited interest in the mechanisms used to coordinate the execution of complex computational tasks. This situation can be attributed to several reasons: (i) there are relatively few applications of grid computing; (ii) the users of grid computing are highly computationally sophisticated

individuals with a high threshold for pain in using computer systems; (iii) powerful scripting languages such as Perl, Python, and tuple-spaces can be used for coordination [105]; and last but not least, (iv) the problems posed by a coordination service are non-trivial.

2.3.1 Process Coordination

Coordination is a very broad subject with applications to virtually all areas of science and engineering, management, social systems, defense systems, education, health care, and so on. Human life is an exercise in coordination, each individual has to coordinate his own activities with the activities of other individuals, groups of individuals have to coordinate their efforts to achieve a meaningful result.

Coordination is critical for the design and engineering of new man-made systems and important for understanding the behavior of existing ones [96]. Coordination is an important dimension of computing. An algorithm describes the flow of control, the flow of data, or both; a program implementing the algorithm coordinates the software and the hardware components involved in a computation. The software components are library modules interspaced with user code executed by a single thread of control in case of sequential computations; in this case, the hardware is controlled through system calls supported by the operating system running on the target hardware platform.

Coordination of distributed and/or concurrent computations is more complex; it involves software components based on higher level abstractions such as objects, agents, and programs as well as multiple communication and computing systems. Figure 2.1 illustrates a three-dimensional space for coordination models. The first dimension reflects the type of the network, i.e., an interconnection network of a parallel system, a local area network, or a wide area network; the second dimension describes the type of coordination, i.e., centralized, or distributed; the third dimension reflects the character of the system, i.e., closed, or open.

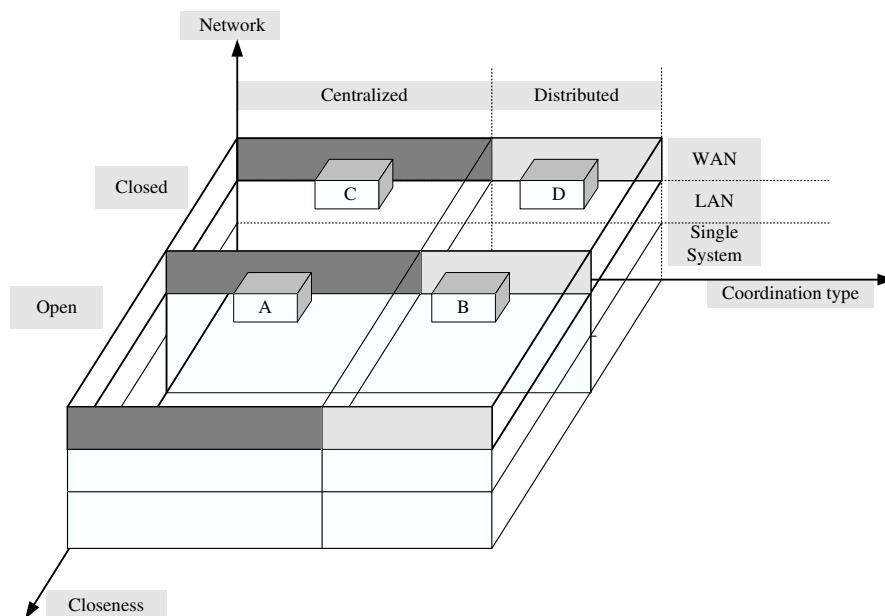


Figure 2.1: The coordination can be centralized, or distributed; the components may be confined to a single system, to a LAN, or to a WAN; the system may be open, or closed.

A computer network provides the communication substrate and its characteristics provide the first dimension of a coordination space. The individual entities can be co-located in space

within a single system. They can be distributed over a LAN, or over a WAN. Coordination in a WAN is a more difficult problem than coordination confined to a LAN or to a single system; we have to deal with multiple administrative domains and in theory communication delays are unbounded. In a WAN it is more difficult to address performance, security, or quality of service issues.

There are two approaches to coordination, a centralized and a distributed one. Centralized coordination is suitable in some instances such as *ad hoc service composition*. Suppose that one user needs a super service involving several services; in this case an agent acting on behalf of the user coordinates the composition. In other cases, a distributed coordination approach has distinct benefits. Consider, for example, a complex weather service with a very large number of sensors, of the order of millions, gathering weather-related data. The system uses information from many databases; some contain weather data collected over the years, others archive weather models. The system generates short-, medium-, and long-term forecasts. Different functions of this service such as data acquisition, data analysis, data management, and weather information dissemination will most likely be coordinated in a distributed fashion. A hierarchy of coordination centers will be responsible for data collected from satellites, another group will coordinate terrestrial weather stations, and yet another set of centers will manage data collected by vessels and sensors from the oceans.

The last dimension of interest of the coordination space reflects whether the system is closed and all entities involved are known at the time when the coordination activity is initiated, or the system is open and allows new entities to join or leave at will. Coordination

in an open system is more difficult than coordination in a closed system. Error recovery and fault tolerance become a major concern because a component may suddenly fail or leave the system without prior notice. The dynamics of coordination changes; we cannot stick to a pre-computed coordination plan and we may have to revise it. The state of the system must be reevaluated frequently to decide if a better solution involving components that have recently joined the system exists.

2.3.2 Coordination Techniques

We distinguish between low-level and high-level coordination issues. Low-level coordination issues are centered on the delivery of coordination information to the entities involved; high-level coordination covers the mechanisms and techniques leading to coordination decisions.

The more traditional distributed systems are based on *direct communication models* like the one supported by remote procedure call protocols to implement the client-server paradigm. A client connected to multiple servers is an example of a simple coordination configuration; the client may access services successively, or a service may in turn invoke additional services.

In this model there is a direct coupling between interacting entities in terms of name, place, and time. To access a service, a client needs to know the name of the service and the location of the service; the interaction spans a certain time interval.

Mediated coordination models ease some of the restrictions of the direct model by allowing an intermediary, e.g., a directory service to locate a server, an event service to support asynchronous execution, an interface repository to discover the interface supported by the remote service, brokerage and matchmaking services to determine the best match between a client and a set of servers, and so on.

Coordination Based on Scripting Languages. Coordination is one application of a more general process called software composition where individual components are made to work together and create an ensemble exhibiting a new behavior, without introducing a new state at the level of individual components. A component is a black box exposing a number of interfaces allowing other components to interact with it.

The components or entities can be “glued” together with scripts. Scripting languages provide “late gluing” of existing components. Several scripting languages are very popular: `Tcl`, `Perl`, `Python`, `JavaScript`, `AppleScript`, `Visual Basic`, languages supported by the `csh` or the `Bourne Unix` shells,

Scripting languages share several characteristics [107]:

(i) Support composition of existing applications; thus, the term “late gluing”. For example, we may glue together a computer-aided design system (CAD), with a database for material properties (MPDB). The first component may be used to design different mechanical parts; then the second may be invoked to select for each part the materials with desirable mechanical, thermal, and electrical properties.

(ii) Rely on a virtual machine to execute bytecode `Tcl`, or interpreted languages. `Perl`, `Python`, and `Visual Basic` are based on a bytecode implementation, whereas `JavaScript`, `AppleScript`, and the `Bourne Shell` need an interpreter.

(iii) Favor rapid prototyping over performance. In the previous example in (i), one is likely to get better performance in terms of response time by rewriting and integrating the two software systems, but this endeavor may require several men-years; writing a script to glue the two legacy applications together could be done in days.

(iv) Allow the extension of a model with new abstractions. For example, if one of the components is a CAD tool producing detailed drawings and specifications of the parts of an airplane engine, then the abstractions correspond to the airplane parts, e.g., wing, tail section, landing gear. Such high-level, domain-specific abstractions can be easily understood and manipulated by aeronautic and mechanical engineers with little or no computer science background.

(v) Generally, scripting languages are weakly typed, offer support for introspection and reflection, and for automatic memory management.

`Perl`, `Python`, `JavaScript`, `AppleScript`, and `Visual Basic` are object-based scripting languages. All five of them are embeddable and can be included into existing applications. For example, code written in `Jython`, a Java version of `Python`, can be embedded into a data stream, sent over the network, and executed by an interpreter at the other site.

Perl, Python, and JavaScript support introspection and reflection. Introspection and reflection allow a user to determine and modify the properties of an object at runtime.

Scripting languages are very popular and widely available. Tcl, Perl, and Python are available on most platforms, JavaScript is supported by all popular web browsers, the Bourne Shell is supported by Unix, and Visual Basic by Windows.

Script-based coordination has obvious limitations. It is most suitable for applications with one coordinator acting as an enactment engine, or in a hierarchical scheme when the legacy applications form the leaves of the tree and the intermediate nodes are scripts controlling the applications in a subtree. A script for a dynamic system, where the current state of the environment determines the course of action, becomes quickly very complex. Building some form of fault tolerance and handling exceptions could be very tedious.

In summary, script-based coordination is suitable for simple, static cases and has the advantage of rapid prototyping but could be very tedious and inefficient for more complex situations.

Coordination Based on Shared-Data Spaces. Tuple space coordination has also been intensely scrutinized [30], [105]. A shared-data space allows agents to coordinate their activities. We use the terminology *shared-data space* because of its widespread acceptance, though in practice the shared space may consist of data, knowledge, code, or a combination of them. The term *agent* means a party to a coordination effort.

In this coordination model all agents know the location of a shared data space and have access to communication primitives to deposit and to retrieve information from it. As in virtually all other coordination models, a prior agreement regarding the syntax and the semantics of communication must be in place before meaningful exchanges of coordination information may take place.

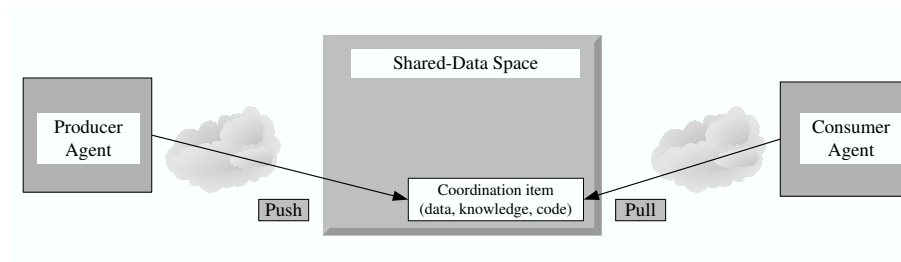


Figure 2.2: A shared-data space coordination model. The producer of the coordination information pushes an item into the shared data space, a consumer pulls it out. Little, or no state information needs to be maintained by the shared-data space. The model supports asynchronous communication between mobile agents. The agents may join and leave at will, the model supports open systems.

The shared-data space coordination model allows *asynchronous communication* between mobile agents in an open system, as seen in Figure 2.2. The communicating components need not be coupled in time or space. The producer and the consumer of a coordination information item act according to their own timing; the producer agent may deposit a message at its own convenience and the consumer agent may attempt to retrieve it according to its own timing. The components need not be co-located; they may even be mobile. The

only constraint is for each agent to be able to access the shared-data space from its current location. Agents may join and leave the system at will.

Another distinctive advantage of the shared-data space coordination model is its tolerance of heterogeneity. The implementation language of the communicating entities, the architecture, and the operating systems of the hosts where the agents are located play no role in this model. An agent implemented in Java, running in a Linux environment and on a SPARC-based platform, could interact with another one implemented in C++, running under Windows on a Pentium platform, without any special precautions.

Traditionally, a shared-data space is a *passive* entity, coordination information is *pushed* into it by a source agent and *pulled* from it by a destination agent. The amount of state information maintained by a shared-data space is minimal, it does not need to know either the location or even the identity of the agents involved. Clearly, there are applications where security concerns require controlled access to the shared information, thus, some state information is necessary. These distinctive features make this model scalable and extremely easy to use.

An alternative model is based on *active* shared-data spaces; here the shared-data space plays an active role as it informs an intended destination agent when information is available. This approach is more restrictive and requires the shared-data space to maintain information about the agents involved in the coordination effort. In turn, this makes the system more cumbersome, less scalable, and less able to accommodate mobility.

Linda [23] was the first system supporting associative access to a shared-data space. *Associative access* raises the level of communication abstraction. Questions such as who produced the information, when was it produced, and who were the intended consumers are no longer critical and applications that do not require such knowledge benefit from the additional flexibility of associative access.

Tuples are ordered collections of elements. In a shared-tuple space agents use templates to retrieve tuples; this means that an agent specifies what type of tuple to retrieve, rather than a specific tuple.

Linda supports a set of primitives to manipulate the shared tuple space; **out** allows an agent to deposit or write a tuple with multiple fields in the tuple space; **in** and **rd** are used to read or retrieve a tuple when a matching has been found; **inp** and **rdp** are nonblocking versions of **in** and **rd**; **eval** is a primitive to create an *active* tuple, one with fields that do not have a definite value but are evaluated using function calls.

Several types of systems extend some of the capabilities of Linda. Some, including **T Spaces** from IBM [78] and **JavaSpaces** from Sun Microsystems, extend the set of coordination primitives, others affect the semantics of the language, yet another group modifies the model. For example, **T Spaces** allows database indexing and event notification, supports queries expressed in the structured query language (SQL), and allows direct thread access when the parties run on the same Java Virtual Machine.

A survey of the state of the art tuple-based technologies for coordination and a discussion of a fair number of systems developed in the last few years are presented in [105]. Several papers referred in [96] provide an in-depth discussion of tuple space coordination.

Coordination Based on Middleware Agents. In our daily life middlemen facilitate transactions between parties, help coordinate complex activities, or simply allow one party to locate other parties. For example, a title company facilitates real estate transactions, wedding consultants and planners help organize a wedding, an auction agency helps sellers locate buyers and help buyers find items they desire.

So it is not very surprising that a similar organization appears in complex software systems. The individual components of the system are called *entities* whenever we do not want to be specific about the function attributed to each component; they are called clients and servers when their function is well defined.

Coordination can be facilitated by agents that help locate the entities involved in coordination, and/or facilitate access to them. Brokers, matchmakers, and mediators are examples of middle agents used to support reliable mediation and to guarantee some form of end-to-end Quality of Service (QoS). In addition to coordination functions, such agents support interoperability and facilitate the management of knowledge in an open system.

A *broker* is a middle agent serving as an intermediary between two entities involved in coordination. All communications between the entities are channeled through the broker. In Figure 2.3 we see the interactions between a client and a server through a broker. In this

case, the broker examines the individual QoS requirements of a client and attempts to locate a server capable of satisfying them; moreover, if the server fails, the broker may attempt to locate another one that can provide a similar service under similar conditions.

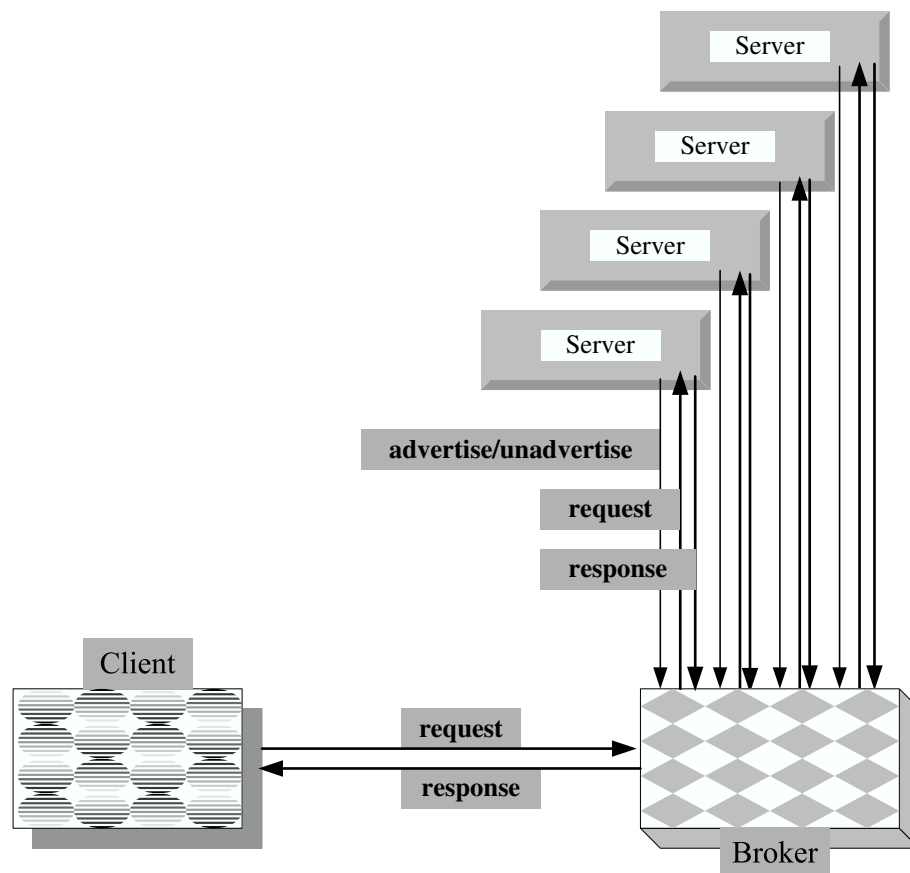


Figure 2.3: A broker acts as an intermediary between a client and a set of servers. The sequence of events: (i) servers register with a broker; (ii) a client sends a request; (iii) the broker forwards the request to a server; (iv) the server provides the response to the broker; (v) the broker forwards the request to the client.

A broker does not actively collect information about the entities active in the environment. Each entity has to make itself known by registering itself with the broker before it can be involved in mediated interactions.

Entities may provide additional information such as a description of services, or a description of the semantics of services. A broker may maintain a knowledge base with information about individual entities involved and may even translate the communication from one party into a format understood by the other parties involved.

A matchmaker is a middle agent whose only role is to pair together entities involved in coordination; once the pairing is done, the matchmaker is no longer involved in any transaction between the parties. For example, a matchmaker may help a client select a server as shown in Figure 2.4. Once the server is selected, the client communicates directly with the server bypassing the matchmaker.

The matchmaker has a more limited role; while the actual selection may be based on a QoS criterion, once made, the matchmaker cannot provide additional reliability support. If one of the parties fails, the other party must detect the failure and again contact the matchmaker. A matchmaker, like a broker, does not actively collect information about the entities active in the environment, each entity has to make itself known by registering itself with the matchmaker.

A mediator can be used in conjunction with a broker, or with a matchmaker to act as a front end to an entity. In many instances it is impractical to mix the coordination primitives with the logic of a legacy application, e.g., a database management system. It is easier for

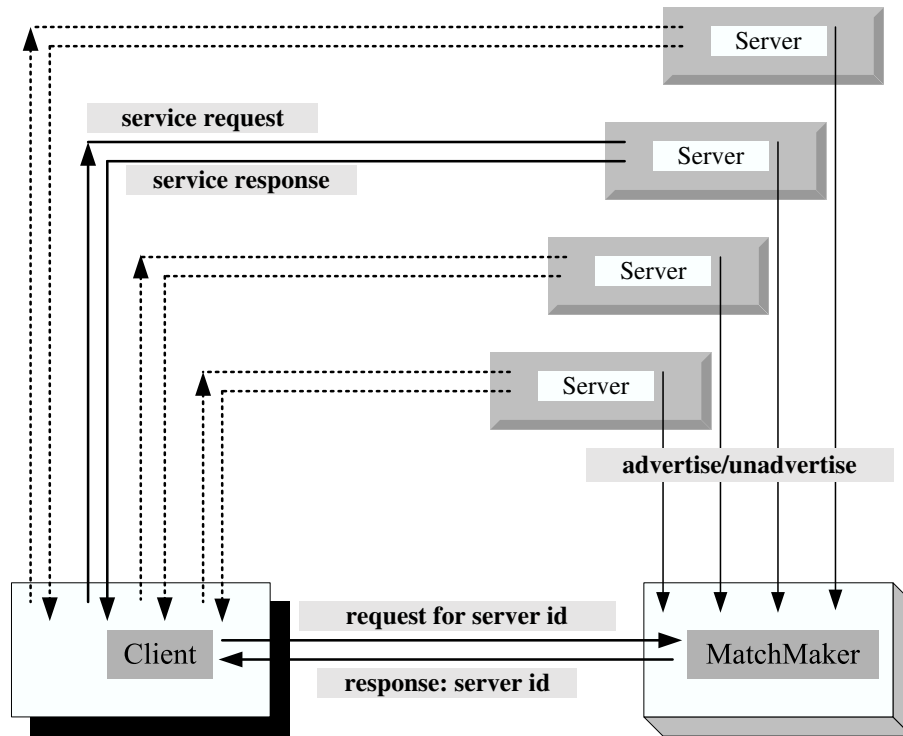


Figure 2.4: A matchmaker helps a client select a server. Then the client communicates directly with the server selected by the matchmaker. The sequence of events: (i) servers register with the matchmaker; (ii) a client sends a request to a broker; (iii) the broker selects a server and provides its ID; (iv) the client sends a request to the server selected during the previous step; (v) the server provides the response to the client.

an agent to use a uniform interface for an entire set of systems designed independently than to learn the syntax and semantics of the interface exposed by each system. A solution is to create a wrapper for each system and translate an incoming request into a format understood by the specific system it is connected to; at the same time, responses from the system are translated into a format understood by the sender of the request.

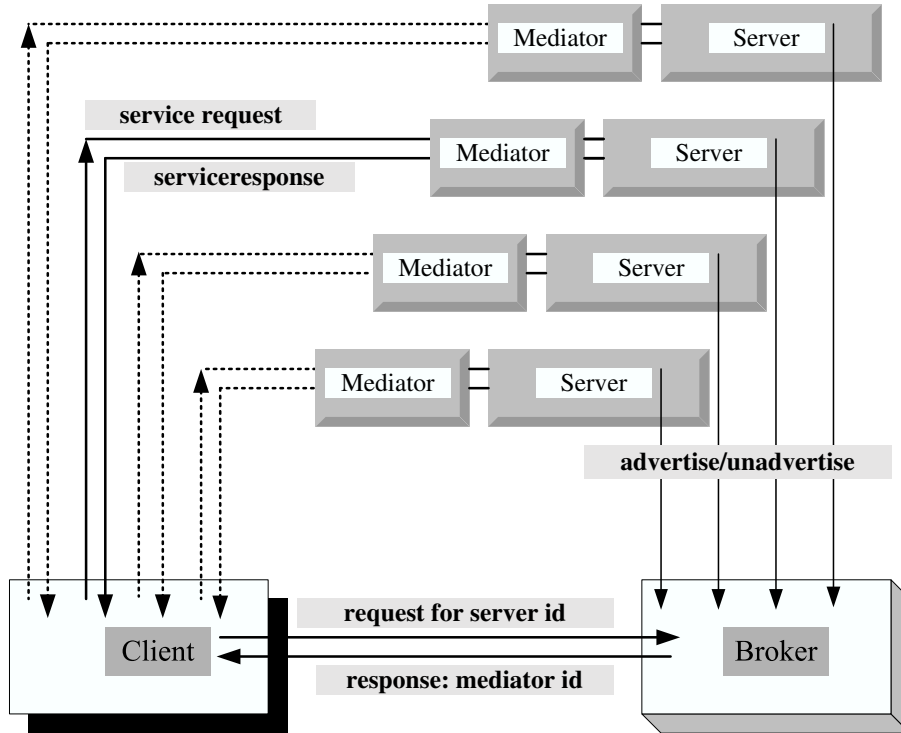


Figure 2.5: A mediator acts as a front end or a wrapper to one or more servers; it translate requests and responses into a format understood by the intended recipient. A mediator may be used in conjunction with brokers or matchmakers.

Agent-based coordination and coordination of agent federations have been investigated by several groups [23] [99].

2.3.3 Process Coordination and Workflow Management

In a large-scale distributed system, users have complex tasks and want to take advantage of the resource-rich environment to solve their problems subject to a set of constrains such as

deadlines, cost, and quality of the solution. A complex task consists of multiple activities. *Activities* are units of work to be performed by the agents, humans, computers, sensors, and other man-made devices. A *process description* is a structure describing the *activities* to be executed and the order of their execution. A process description contains one start and one end symbol and includes various patterns [123].

The term *workflow* has been used for some time in the business community to describe a complex task. Originally, workflow management was considered a discipline confined to the automation of business processes. Today most business processes depend on the Internet and workflow management has evolved into a network-centric discipline. The scope of workflow management has broadened. The basic ideas and technologies for automation of business processes can be extended to virtually all areas of human endeavor from science and engineering to entertainment.

Production, administrative, collaborative, and ad hoc workflows require that documents, information, or tasks be passed from one participant to another for action, according to a set of procedural rules. Production workflows manage a large number of similar tasks with the explicit goal of optimizing productivity. Administrative workflows define processes, while collaborative workflows focus on teams working toward common goals. E-commerce and Business-to-Business are probably the most notable examples of Internet-centric applications requiring some form of workflow management. E-commerce has flourished in recent years; many businesses encourage their customers to order their products online and some, including

PC makers, only build their products on demand. Various Business-to-Business models help companies reduce their inventories and outsource major components.

There are several distinctions between the workflows in large-scale distributed systems and the traditional workflows encountered in business management, office automation, or production management, see [82]:

(a) The emphasis in a traditional workflow model is placed on the contractual aspect of a transaction. For a workflow in a large-scale distributed system, the enactment of a case is sometimes based on a “best-effort model” where the agents involved do their best to attain the goal state but there is no guarantee of success.

(b) An important aspect of a transactional model is to maintain a consistent state of the system. A large-scale distributed system is an open system, thus, the state of a system is considerably more difficult to define than the state of a traditional system.

(c) A traditional workflow consists of a set of well-defined activities that are unlikely to be altered during the enactment of the workflow. However, the process description for a workflow in a large-scale distributed system may change during the lifetime of a case. After a change, the enactment of a case may continue based on the older process description, while under some circumstances it may be based on the newer process description. In addition to static workflows we have to support dynamic ones.

(d) The activities of a workflow in a large-scale distributed system could be long lasting. Some of the activities supported by the system are collaborative in nature and the workflow management should support some form of merging of partial process descriptions.

(e) The individual activities of a workflow in a large-scale distributed system may not exhibit the traditional properties of transactions. Consider, for example, durability; at any instance of time before reaching the goal state a workflow may roll back to some previously encountered state and continue from there on an entirely different path. An activity of a workflow in a large-scale distributed system could be either *reversible* or *irreversible*. Sometimes, paying a penalty for reversing an activity is more profitable in the long run than continuing on a wrong path.

(f) Resource allocation is a critical and very delicate aspect of workflow enactment in a large-scale distributed system. The system provides a resource-rich environment with multiple classes of resources and many administrative domains; there is a large variability of resources in each class; resource utilization is bursty in nature. Thus, we need: resource discovery services, support for negotiations among multiple administrative domains, matchmaking and brokerage services, reservations mechanisms, support for dynamic resource allocation, and other sophisticated resource management mechanisms and services.

(g) Mobility of various agents involved in the enactment of a case is important for workflows in a large-scale distributed system [82]. The agents may relocate to the proximity of the sites where activities are carried out to reduce communication costs and latency.

A workflow has three dimensions:

- (1) The process; the *process dimension* refers to the creation and the eventual modification of the process description.
- (2) The case; the *case dimension* refers to a particular instance of the workflow when the attributes required by the process enactment are bound to specific values.
- (3) The resources; the *resource dimension* refers to discovery and allocation of resources needed for the enactment of a case.

Workflow enactment is the process of carrying out the activities prescribed by the process description for a particular case.

The creation of a process description is similar to writing a program, the instantiation of a case is analogous to the execution of the program with a particular set of input data, while resource allocation has no direct correspondent in traditional computing where resources are under the control of the operating system. Static workflows correspond to traditional programs while dynamic workflows correspond to self-modifying ones.

The milestones in the life of a workflow are: (i) the creation of the *process description (PD)*; (ii) verification of the PD; (iii) the creation of a *case description (CD)*; and (iv) the enactment of a case.

Scripts, specialized workflow description languages, and formal methods can be used for process description [105], [107]. Several workflow description languages exist. Petri Nets (PNs) and their restrictions have provided for many years the formal methods of choice for

process description for business-oriented workflows; there is a vast literature on PNs and numerous algorithms and tools for PNs analysis have been developed along the years.

To avoid enactment errors we need to verify the process description and check for desirable properties such as *safety* and *liveness*. Some process description methods are more suitable for verification than others.

In an open system it is desirable to support multiple process description methods but a single internal representation method.

We distinguish two types of workflows, *static* and *dynamic*. The process description of a static workflow is invariant in time. The process description of a dynamic workflow changes during the workflow enactment phase due to circumstances unforeseen at the process definition time. Exceptional conditions are handled by a static workflow using its exception handling mechanisms while unforeseen circumstances trigger planning and generation of a new process description. For example, an activity in a process description involves a service that has been discontinued, but there are several new services whose composition is equivalent to the missing service.

It is conceivable that multiple variations of a process description may co-exist and it is useful to define the concept of *workflow inheritance* and exploit it in the implementation of a coordination architecture.

2.3.4 Process Description and Case Description

A *process description* is a formal description of the complex problem a user wishes to solve. For the process description, we use a formalism similar to the one provided by Augmented Transition Networks (ATNs) [134]. The coordination service implements an abstract ATN machine. A *case description* provides additional information for a particular instance of the process the user wishes to perform, e.g., it provides the location of the actual data for the computation, additional constraints related to security, cost, or the quality of the solution, a soft deadline, and/or user preferences [82].

The process description used by BondGrid coordination is based upon the BNF grammar presented in Figure 2.6. The symbol S denotes the start symbol, while e stands for an empty string.

2.3.5 Coordination Services

Let us now examine the question: why are coordination services needed in an intelligent environment and how can they fulfill their mission? First of all, some of the computational activities are long lasting and it is not uncommon to have a large simulation running for 24 hours or more. An end-user may be intermittently connected to the network so there is a need for a proxy whose main function is to wait until one step of the complex computational procedure involving multiple programs is completed and launch the next step of the compu-

```

S ::= <ProcessDescription>

<ProcessDescription> ::= BEGIN <Activities> END
<Activities> ::= <SequentialActivities> | <ConcurrentActivities>
               | <IterativeActivities> | <SelectiveActivities> | <Activity>
<SequentialActivities> ::= <Activities> ; <Activities>
<ConcurrentActivities> ::= FORK <Activities> ; <Activities> JOIN
<IterativeActivities> ::= ITERATIVE <ConditionalActivity>
<SelectiveActivities> ::= CHOICE <ConditionalActivity> ;
                        <ConditionalActivitySet> MERGE
<ConditionalActivitySet> ::= <ConditionalActivity>
                           | <ConditionalActivity> ; <ConditionalActivitySet>
<ConditionalActivity> ::= { COND <Conditions> } { <Activities> }
<Activity> ::= <String>
<Conditions> ::= ( <Conditions> AND <Conditions> )
               | ( <Conditions> OR <Conditions> )
               | NOT <Conditions> | <Condition>
<Condition> ::= <DataName>.<Attribute> <Operator> <Value>
<DataName> ::= <String>
<Attribute> ::= <String>
<Operator> ::= < > | = | <= | >=
<Value> ::= <String>
<String> ::= <Character> <String> | <Character>
<Character> ::= <Letter> | <Digit>
<Letter> ::= a | b | ... | z | A | B | ... | Z
<Digit> ::= 0 | 1 | ... | 9

```

Figure 2.6: BNF grammar for the process description.

tation. Of course, a script will do, but during this relatively long period of time unexpected conditions may occur and the script would have to handle such conditions. On the other hand, porting a script designed for a cluster to a large-scale distributed system is a non-trivial task. The script would have to work with other services, e.g., with the *information* service, or directory services to locate other core services, with the *brokerage* service to select

systems which are able to carry out different computational steps, with a *monitoring* service to determine the current status of each resource, with a *persistent storage* service to store intermediary results, with an authentication service for security considerations, and so on.

While automation of the execution of a complex task in itself is feasible using a script, very often such computations require human intervention. Once a certain stage is reached, while some conditions are not met, we may have to backtrack and restart the process from a previous checkpoint using a different set of model parameters, or a different input data. For example, during the computation of the correlation coefficient indicating the resolution of the sample electron density map, we may decide to eliminate some of the original virus particle projections which introduce too much noise in the reconstruction process. It would be very difficult to automate such a decision which requires the expertise of a highly trained individual. In such a case the coordination service should checkpoint the entire computation, release most resources, and attempt to contact an individual capable of making a decision. If the domain expert is connected to the Internet with a palmtop computer with a small display and a wireless channel with low bandwidth, the coordination service should send low resolution images and summary data enabling the expert to make a decision.

In summary, the coordination service acts as a proxy for the end-user and interacts with core and other services on user's behalf. It hides the complexity of the system from the end-user and allows user interfaces running on the network access devices to be very simple. The coordination service should be reliable and able to match user policies and constraints

(e.g., cost, security, deadlines, quality of solution) with the corresponding system policies and constraints.

A coordination service relies heavily on shared ontologies. It implements an abstract machine which understands a description of the complex task, we call it a *process description*, and a description of a particular instance of the task, we call it a *case description*.

2.4 A Case Study: the Coordination Service in BondGrid

In the following sections we describe the process description and the case description, the ontologies used for BondGrid task coordination, and the coordination service.

2.4.1 Process Description and Case Description

A *process description* defines the data dependencies among the activities of a complex task and consists of *end-user activities* and *flow control activities*. The execution of an end-user activity corresponds to the execution of an end-user service thus of an application program. The specification of an end-user activity may include the symbolic names of the input and/or output data sets of the corresponding program. Figure 2.7 shows a sample process description for the 3D atomic structure determination of macromolecules based upon electron microscopy.

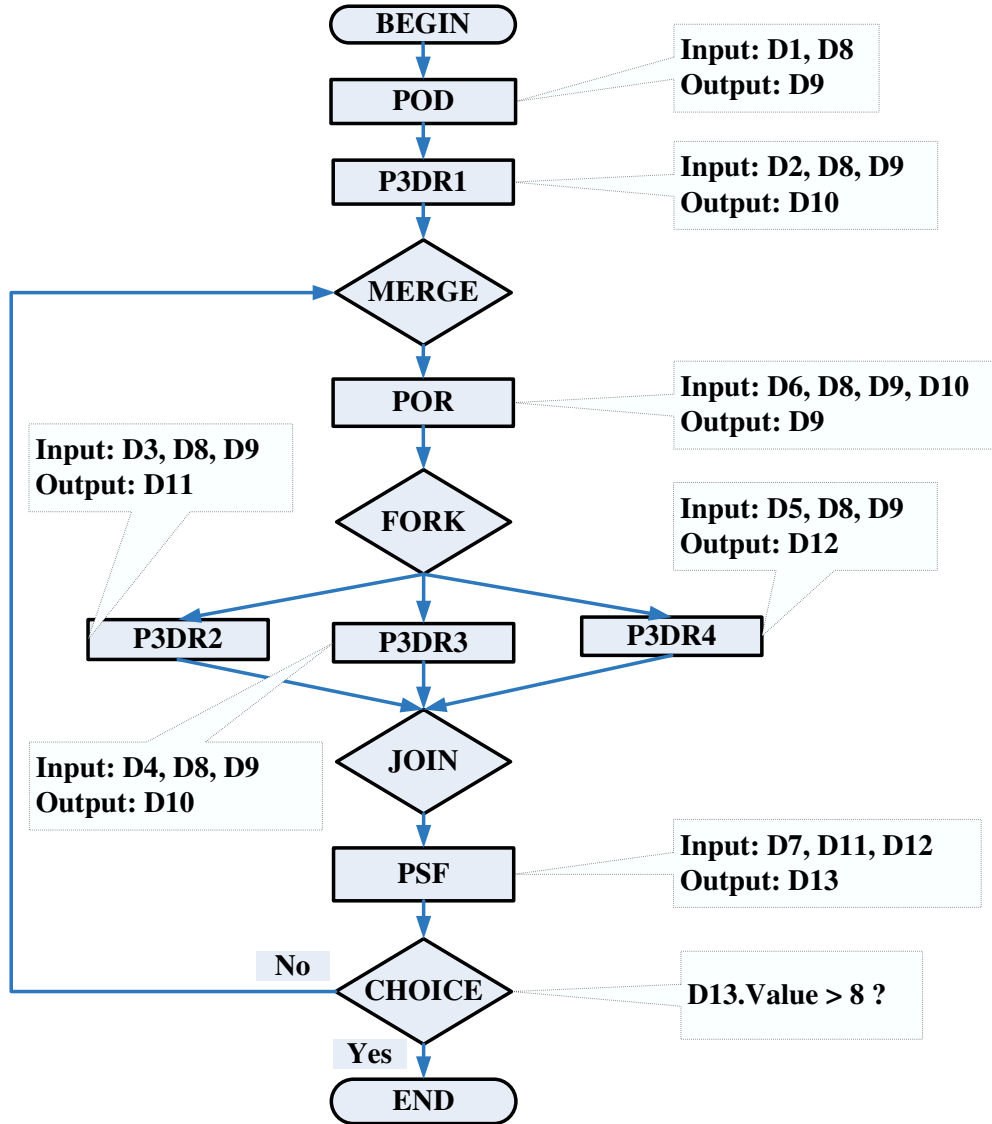


Figure 2.7: A process description for the 3D structure determination. D_1, D_2, \dots, D_{13} are the symbolic names of the input and output data files for the programs carrying out different end-user activities.

Flow control activities do not have associated end-user services. They are used to control the execution of end-user activities. We define six flow control activities: **Begin**, **End**, **Choice**,

Fork, **Join**, and **Merge**. Every process description should start with a **Begin** activity and conclude with an **End** activity. The **Begin** activity and the **End** activity should occur exactly once in a process description.

The *direct precedence* relation reflects the causality among activities. If activity \mathcal{B} can only be executed directly after the completion of activity \mathcal{A} , we say that \mathcal{A} is a *direct predecessor* activity of \mathcal{B} and that \mathcal{B} is a *direct successor* activity of \mathcal{A} . An activity may have a *direct predecessor set* of activities and a *direct successor set* of activities. We use the term “direct” rather than “immediate” to emphasize the fact that there may be a gap in time from the instance an activity terminates and the instance its direct successor activity is triggered. For the sake of brevity we drop the word “direct” and refer to predecessor activity set, or predecessor activity and successor activity set, or successor activity.

A **Choice** flow control activity has one predecessor activity and multiple successor activities. It can be executed only after its predecessor activity has been executed. Following the execution of a **Choice** activity, only *one* of its successor activities may be executed.

A **Fork** flow control activity has one predecessor activity and multiple successor activities. The difference between **Fork** and **Choice** is that after the execution of a **Fork** activity, all the activities in its successor set are triggered.

A **Merge** flow control activity is paired with a **Choice** activity. It has a predecessor set consisting of two or more activities and only one successor activity. A **Merge** activity is triggered after the completion of *any* activity in its predecessor set.

A **Join** flow control activity is paired with a **Fork** activity. Like a **Merge** activity, a **Join** activity has multiple predecessor activities and only one successor activity. The difference is that a **Join** activity can be triggered only after all of its predecessor activities are completed.

A *case description* associates symbolic data names referred to by the corresponding process description with real data. If the data refers to a file, one or more URLs are specified. Multiple URLs refer to multiple copies of the same file. Various constraints (e.g., deadlines, cost, exclusion of some resources, special resource requirements) are often provided by case descriptions.

Process and case descriptions are part of the system-wide ontologies, as seen in Figure 2.8. Their instances can be stored in knowledge bases and exchanged in XML format.

2.4.2 Ontologies for BondGrid Coordination

Figure 2.8 shows the logic view of the main ontologies used for BondGrid coordination and their relations. A non-exhaustive list of classes in this ontology includes: **task**, **process description**, **case description**, **activity**, **data**, **service**, **resource**, **hardware**, and **software**. A **task** class is related to the **process description** and the **case description** classes. A **process description** contains a set of **activities**. A **service** class is associated with a **resource** class. In turn a **resource** class is associated with **hardware** and

software classes. The `data` class is connected to the `in activity`, `service`, and `case description` classes.

2.4.3 The Coordination Service

The coordination service consists of a message handler, a coordination engine, and a service manager. The message handler is responsible for inter-agent communication. The coordination engine manages the execution of tasks submitted to the coordination service. The service manager provides a GUI for monitoring the execution of tasks and the interactions between coordination service and other services. These three components run concurrently on different planes of the agent and share the same knowledge base as shown in Figure 2.9. Any modifications performed on the knowledge base by one component affects the other components.

As pointed out earlier, a task consists of a process description and a case description. The execution of a task is associated with a common data space shared by all activities. Each symbolic name in the process description corresponds to an entry in this data space. Initially, the case description may provide some binding of symbolic names to existing data files. As the execution of the task progresses, more symbolic names are bound to the data produced as the result of end-user services. After the successful completion of a task, the data files containing the results are disposed of as specified by the case description, e.g., sent to a persistent storage service.

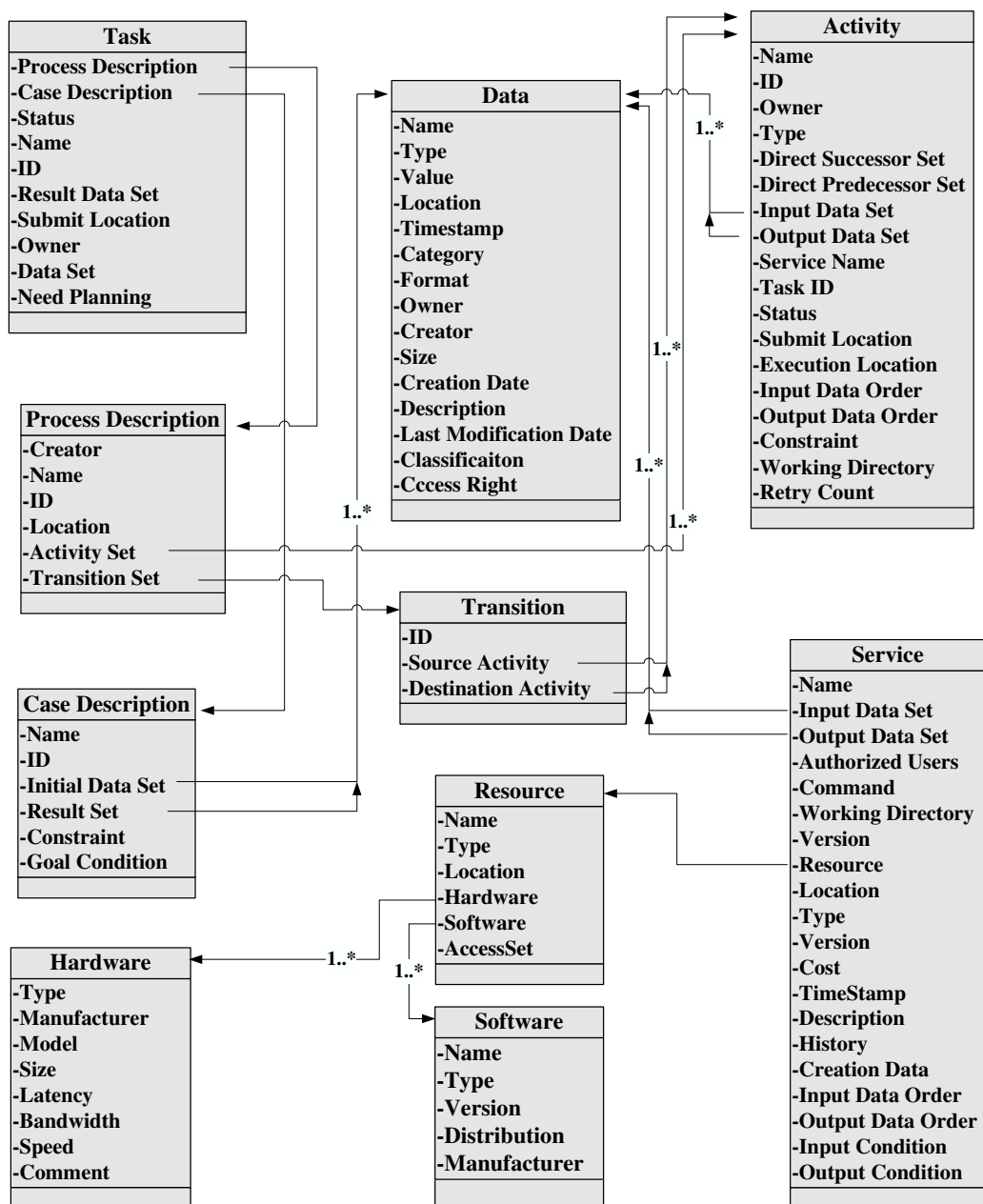


Figure 2.8: Logic view of the main ontology for BondGrid coordination.

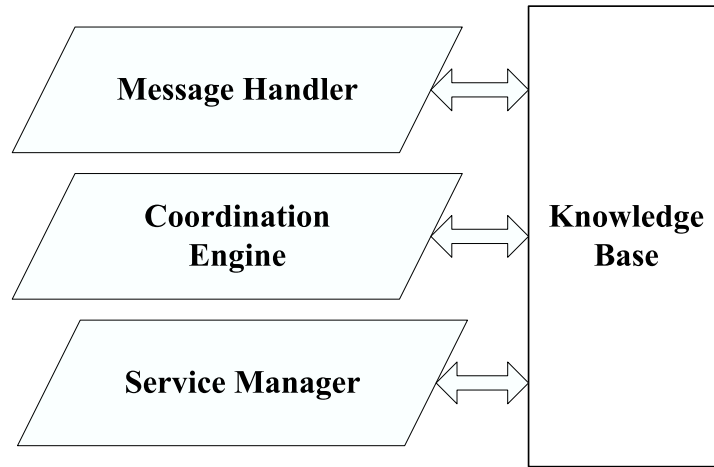


Figure 2.9: The components of the coordination service.

Figure 2.10 shows the state transition diagram of a task. The states of a task are: **SUBMITTED**, **WAITING**, **RUNNING**, **PLANNING**, **REPLANNING**, **FINISHED**, and **ERROR**. Once a *task submission* message is received it is queued by the message handler of the coordination service. Then the message handler creates a *task instance* in the knowledge base. The initial state of the newly created task is **SUBMITTED**.

The coordination engine keeps checking the state of all task instances in the knowledge base. When it finds a task instance in **SUBMITTED** state it attempts to initiate its execution. One of the slots of the task class indicates if the task needs planning (the slot is set to **PlanningNeeded**). If the task has already been sent to the planning engine and awaits the creation of a process description the slot is set to **Waiting**. If the process description has been created the slot is set to **PlanningComplete**.

If the task needs planning, the coordination engine waits until the new process description is ready, then it updates the task instance accordingly, and sets its state to **RUNNING**. When the execution of the task cannot continue (e.g., due to resource unavailability) the coordination engine may send the task to a planning service for replanning. In such a case the state of the task is set to **REPLANNING**. After the successful completion of a task its the state is **FINISHED**, while in case of an error it is set to **ERROR**.

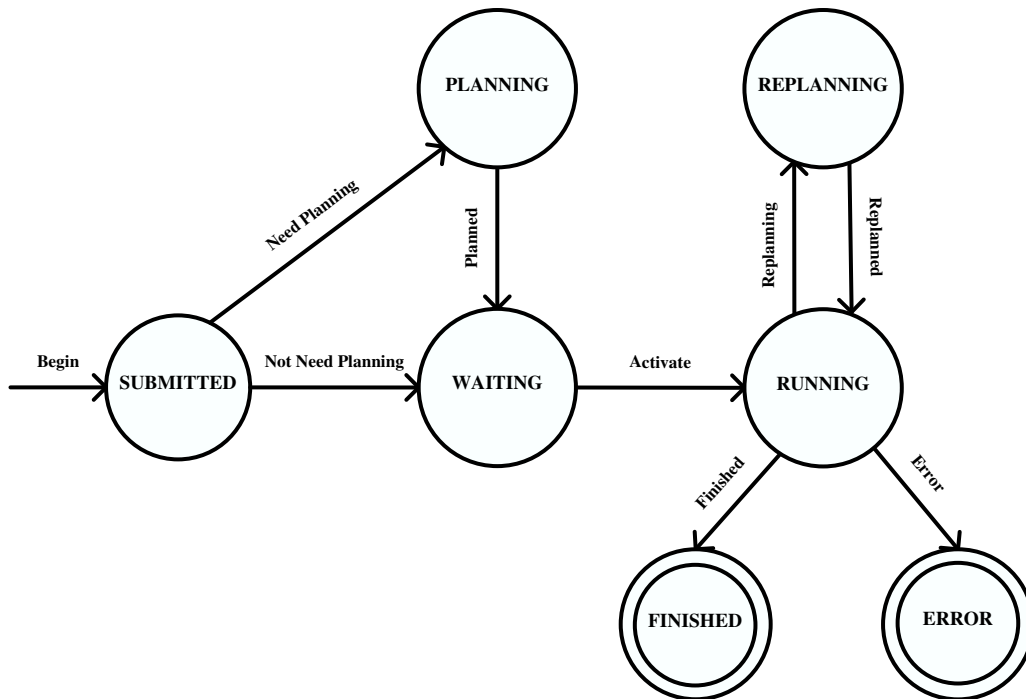


Figure 2.10: The task state transition diagram.

The coordination engine takes different actions according to the type of each activity. The handling of flow control activities depends on their semantics. For an end-user activity, the coordination service collects the necessary input data and performs data staging of each

data set, bringing it to the site of the corresponding end-user service. Upon completion of an activity, the coordination service triggers a data staging phase, collects partial results, and updates the data space.

The activity class has a slot describing the state of an activity, `INACTIVE`, `ACTIVE`, `DISPATCHED`, `NOSERVICE`, `FINISHED`, or `ERROR`, as shown in Figure 2.11. Initially, an activity is in the `INACTIVE` state. The coordination engine sets the state of its `begin` activity as `ACTIVE` when the state of a task transitions from `WAITING` to `RUNNING`. When the coordination engine finds an `ACTIVE` activity it checks the type slot of the activity class. In case of a flow control activity, the coordination engine sets: (i) the state of one or more successor activities to `ACTIVE` and (ii) the state of the current activity to `FINISHED`. In case of an end-user activity, the coordination engine attempts to find an end-user service for this activity subject to a time and/or a retry count limit. If the coordination engine finds an end-user service, the state of this activity becomes `DISPATCHED`. Otherwise, the state becomes `NOSERVICE`. When the end-user service signals the successful completion of an activity the coordination engine sets (i) the state of the corresponding activity to `FINISHED` and (ii) the state of the successor activity to `ACTIVE`; otherwise, the state is set as `ERROR`.

The coordination engine executes iteratively the procedure shown in Figure 2.12. The message handler executes iteratively the procedure shown in Figure 2.13.

The interaction of the coordination service with the end-user. Figure 2.14 summarizes the interactions between the user and the coordination service. Such interactions can be initiated by the user when submitting a task or requesting task status information, or

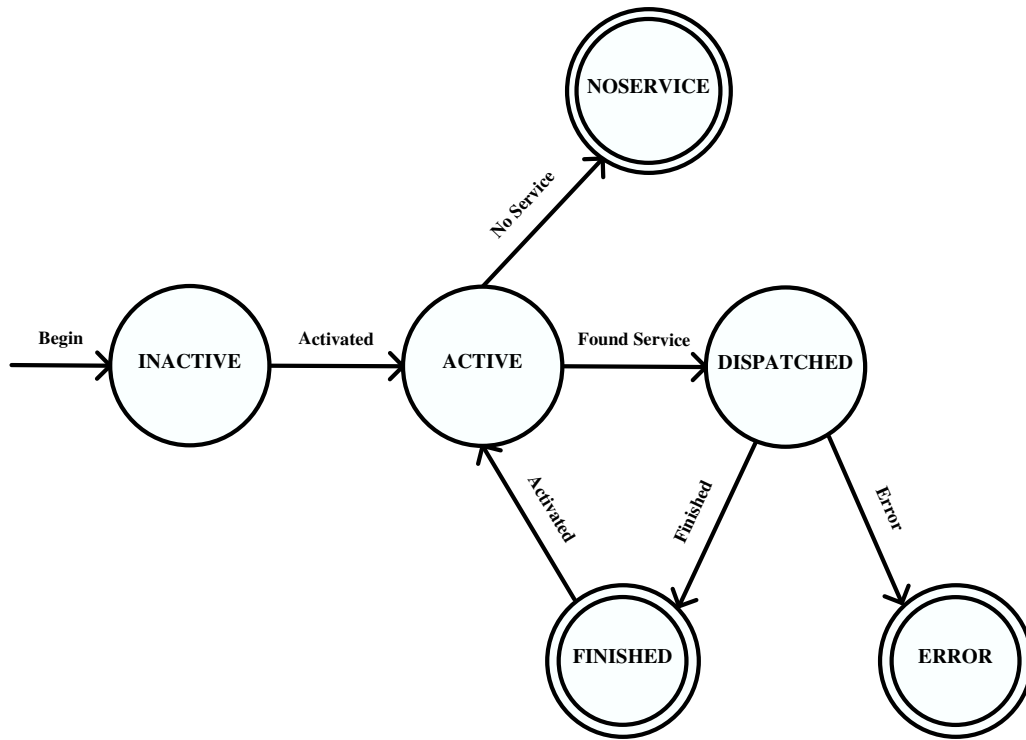


Figure 2.11: The activity state transition diagram.

by the coordination service when reporting an error condition or the successful completion of the task.

A request for coordination is triggered by the submission of a task initiated by a user. When receiving such a message the coordination engine first checks the correctness of the process and task description. Next, the task activation process presented earlier is triggered. The user interface then subscribes to the relevant events produced by the coordination service.


```

for each task in the knowledgeBase
  if (task.Status .eq. SUBMITTED)
    if (task.NeedPlanning .eq. TRUE)
      send (task to a planningService);
      task.Status = PLANNING;
    else
      task.Status = WAITING;
    end if;
  else if (task.Status .eq. WAITING)
    status.BeginActivity = ACTIVE;
    task.Status = RUNNING
  else if (task.Status .eq. RUNNING)
    for (each activity with (activity.Status .eq. ACTIVE))
      if (activity.Type .eq. flowControlActivity)
        carry out flowControlActivity;
        activity.Status = FINISHED;
        for every activity in postset(flowControlActivity)
          activities.Status = ACTIVE;
        end for;
      else if (activity.Type .eq. endUserActivity)
        search for endUserService with time constraint;
        if found
          dispatch (activity to endUserService);
          activity.Status = DISPATCHED;
        else
          activity.Status = NOSERVICE;
          if (task.NeedPlanning .eq. TRUE)
            send task to a planning service for replanning;
            task.Status = REPLANNING;
          else
            task.Status = ERROR;
          end if;
        end if;
      end if;
    end for;
  end if;
end for;

```

Figure 2.12: The coordination engine executes iteratively the procedure.

A user may send a query message to the coordination service requesting task state information. The message handler parses the request and fetches from its knowledge base the relevant slots of the task instance.

```

pick up a message from message queue;

if (message.Type .eq. PLANNEDTASK)
    task.Status = WAITING;
else if (message.Type .eq. RE-PLANNEDTASK)
    if (replanning fails)
        task.Status = ERROR;
    else
        task.Status = RUNNING;
    end if;
else if (message.Type .eq. RESULT)
    if (computation.Status .eq. SUCCESS)
        activity.Status = FINISHED;
        status(successor.activity = ACTIVE);
    else
        activity.Status = ERROR;
        task.Status = ERROR;
    end if;
end if;

```

Figure 2.13: The message handler executes iteratively the procedure.

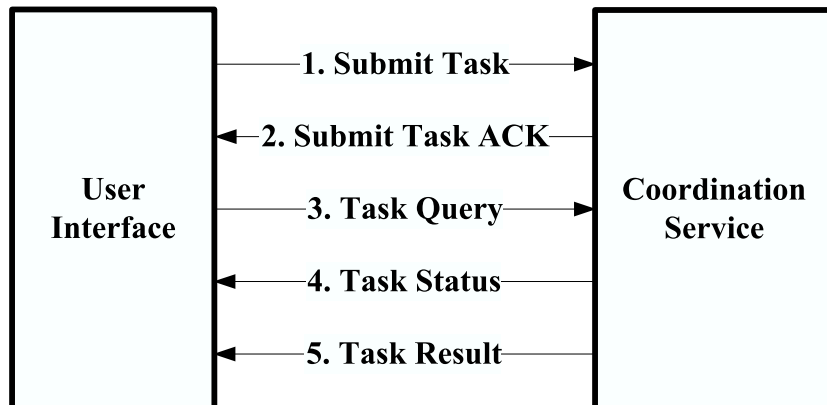


Figure 2.14: The interactions between the coordination service and the end-user.

Upon completion of the task, or in case of an error condition, the coordination service posts the corresponding events for the user interface.

The interaction of the coordination service with other core services and application containers. A coordination service acts as a proxy for one or more users and interacts on behalf of the user with other core services such as the brokerage service, the matchmaking service, the planning service, and the information service, as shown in Figure 2.15.

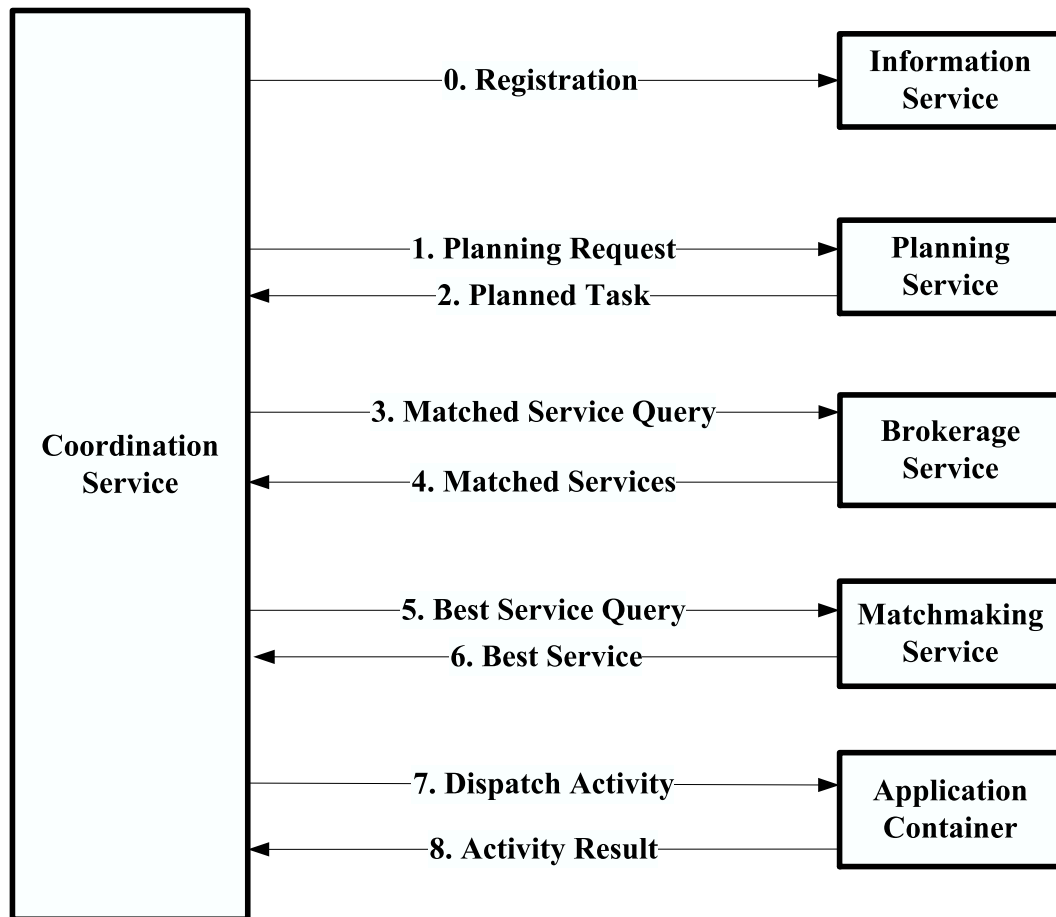


Figure 2.15: The interactions between the coordination service and other core services.

If a task submitted by the user does not have a valid process description, the coordination service forwards this task to a planning service. During the execution of a task, when the coordination service needs to locate an end-user service for an activity, it interacts with the brokerage and matchmaking services. A brokerage service has up-to-date information regarding end-user services and their status. A matchmaking service is able to determine a set of optimal or suboptimal matchings between the characteristics of an activity and each service provider.

The event service supports asynchronous communication. For example, a user submits a task using a PDA connected via a wireless network to the Internet and subscribes for several events. Then the user is disconnected from the network. After the completion of the task the coordination service posts a termination event. When the end-user is reconnected to the network and inquires about the status of the task, the termination event is delivered to the user interface.

Besides core services, a coordination service interacts with application containers. When a coordination service attempts to locate the optimal end-user service for an activity, the status and the availability of data on the node providing the end-user service ought to be considered in order to minimize communication costs.

2.4.4 Performance Measurements

While we cannot attempt a realistic performance evaluation study of the coordination service before all the components of the BondGrid environment are fully implemented, we have conducted some performance studies on the communications between the coordination service and other societal (core) services.

The coordination service uses messages to exchange ontologies with other components of the environment. An ontology is first converted to an XML format and then packed into a BondGrid message.

The footprint of instances varies function of the class; task instances and process description instances, often consist of tens to hundreds of kilobytes, data instances, are often less than one kilobyte. We experimented with ontologies of different sizes. Figure 2.16 shows the relationship between the size of the ontologies and the encoding, transmission, and the decoding time. Our testing environment was provided by two systems with 1.8 GHz Pentium IV processors and 1 GB of main memory, under Linux. The two machines are connected to the same hub. The resolution of our clock is one millisecond. Figure 2.16 indicates that as the size of the ontology (in XML format) increases, the time needed to encode, transmit, and decode increases as well. Encoding time, transmission time and decoding time are generally less than 30 milliseconds. We observed considerably large transmission times (seconds), when the network load was high.

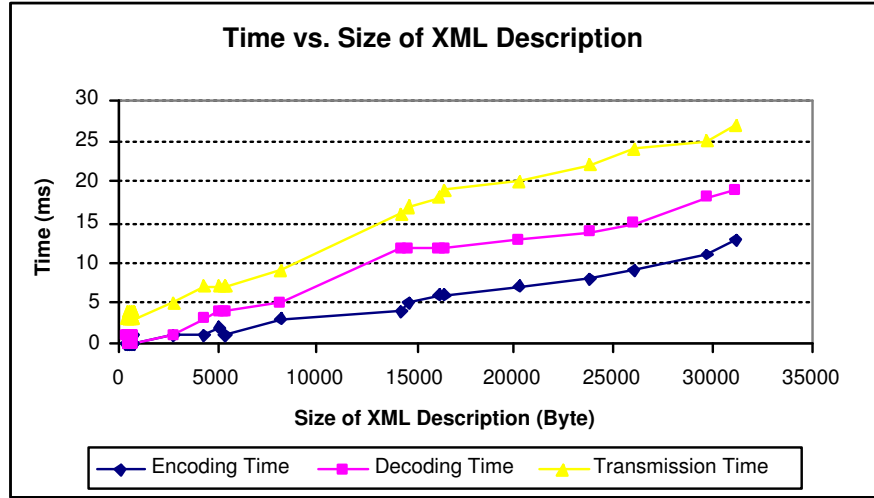


Figure 2.16: The performance of encoding, transmitting, and decoding instances between the coordination service and other components of the environment.

We also studied the message exchange rate between a coordination service and an agent. The coordination service is running on a system with a 1.8 GHz Pentium IV processor and 1 GB of main memory under Windows. The agent runs on a system with a 900 MHz Pentium III processor and 256 MB of memory under Windows. The agent keeps submitting tasks to the coordination service, which is able to process about 100 submissions per second while the incoming rate is about 6,000 submissions per second.

CHAPTER 3

MATCHMAKING

In this chapter, we study the problem of matchmaking for large-scale distributed systems. The matchmaking process in a large-scale distributed system involves three types of entities or agents: the consumers called requesters; the producers called providers; and the matchmaking service. The matchmaking services mediate among the providers and the requesters and use a matching algorithm to evaluate a matching function that returns the matching degree. We introduced for the first time a more comprehensive ontology-based resource matching scheme for large-scale distributed systems. We defined the ontologies for commonly used resources in a large-scale distributed system. The scheme supports a variety of matching functions including boolean function, arithmetic function, and fuzzy function. We implemented a matchmaking service in BondGrid. The experimental results indicate that the response time of matchmaking requests is dominated by the time to access the resource knowledge base. The complexity of the matching function has little effect on the response time. We also tested two matchmaking algorithms, a simple algorithm that requires an exhaustive search of all resource advertisements, and a modified algorithm that only covers

a portion of the knowledge base. The modified algorithm is able to find the near optimal matching resources with a sufficiently lower computational cost than the simple algorithm.

3.1 Introduction and Motivation

Webster dictionary defines the verb “to match” as “to be equal, similar, suitable, or corresponding to in some way”. In computer science, the term *matching* refers to a process of evaluation of the degree of similarity of two objects. Each object is characterized by a set of properties/attributes; each property is a tuple $(name, value)$, with *name* a string of characters and *value* either a constant (a number - integer or real, a Boolean constant - “true” or “false”, or a string of characters), or an expression that returns a constant. The “matching degree”, m , is a real number typically $0 \leq m \leq 1$, with $m = 0$ meaning a *total mismatch* and $m = 1$ a *perfect match*.

Matching is a common operation in many areas of computer science, e.g., in stringology and its applications to bioinformatics. In this dissertation we discuss application of matching in the area of resource discovery and resource allocation in large-scale distributed systems. In this context a matching operation involves a *reference object* and a *current object*. Oftentimes, the current object has to be matched against a set of reference objects, to identify the object, or the subset of objects in the set, that best match the current object. In this case the operation is referred to as *matchmaking*.

A large-scale distributed system is an open system, a large collection of autonomous systems giving individual users the image of a single virtual machine with a rich set of hardware and software resources. A set of *core services* provide access to various resources. For example, a *resource discovery* service assists users, or their proxies, to locate needed resources in the system. In more traditional computing systems, resources are managed centrally under the control of a single administrative authority by the resource management component of an operating system or by a distributed operating system. The central management of resources in a large-scale distributed system is unthinkable because of the scale of the system and because it would violate the autonomy of individual resource providers, a critical aspect of the system.

The matchmaking process in large-scale distributed system involves three types of entities or agents: the consumers called *requesters*, the producers called *providers*, and the *matchmaking service*. The matchmaking services mediate among the providers and the requesters and it uses a *matching algorithm* to evaluate a *matching function* which produces the *matching degree*. The first step in making resources available to the community is to advertise them. We call the description of a resource a *resource advertisement*, or simply *resource*. A resource request, simply called a *request*, consists of a function to be evaluated in the context of a resource. For example, the request “*CpuClockRate* > 2Gflops” will be evaluated by determining if a resource has an attribute called “*CpuClockRate*” and if so the value of this attribute satisfies the condition “*Value(CpuClockRate)* > 2Gflops”. If the request can be successfully satisfied, the matchmaking service responds with a list of ranked

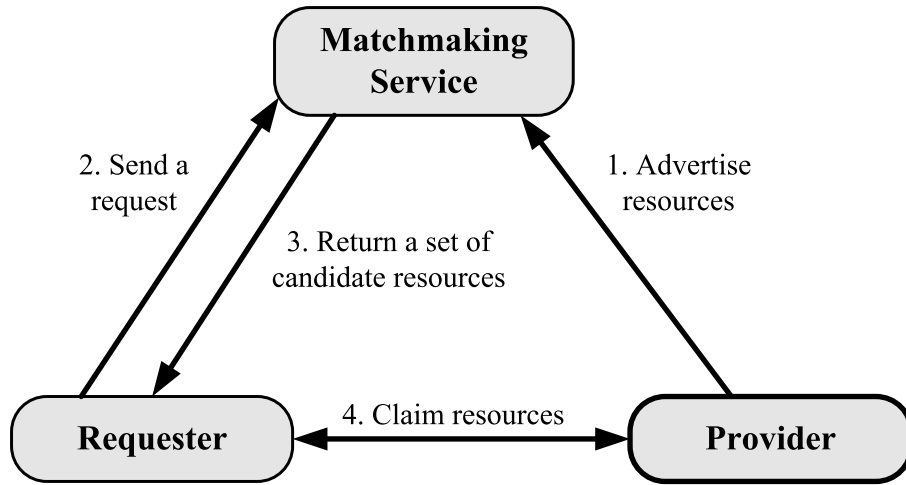


Figure 3.1: The matchmaking process in large-scale distributed systems: 1) Providers send resource descriptions to the matchmaking service; 2) A request is sent to the matchmaking service; 3) The matchmaking service executes a matchmaking algorithm and returns a set of ranked resources to the requester; 4) The requester chooses a resource from the set and contacts the corresponding resource provider.

resources. Figure 3.1 summarizes the matchmaking process in large-scale distributed systems. Typically, a *coordination service* which acts as a proxy for the end-user, is responsible to trigger the next activity in the activity graph (or process description) and it uses a *broker* to find the most suitable site for the execution of the next ready to run activity. In turn, the broker uses the matchmaking service. A matchmaking framework should be extensible and should support exact and inexact matchmaking. In some cases, requesters need to find the resource that exactly matches the request. In other cases, resources that partially match the request are also acceptable.

In the next sections we talk a formal definition of matchmaking in large-scale distributed systems, the development of ontologies necessary for resource management, an implementation of algorithms to evaluate different types of matchmaking expressions, and an evaluation of a matchmaking service.

3.2 Background and Related Work

Matchmaking has been a hot topic of MAS (Multi-Agent Systems) research, related to question on how to find a suitable agent for a specific problem. The notable results in this area are ACLs (Agent Communication Languages) and matchmaking algorithms based on these languages. One of the earliest results in this area is ABSI [114] (Agent-Based Software Interoperability) facilitator which uses KQML (Knowledge Query and Manipulation Language) specification and KIF (Knowledge Interchange Format) as the content language. The matchmaking of an advertisement and a request is made through the unification of equality predicate. In the COIN system [73], the matchmaking algorithm is based on a unification process similar to Prolog. The InfoSleuth [15] uses KIF as the content language. The matchmaking algorithm is based on constraints, i.e., the advertisement and the request match if the constraints are satisfied. SDL (Service Description Language) was proposed in [117] to describe services. The matchmaking algorithm finds k-nearest services for a request according to the distance between the service names (pairs of verb and noun terms) and the request. CDL (Capability Description Language) was proposed in [133]. It supports

reasoning through the notions of subsumption and instantiation. LARKS (Language for Advertisement and Request for Knowledge Sharing) was proposed in [120] for describing service capability and service request. It supports ITL (Information Terminological Language) [119] concept language. The relations among concepts are used to compute semantic similarities. The matchmaking in MAS involves semantic service matchmaking using the concept relationship and word distance to determine the semantic similarities of advertisements and requests. The matchmaking in MAS does not involve other resource types and the matchmaking results are exact, i.e., only "true" and "false" are allowed.

Research for service discovery in the Internet involves ontology-based matchmaking. The traditional methods of service discovery include name matchmaking and keyword matchmaking. Some new methods are based on ontologies. In [98] a semantic matchmaking framework based on DAML-S, a DAML (DARPA Agent Markup Language)-based language for service description, was proposed for semantic matchmaking of web services capabilities. The basic idea is that an advertisement matches a request when the service provided by the advertisement can be of some use to the requester. The matchmaking is performed on the outputs and inputs of the advertisement and the request based on the ontologies available to the matchmaker. Through the subsumption relationship of one concept of the input/output of the advertisement and one concept of the input/output of the request, four levels of matching can be determined: exact, plug-in, subsume, and fail. The idea of checking the concepts of input and output is similar to the one in the MAS research.

The matchmaking framework of Condor [102] system uses a semi-structured data model [89] called classified advertisements (classads) to describe resources and requests. A classad is a mapping from attribute names to expressions, see Figure 3.2. Condor matchmaking takes two classads and evaluates each one with the other. A classad has an attribute named "Constraint" that is used to be evaluated in the context of this classad and the classad being matched with this classad. Only when the values of attribute "Constraint" of both classads are evaluated to be true, can these two classads be thought to be matched. A classad has an attribute named "Rank" that measures the desirability of a match. The evaluation value of "Rank" identifies how much the two classads match. The larger the value, the better they match. The Condor system requires the provider and the requester to know each other's classad structure. The evaluation result of the attribute "Rank" is generally not normalized and can not tell explicitly how well two classads match.

The matchmaking framework in Condor supports the selection of only one resource. Based on the Condor matchmaking, in [79], an extension, called set-extended classad syntax, was proposed to support the multiple resource selection. The matchmaking algorithm evaluates a set-extended classad with a set of classads and returns a classad set with the highest rank. When the size of the classad set is large, it is not feasible to evaluate all of the possible combinations of the resources. A greedy heuristic algorithm is used to find the classad set with the highest rank. This set-extended resource selection framework can perform both resource discovery and resource allocation.

```

[
  Type          = "Laser Printer"
  Memory        = "64"           // megabytes
  Name          = "csb110.cs.ucf.edu"
  PriorityGroup1 = { "xbai" , "yji" }
  PriorityGroup2 = { "xliu" }
  Constraint     =
    other.Type == "Print Job"
    && (member( other.Owner, PriorityGroup1)
        || member( other.Owner, priorityGroup2) )
  Rank          =
    member( other.Owner, PriorityGroup1) * 10
    + member ( other.Owner, PriorityGroup2) * 5
]

[
  Type          = "Print Job"
  Owner         = "xbai"
  Constraint     =
    other.Type == "Laser Printer"
  Rank          =
    other.Memory / 32
]

```

Figure 3.2: Classads describing a laser jet printer (left) and a print job (right).

In [125], a service selection model was proposed based on the quality of service of different service providers. Each time after a service is used the user sends a feedback to the matchmaker. When a request arrives, the matchmaker finds the most appropriate service according to the inputs specified by the request based on the feedbacks it received. The problem is that different users may have different criteria for the service evaluation and even the same user may have different criteria for the service evaluation at different times. Fur-

thermore, for the same request the matchmaker always finds the same service as the best service. This is not good for the throughput of the whole system.

3.3 Resource Ontologies

Resource ontologies are a critical component of the matchmaking framework. Entities in the matchmaking framework, i.e. the providers, the requesters, and the matchmaking services, which are generally not in the same domain, must share the same ontology structure.

The structure of a category of entities is described as a *class* in Protégé knowledge base [101]. Figure 3.3 shows the hierarchical relationship among resource classes. Figure 3.4 shows the structures of the Workstation, Cluster, MPP, and SMP classes. A class consists of one or more slots. A *slot* describes one attribute of the class and consists of a name and a value. An instantiation of a class is called an *instance* of that class. The type of a slot value may be simple types, such as integer, float, Boolean, and string. For example, in Figure 3.4, the value type of the slot “*NumberOfNodes*” of the class “*Cluster*” is an integer. The type of a slot value may also be an instance of a class. For example, in Figure 3.4, the value type of the slot “*CPU*” of the class “*Cluster*” is an instance of the class “*CPU*”. A class may be extended from another class and inherit all the slots of that class. Figure 3.5 shows the Program, Library, and Package classes, the Service, Data, and Storage classes and the CPU, Memory, Harddisk, NIC, and OS classes.

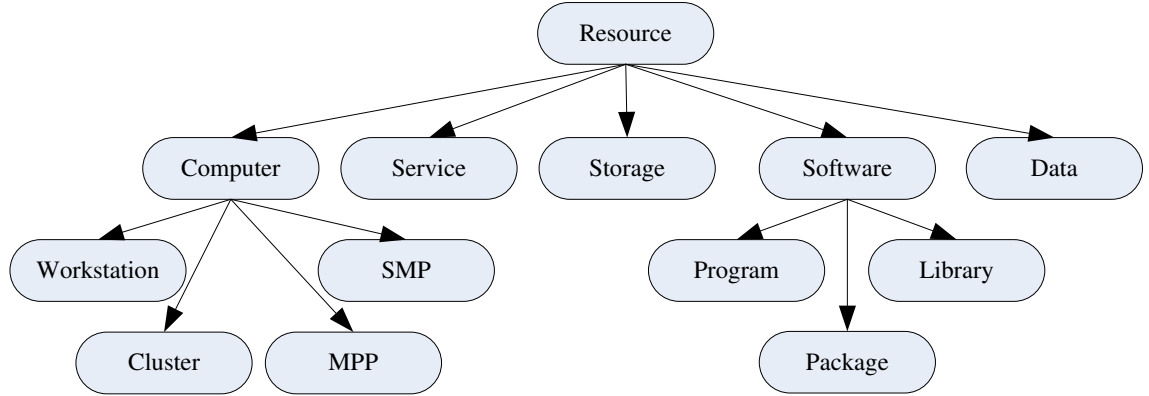


Figure 3.3: The hierarchical relationship among resource classes.

3.4 The Matchmaking Problem

Given a request and a set of resources, we aim to find a set of resources that best match the request. A *request* is a $(n + 1)$ -tuple consisting of n attributes (a_1, a_2, \dots, a_n) and a function of these attributes to be evaluated in the context of resources, i.e., $request = [a_1, a_2, \dots, a_n, f(a_1, a_2, \dots, a_n)]$. An attribute of a request is a mapping from an attribute name to an attribute expression. A *resource* is an m -tuple consisting of m attributes (a_1, a_2, \dots, a_m) . An attribute of a resource is a mapping from an attribute name to an attribute value. The resource that returns the largest value of function f is the one that best matches the request. Figure 3.6 shows the input and output of the matchmaking problem.

Attribute names should be constructed according to the corresponding resource ontologies. The rules for constructing the attribute name for an attribute are:

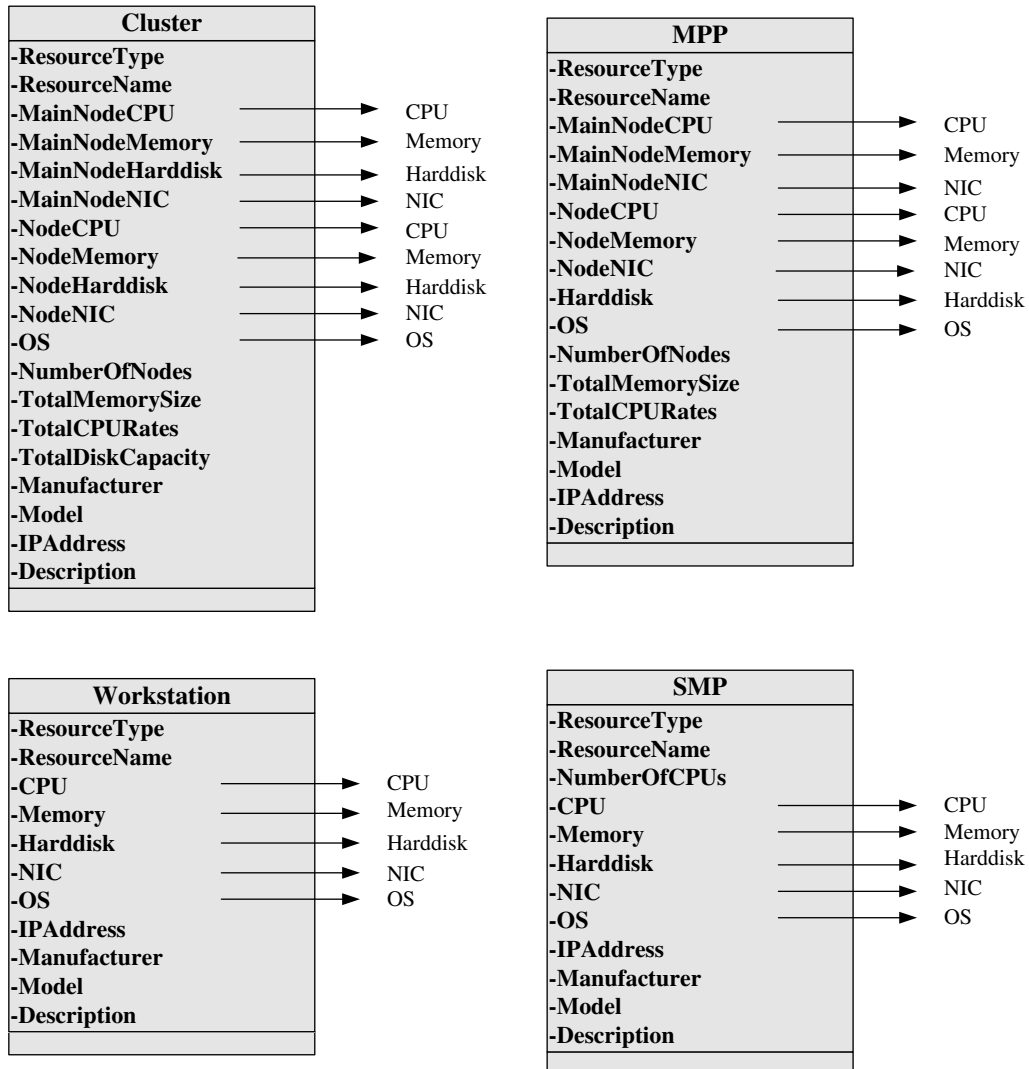


Figure 3.4: The Workstation, Cluster, MPP, and SMP classes. The arrows to the right of some slots indicate that the values of these slots are instances of other classes.

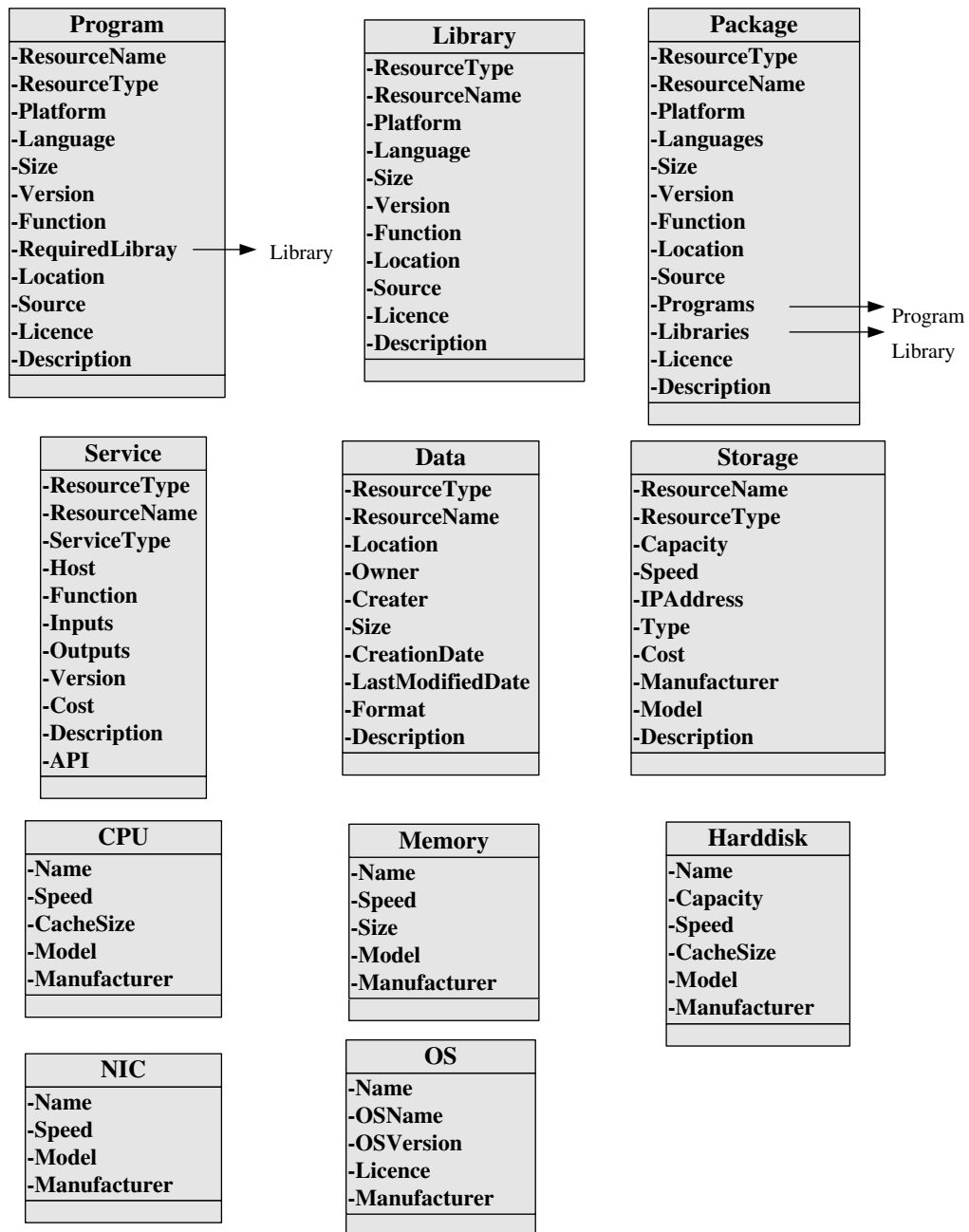


Figure 3.5: Program, Library, and Package classes (top). Service, Data, and Storage classes (middle). CPU, Memory, Harddisk, NIC, and OS classes (bottom).

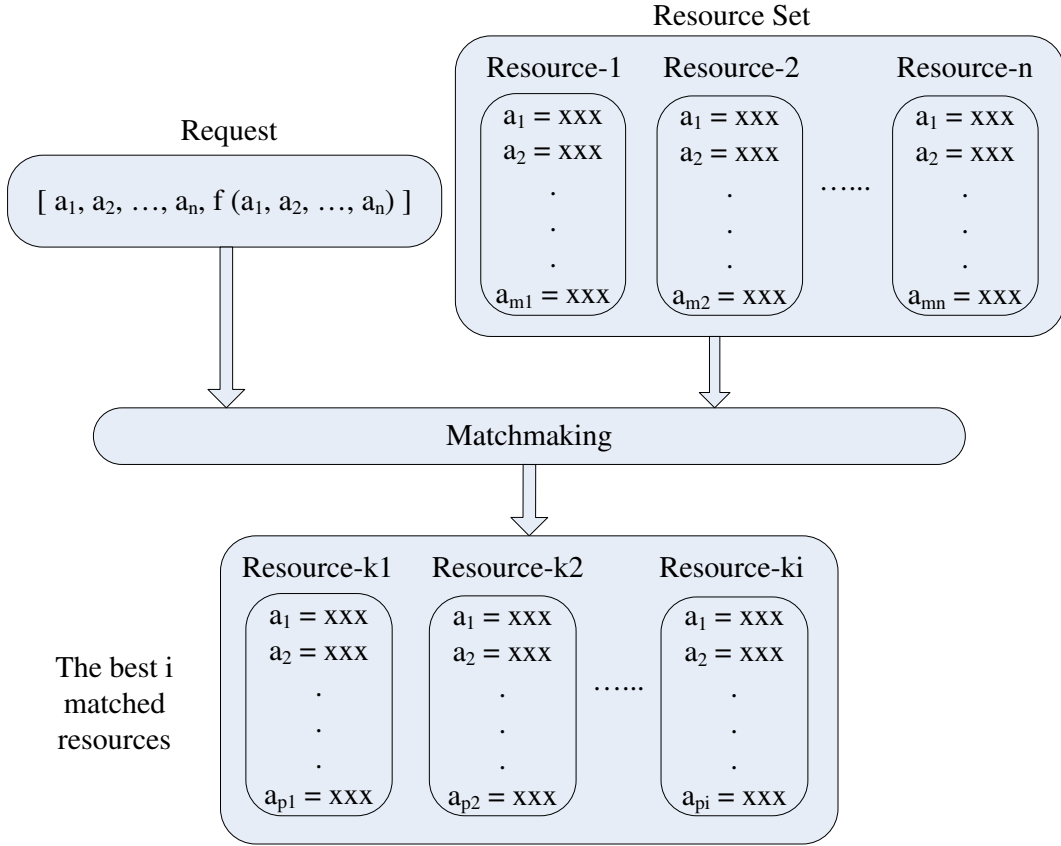


Figure 3.6: The input and output of the matchmaking problem.

1. If the attribute refers to a slot of the resource class, the attribute name is the slot name. For example, for a cluster, the attribute name for the total CPU rates is “*TotalCPURates*” according to the structure of the “*Cluster*” class in Figure 3.4.
2. If a slot of the resource class refers to an instance of other class and the attribute refers to a slot of this instance, the attribute name is the combination of the two slot names connected by ‘.’. For example, in Figure 3.4 the “*NodeMemory*” slot of the “*Cluster*” class refers to an instance of the “*Memory*” class, and in Figure 3.5 the

“*Memory*” class has the “*Size*” slot, so the attribute name for the node memory size is “*NodeMemory.Size*”.

The attribute names of a workstation and three clusters in Figure 3.7 are constructed according to the above rule. For a request = $[a_1, a_2, \dots, a_n, f(a_1, a_2, \dots, a_n)]$, the function f is an expression that is the combination of attribute expressions $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ through mathematical and/or logical operators, where $f_1(a_1)$, $f_2(a_2)$, ..., and $f_n(a_n)$ are to be evaluated in the context of the corresponding attributes of the resource.

We define three types of matching functions. A matching function f could consist of Boolean expressions and return a Boolean constant (“true”, 1 or “false”, 0), see Figure 3.8(a). f could also consist of arithmetic expressions and return a positive real number, see Figure 3.8(b). We also allow f to consist of fuzzy expressions and return a fuzzy number in $[0,1]$, as shown in Figure 3.8(c). The higher the matching degree, the better a request is satisfied.

The expression for function f may involve:

1. Boolean expressions can be combined with the use of Boolean operators “&” and/or “|”. Figure 3.8(a) is an example for this case.
2. Arithmetic expressions can be combined with the use of arithmetic operators, such as “+”, “-”, “*”, and “/”. Figure 3.8(b) is an example for this case.

<pre> Workstation-110{ ResourceType=Workstation ResourceName=workstation-110 Manufacturer=Dell Model=Dimension 8200 IPAddress=xin.cs.ucf.edu CPU.Speed=2.4 GHz Memory.Size=1024 MB NIC.Speed=100 Mbps Harddisk.Capacity=80 GB ... } </pre>	<pre> Boticelli{ ResourceType=Cluster ResourceName=Boticelli IPAddress=botticelli.cs.ucf.edu NodeMemory.Size=3 GB NodeCPU.Speed=1.6 GHz NodeHarddisk.Capacity=80 GB NumberOfNodes=44 TotalMemorySize=132 GB TotalCPURates=140.8 GHz TotalDiskCapacity=3520 GB ... } </pre>
<pre> Hector{ ResourceType=Cluster ResourceName=Hector IPAddress=hector.cs.ucf.edu NodeMemory.Size=1 GB NodeCPU.Speed=2.4 GHz NodeHarddisk.Capacity=40 GB NumberOfNodes=64 TotalMemorySize=64 GB TotalCPURates=153.6 GHz TotalDiskCapacity=2560 GB ... } </pre>	<pre> Bond{ ResourceType=Cluster ResourceName=Bond IPAddress=bond.cs.ucf.edu NodeMemory.Size=4 GB NodeCPU.Speed=3.6 GHz NodeHarddisk.Capacity=120 GB NumberOfNodes=32 TotalMemorySize=128 GB TotalCPURates=115.2 GHz TotalDiskCapacity=3840 GB ... } </pre>

Figure 3.7: Examples of resource instances: a workstation and three clusters.

3. Fuzzy expressions can be combined with the use of fuzzy operators “&&”. The evaluation result of multiple fuzzy numbers connected by “&&” are the average of these fuzzy numbers. Figure 3.8(c) is an example of this case.

```

Request1 {
    f = ( ResourceType == "Cluster" ) &
        ( NodeCPU.Speed >= 1.6 GHz ) &
        ( if ( NodeMemory.Size < 2 GB ) then ( NumberOfNodes > 30 )
          elseif ( NodeMemory.Size < 3 GB ) then ( NumberOfNodes > 20 )
          else ( NumberOfNodes > 10 ) )
}
(a)

Request2 {
    f = ( ResourceType == "Cluster" ) &
        ( ( TotalMemorySize/40 GB ) + ( TotalCPURates/50 GHz ) +
          ( TotalDiskCapacity/30 GB ) )
}
(b)

Request3 {
    f = ( ResourceType == "Cluster" ) &
        ( NodeMemory.Size >= 1 GB ) &
        ( NodeCPU.Speed >= 1.6 GHz ) &
        ( NodeHarddisk.Capacity >= 30 GB ) &
        ( TotalDiskCapacity > 40 GB ) &
        (
            ( if ( NodeMemory.Size > 4 GB ) then 1
              else ( NodeMemory.Size / 4 GB ) ) &&
            ( if ( NumberOfNodes > 40 ) then 1
              else ( NumberOfNodes / 40 ) )
        )
}
(c)

```

Figure 3.8: Boolean, arithmetic, and fuzzy requests.

4. A Boolean expression can be combined with an arithmetic expression or a fuzzy expression through Boolean operator “&”. If the Boolean expression returns 1, they are evaluated to the value returned by the arithmetic expression or the fuzzy expression. If the Boolean expression returns 0, they are evaluated as 0. Figure 3.8(b) and Figure 3.8(c) are examples for this case.

5. Expressions are combined through “*if*”, “*then*”, and “*else*” constructs. Figure 3.8(b) and Figure 3.8(c) are examples for this case.

Figure 3.9 shows the grammar of the resource in BNF form. Figure 3.10 shows the grammar of the request in BNF form. To evaluate f of Figure 3.8(a) in the context of the three clusters in Figure 3.7, cluster *boticelli*, *bond*, and *hector* return 1. To evaluate f of Figure 3.8(b) in the context of the cluster *boticelli* in Figure 3.7, “ $f = 1 \& (132/40) + (140.8/50) + (3520/30) = 123.45$ ”. To evaluate f in the context of the cluster *hector* in Figure 3.7, “ $f = 1 \& (64/40) + (153.6/50) + (2560/30) = 90$ ”. To evaluate f in the context of the cluster *bond* in Figure 3.7, “ $f = 1 \& (128/40) + (115.2/50) + (3480/30) = 121.5$ ”. The cluster that best matches the request is *boticelli*. To evaluate f of Figure 3.8(c) in the context of the cluster *boticelli* in Figure 3.7, “ $f = 1 \& 1 \& 1 \& 1 \& 1 \& (3/4) \&\& 1 = 0.875$ ”. To evaluate f in the context of the cluster *hector* in Figure 3.7, “ $f = 1 \& 1 \& 1 \& 1 \& 1 \& (1/4) \&\& 1 = 0.625$ ”. To evaluate f in the context of the cluster *bond* in Figure 3.7, “ $f = 1 \& 1 \& 1 \& 1 \& 1 \& 1 \&\& 1 = 1$ ”. The cluster best matching the request is *bond*.

```

S ::= <Resource>
<Resource> ::= <Attribute> | <Resource>; <Attribute>
<Attribute> ::= <AttributeName> = <AttributeValue>
<AttributeName> ::= <String>
<AttributeValue> ::= <String>

```

Figure 3.9: BNF grammar for the resource.

```

S ::= <Request>
<Request> ::= RequesterName=<RequesterName>; RequestID=<RequestID>;
    CardinalityThreshold=<CardinalityThreshold>;
    MatchingDegreeThreshold=<MatchingDegreeThreshold>; RequestFunction=<RequestFunction>
<RequesterName> ::= <String>
<RequestID> ::= <String>
<CardinalityThreshold> ::= <PositiveInteger>
<MatchingDegreeThreshold> ::= <PositiveRealNumber>
<RequestFunction> ::= <BooleanExpression> | <FuzzyExpression> | <ArithmeticExpression>

<BooleanExpression> ::= <BooleanExpression> <BooleanOperator> <BooleanExpression>
    | (<BooleanExpression>) | <AtomicBooleanExpression> | <ConditinalBooleanExpression>
<BooleanOperator> ::= <AND> | <OR>
<AND> ::= &
<OR> ::= |
<AtomicBooleanExpression> ::=
    <AttributeName> <RelationalOperator> <Value> | <PredefinedBooleanFunction>
<AttributeName> ::= <String>
<RelationalOperator> ::= > | < | >= | <= | ==
<Value> ::= <String>
<PredefinedBooleanFunction> ::= <KeywordMatchingFunction>
<KeywordMatchingFucntion> ::= KeywordMatching(<AttributeName>,<KeywordList>)
<KeywordList> ::= "<String>"
<ConditinalBooleanExpression> ::= if ( <BooleanExpression> ) then ( <BooleanExpression> )
    <ElseifBooleanClause> else ( <BooleanExpression> )
<ElseifBooleanClause> ::= <EmptyString>
    | <ElseifBooleanClause> elseif ( <BooleanExpression> ) then ( <BooleanExpression> )

<FuzzyExpression> ::= <FuzzyExpression> <FuzzyOperator> <FuzzyExpression>
    | (<FuzzyExpression>) | <FuzzyFunction>
    | <BooleanExpression> <BooleanOperator> <FuzzyExpression>
    | <FuzzyExpression> <BooleanOperator> <BooleanExpression>
<FuzzyOperator> ::= &&
<FuzzyFunction> ::= <AtomicFuzzyFunction> | <ConditinalFuzzyExpression>
<AtomicFuzzyFunction> ::= <AttributeName> <ArithmeticOperator> <Value>
<ArithmeticOperator> ::= + | - | * | /
<ConditinalFuzzyExpression> ::=
    if ( <BooleanExpression> ) then ( <FuzzyFunction> ) <ElseifFuzzyClause> else ( <FuzzyFunction> )
<ElseifFuzzyClause> ::= <EmptyString>
    | <ElseifFuzzyClause> elseif ( <BooleanExpression> ) then ( <FuzzyFunction> )

<ArithmeticExpression> ::= <ArithmeticExpression> <ArithmeticOperator> <ArithmeticExpression>
    | (<ArithmeticExpression>) | <BooleanExpression> <BooleanOperator> <ArithmeticExpression>
    | <ArithmeticExpression> <BooleanOperator> <BooleanExpression> | <AttributeName> | <Value>

```

Figure 3.10: BNF grammar for the request.

3.5 A Case Study: the Matchmaking Service in BondGrid

In this section we describe our work on a matchmaking service in BondGrid and a set of performance studies.

3.5.1 The Matchmaking Service

We implemented a matchmaking service in BondGrid. The matchmaking service has a knowledge base that holds the resource advertisements from providers. Requests from consumers are evaluated with resource advertisements in the knowledge base. The matchmaking service consists of a message handler, a service manager, and a matchmaking engine. The message handler is responsible for the communication between the matchmaking service and other entities in the system. The service manager provides a GUI for monitoring the matchmaking engine and the interactions between matchmaking service and other services. The matchmaking engine handles the matchmaking requests submitted to the matchmaking service.

Providers advertise their resources as instances of corresponding resource classes. Figure 1.2 and Figure 1.3 show the XML specification for instances and classes, respectively. A request specification includes a matchmaking function and possibly two additional constraints, a *cardinality threshold* and a *matching degree threshold*. The element *CardinalityThreshold* specifies how many resources are expected to be returned by the matchmaking service. The

element *MatchingDegreeThreshold* specifies the least matching degree of one of resources returned by the service. Figure 3.11 shows the request specification. Figure 3.12, 3.13, and 3.14 show the request of Figure 3.8(a), 3.8(b), and 3.8(c) respectively according to the request specification.

```
<?xml version="1.0" encoding="UTF-8"?>

<Request>
  <RequesterName> a name </RequesterName>
  <RequestID> an id </RequestID>
  <CardinalityThreshold>
    a positive integer
  </CardinalityThreshold>
  <MatchingDegreeThreshold>
    a positive real number
  </MatchingDegreeThreshold>
  <RequestFunction>
    a request function
  </RequestFunction>
</Request>
```

Figure 3.11: XML specification for a request.

The matchmaking service executes a matchmaking algorithm for each request sent by the requester. The input of the algorithm is the request and the resource instances stored in the knowledge base of the matchmaking service. The output of the algorithm is a number of resources ranked according to their matching degrees with the request. Let n denote the *CardinalityThreshold* specified by the request. The matchmaking algorithm returns the resources that have the n largest matching degrees with the request to the requester. Figure 3.15 shows the pseudo code of the matchmaking algorithm. Generally, a matchmaking

```

<?xml version="1.0" encoding="UTF-8"?>

<Request>

  <RequesterName>xin</RequesterName>
  <RequestID>id_001</RequestID>
  <CardinalityThreshold>5</CardinalityThreshold>
  <MatchingDegreeThreshold>1</MatchingDegreeThreshold>
  <RequestFunction>
    <![CDATA[
      ( ResourceType == "Cluster" ) &
      ( NodeCPU.Speed >= 1.6 GHz ) &
      ( if ( NodeMemory.Size < 2 GB ) then ( NumberOfNodes > 30 )
        else if ( NodeMemory.Size < 3 GB ) then ( NumberOfNodes > 20 )
        else ( NumberOfNodes > 10 )
      )
    ]]>
  </RequestFunction>

</Request>

```

Figure 3.12: A Boolean matching function.

service maintains a knowledge base with a large number of resource instances. Performing an exhaustive matchmaking involving all resources in the knowledge base is very expensive for large knowledge bases. In a modified matchmaking algorithm shown in Figure 3.16, the algorithm finishes searching the knowledge base when $k * n$ (where k is a constant) resources are found with the required matching degrees (not less than the matching degree threshold). The searching process starts from a random position in the knowledge base to avoid hitting the same instances repeatedly, and continues circularly past the end of the knowledge base, up to the initial search starting point.

```

<?xml version="1.0" encoding="UTF-8"?>

<Request>

  <RequesterName>xin</RequesterName>
  <RequestID>id_003</RequestID>
  <CardinalityThreshold>5</CardinalityThreshold>
  <MatchingDegreeThreshold>40</MatchingDegreeThreshold>
  <RequestFunction>
    <![CDATA[
      ( ResourceType == "Cluster" ) &
      ( ( TotalMemorySize/40 GB ) + ( TotalCPURates/50 GHz )
        + ( TotalDiskCapacity/30 GB ) )
    ]]>
  </RequestFunction>

</Request>

```

Figure 3.13: An arithmetic matching function.

3.5.2 Performance Measurements

We evaluated the performance of the matchmaking service using two systems, each with a 1.8 GHz Pentium IV processor and 1 GB of physical memory. The matchmaking service was running under Linux on one of the systems, while the client (requester) was running on the other system under Windows XP. The two machines were connected to the same hub, so the communication time was smaller than in real applications. We conducted performance studies regarding the variation of the response time with the size of the knowledge base. The response time includes the transmission time between the client and the server, the time for the function evaluation, and the knowledge base access time. The resolution of time

```

<?xml version="1.0" encoding="UTF-8"?>

<Request>

  <RequesterName>xin</RequesterName>
  <RequestID>id_002</RequestID>
  <CardinalityThreshold>5</CardinalityThreshold>
  <MatchingDegreeThreshold>0.5</MatchingDegreeThreshold>
  <RequestFunction>
    <![CDATA[
      ( ResourceType == "Cluster" ) &
      ( NodeMemory.Size >= 1 GB ) &
      ( NodeCPU.Speed >= 1.6 GHz ) &
      ( NodeHarddisk.Capacity >= 30 GB ) &
      ( TotalDiskCapacity > 40 GB ) &
      (
        ( if ( NodeMemory.Size > 4 GB ) then 1
          else ( NodeMemory.Size / 4 GB ) ) &&
        ( if ( NumberOfNodes > 40 ) then 1 else ( NumberOfNodes / 40 ) )
      )
    ]]>
  </RequestFunction>

</Request>

```

Figure 3.14: A fuzzy matching function.

is one millisecond. We randomly generated knowledge bases holding different numbers of resource instances with uniformly distributed resource types. We run each case ten times, each time with the same set of randomized resources. We calculated the average value and 95% confidence interval of the response time over 10 runs. We used three requests (shown in Figure 3.12, 3.13, and 3.14) with a different number of operators for the evaluation of the request function.

```

MATCHMAKING ALGORITHM
INPUT request req, a finite set of resource instances rs
OUTPUT a finite set of candidate resource instances cs
BEGIN
  cs =  $\Phi$ 
  n = req.CardinalityThreshold
  m = req.MatchingDegreeThreshold
  FOR each resource r in rs
    md = evaluate req.RequestFunction in the context of r
    IF (md>0) AND (md>=m)
      add r into cs
    END IF
  END FOR
  sort items in cs according to their matching degrees
  keep the items in cs that have the highest n matching degrees and remove the rest
END

```

Figure 3.15: The original matchmaking algorithm performs an exhaustive database search.

BondGrid uses Protégé knowledge base [101] to store instances. A Protégé knowledge base can be stored in a local file or a database. If a knowledge base stored in a local file holds more than 50,000 frames (classes and instances), significant time (about 1 minute) is needed to load the instances from the file to the main memory and a large physical memory ($> 1GB$) is required. If a knowledge base is stored in a database, these limitations are removed due to caching techniques; the required frames are brought into memory as needed and the frames that are no longer needed are removed.

Figure 3.17 shows the response time versus the number of resources when the knowledge base is stored on a local file. As the number of resources increases, the request response time increases as well. When the number of resources is less than 3000, the request response time

```

MATCHMAKING ALGORITHM
INPUT request req, a finite set of resource instances rs
OUTPUT a finite set of candidate resource instances cs
BEGIN
  cs =  $\Phi$ 
  n = req.CardinalityThreshold
  m = req.MatchingDegreeThreshold
  FOR each resource r in rs from a random beginning position
    md = evaluate req.RequestFunction in the context of r
    IF (md>0) AND (md>=m)
      add r into cs
    ENDIF
    IF ( the size of the candidate set > k * n )
      break
    ENDIF
  END FOR
  sort items in cs according to their matching degrees
  keep the items in cs that have the highest n matching degrees and remove the rest
END

```

Figure 3.16: The modified matchmaking algorithm performs a restricted database search; it stops when the cardinality of a set of resources that match the request reaches $k * n$.

is less than 5 seconds, which is quite acceptable. When the number of resources exceeds 3000, the response time increases sharply. In this case the physical memory of the machine running the service is insufficient and leads to frequent paging. Figure 3.17 also shows that three requests with different levels of complexity of the evaluation function have similar response time. This indicates to us that the complexity of a request has small impact on the efficiency of the algorithm, and the knowledge base access time is the major contributor to the response time. Figure 3.18 amplifies a part of Figure 3.17. It indicates that when the number of resources in the knowledge base is less than 3000, the response time increases

linearly with the number of resources. The three requests take slightly different response times because their request functions have different complexities.

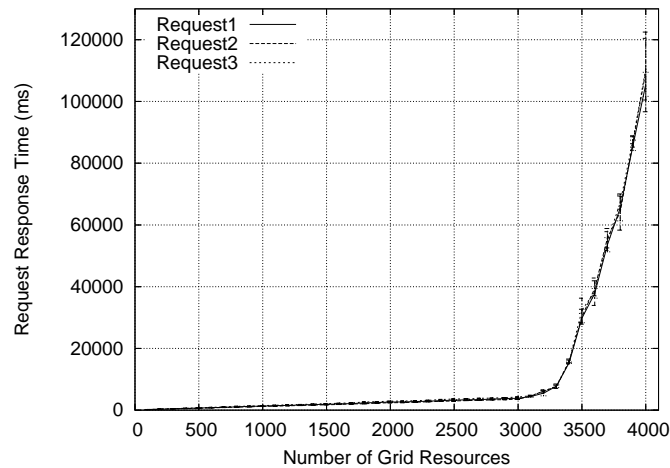


Figure 3.17: Response time vs. number of resources for three requests of different complexities when the knowledge base is a local file.

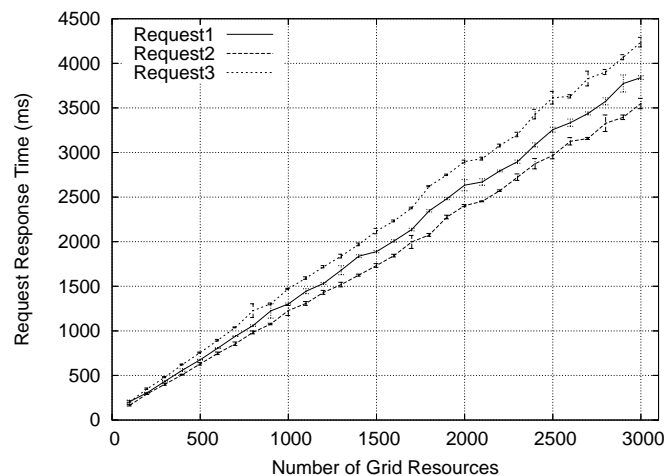


Figure 3.18: Response time vs. number of resources for three requests when the knowledge base is a local file.

Figure 3.19 shows the response time over various number of resources when the knowledge base is stored in a database. We used a MySQL 4.0 database deployed on the machine holding the matchmaking service. As the number of resources increases, the response time increases smoothly. When this number is less than 3000, the response time is larger than that of Figure 3.17. When the number of resources is larger than 3000, the request response time is much smaller than that of Figure 3.17. Using database to store knowledge bases turns out to be a better choice for large knowledge bases as will be the case of realistic applications.

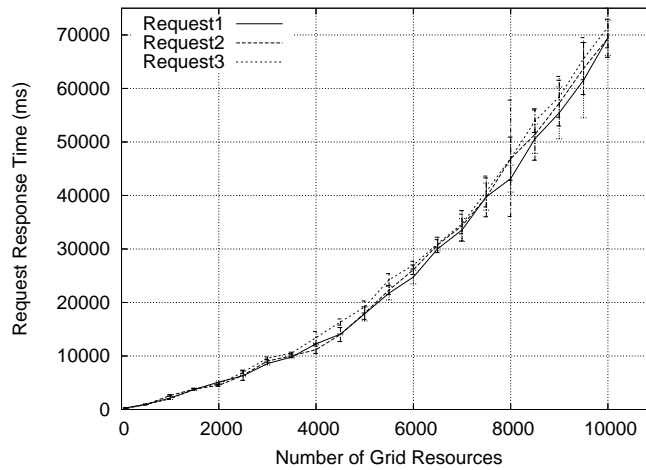


Figure 3.19: Response time vs. number of resources for three requests when the knowledge base is stored as a database.

Figure 3.20 shows the response time over various number of resources for Request3 from Figure 3.14 when the knowledge base is stored in a database and the matchmaking service is running the modified matchmaking algorithm (see Figure 3.16). The response time is considerably lower than that of Figure 3.19. Figure 3.21 shows the corresponding average

matching degree over various number of resources for the same run, where the average matching degree is the average of the matching degrees of the returned resources. As k increases, the average matching degree increases as well. When $k = 3$, the average matching degree reaches 1 most of the time, which indicates that the modified matchmaking algorithm is likely to produce optimal results when $k = 3$. The modified algorithm greatly improves the efficiency of matchmaking without sacrificing the quality of the matchmaking results.

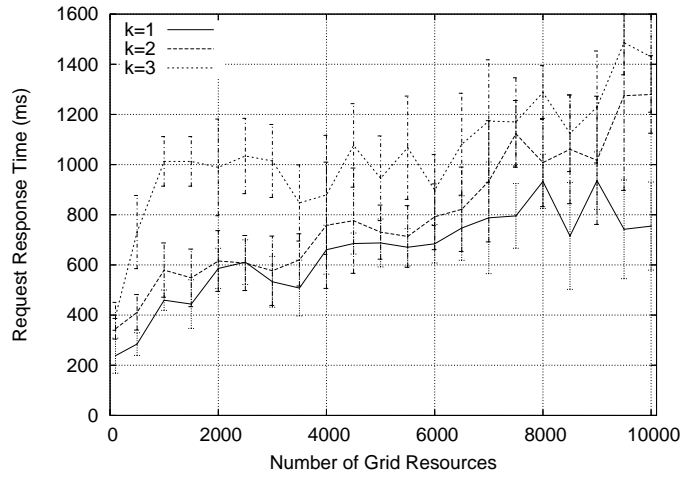


Figure 3.20: Response time vs. number of resources for Request3 when the knowledge base is stored as a database. The matchmaking service runs the modified matchmaking algorithm.

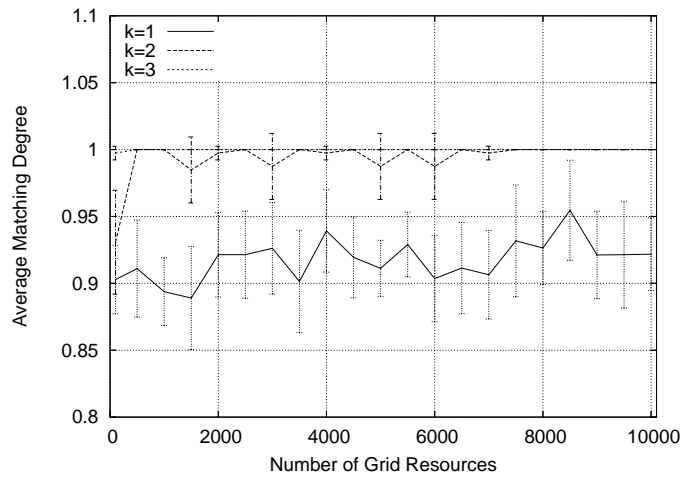


Figure 3.21: Average matching degree vs. number of resources for Request3 when the knowledge base is stored as a database. The matchmaking service runs the modified matchmaking algorithm. The average matching degree is the average of the matching degrees of the resources returned by the service.

CHAPTER 4

A MACRO-ECONOMIC RESOURCE ALLOCATION MODEL

Sharing scarce resources is an enduring problem. In this chapter, we discuss a macro-economic model suitable for resource sharing when computing and communication resources as well as consumers belong to different administrative domains. The formalism captures traditional concepts such as consumer utility and provider revenues. The model allows us to analyze the effect of different pricing strategies upon measures of performance important for the consumers (satisfaction and utility) and providers (revenue and acceptance ratios). We report on simulation experiments which confirm the important role played by brokers and the mechanisms used by them to achieve societal objectives.

4.1 Introduction and Motivation

As we become increasingly more dependent upon digital information, we pay more attention to computing and communication resources and the mechanisms to share them efficiently and

in an equitable manner. When all resources are controlled by a single agent we have adequate means to tightly control resource allocation using priority scheduling and rejecting consumers when the system is overloaded. Computational, service, and data grids, peer-to-peer systems, and ad-hoc wireless networks are examples of *open systems* where individual members of the community contribute computing cycles, storage, services, and communication bandwidth to the pool of resources available to the entire community and resources and consumers belong to different administrative domains. In this case it is difficult to devise resource allocation schedules and there is no central authority to enforce any such schedules.

Market-oriented economies have proved their advantages over alternative means to control and manage resource allocation in social as well as man-made systems. It seems reasonable to adapt some of the successful ideas of economical models to resource allocation in large-scale computing systems and to study market-oriented resource allocation algorithms. Economic models are attractive for resource providers, beneficial for the consumers of resources, and have societal benefits. Providers benefit from contributing their resources and are encouraged to re-invest some of their profits into additional resources; consumers enjoy fair treatment as the resource allocation is governed by rules that do not depend on the individual consumer. Moreover, providers and consumers can make their own decisions to maximize their utility and/or profits. When system-centric scheduling policies are replaced by consumer-centric policies the system becomes more responsive to consumer needs and important problems are solved with higher priority. Economic models allow resource allocation and management to be more efficient, the demand and supply is regulated through economic

activities and fewer resources are wasted, while excess capacity and overloading are averaged over a very large number of providers and consumers. Resources, e.g., CPU cycles, main memory, secondary storage, and network bandwidth/latency, are treated uniformly and this can facilitate the design of large-scale distributed systems, such as computational grids. The system is more scalable and decision-making is distributed. In an economic model all participants are considered self-interested. The resource providers are trying to maximize their revenues. The consumers want to obtain the maximum possible resources for the minimum possible price. The large number of participants makes one-to-one negotiations expensive and unproductive.

This process of migration to market-oriented management strategies requires a major rethinking of our philosophy for allocation of computing and communication resources. Some of the steps required by this transition are:

- Extended the mechanisms for resource management from a single controlling authority to large-scale systems consisting of multiple administrative domains. This transition requires new ideas to bridge the gap between micro- and macro-economics of computing resources. Negotiation protocols for resource allocation and for contract renegotiations, and agents with a global view of the computing economy, are only some of the new elements of this macro-economic landscape.
- Formalize traditional concepts such as consumer satisfaction and utility. Informally, utility quantifies the benefits obtained as the result of being granted a certain amount

of resources. Utility based resource allocation models have proved their potential in a different context, e.g., when the only resource is the radio bandwidth, the size of the population is limited, and each participant has a unique role (e.g., is a consumer) [8]. The heterogeneity of a large-scale distributed system, the large spectrum of resources and demands placed upon these resources, the scale of the system, the autonomy of individual resource providers, and the dual role of individual actors, as consumer of some resources and provider for others, add complexity to the models.

- Transform imperative requests into *elastic* ones which reflect the level of utility. Rather than requesting precisely 100 nodes of a cluster to carry out a computation, the user should specify how useful are 30, 40, 50, or 80 processors. At the same time, we have to be concerned with the possibility of failure and the need for consumers to seek alternative ways to achieve their goals when resources they request at a given time are unavailable. In the long run, we foresee an adaptive and intelligent user behavior based upon the idea that in the general case the same goal may be achieved through different means. While scripts provide an efficient way to coordinate execution of multiple tasks we have to consider dynamic coordination models which require possible changes of the process and case description to adapt to the very dynamic grid environment.
- Devise multiple pricing strategies. There is no reason to optimize one's algorithms if the resources are free or available based upon an arbitrary allocation, thus pricing strategies are likely to play a major role in resource management.

- Include within the social computing landscape middlemen to mediate access to system resources. The role of a broker is to reconcile the selfish objectives of individual resource providers and consumers with some global, societal objectives, e.g., to maximize the resource utilization of the system, or to maximize the average consumer satisfaction. The existence of multiple brokers forces each one of them to be socially responsible and act in a fair and equitable manner.

In this chapter we address these concerns and introduce a macro-economic model based upon utility and price functions and discuss several pricing strategies. Different utility functions can be considered and we concentrate on sigmoids. Sigmoid functions have some desirable properties and have been used in many economic models and in some other areas, as we discuss in Section 4.3. We define a measure of consumer's satisfaction which takes into account the utility resulting from resource consumption and the price paid by the consumer. We introduce three pricing policies and investigate the effect of several parameters upon critical measures of performance for producers and consumers. The pricing policies are affected by the relationship between the amount of resources required and the total amount we pay for them, as well as the overall state of the system. We analyze the case when the price per unit is constant regardless of the amount of resources consumed (linear pricing); pricing to encourage consumption, i.e., the more resources are used the lower becomes the average unit price (sub-linear pricing); and pricing to discourage consumption, i.e., the more resources are used the higher becomes the average unit price (super-linear pricing). We also analyze the effect of resource abundance upon pricing strategies.

In our model each resource is characterized by a vector with several components. As expected, even with a set of simplifying assumptions, the models are extremely complex and can only be evaluated through simulation.

We show that brokers play a very important role and can influence positively the market. We also show that consumer satisfaction does not track the consumer utility, these two important performance measures for consumers behave differently under different pricing strategies. Pricing strategies also affect the revenues obtained by providers, as well as, the request acceptance ratio.

4.2 Background and Related Work

Economic models have been proposed independently in several areas of distributed systems. In certain cases this was motivated by the desire of companies to maximize their profit in a market of computational resources based on real currency. Examples of these types of systems are IBM E-Business On Demand [61], HP Adaptive Enterprise [58] and Sun Microsystems pay-as-you-go [118]. Several startup companies such as Entropia, ProcessTree, Popular Power, Mojo Nation, United Devices, and Parabon have also proposed innovative market based computational models.

In many other cases however, the economic models were introduced because of the perceived benefits which economic models of interaction such as auctions, negotiation, bartering

or tendering can offer to resource allocation in distributed systems. These systems usually rely on a *virtual currency* which has only a loose relation, or none to real money. It was found that using an allocation algorithm based on self-interested participants interacting in a competitive environment leads to more efficient resource allocation. This is true even if the components are part of the same computational project, thus their relationship is essentially cooperative. Virtual currencies allow us to introduce competition in these environments.

The projects in economic model based resource allocation can be categorized based on two major criteria (a) the application domain and the nature of the traded resources and (b) the economic models and algorithms deployed. These two criteria are independent: virtually all the market models can be applied to almost any application domain.

Looking from the point of view of the application domain we find that the goods traded by the economic model can be:

- *Computation*: usually including the processor time, together with the associated RAM memory, hard drive space, and communication bandwidth necessary to perform the computation. These systems are deployed in the context of clusters, parallel and distributed computing, and more recently, grid systems.
- *Storage*: the traded goods are either unstructured hard drive space, or structured storage in distributed databases [5, 116]. The recently emerging field of distributed storage systems such as the Stanford Peers Initiative [35] and GnuNet [55] also falls in this category.

- *Services*: a series of efforts are using economic models for the trading of services on the Web: Java Market [4], JaWS [76], Xenoservers [104] and others.

The other criteria of categorization is the nature of the economic models and algorithms deployed. Most of these algorithms are direct applications of existing economic interaction patterns. Others, such as the Contract Net, are agent based formalizations of interaction patterns such as tendering and subcontracting. We observe, that some algorithms are more popular in distributed systems domain than in the real economy. For instance, Vickrey auctions [126], despite their desirable properties, are counterintuitive for humans, and therefore less popular than English and Dutch auctions. However, Vickrey auctions are quite frequently used in computer based auctions, such as the Google Adwords auction system.

We survey the economic models used for trading computational resources based on [22]:

- *Commodity Market*: resource providers advertise their resource prices and charge users based on the amount of resources used. The pricing policy could be based on a flat fee, the resource usage duration, the subscription, and the demand and supply [86]. Mungi [57], Enhanced MOSIX [3], and Nimrod/G [21] are some of the systems based upon the commodity market model.

- *Posted Price*: this model, used by Nimrod/G [21], is similar to the Commodity Market Model except that it advertises special offers to attract users.

- *Bargaining*: resource owners and users/brokers negotiate with each other until they reach a mutually agreeable price. This model is mostly used in a market that does not have a clear

demand-and-supply relationship and price. Examples of systems include Mariposa [85] and Nimrod/G [21].

- *Tendering/Contract-Net*: first users/brokers advertise their requirement, then resource owners respond with their bids, and at last users/brokers choose a resource owner to use its resource. Mariposa [85] applies this strategy.

- *Auction*: resource owners announce their resources and invite bids, then an auction process is performed with users/brokers, and at last the winner user/broker uses the resource. Different auction policies can be used: 1) English auction; 2) first-price sealed-bid auction; 3) Vickrey auction (second-price sealed-bid); and 4) Dutch auction. In English auction, all bidders are free to increase their bids exceeding other offers; when no bidder is willing to increase the bid, the auction ends and the highest bidder wins. In first-price sealed-bid auction, every bidder submits a sealed-bid and the highest bidder wins. In Vickrey auction, every bidder submits a sealed-bid and the highest bidder wins at the price of the second highest bidder. In Dutch auction, the resource owners start by a high price and continuously decrease the price until a bidder is willing to take the resource at the current price. Spawn [129] and Popcorn [91] use this model.

- *Bid-based Proportional Resource Sharing*: the percentage of resources allocated a user is a function of the user's bid and other users' bids. Rexec/Anemone [31] implements this model.

- *Community/Coalition/Bartering*: a community of resource owners share each other's resources. The resource owners contributing to the community get credits by sharing their

resources. The credit of a resource owner decides how much resources he can get from others. Condor [34], SETI@home [110], and Mojo Nation [87] are based on this model.

- *Monopoly/Oligopoly*: one or a small number of resource owners decide the price and it is not possible to negotiate the price. Nimrod/G [21] embraces this model.

Some systems use more than one strategy. For example, Nimrod/G supports four different models: Commodity Market, Posted Price, Bargaining, and Monopoly / Oligopoly. Arguments that commodities markets are better choices for controlling grid resources than auction strategies are presented in [135, 136] based upon concepts such as price stability, market equilibrium, consumer efficiency, and provider efficiency. An approach to implement automatic selection of multiple negotiation models to adapt to the computation needs and change of resource environment is discussed in [111]. A task-oriented mechanism for measuring the economic value of using heterogeneous resources as a common currency is analyzed in [56]; resource consumers can compare the advantage of participating in a computational grid with the alternative of purchasing their own resources necessary, and resource providers can evaluate the profit of putting their resources into a grid. A comparative analysis of market-based resource allocation by continuous double auctions and by the proportional share protocol versus a conventional round-robin approach is presented in [51]. A game theoretic pricing strategy for efficient job allocation in mobile grids is discussed in [50].

4.3 Basic Concepts

An efficient and fair utilization of the resources can be obtained only through a scheme that gives incentives to the providers to share their resources and that encourages the consumers to maximize the utility of the received resources. A well-tested model for such a scheme is based on an economic model, in which the resources need to be paid for in a real or virtual currency. This model has the advantage of being provably scalable, and we can successfully reuse or adapt the models that govern the economy in our society.

To study possible resource management policies, we have to develop resource consumption models that take into account different, possibly contradictory, views of the benefits associated with resource consumption as well as the rewards for providing resources to the consumer population. Such models tend to be very complex and only seldom amenable to analytical solutions.

In this section we introduce the basic concepts and notations for our model. First, we introduce price, utility, and satisfaction functions; then we present our resource provider-consumer model. To capture the objectives of the entities involved in the computational economy we use: (i) a consumer *utility function*, $0 \leq u(r) \leq 1$, to represent the utility provided to an individual consumer, where r represents the amount of allocated resource; (ii) a provider *price function*, $p(r)$, imposed by a resource provider, and (iii) a consumer *satisfaction function*, $s(u(r), p(r))$, $0 \leq s \leq 1$, to quantify the level of satisfaction; the satisfaction depends on both the provided utility and the paid price.

4.3.1 Price Function

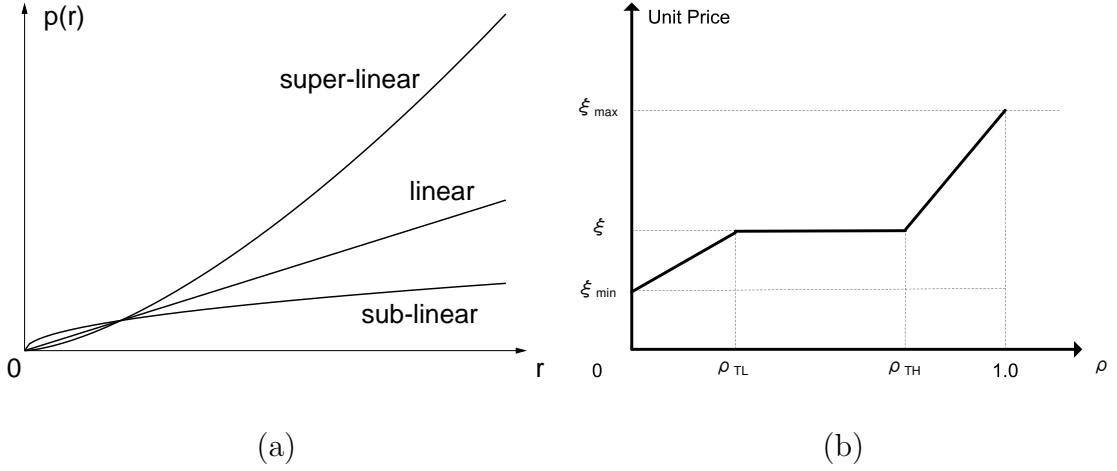


Figure 4.1: (a) Sub-linear, linear, and super-linear price functions. (b) The unit price varies with ρ , the load index of the provider.

We discuss the three pricing functions in Figure 4.1(a). Given the constant, ξ , the three particular pricing functions we choose are:

(a) The price per unit is constant regardless of the amount of resources consumed (linear pricing):

$$p(r) = \xi \cdot r \quad (4.1)$$

(b) Discourage consumption: the more resources are used, the higher becomes the average unit price (super-linear pricing):

$$p(r) = \xi \cdot r^d \quad (4.2)$$

where $d > 1$. For this equation, we use $d = 1.5$ throughout the remainder of the paper.

(c) Encourage consumption: the more resources are used, the lower becomes the average unit price (sub-linear pricing):

$$p(r) = \xi \cdot r^e \quad (4.3)$$

where $e < 1$. For this equation, we use $e = 0.5$ throughout the remainder of the paper.

We also analyze the effect of resource abundance; in this case we define the load index ρ as the ratio of total amount of allocated resources to the capacity of the provider and consider three regions: low, medium, and high load index. We denote the low, medium, and high regions as the interval of $[0, \rho_{TL})$, $[\rho_{TL}, \rho_{TH}]$, and $(\rho_{TH}, 1]$, respectively, as shown in Figure 4.1(b). The pricing strategy for each region is different. We consider two models, *EDL* - *Encourage/Discourage Linear*, and *EDN* - *Encourage/Discourage Nonlinear*. The choice of the ρ_{TL} , ρ_{TH} is basically a policy decision. However, in order to have the desired influence on the system as a whole, the three intervals need to be of a sufficient size. Values such as $\rho_{TL} = 0.49$ and $\rho_{TH} = 0.51$ make the target interval unreasonably small; very low ρ_{TL} and very high ρ_{TH} values make the pricing strategy degenerate into a constant price strategy. The values used throughout this dissertation are $\rho_{TL} = 0.3$ and $\rho_{TH} = 0.7$.

For the first model, the unit price is constant in each region, but different in different regions, as defined in Equation 4.4, and shown in Figure 4.1(b). We introduce three prices, each corresponding to a range of the system load: minimal, ξ_{min} , maximal, ξ_{max} , and ξ , the price corresponding to medium load. For low load the providers use lower prices to encourage resource consumption, but do not lower the price below ξ_{min} . For high load, the providers gradually increase the price, up to ξ_{max} . The choice of the ξ_{min} and ξ_{max} is a matter of

policy. However, too low values for ξ_{min} would make resources basically free for nodes with low utilization, and very high values of ξ_{max} would make resources too expensive. We use the values of $\xi_{max} = 2 \times \xi$, and $\xi_{min} = 0.5 \times \xi$ throughout the remainder of the paper.

$$p(r) = \begin{cases} \left(\xi_{min} + \frac{\rho}{\rho_{TL}}(\xi - \xi_{min}) \right) \cdot r & \text{if } \rho \in [0, \rho_{TL}); \\ \xi \cdot r & \text{if } \rho \in [\rho_{TL}, \rho_{TH}]; \\ \left(\xi + \frac{\rho - \rho_{TH}}{1.0 - \rho_{TH}}(\xi_{max} - \xi) \right) \cdot r & \text{if } \rho \in (\rho_{TH}, 1.0]. \end{cases} \quad (4.4)$$

For the second model, when ρ is low, the provider uses a sub-linear price function; when ρ is high, the provider uses a super-linear price function; otherwise, the provider uses a linear price function, as expressed by Equation 4.5:

$$p(r) = \begin{cases} \xi \cdot r^e & \text{if } \rho \in [0, \rho_{TL}); \\ \xi \cdot r & \text{if } \rho \in [\rho_{TL}, \rho_{TH}]; \\ \xi \cdot r^d & \text{if } \rho \in (\rho_{TH}, 1.0]. \end{cases} \quad (4.5)$$

where $e < 1$ and $d > 1$. The choice of e and d follow similar considerations like the choice of parameters for the EDL model: we need to encourage and discourage the customers, while still maintaining the prices in a justifiable range. In this dissertation we are using the values of $e = 0.5$ and $d = 1.5$, which provide an appropriate range of prices.

4.3.2 Utility Function

The utility function should be a non-decreasing function of r , i.e., we assume that the more resources are allocated to the consumer, the higher the consumer utility is. However, when enough resources have been allocated to the consumer, i.e., some threshold is reached, an increase of allocated resources would bring no improvement of the utility. On the other hand, if the amount of resources is below some threshold the utility is extremely low. Thus, we expect the utility to be a concave function and reach saturation as the consumer gets all the resources it can use effectively. These conditions are reflected by the following equations:

$$\frac{du(r)}{dr} \geq 0, \quad \lim_{r \rightarrow \infty} \frac{du(r)}{dr} = 0 \quad (4.6)$$

For example, if a parallel application could use at most 100 nodes of a cluster, its utility reflected by a utility function does not increase if its allocation increases from 100 to 110 nodes. If we allocate less than 10 nodes then the system may spend most of its time paging and experiencing cache misses and the execution time would be prohibitively high.

Different functions can be used to model this behavior and we choose one of them, a sigmoid:

$$u(r) = \frac{(r/\omega)^\zeta}{1 + (r/\omega)^\zeta} \quad (4.7)$$

where ζ and ω are constants provided by the consumer, $\zeta \geq 2$, and $\omega > 0$. Clearly, $0 \leq u(r) < 1$ and $u(\omega) = 1/2$.

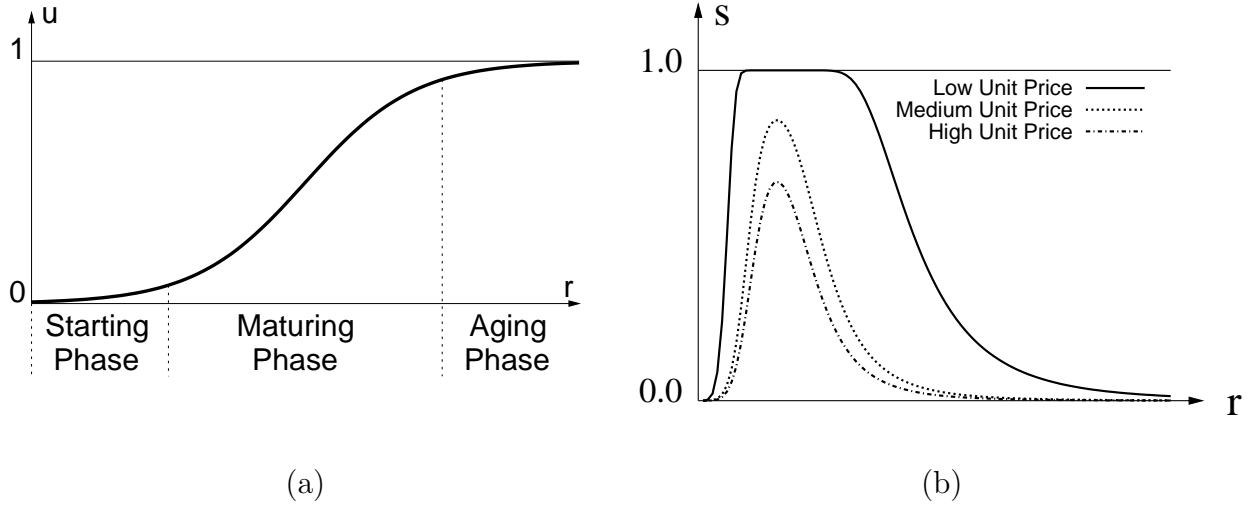


Figure 4.2: (a) A sigmoid is used to model the utility function; a sigmoid includes three phases: the starting phase, the maturing phase, and the aging phase. (b) The satisfaction function for a sigmoid utility function and three linear price functions with low, medium, and high unit price.

A sigmoid is a tilted S-shaped curve that could be used to represent the life-cycles of living, as well as man-made, social, or economical systems. It has three distinct phases: an incipient or starting phase, a maturing phase, and a declining or aging phase, as shown in Figure 4.2(a).

4.3.3 Satisfaction Function

A consumer satisfaction function takes into account both the utility provided to the consumer and the price paid for the resources. For a given utility, the satisfaction function should

increase when the price decreases and, for a given price, the satisfaction function should increase when the utility u increases. These requirements are reflected by Equation (4.8).

$$\frac{\partial s}{\partial p} \leq 0, \quad \frac{\partial s}{\partial u} \geq 0 \quad (4.8)$$

Furthermore, a normalized satisfaction function should satisfy the following conditions:

- the degree of satisfaction, $s(u(r), p(r))$, for a given price $p(r)$, approaches the minimum, 0, when the utility, $u(r)$, approaches 0;
- the degree of satisfaction, $s(u(r), p(r))$, for a given price $p(r)$, approaches the maximum, 1, when the utility, $u(r)$, approaches infinity;
- the degree of satisfaction, $s(u(r), p(r))$, for a given utility $u(r)$, approaches the maximum, 1, when the price, $p(r)$, approaches 0; and
- the degree of satisfaction, $s(u(r), p(r))$, for a given utility $u(r)$, approaches the minimum, 0, when the price, $p(r)$, approaches infinity.

These requirements are reflected by Equations (4.9) and (4.10).

$$\forall p > 0, \lim_{u \rightarrow 0} s(u, p) = 0, \quad \lim_{u \rightarrow \infty} s(u, p) = 1 \quad (4.9)$$

$$\forall u > 0, \lim_{p \rightarrow 0} s(u, p) = 1, \quad \lim_{p \rightarrow \infty} s(u, p) = 0 \quad (4.10)$$

A candidate satisfaction function is [124]:

$$s(u, p) = 1 - e^{-\kappa \cdot u^\mu \cdot p^{-\epsilon}} \quad (4.11)$$

where κ , μ , and ϵ are appropriate positive constants. The satisfaction function based upon the utility function in Equation 4.7 is normalized; given a reference price ϕ we consider also a normalized price function and we end up with a satisfaction function given by:

$$s(u, p) = 1 - e^{-\kappa \cdot u^\mu \cdot (p/\phi)^{-\epsilon}}. \quad (4.12)$$

Because u and p are functions of r , satisfaction increases as more resources are allocated, reaches an optimum, and then declines, as shown in Figure 4.2(b). The optimum satisfaction depends upon the pricing strategy; not unexpectedly, the higher the unit price, the lower the satisfaction.

The 3D surfaces representing the relationship $s = s(r, \xi)$ between satisfaction s and the unit price ξ and amount of resources r for several pricing functions (super-linear, linear, and sub-linear) are presented in Figure 4.3. As we can see from the cut through the surfaces $s = s(r, \xi)$ at a constant ξ when we discourage consumption (super-linear pricing) the optimum satisfaction is lower and occurs for fewer resources; when we encourage consumption (sub-linear pricing) the optimum satisfaction is improved and occurs for a larger amount of resources. These plots reassure us that the satisfaction function has the desired behavior.

4.3.4 Resource Provider-Consumer Model

Consider a system with n providers offering computing resources and m consumers. To simplifying the model, we assume that the two sets are disjoint. Call \mathcal{U} the set of consumers

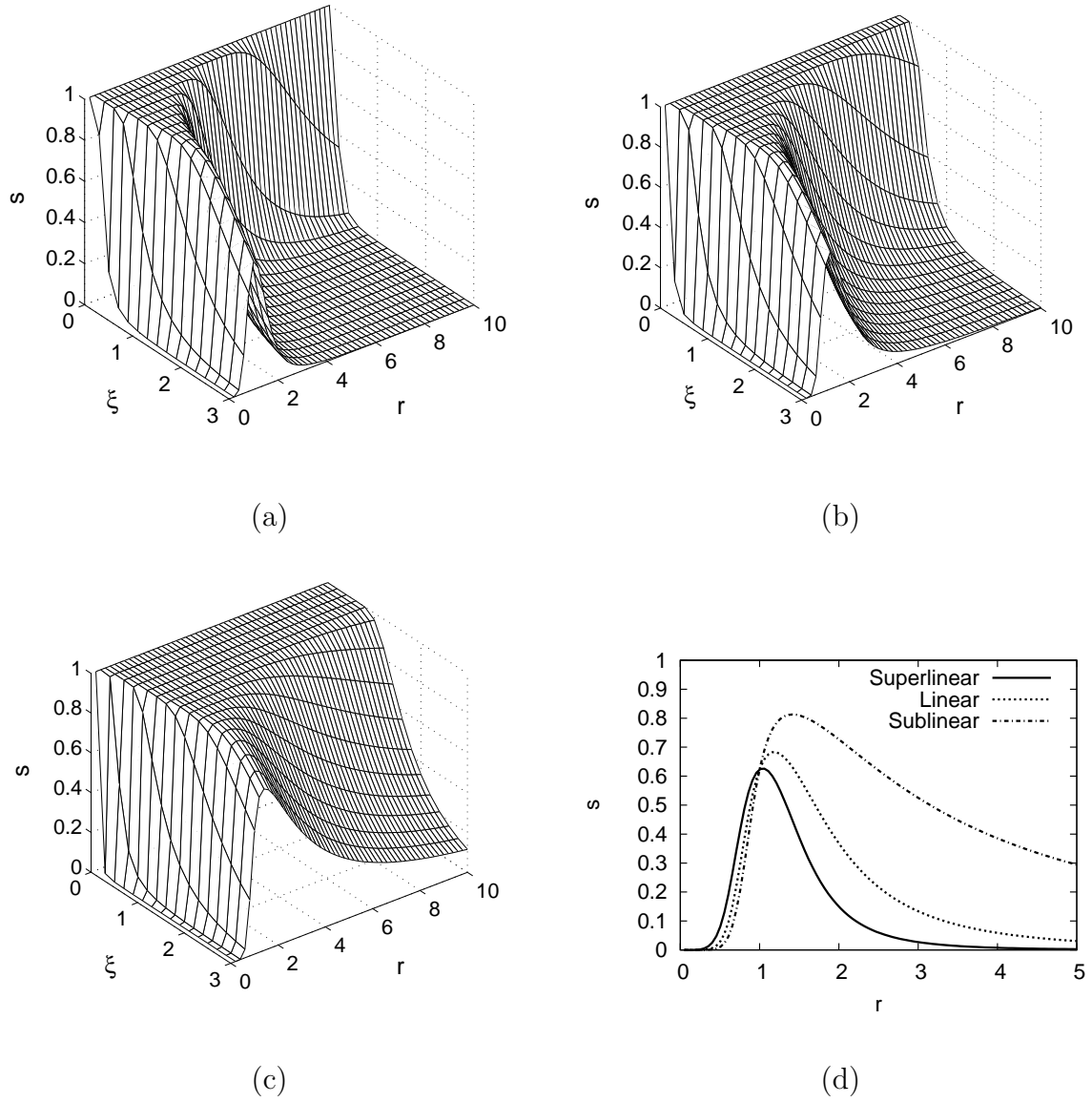


Figure 4.3: The relationship between satisfaction s and the unit price ξ and amount of resources r . The satisfaction function is based on a sigmoid utility function and different price functions: (a) discourage consumption (super-linear); (b) linear; (c) encourage consumption (sub-linear); (d) a cut through the three surfaces at a constant ξ .

and \mathcal{R} the set of providers. The n providers are labeled 1 to n and the m consumers are labeled 1 to m . Consider provider R_j , $1 \leq j \leq n$, and consumer U_i , $1 \leq i \leq m$, that could potentially use resources of that provider.

Let r_{ij} denote the resource (defined below) of R_j allocated to consumer U_i and let u_{ij} denote its utility for consumer U_i . Let p_{ij} denote the price paid by U_i to provider R_j . Let t_{ij} denote the time U_i uses the resource provided by R_j . Let c_j denote the resource capacity of R_j , i.e., the amount of resources regulated by R_j .

The term “resource” here means a vector with components indicating the actual amount of each type of resource:

$$r_{ij} = (r_{ij}^1 \ r_{ij}^2 \ \dots \ r_{ij}^l)$$

where l is a positive integer and r_{ij}^k corresponds to the amount of resource of the k -th type. The structure of r_{ij} may reflect the rate of CPU cycles, the physical memory required by the application, the secondary storage, the number of nodes and the interconnection bandwidth (for a multiprocessor system or a cluster), the network bandwidth (required to transfer data to/from the site), the graphics capabilities, and so on.

The utility of resource of the k -th type provided by R_j for consumer U_i is a sigmoid:

$$u_{ij}^k = u(r_{ij}^k) = \frac{(r_{ij}^k/\omega_i^k)^{\zeta_i^k}}{1 + (r_{ij}^k/\omega_i^k)^{\zeta_i^k}}$$

where ζ_i^k and ω_i^k are constants provided by consumer U_i , $\zeta_i^k \geq 2$, and $\omega_i^k > 0$. Clearly, $0 < u(r_{ij}^k) < 1$ and $u(\omega_i^k) = 1/2$.

The overall utility of resources provided by R_j to U_i is:

- the product over the set of resources provided by R_j , i.e., $u_{ij} = \prod_{k=1}^l u_{ij}^k$, or
- the weighted average over the set of resources provided by R_j , i.e., $u_{ij} = \frac{1}{l} \sum_{k=1}^l a_{ij}^k u_{ij}^k$,
where a_{ij}^k values are provided by consumer U_i and $\sum_{k=1}^l a_{ij}^k = 1$.

Let p_{ij}^k denote the price consumer U_i pays to provider R_j for a resource of type k . The total price for consumer U_i for resources provided by provider R_j is:

$$p_{ij} = \sum_{k=1}^l p_{ij}^k.$$

The total cost for consumer U_i for resources provided by provider R_j is $p_{ij} \times t_{ij}$.

Based on Equation 4.12, we define the degree of satisfaction of U_i for a resource of the k -th type provided by provider R_j as:

$$s_{ij}^k(u_{ij}^k, p_{ij}^k) = 1 - e^{-\kappa_i^k u_{ij}^k \mu_i^k (p_{ij}^k / \phi_i^k)^{-\epsilon_i^k}}, \quad \kappa_i^k, \phi_i^k, \mu_i^k, \epsilon_i^k > 0$$

where κ_i^k , μ_i^k and ϵ_i^k are appropriate positive constants and ϕ_i^k is a reference price.

The overall satisfaction of consumer U_i for resources provided by R_j is:

- the product over the set of resources provided by R_j , i.e., $s_{ij} = \prod_{k=1}^l s_{ij}^k$, or
- the weighted average over the set of resources provided by R_j , i.e., $s_{ij} = \frac{1}{l} \sum_{k=1}^l b_{ij}^k s_{ij}^k$,
where b_{ij}^k values are provided by consumer U_i and $\sum_{k=1}^l b_{ij}^k = 1$.

4.4 The Role of Brokers in the Macro-Economic Model

In this chapter, we concentrate on optimal resource management policies. A policy is optimal when the satisfaction function, which reflects both the price paid to carry out a task and the utility resulting from the completion of the task, reaches a maximum. A broker attempts to operate at or near this optimum.

The role of a broker is to mitigate access to resources. In this chapter, we consider *provider-broker-consumer models* that involve the set of resource providers \mathcal{R} , the set of consumers \mathcal{U} , and broker B . These models assume that a consumer must get all of its resources from a single provider. Brokers have “societal goals” and attempt to maximize the average utility and revenue, as opposed to providers and consumers that have individualistic goals; each provider wishes to maximize its revenue, while each consumer wishes to maximize its utility and do so for as the lowest cost possible. To reconcile the requirements of a consumer and the candidate providers, a broker chooses a subset of providers such that the satisfaction is above a high threshold and all providers in the subset have equal chances to be chosen by the consumer. We call the size of this subset *satisficing size*, and denote it by σ ; the word “satisfice” was coined by Nobel Prize winner Herbert Simon in 1957 to describe the desire to achieve a minimal value of a variable instead of its maximum [113].

The resource negotiation protocol consists of the following steps:

1. All providers reveal their capacity and pricing parameters to the broker: $\forall R_j \in \mathcal{R}$ send vectors c_j and ξ_j where each element corresponds to one type of resource.

2. A consumer U_i sends to the broker a request with the following information :
 - (a) the parameters of its utility function: vectors ζ_i and ω_i where each element corresponds to one type of resource,
 - (b) the parameters of its satisfaction function: vectors μ_i , ϵ_i , κ_i and ϕ_i where each element corresponds to one type of resource, and
 - (c) the number of candidate resource providers to be returned.
3. The broker performs a brokering algorithm and returns a list of candidate resource providers \mathcal{R}^i to consumer U_i .
4. Consumer U_i selects the first provider from \mathcal{R}^i and verifies if the provider can allocate the required resources. If it can not, the consumer moves to the next provider from the list until the resources are allocated by a provider R_j .
5. R_j notifies the broker about the resource allocation to U_i .

The algorithm performed by the broker is summarized in Figure 4.4. The amount of resources to be allocated is determined during the algorithm according to a *broker strategy*. Simple strategies would be to allocate the same amount of resources to every consumer, or to allocate to every consumer a random amount of resources. A better strategy, used by our system, is to allocate an amount of resources such that the utility of each type of resource to the consumer reaches a certain *target utility* τ . To determine the amount of resources allocated to the consumer, the broker uses Equation 4.13(a) derived from the definition of

BROKERING ALGORITHM

INPUT request req , τ , σ , a finite set of resource providers ps

OUTPUT a finite set of suggested resource providers ss

BEGIN

 determine $amount$ so that $req.u(amount) = \tau$

 FOR each resource provider rp in ps

$r = \min(amount, \text{available resources of } rp)$

$satisfaction = req.s(req.u(r), rp.p(r))$

 END FOR

 sort elements in ps according to their $satisfactions$

 randomize the sequences of the first σ items in ps

 keep the elements in ps that have the highest $req.cardinality$ satisfaction degrees and remove the rest

$ss = ps$

END

Figure 4.4: The algorithm performed by the broker. The consumer request, **req**, is elastic. It contains the parameters describing **u** and **s**, the utility and satisfaction functions. τ is the target utility and σ is the satisficing size. The **cardinality** specifies the number of resource providers to be returned by the broker.

$u(r)$, Equation 4.13(b):

$$r = e^{\frac{\ln(\frac{\tau}{1-\tau})}{\zeta} + \ln(\omega)} \quad (a) \quad u(r) = \frac{(r/\omega)^\zeta}{1 + (r/\omega)^\zeta} \quad (b) \quad (4.13)$$

Several quantities characterize the resource management policy for broker B and its associated providers and consumers:

- (a) *Average hourly revenue*. The average is over the set of providers connected to broker B ; the revenue of a provider is the sum of revenues from all resources it controls.
- (b) *Request acceptance ratio*. The ratio is the number of accepted requests over the number of requests submitted by the consumers connected to broker B . A request is accepted

if a provider able to allocate resources exists, otherwise the request is rejected and the corresponding satisfaction and utility are set to 0.

(c) *Average consumer satisfaction.* The average is over the set of all consumers connected to broker B .

(d) *Average consumer utility.* This average is over the set of consumers connected to broker B .

In our model, a broker receives a percentage of the revenues collected by the providers connected to it. More sophisticated mechanisms are possible, for example, in addition to the percentage of the revenues collected from the providers, a broker may receive a premium from consumers based upon their level of satisfaction. This policy would encourage brokers to balance the interests of providers and consumers. Different brokers may have different policies and may be required to disclose the average values for critical parameters, such as τ and σ , and their fee structure, during the initial negotiation phase; thus, consumers and providers will have the choice to work with a broker that best matches their own objective.

4.5 A Simulation Study

Market-oriented resource allocation algorithms are very difficult to analyze analytically. To understand the behavior of the system we conducted a simulation study using YAES [19]. A thorough investigation would require multiple brokers, but the model is already very

complex and would require additional protocols for broker selection and renegotiations so we are considering the case of a single broker.

The resource allocated by provider R_j to consumer U_i are represented by a resource vector $r_{ij} = (r_{ij}^1 \ r_{ij}^2 \ \dots \ r_{ij}^l)$. For example, if the k -th component is secondary storage, then $r_{ij}^k = 20GB$ is the amount of secondary storage provided by R_j to consumer U_i . The associated utility and satisfaction vectors are: $u_{ij} = (u_{ij}^1 \ u_{ij}^2 \ \dots \ u_{ij}^l)$ and $s_{ij} = (s_{ij}^1 \ s_{ij}^2 \ \dots \ s_{ij}^l)$. The *demand to capacity ratio* for resource type k is the ratio of the amount requested by all consumers to the total capacity of providers for resource k , $\sum_j c_j^k$. The level of demand is limited by the sigmoid shape of the utility curve and the finite financial resources of the consumers. In the computation of the demand-capacity ratio, for each consumer and each resource, it is assumed that for the requested r_{ij}^k value the corresponding utility value $u_{ij}^k = 0.9$, i.e., the consumers request an amount of r_{ij}^k that results in $u_{ij}^k = 0.9$. The *demand to capacity ratio* vector for all resource types is $\eta = (\eta^1 \ \eta^2 \ \dots \ \eta^l)$. For the sake of simplifying the simulation, we only consider the case when $\eta^1 = \eta^2 = \dots = \eta^l = \eta$.

We run multiple simulation experiments for each case (50 runs/case) and compute 95% confidence intervals for the results. The parameters for our experiments are:

- τ - target utility for the consumers,
- σ - satisficing size; reflects the choices given to the consumer by the broker, and
- η - demand to capacity ratio; measures the commitment and, thus, the load placed upon providers.

We study the evolution in time of

- average hourly revenue,
- request acceptance ratio (the ratio of resource requests granted to total number of requests),
- average consumer satisfaction, and
- average consumer utility.

We investigate the performance of the model for different target utilities, τ , satisfying sizes, σ , and demand to capacity ratios, η . We study several scenarios, for the linear (Equation 4.1), EDN (Equation 4.5), and EDL (Equation 4.4) pricing strategies.

We simulate a system of 100 clusters and one broker. The number of nodes of each cluster is a random variable normally distributed with the mean of 50 and the standard deviation of 30. Each node is characterized by a resource vector containing the CPU rate, the main memory, and the disk capacity. For example, the resource vector for a node with one 2 GHz CPU, 1 GB of memory, and a 40 GB disk is $(2GHz, 1GB, 40GB)$.

Initially, there is no consumer in the system. Consumers arrive with an inter-arrival time exponentially distributed with the mean of 2 seconds. The service time t_{ij} is exponentially distributed with the mean of λ seconds. By varying the λ value we modify demand-capacity ratio so that we can study the behavior of the system under different loads.

Table 4.1: The parameters for the simulation are uniformly distributed. The parameters and the corresponding intervals are shown.

Parameter	CPU	Memory	Disk
ξ	[5, 10]	[5, 10]	[5, 10]
ω	[0.4, 0.9]	[0.5, 1.5]	[10, 30]
κ	[0.02, 0.04]	[0.02, 0.04]	[0.02, 0.04]
μ	[2, 4]	[2, 4]	[2, 4]
ϵ	[2, 4]	[2, 4]	[2, 4]
ϕ	[10, 20]	[20, 40]	[400, 800]

The request is elastic, i.e., instead of requesting a precise amount, consumers only specify their utility and satisfaction functions. The parameters of the utility and satisfaction functions are uniformly distributed in the intervals shown in Table 4.1. A request provides the parameters of the utility function, ω and ζ , for each element of the resource vector (CPU, Memory, Disk). We generate ω and ζ such that with a utility of 0.9, the CPU rate, memory space, and disk space of a request are exponentially distributed with means of $2GHz$, $4GB$, and $80GB$, and ranges of $[0.1GHz, 100GHz]$, $[0.1GB, 200GB]$, and $[0.1GB, 1000GB]$, respectively. More precisely, for each element: (a) we generate the amount r according to the corresponding distribution; (b) we choose a value for ω ; (c) set $u = 0.9$ and compute the corresponding value of ζ . For a resource vector, we let the overall utility be the product of the

utilities of its scalar resources, and the overall satisfaction be the product of the satisfaction for its scalar resources.

When we study the effect of the target utility τ , we use $\sigma = 1$ and $\eta = 1.0$; when we study the effect of σ , we use $\tau = 0.9$ and $\eta = 1.0$; and when we study the effect of η , we use $\tau = 0.9$ and $\sigma = 1$. We also compare the system performance of our scheme for several σ values with a *random strategy*. In this case, we randomly choose a provider from the set of all providers, without considering the satisfaction function. To make the model more realistic, we allow a resource provider to reject a consumer's request if the available resources are insufficient to permit both satisfaction and utility to reach 0.1.

Figures 4.5, 4.6, 4.7, and 4.8 summarize our findings. In each case, we present the three pricing strategies, linear, EDN, and EDL. The parameters for the graphs illustrating the effect of the target utility, τ , at the top of the figure are: $\sigma = 1$, $\eta = 1.0$, and $\tau = 0.8, 0.85, 0.9$, and 0.95 . The graphs illustrating the effect of the satisficing size, σ , in the middle of the figure use the following parameters: $\tau = 0.9$, $\eta = 1.0$, and $\sigma = 1, 10$, and 20 ; for the random strategy, $\sigma = |\mathcal{R}| = 50$. The parameters for the graphs illustrating the effect of the demand to capacity ratio, η , at the bottom of the figure are: $\tau = 0.9$, $\sigma = 1$, and $\eta = 0.5, 1.0, 1.5$, and 2.0 .

The average hourly revenue is an important consideration for resource providers. We notice that the three pricing strategies exhibit similar behavior: the average hourly revenue increases rapidly during the transient period, reaches a maximum, and then converges to a steady state, as shown in Figure 4.5. For the same value of the target utility, τ , the steady

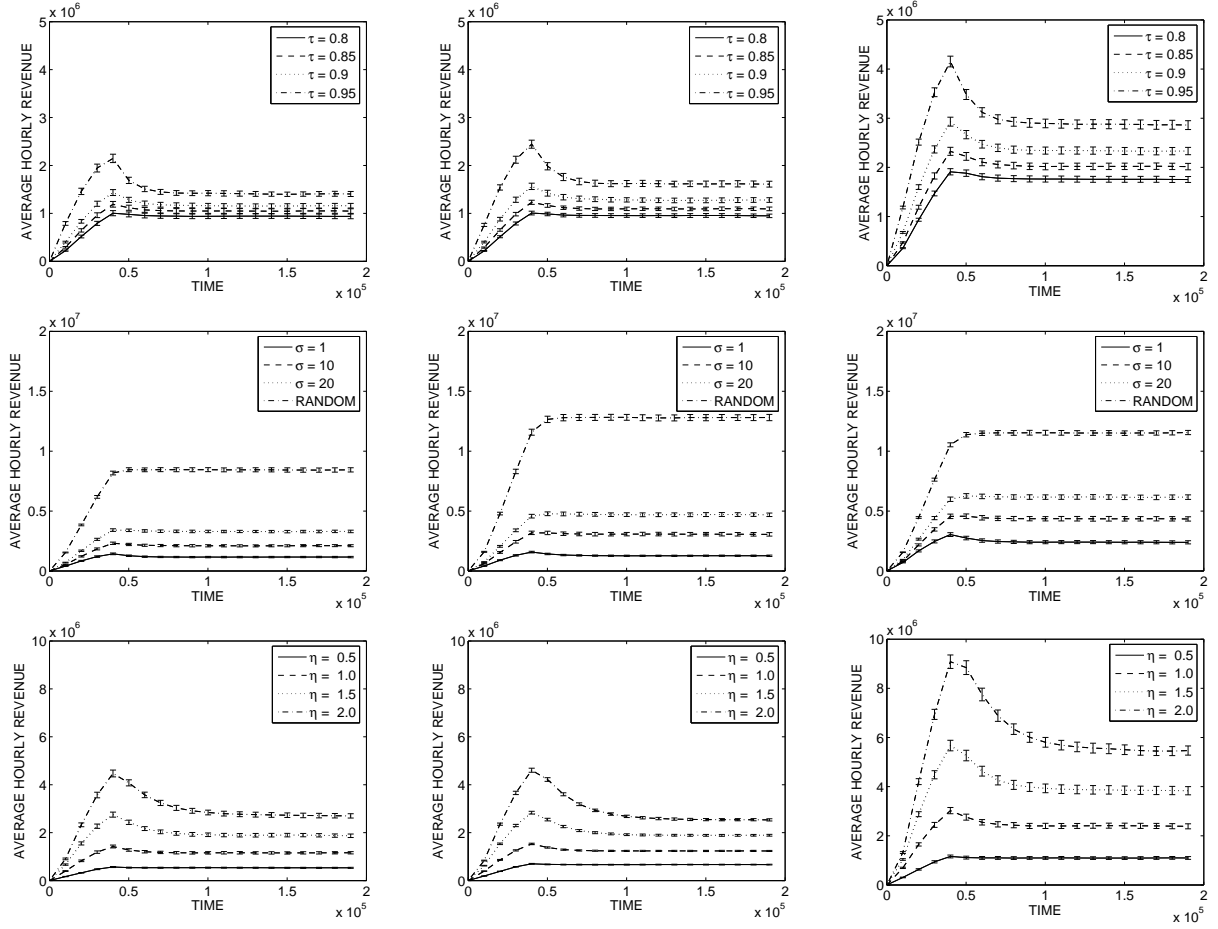


Figure 4.5: Average hourly revenue vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).

state value for the linear and the EDN pricing strategies are close to one another and almost half of those for EDL, as shown in the top row of Figure 4.5. In all cases, the larger τ the higher the revenue. In these simulations, $\sigma = 1$ (the broker provides a single choice) and the demand to capacity ratio is $\eta = 1.0$. We believe that resource fragmentation is the reason

why the steady state value is lower than the maximum attained at the end of the transient period.

Resource fragmentation is an undesirable phenomena where the amount of resources available cannot meet the target utility value for any request and resources remain idle. This effect is more pronounced for larger utility values, for example for $\tau = 0.95$ the steady state value is some 20% lower than its corresponding maximum, while for $\tau = 0.8$ the steady state value is close to its corresponding maximum.

The next question is if larger satisficing size affects the average revenue. A small value of σ limits the number of choices to consumers and this restriction leads to lower average hourly revenues. In our experiments $\tau = 0.9$ and $\eta = 1.0$, as shown in the middle row of Figure 4.5. EDN and EDL are superior to linear pricing. The larger σ , the higher the average hourly revenue for the provider. The random strategy, which corresponds to the maximum value of $\sigma = |\mathcal{R}|$ leads to the highest average hourly revenue.

Lastly, we see that the demand to capacity ratio also has an impact upon the average hourly revenue that is larger for larger η for for all three pricing strategies, as shown in the bottom row of Figure 4.5. The conclusion we draw from these results is that the average hourly revenue increases when we provide a higher target utility (τ closer to 1), increase the satisficing size (larger σ), and increase the demand to capacity ratio, η , and that differential pricing strategies (EDN and EDL) are preferable to the linear one.

The request acceptance ratio for various pricing policies and choices of parameters is shown in Figure 4.6. We find that the request acceptance ratio shows variations during

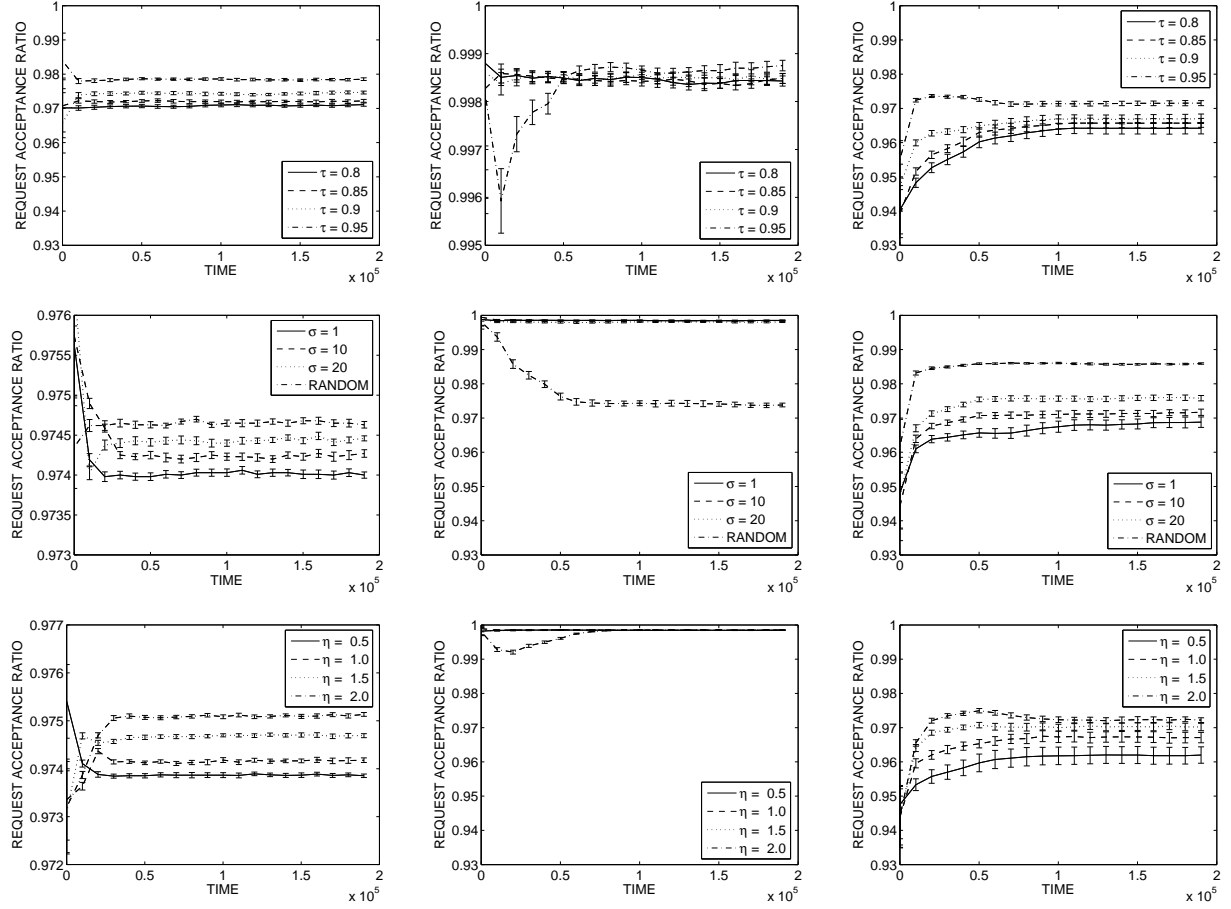


Figure 4.6: Request acceptance ratio vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).

the transient period but converges to constant values in the steady state. The EDN pricing strategy appears optimal, leading to steady state values close to 1.0 for virtually every choice of parameters, except for the random σ . The steady state values for the linear and EDL strategies are also high, with values larger than 0.95, but the exact amount is determined by

the values of τ , η and σ . We find that the higher the values of any of these parameters, the higher the request acceptance ratio.

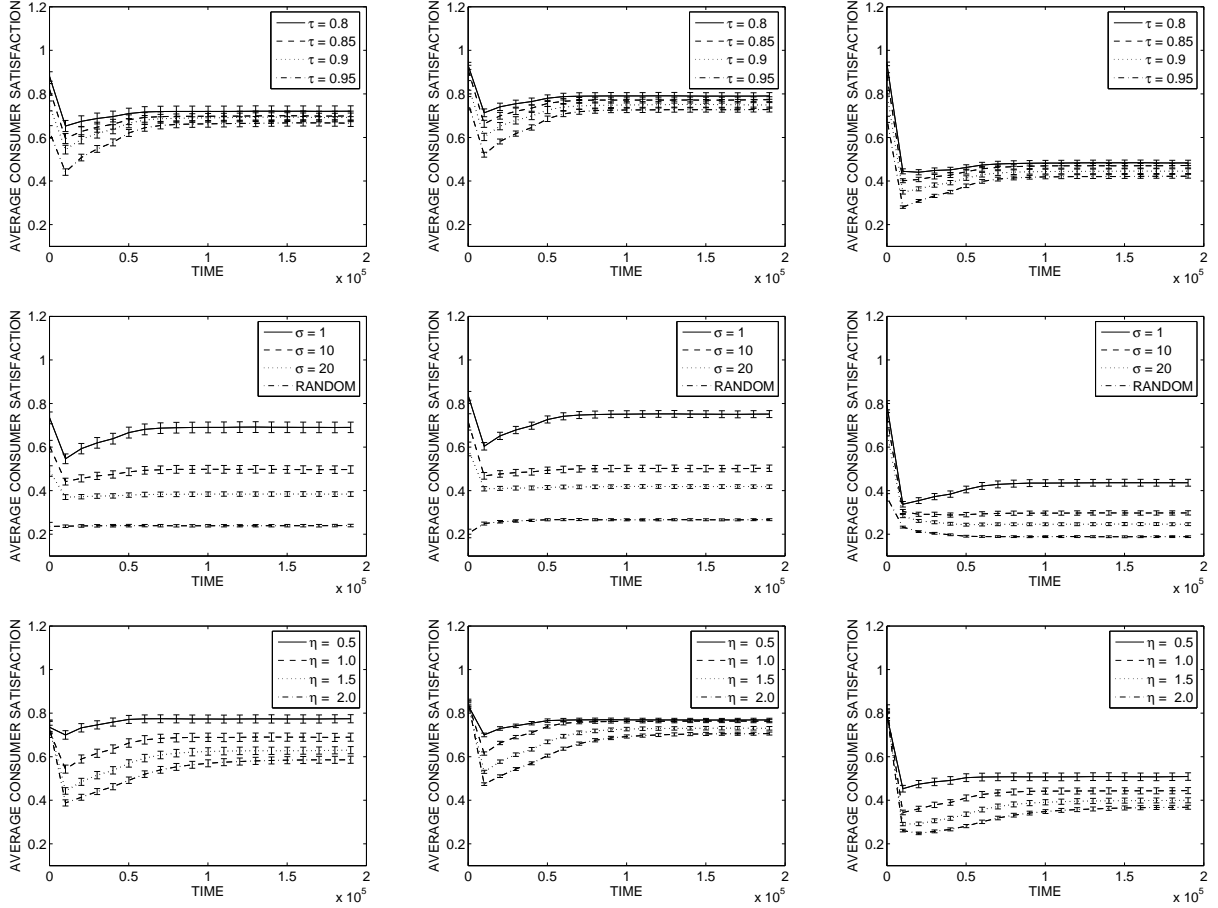


Figure 4.7: Average consumer satisfaction vs. time (in seconds) for different target utilities, τ (top), satiscing sizes, σ (middle), and demand to capacity ratio, η (bottom). The three pricing strategies are: linear (left), EDN (center), and EDL (right).

The three pricing strategies lead to very different consumer satisfaction for the same set of parameters of the simulation, even though the qualitative behavior is somehow similar

in that the average consumer satisfaction decreases during the transient period and then increases and reaches a stable value in steady state, as shown in Figure 4.7. EDN appears to be best strategy. The larger the target utility, the lower the consumer satisfaction. The highest steady state average satisfaction is about 80% when $\tau = 0.8$ and when we use the EDN strategy as compared with less than 50% for EDL and about 70% for linear pricing strategy in terms of σ . The highest satisfaction occurs when $\sigma = 1$. Though this seems counterintuitive it is well justified; in this case the broker directs the consumer to that resource provider that best matches the request. When we select at random one provider from the list of all providers supplied by the broker we observe the lowest average consumer satisfaction because we have a high probability to select a less than optimal match for a given request. Recall that the optimal match is the top ranked element of the list of providers supplied by the broker. We also notice that a high demand to capacity ratio has a negative impact upon user satisfaction. The largest impact of the demand to capacity ratio upon the steady state average consumer satisfaction is visible for the linear pricing strategy, when the average consumer satisfaction ranges from about 55% for $\eta = 2.0$ to about 75% for $\eta = 0.5$.

For the same set of parameters of the simulation the three pricing strategies lead to slightly different average consumer utility values, but the qualitative behavior is similar, as shown in Figure 4.8. The average consumer utility decreases slowly during the transient period because of system fragmentation; some resources are allocated to consumers due to their cheaper price, although they are not enough to allow the utility to reach the target value, τ . In steady state, the average utility reaches a stable value. Overall, the differentiated

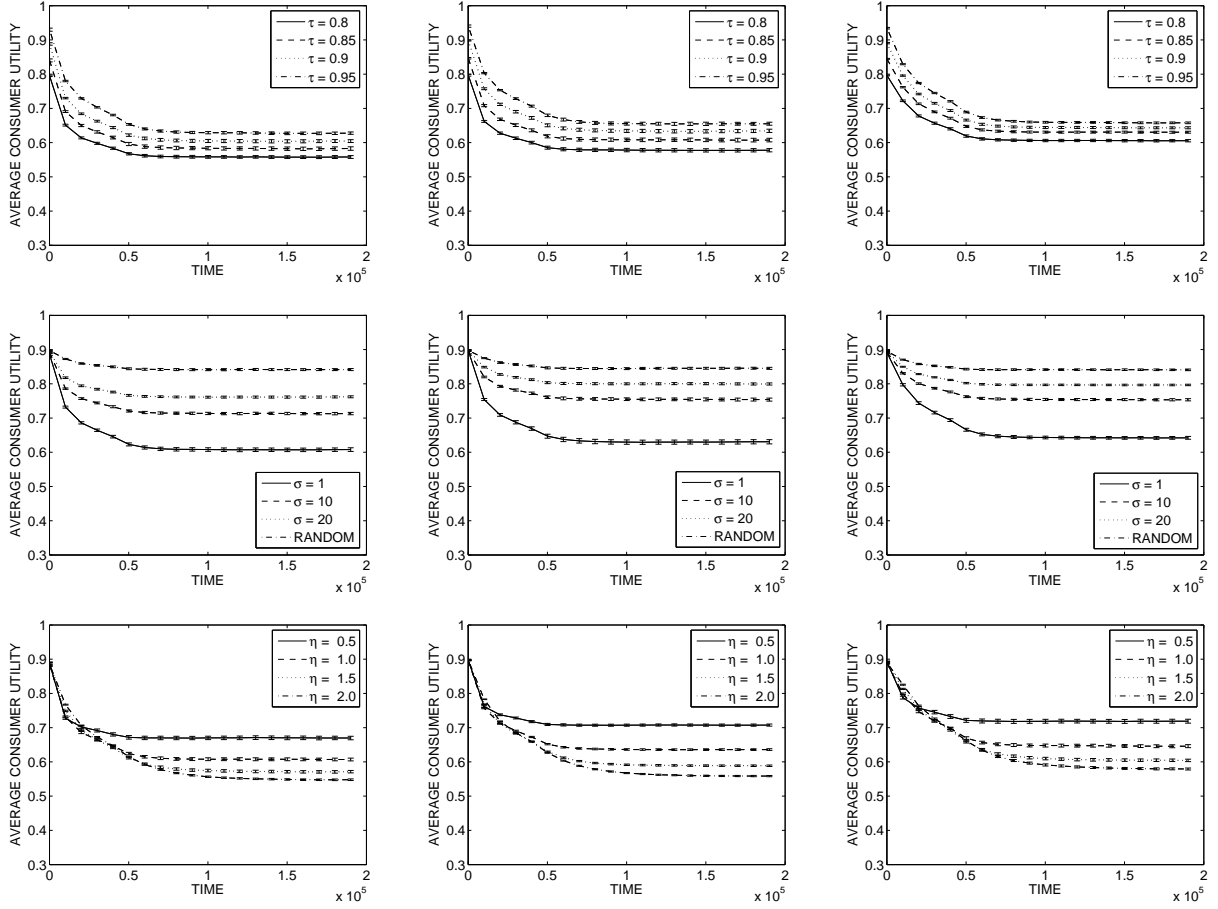


Figure 4.8: Average consumer utility vs. time (in seconds) for different target utilities, τ (top), satisficing sizes, σ (middle), and demand to capacity ratios, η (bottom). The three pricing strategies: linear (left), EDN (center), and EDL (right).

pricing strategies, EDN and EDL, perform better and reach higher steady state values. The higher the target utility, the larger the actual utility; the highest steady-state utility is about 70% for $\tau = 0.95$ for EDN and EDL, as shown in the top of Figure 4.8. The larger the satisficing size, the higher the actual utility; the random strategy leads to 90% utility, as

shown in the middle of Figure 4.8. The lower the demand to capacity ratio, the higher the satisfaction.

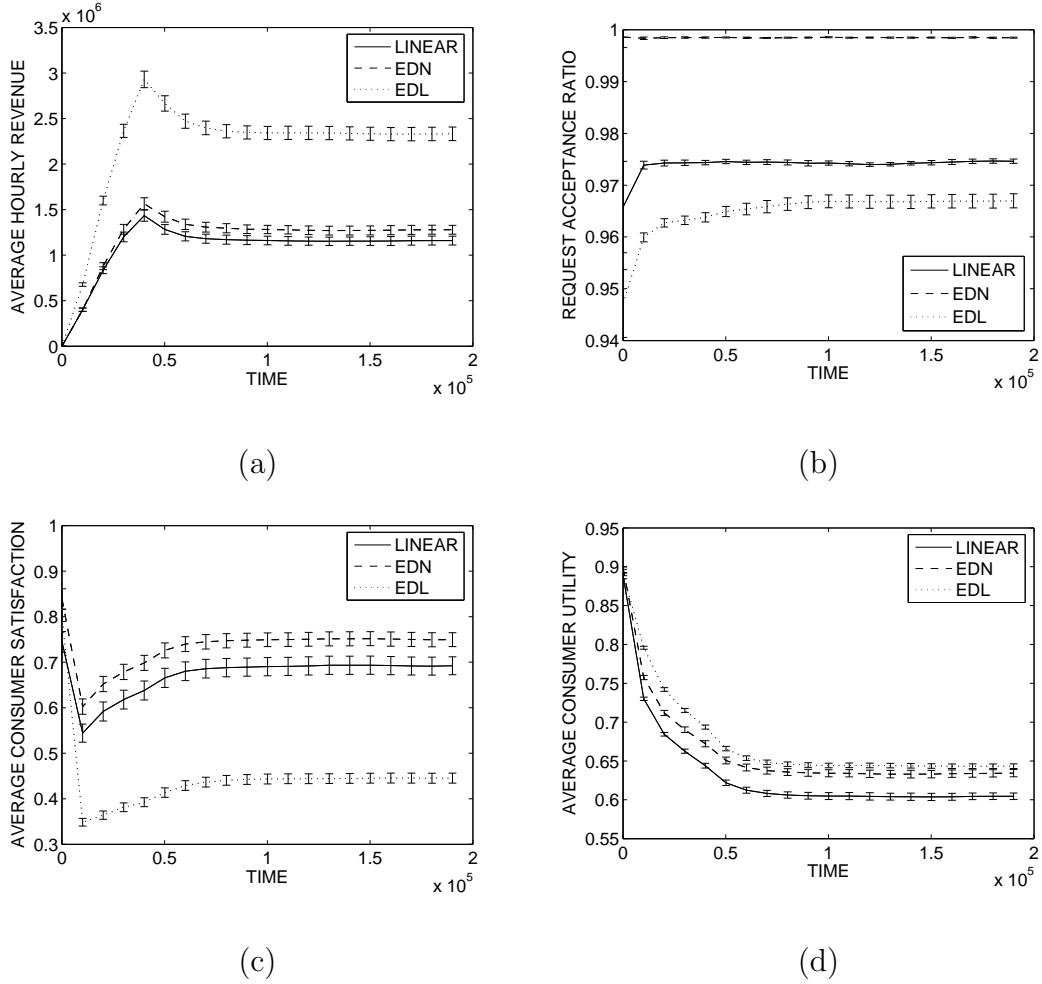


Figure 4.9: (a) The average hourly revenue, (b) the request acceptance ratio, (c) the average consumer satisfaction, and (d) the average consumer utility vs. time (in seconds) for $\sigma = 1$, $\tau = 0.9$, and $\eta = 1.0$, with different price functions.

Figure 4.9 summarizes the effect of the three pricing strategies upon the four quantities we monitored in our experiments, for a particular set of parameters: $\tau = 0.9$, $\sigma = 1$, and $\eta = 1.0$.

EDL allows the highest average hourly revenue while the linear pricing strategy leads to the lowest one, as shown in Figure 4.9(a). EDN leads to the highest request acceptance ratio while EDL leads to the lowest one, as shown in Figure 4.9(b). EDN leads to the highest consumer satisfaction while EDL leads to the lowest one, as shown in Figure 4.9(c). EDL allows the highest average hourly revenue while linear pricing strategy leads to the lowest one, as shown in Figure 4.9(d).

CHAPTER 5

CONCLUSIONS

In this dissertation, we address the problems of coordination, matchmaking, and resource allocation for large-scale distributed systems. Although deterministic approaches for coordination, matchmaking, and resource allocation have been well studied, they are not suitable for large-scale distributed systems due to the large-scale, the autonomy, and the dynamics of the systems. We have to seek for nondeterministic solutions for large-scale distributed systems. We present two services in large-scale distributed systems: the coordination service and the matchmaking service. The coordination service coordinates the execution of complex tasks in a dynamic environment and the matchmaking service supports finding the appropriate resources for users. We also presents a macro-economic resource allocation model based upon utility, price, and satisfaction functions. In this model, a broker mediates resource providers who want to maximize their revenues and resource consumers who want to get the best resources at the lowest possible price, with some global objectives, e.g., to maximize the resource utilization of the system. We summarize and conclude our research for each addressed problem as follows.

a) Coordination

A large-scale distributed system is a complex system. The state space of a complex system is very large and it is unfeasible to create a rigid infrastructure implementing optimal policies and strategies that take into account the current state of the system. We need a coordination service to hide the complexity of the system from the end-user. The coordination service acts as a proxy on behalf of end-users to react to unforeseen events and to plan how to carry out complex tasks. It should be reliable and able to match user policies and constraints (e.g., cost, security, deadlines, quality of solution) with the corresponding system policies and constraints.

We use a process description and a case description to describe a complex task. A process description is a formal description of the complex problem a user wishes to solve. For the process description, we use a formalism similar to the one provided by Augmented Transition Networks (ATNs). A process description defines the data dependencies among the activities of a complex task and consists of end-user activities and flow control activities. The execution of an end-user activity corresponds to the execution of an end-user service thus of an application program. Flow control activities do not have associated end-user services. They are used to control the execution of end-user activities. We define six flow control activities: Begin; End; Choice; Fork; Join; and Merge. Every process description should start with a Begin activity and conclude with an End activity. The Begin activity and the End activity should occur exactly once in a process description. A case description provides additional information for a particular instance of the process the user wishes to

perform, e.g., it provides the location of the actual data for the computation, additional constraints related to security, cost, or the quality of the solution, a soft deadline, and/or user preferences. Based on the concept of process and case description, we designed an algorithm to coordinate the execution of a task and to supervise the execution of each activity of a task.

We implemented a coordination service in BondGrid. The coordination service interacts with other core and end-user services for the execution of computing tasks. The coordination service consists of a message handler, a service manager, and a coordination engine. The message handler is responsible for the communication between the coordination service and other entities in the system. The service manager provides a GUI for monitoring the execution of tasks and the interactions between coordination service and other services. The coordination engine manages the execution of tasks submitted to the coordination service.

The coordination service relies heavily on shared ontologies. An ontology is a catalog of and reveals the relationships among a set of concepts assumed to exist in a well defined area. Creating ontologies in the context of large-scale distributed system represents a monumental task. We defined the main ontologies used in BondGrid including task, process description, case description, activity, data, service, resource, hardware, and software.

We studied the time the coordination service needs to encode, transmit, and decode an ontology (in XML format) as the size of the ontology increases. We studied the maximum request handling rate of the coordination service. We also tested the coordination service in BondGrid for an important application of biology computation, the 3D atomic structure

determination of macromolecules based upon electron microscopy. For every experiment we use the same process description and different case descriptions with different input data sets and different target resolutions. The coordination service is able to supervise the execution of the computation and provides the results upon completion successfully.

From our past experience it is abundantly clear that the development of complex and scalable systems requires some form of intelligence. We cannot design general policies and strategies that do not take into account the current state of a system. But the state space of a complex system is very large and it is unfeasible to create a rigid control infrastructure. The only alternative left is to base our actions on logical inference. This process requires a set of policy rules and facts about the state of the system, gathered by a monitoring agent. Similar arguments show that we need to plan if we wish to optimally use the resource-rich environment of a large-scale distributed system, subject to quality of service constraints. Further optimization is only possible if various entities making decisions have also the ability to learn. Yet, it is not so clear that the current AI technologies are at the point where their application to large-scale distributed systems is unproblematic. Our limited experiments point out that there are limitations of the current agent and knowledge base technologies. We need to perform more comprehensive measurements. Data collected from these experiments will allow us to create realistic models of large-scale systems and study their scalability.

b) Matchmaking

A large-scale distributed system is an open system, a large collection of autonomous systems giving individual users the image of a single virtual machine with a rich set of hardware

and software resources. In traditional computing systems, resources are managed centrally under the control of a single administrative authority by the resource management component of an operating system or by a distributed operating system. The central management of resources on a large-scale distributed system is unthinkable because of the large-scale of the system and because it would violate the autonomy of individual resource providers, a critical aspect of large-scale distributed systems. A matchmaking service allow users or agents on behalf of users to describe their needs and get a list of candidate resources ranked according to their matching degree to users' needs so that further decision can be made.

The matchmaking process in large-scale distributed systems involves three types of entities or agents: the consumers called requesters; the producers called providers; and the matchmaking service. The matchmaking services mediate among the providers and the requesters and use a matching algorithm to evaluate a matching function that returns the matching degree. We call the description of a resource a resource advertisement, or simply resource. A resource request, simply called a request, consists of a function to be evaluated in the context of a resource. If the request can be successfully satisfied, the matchmaking service responds with a list of ranked resources. A matchmaking scheme should be extensible and should support exact and inexact matchmaking. In some cases, requesters need to find the resource that exactly matches the request. In other cases, resources that partially match the request are also acceptable.

We introduced for the first time a more comprehensive ontology-based resource matching scheme for large-scale distributed systems. We defined the ontologies for commonly used

resources in large-scale distributed systems such as computer, service, storage, software, and data. The matchmaking service has a knowledge base that holds the resource advertisements from providers. Requests from consumers are evaluated with resource advertisements in the knowledge base. We designed two matchmaking algorithms: a simple algorithm that requires an exhaustive search of all resource advertisements; and a modified algorithm that only covers a portion of the knowledge base. The scheme supports a variety of matching functions including boolean function, arithmetic function, and fuzzy function. A Boolean function returns a Boolean constant, i.e., “true” or “false”. An arithmetic function returns a positive real number. A fuzzy expression returns a fuzzy number in $[0, 1]$. The higher the returned value, the better a request can be satisfied.

We implemented a matchmaking service in BondGrid. The matchmaking service consists of a message handler, a service manager, and a matchmaking engine. The message handler is responsible for the communication between the matchmaking service and other entities in the system. The service manager provides a GUI for monitoring the matchmaking engine and the interactions between matchmaking service and other services. The matchmaking engine handles the matchmaking requests submitted to the matchmaking service.

We studied the response time of the matchmaking service. The experimental results indicate that the response time is dominated by the time to access the resource knowledge base. The complexity of the matching function has little effect on the response time. We studied the response time of the matchmaking service versus the number of resource advertisements when the knowledge base is stored in a local file or a database. For a large

knowledge base, the response time of the matchmaking service is greatly lower when the knowledge base is stored in a database. We also tested the simple matchmaking algorithm and the modified matchmaking algorithm. The modified matchmaking algorithm is able to find the near optimal matching resources with a sufficiently lower computational cost than the simple algorithm.

c) Resource Allocation

Resource management in a large-scale distributed system poses serious challenges due to the scale of the system, the heterogeneity and inherent autonomy of resource providers, and the large number of consumers and the diversity of their needs. In an economic model, all the participants are considered self-interested. The resource providers are trying to maximize their revenues. The consumers want to obtain the maximum possible resources for the minimum possible cost. The large number of participants makes one-to-one negotiations expensive and unproductive. We need a broker to mediate access to resources from different providers. A broker is able to reconcile the selfish objectives of individual resource providers who want to maximize their revenues, with the selfish objectives of individual consumers who want to get the most possible utility at the lowest possible cost, and with some global, societal objectives, e.g., to maximize the utility of the system.

To formalize the objectives of the participants, we use: (i) a consumer *utility function*, $0 \leq u(r) \leq 1$, to represent the utility provided to an individual consumer, where r represents the amount of allocated resources; (ii) a provider *price function*, $p(r)$, imposed by a resource

provider, and (iii) a consumer *satisfaction function*, $s(u(r), p(r))$, $0 \leq s \leq 1$, to quantify the level of satisfaction; the satisfaction depends on both the provided utility and the paid price.

We proposed a macro-economic resource allocation model based upon utility, price, and satisfaction functions for large-scale distributed systems. In this model, resource providers advertise their resource capacities and pricing strategy to the broker, and resource consumers send their description about their utility and satisfaction to the broker. The broker performs a brokering algorithm based on some system parameters for each request from the consumer.

Economic models are notoriously difficult to study. The complexity of the utility, price, and satisfaction-based models precludes analytical studies and in this dissertation we reported on a simulation study. The goal of our simulation study is to validate our choice of utility, price, and satisfaction function, to study the effect of the many parameters that characterize our model, and to get some intuition regarding the transient and the steady-state behavior of our models. We are primarily interested in qualitative rather than quantitative results, i.e., we are interested in trends, rather than actual numbers.

The function of a broker is to monitor the system and set τ and σ for optimal performance. For example, if the broker perceives that the average consumer utility is too low, it has two choices: increase τ or increase σ . At the same time, the system experiences an increase of the average hourly revenue and a decrease of the average consumer satisfaction. The fact that increasing utility could result in lower satisfaction seems counterintuitive, but reflects the consequences of allocating more resources; we increase the total cost possibly beyond the optimum predicated by the satisfaction function. The simulation results shown in this

dissertation are consistent with those in [9, 10] where we use linear pricing and simpler models based upon a synthetic quantity to represent a vector of resources.

We tested three pricing strategies in the simulation: linear, EDL, and EDN. The EDL pricing strategy leads to the highest average consumer utility and the highest average hourly revenue, while it gives the lowest request acceptance ratio and the lowest average consumer satisfaction. The EDN pricing strategy allows the highest request acceptance ratio and the highest average consumer satisfaction, while it leads to lower average consumer utility and average hourly revenue than EDL. It is also remarkable that the average consumer satisfaction does not track the average consumer utility. This shows the importance of the satisfaction function.

One could argue that in practice it would be rather difficult for users to specify the parameters of their utility and satisfaction function. Though this is true in today's environments, it is entirely feasible in intelligent environments where such information could be provided by societal services [14]. The advantages of elastic requests is likely to motivate the creation of such services in the computational economy of the future.

Even though we limit our analysis to a single broker system, we are confident that the most important conclusions we are able to draw from our model are:

- (i) Given a particular set of model parameters the satisfaction reaches an optimum; this value represents the perfect balance between the utility and the price paid for resources,
- (ii) The satisfaction does not track the utility,

(iii) Differentiated pricing perform better than linear pricing,

(iv) Brokers can effectively control the computing economy

will still be valid for multiple broker systems. In such an environment, individual brokers could enforce different policies; providers and consumers could join the one that best matches their individual goals. The other simplifying assumptions for our analysis, e.g., the uniformity of the demand to capacity ratio for all resources available at a consumer's site, will most likely have second order effects. The restriction we impose by requiring a consumer to obtain all necessary resources from a single broker is also unlikely to significantly affect our findings.

It is too early to compare our model with other economic models proposed for resource allocation in distributed systems, but we are confident that a model that formalizes the selfish goals of consumers and providers, as well as societal goals, has a significant potential. Our intention is to draw the attention of the community to the potential of utility, price, and satisfaction-based resource allocation models.

A fair number of questions require further investigations including: (a) Are there better alternatives to the utility, price, and satisfaction functions we introduced? (b) Is the policy aiming to achieve maximum satisfaction sound, e.g., how should we take into account the societal importance of activities carried out by individual resource consumers? (c) How can we apply the models to more complex networks of resource managers? (d) What composition rules should be used to describe the utility and/or the satisfaction for a group of consumers?

(e) How can we define more complex utility functions that take into account additional constraints related to quality of service, system reliability, and deadlines?

LIST OF REFERENCES

- [1] J. Almond and D. Snelling. UNICORE: uniform access to supercomputing as an element of electronic commerce. *Future Generation Computer Systems*, 15(5–6):539–548, October 1999.
- [2] K. Amin, G. von Laszewski, M. Hategan, N. J. Zaluzec, S. Hampton, and A. Rossi. GridAnt: A client-controllable grid workflow system. In *Proceedings of the 37th Hawaii International Conference on System Sciences*, 2004.
- [3] Y. Amir, B. Awerbuch, A. Barak, R. S. Borgstrom, and A. Keren. An opportunity cost approach for job assignment in a scalable computing cluster. *IEEE Transactions on Parallel and Distributed Systems*, 11(7):760–768, 2000.
- [4] Y. Amir, B. A. B, and R.S.Borgstrom. A cost-benefit framework for online management of a metacomputing system. In *Proceedings of the 1st International Conference on Information and Computation Economics (ICE 98)*, pages 140–147. ACM Press, October 1998.
- [5] A. Anastasiadi, S. Kapidakis, C. Nikolaou, and J. Sairamesh. A computational economy for dynamic load balancing and data replication. In *Proceedings of the 1st International Conference on Information and Computation Economics (ICE 98)*. ACM Press, October 1998.
- [6] Ant. URL <http://ant.apache.org/>.
- [7] Autonomic Computing. URL <http://www.research.ibm.com/autonomic/index.html>.
- [8] L. Badia and M. Zorzi. On utility-based radio resource management with and without service guarantees. In *Proceedings of ACM Modelling, Analysis, and Simulation of Wireless and Mobile Systems (MSWiM 2004)*, pages 244–251. ACM Press, 2004.
- [9] X. Bai, L. Bölöni, D. C. Marinescu, H. J. Siegel, R. A. Daley, and I.-J. Wang. Are utility, price, and satisfaction based resource allocation models suitable for large-scale distributed systems? In *Proceedings of the 3rd International Workshop on Grid Economics and Business Models (GECON 2006) (to appear)*, Singapore, May 2006.
- [10] X. Bai, L. Bölöni, D. C. Marinescu, H. J. Siegel, R. A. Daley, and I.-J. Wang. A brokering framework for large-scale heterogeneous systems. In *Proceedings of the 20th*

IEEE International Parallel and Distributed Processing Symposium (IPDPS 2006) (to appear), Rhodes Island, Greece, April 2006.

- [11] X. Bai, H. Yu, Y. Ji, and D. C. Marinescu. Resource matching and a matchmaking service for an intelligent grid. In *Proceedings of International Conference on Computational Intelligence (ICCI2004)*, pages 262–265, 2004.
- [12] X. Bai, H. Yu, Y. Ji, and D. C. Marinescu. Resource matching and a matchmaking service for an intelligent grid. *International Journal of Computational Intelligence*, 1(3):197–205, 2004.
- [13] X. Bai, H. Yu, G. Wang, Y. Ji, D. C. Marinescu, G. M. Marinescu, and L. Bölöni. Intelligent grids. In J. C. Cunha and O. F. Rana, editors, *Grid Computing: Software Environments and Tools*, pages 45–74. Springer Verlag, Heidelberg, 2006.
- [14] X. Bai, H. Yu, G. Wang, Y. Ji, G. M. Marinescu, D. C. Marinescu, and L. Bölöni. Coordination in intelligent grid environments. *Proceedings of the IEEE*, 93(3):613–630, March 2005.
- [15] R. J. Bayardo, Jr., W. Bohrer, R. Brice, A. Cichocki, J. Fowler, A. Helal, V. Kashyap, T. Ksiezyk, G. Martin, M. Nodine, M. Rashid, M. Rusinkiewicz, R. Shea, C. Unnikrishnan, A. Unruh, and D. Woelk. InfoSleuth: Agent-based semantic integration of information in open and dynamic environments. In *Proceedings of the ACM International Conference on Management of Data*, volume 26, pages 195–206, New York, 1997. ACM Press.
- [16] D. Bhatia, V. Burzevski, M. Camuseva, G. Fox, W. Furmanski, and G. Premchandran. WebFlow - a visual programming paradigm for Web/Java based coarse grain distributed computing. *Concurrency - Practice and Experience*, 9(6):555–577, 1997.
- [17] H. P. Bivens. Grid workflow. In *Grid Computing Environments Working Group Document*, Global Grid Forum, 2001.
- [18] L. Bölöni, K. Jun, K. Palacz, R. Sion, and D. Marinescu. The Bond agent system and applications. In D. Kotz and F. Mattern, editors, *Agent Systems, Mobile Agents, and Applications, Lecture Notes on Computer Science, volume 1882*, pages 99–112. Springer Verlag, 2000.
- [19] L. Bölöni and D. Turgut. YAES - a modular simulator for mobile networks. In *Proceedings of the 8th ACM/IEEE International Symposium on Modeling, Analysis and Simulation of Wireless and Mobile Systems (MSWIM 2005)*, pages 169–173, October 2005.
- [20] BPEL4WS. URL <http://www.oasis-open.org/cover/bpel4ws.html>.

- [21] R. Buyya, D. Abramson, and J. Giddy. Nimrod/G: An architecture of a resource management and scheduling system in a global computational grid. In *High Performance Computing in Asia 2000*. IEEE Press, September 2000.
- [22] R. Buyya, H. Stockinger, J. Giddy, and D. Abramson. Economic models for management of resources in peer-to-peer and grid computing. In *Proceedings of the SPIE International Conference on Commercial Applications for High-Performance Computing*, Denver, USA, August 20-24 2001.
- [23] G. Cabri, L. Leonardi, and F. Zambonelli. Reactive tuple spaces for mobile agent coordination. In *Mobile Agents*, pages 237–248, 1998.
- [24] J. Cao. ARMSim: A modeling and simulation environment for agent-based grid computing. *SIMULATION*, 80(4):221–229, 2004.
- [25] J. Cao, S. A. Jarvis, and S. Saini. ARMS: An agent-based resource management system for grid computing. *Scientific Programming*, 10(2):135–148, 2002.
- [26] J. Cao, S. A. Jarvis, S. Saini, and G. R. Nudd. GridFlow: Workflow management for grid computing. In *Proceedings of the International Symposium on Cluster Computing and the Grid 2003*, pages 198–205, 2003.
- [27] J. Cao, D. J. Kerbyson, and G. R. Nudd. Performance evaluation of an agent-based resource management infrastructure for grid computing. In *Proceedings of the International Symposium on Cluster Computing and the Grid 2001*, pages 311–319, 2001.
- [28] J. Cao, D. J. Kerbyson, and G. R. Nudd. Use of agent-based service discovery for resource management in metacomputing environment. In *Proceedings of Euro-Par 2001*, pages 882–886, 2001.
- [29] J. Cao, D. P. Spooner, S. A. Jarvis, S. Saini, and G. R. Nudd. Agent-based grid load balancing using performance-driven task scheduling. In *Proceedings of the 17th International Parallel and Distributed Processing Symposium (IPDPS 2003)*, pages 49–58, 2003.
- [30] N. Carriero and D. Gelernter. Linda in context. *Communications of the ACM*, 32(4):444–458, 1989.
- [31] B. N. Chun and D. E. Culler. Market-based proportional resource sharing for clusters. Technical report, University of California at Berkeley, Berkeley, CA, USA, 2000.
- [32] L. Chunlin and L. Layuan. Agent framework to support the computational grid. *Journal of Systems and Software*, 70(1):177–187, 2004.
- [33] D. Clark. Face-to-face with peer-to-peer networking. *Computer*, 34(1):18–21, 2001.

- [34] CONDOR. URL <http://www.cs.wisc.edu/condor/>.
- [35] B. F. Cooper and H. Garcia-Molina. Peer-to-peer data preservation through storage auctions. *IEEE Transaction on Parallel and Distributed Systems*, 16(3):246–257, 2005.
- [36] COSA. URL <http://www.cosa.de/>.
- [37] E. Deelman, J. Blythe, Y. Gil, C. Kesselman, G. Mehta, K. Vahi, K. Blackburn, A. Lazzarini, A. Arbree, R. Cavanaugh, and S. Koranda. Mapping abstract complex workflows onto grid environments. *Journal of Grid Computing*, 1(1):25–39, 2003.
- [38] Domino Workflow. URL <http://www.lotus.com/workflow/>.
- [39] Eastman Software. URL <http://www.eastmansoftware.com/>.
- [40] J. Eder, E. Panagos, and M. Rabinovich. Time constraints in workflow systems. In *Proceedings of the 11th International Conference on Advanced Information Systems Engineering (CAiSE 99)*, pages 286–300, 1999.
- [41] C. A. Ellis, K. Keddara, and G. Rozenberg. Dynamic change within workflow systems. In *Proceedings of Conference on Organizational Computing Systems*, pages 10–21, 1995.
- [42] FAFNER. URL <http://www.npac.syr.edu/factoring.html>.
- [43] I. Foster, J. Geisler, W. Nickless, W. Smith, and S. Tuecke. Software infrastructure for the I-WAY high-performance distributed computing experiment. In *Proceeding of the 5th IEEE Symposium on High Performance Distributed Computing*, pages 562–571. IEEE Computer Society Press, 1996.
- [44] I. Foster and C. Kesselman. Globus: A metacomputing infrastructure toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, Summer 1997.
- [45] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, San Francisco, CA, 1999.
- [46] I. T. Foster, N. R. Jennings, and C. Kesselman. Brain meets brawn: Why grid and agents need each other. In *Proceedings of the 3rd international joint conference on Autonomous Agents and Multi-Agent Systems*, pages 8–15, 2004.
- [47] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. In *Proceedings of the 10th IEEE Symposium on High Performance Distributed Computing*, pages 7–9, San Francisco, California, August 2001.
- [48] A. Geppert and D. Tombros. Event-based distributed workflow execution with EVE. Technical Report ifi-96.05, University of Zurich, 20, 1996.

- [49] Global Grid Forum (GGF). URL <http://www.ggf.org/>.
- [50] P. Ghosh, N. Roy, S. K. Das, and K. Basu. A pricing strategy for job allocation in mobile grids using a non-cooperative bargaining theory framework. *Journal of Parallel and Distributed Computing*, 65(11):1366–1383, 2005.
- [51] J. Gomoluch and M. Schroeder. Market-based resource allocation for grid computing: A model and simulation. In *Middleware Workshops*, pages 211–218, 2003.
- [52] A. Grasso, J.-L. Meunier, D. Pagani, and R. Pareschi. Distributed coordination and workflow on the World Wide Web. *Computer Supported Cooperative Work*, 6(2):175–200, 1997.
- [53] A. S. Grimshaw and W. A. Wulf. The legion vision of a worldwide virtual computer. *Communication of the ACM*, 40(1):39–45, January 1997.
- [54] W. Grosso, H. Eriksson, R. Ferguson, J. Gennari, S. Tu, and M. Musen. Knowledge modeling at the millennium – the design and evolution of Protégé, 2000.
- [55] C. Grothoff. Resource allocation in peer-to-peer networks - an excess-based economic model. *Wirtschaftsinformatik*, 45(3):285–292, 2003.
- [56] L. He and T. R. Ioerger. Task-oriented computational economic-based distributed resource allocation mechanisms for computational grids. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI 04)*, pages 462–468, 2004.
- [57] G. Heiser, F. Lam, and S. Russel. Resource management in the mungi single-address-space operating system, 1998.
- [58] HP Grid computing. URL <http://www.hp.com/techservers/grid>.
- [59] S. Hwang and C. Kesselman. GridWorkflow: A flexible failure handling framework for the grid. In *Proceeding of the 12th IEEE Symposium on High Performance Distributed Computing*, pages 126–137, 2003.
- [60] I-Flow: Fijitsu. URL <http://www.i-flow.com/>.
- [61] IBM Grid computing. URL <http://www.ibm.com/grid>.
- [62] InConcert: Tibco Software. URL <http://www.tibco.com/>.
- [63] Jade. URL <http://sharon.cselt.it/projects/jade/>.
- [64] N. R. Jennings. An agent-based approach for building complex software systems. *Communications of the ACM*, 44(4):35–41, 2001.

- [65] N. R. Jennings and M. J. Wooldridge. *Agent Technology: Foundations, Applications, Markets*. Springer-Verlag, Berlin, 1997.
- [66] G. Jezic, M. Kusek, T. Marenic, I. Ljubi, I. Lovrek, S. Desic, and B. Dellas. Mobile agent-based software management in grid. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies (WETICE 2004)*, pages 345–346, 2004.
- [67] JINI. URL <http://www.jini.org/>.
- [68] D. Judge, B. Odgers, J. Shepherdson, and Z. Cui. Agent enhanced workflow, 1998.
- [69] K. Jun, L. Bölöni, K. Palacz, and D. C. Marinescu. Agent-based resource discovery. In *Proceedings of the 9th Heterogeneous Computing Workshop*, pages 43–52, 2000.
- [70] G. Kappel, S. Rausch-Schott, and W. Retschitzegger. Coordination in workflow management systems - a rule-based approach. In *Proceedings of Coordination Technology for Collaborative Applications 96*, pages 99–120, 1996.
- [71] H. N. L. C. Keung, J. Cao, D. P. Spooner, S. A. Jarvis, and G. R. Nudd. Grid information services using software agents. In *Proceedings of the 18th Annual UK Performance Engineering Workshop (UKPEW 2002)*, pages 187–198, July 2002.
- [72] S. Krishnan, P. Wagstrom, and G. von Laszewski. GSFL: A workflow framework for grid services. Technical report, Argonne National Laboratory, 2002.
- [73] D. Kuokka and L. Harada. Matchmaking for information agents. In *Proceedings of the 1st International Joint Conferences on Artificial Intelligence*, pages 672–678, 1995.
- [74] S. Kurkovsky and Bhagyavati. Agent-based distributed IDA* search algorithm for a grid of mobile devices. White Paper, 2003.
- [75] S. Kurkovsky and Bhagyavati. Modeling a computational grid of mobile devices as a multi-agent system. In *Proceedings of the International Conference on Artificial Intelligence (IC-AI 03)*, pages 36–44, 2003.
- [76] S. Lalis and A. Karipidis. JaWS: An open market-based framework for distributed computing over the internet. In R. Buyya and M. Baker, editors, *GRID*, volume 1971 of *Lecture Notes in Computer Science*, pages 36–46. Springer, 2000.
- [77] P. Lawrence, editor. *Workflow handbook 1997*. John Wiley & Sons, Inc., 1997.
- [78] T. J. Lehman, S. W. McLaughry, and P. Wycko. T Spaces: The next wave. In *Proceedings of the 32th Hawaii International Conference on System Sciences*, 1999.
- [79] C. Liu, L. Yang, I. T. Foster, and D. Angulo. Design and evaluation of a resource selection framework for grid applications. In *Proceeding of the 11th IEEE Symposium on High Performance Distributed Computing*, pages 63–72, 2002.

- [80] M. Lorch and D. G. Kafura. Symphony - a Java-based composition and manipulation framework for computational grids. In *International Symposium on Cluster Computing and the Grid 2002*, pages 136–143, 2002.
- [81] F. Manola and C. Thompson. Characterizing the agent grid. Technical report, Object Services and Consulting, Inc., 1999.
- [82] D. C. Marinescu. *Internet-Based Workflow Management: Toward a Semantic Web*. Wiley, New York, 2002.
- [83] D. C. Marinescu and Y. Ji. A computational framework for the 3d structure determination of viruses with unknown symmetry. *Journal of Parallel and Distributed Computing*, 63(7-8):738–758, 2003.
- [84] D. C. Marinescu, G. M. Marinescu, and Y. Ji. The complexity of scheduling and coordination on computational grids. In D. C. Marinescu and C. Lee, editors, *Process Coordination and Ubiquitous Computing*, pages 119–132. CRC Press, 2002.
- [85] Mariposa. URL <http://mariposa.cs.berkeley.edu/>.
- [86] L. W. McKnight and J. Boroumand. Pricing internet services: Approaches and challenges. *IEEE Computer*, 33(2):128–129, 2000.
- [87] Mojo Nation. URL <http://www.mojonation.net/>.
- [88] M. Z. Muehlen and J. Becker. Workflow process definition language - development and directions of a meta-language for workflow processes. In L. a. Bading, editor, *Proceedings of the 1st KnowTech Forum*, Potsdam, September 1999.
- [89] S. Nestorov, S. Abiteboul, and R. Motwani. Inferring structure in semistructured data. In *Workshop on Management of Semistructured Data*, 1997.
- [90] Nimrod. URL <http://www.csse.monash.edu.au/~davida/nimrod/>.
- [91] N. Nisan, S. London, O. Regev, and N. Camiel. Globally distributed computation over the internet - the POPCORN project. In *Proceedings of the 18th International Conference on Distributed Computing Systems*, pages 592–601, 1998.
- [92] NSF Middleware Initiative (NMI). URL <http://www.nsf-middleware.org/>.
- [93] G. R. Nudd, D. J. Kerbyson, E. Papaefstathiou, S. C. Perry, J. S. Harper, and D. V. Wilcox. PACE — A toolset for the performance prediction of parallel and distributed systems. *The International Journal of High Performance Computing Applications*, 14(3):228–251, Fall 2000.
- [94] P. O’Brien and M. Wiegand. Agent based process management: applying intelligent agents to workflow. *The Knowledge Engineering Review*, 13(2), 1998.

- [95] OGSA. URL <http://www.globus.org/ogsa/>.
- [96] A. Omicini, F. Zambonelli, M. Klusch, and R. Tolksdorf, editors. *Coordination of Internet Agents: Models, Technologies, and Applications*. Springer, 2001.
- [97] B. Overeinder, N. Wijngaards, M. van Steen, and F. Brazier. Multi-agent support for internet-scale grid management. In *Proceedings of the AISB'02 Symposium on AI and Grid Computing*, pages 18–22, 2002.
- [98] M. Paolucci, N. Srinivasan, K. P. Sycara, and T. Nishimura. Towards a semantic choreography of web services: From WSDL to DAML-S. In *Proceedings of the 2003 International Conference on Web Services (ICWS 2003)*, pages 22–26, 2003.
- [99] C. J. Petrie, S. Goldmann, and A. Raquet. Agent-based project management. In *Artificial Intelligence Today*, pages 339–363, 1999.
- [100] A. Poggi, M. Tomaiuolo, and P. Turci. Extending JADE for agent grid applications. In *Proceedings of the 13th IEEE International Workshops on Enabling Technologies (WETICE 2004)*, pages 352–357, 2004.
- [101] Protégé. URL <http://protege.stanford.edu/>.
- [102] R. Raman, M. Livny, and M. H. Solomon. Matchmaking: Distributed resource management for high throughput computing. In *Proceeding of the 7th IEEE Symposium on High Performance Distributed Computing*, pages 140–146, 1998.
- [103] O. F. Rana and D. W. Walker. The agent grid: agent-based resource integration in PSEs. In *Proceedings of the 16th IMACS World Congress on Scientific Computation, Applied Mathematics and Simulation*. Lausanne, Switzerland, 2000.
- [104] D. Reed, I. Pratt, P. Menage, S. Early, and N. Stratford. Xenoservers: Accountable execution of untrusted programs. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, pages 136–141, 1999.
- [105] D. Rossi, G. Cabri, and E. Denti. Tuple-based technologies for coordination. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 83–109, 2001.
- [106] D. D. Roure, M. A. Baker, N. R. Jennings, and N. R. Shadbolt. The evolution of the grid. In *Grid Computing: Making The Global Infrastructure a Reality*, pages 65–100. John Wiley & Sons, 2003.
- [107] J.-G. Schneider, M. Lumpe, and O. Nierstrasz. Agent coordination via scripting languages. In *Coordination of Internet Agents: Models, Technologies, and Applications*, pages 153–175, 2001.

- [108] Semantic Grid. URL <http://www.semanticgrid.org/>.
- [109] Semantic Web. URL <http://www.w3.org/2001/sw/>.
- [110] SETI@home. URL <http://setiathome.ssl.berkeley.edu/>.
- [111] W. Shen, Y. Li, H. H. Genniwa, and C. Wang. Adaptive negotiation for agent-based grid computing. In *Proceedings of AAMAS2002 Workshop on Agentcities: Challenges in Open Agent Environments*, pages 32–36, 2002.
- [112] R. Siebert. An open architecture for adaptive workflow management systems. In *Issues and Applications of Database Technology (IADT)*, pages 79–85, 1998.
- [113] H. A. Simon. *Models of Man*. Wiley, 1957.
- [114] N. Singh. A common Lisp API and facilitator for ABSI: version 2.0.3. Technical Report Logic-93-4, Logic Group, Computer Science Department, Stanford University, 1993.
- [115] Staffware. URL <http://www.staffware.com/>.
- [116] M. Stonebraker, R. Devine, M. Kornacker, W. Litwin, A. Pfeffer, A. Sah, and C. Staelin. An economic paradigm for query processing and data migration in mariposa. In *Proceedings of 3rd International Conference on Parallel and Distributed Information Systems*, September 1994.
- [117] V. S. Subrahmanian, J. Dix, and F. Ozcan. *Heterogeneous Agent Systems*. MIT Press, 2000.
- [118] Sun Microsystems utility computing. URL <http://www.sun.com/service/utility>.
- [119] K. Sycara, J. Lu, and M. Klusch. Interoperability among heterogeneous software agents on the internet. Technical Report CMU-RI-TR-98-22, Carnegie Mellon University, PA (USA), 1998.
- [120] K. P. Sycara, S. Widoff, M. Klusch, and J. Lu. Larks: Dynamic matchmaking among heterogeneous software agents in cyberspace. *Autonomous Agents and Multi-Agent Systems*, 5(2):173–203, 2002.
- [121] G. Tesauro, D. M. Chess, W. E. Walsh, R. Das, A. Segal, I. Whalley, J. O. Kephart, and S. R. White. A multi-agent systems approach to autonomic computing. In *Proceedings of the 3rd international joint conference on Autonomous Agents and Multi-Agent Systems*, pages 464–471, 2004.
- [122] A. Tveit. jfipa - an architecture for agent-based grid computing. In *Proceedings of the Symposium of AI and Grid Computing, AISB Convention*. AISB, April 2002.

- [123] W. M. P. van der Aalst, A. P. Barros, A. H. M. ter Hofstede, and B. Kiepuszewski. Advanced workflow patterns. In *Proceedings of the 7th International Conference on Cooperative Information Systems (CoopIS 2000)*, pages 18–29, 2000.
- [124] H. R. Varian. *Intermediate Microeconomics: A Modern Approach*. Norton, New York, March 1999.
- [125] D. Veit, J. P. Muller, and C. Weinhardt. An empirical evaluation of multidimensional matchmaking. In *Proceedings of the 3rd International Workshop on Agent Mediated Electronic Commerce (AMEC)*, 2003.
- [126] W. Vickrey. Counterspeculation and competitive sealed tenders. *Journal of Finance*, 16(1):8–37, 1961.
- [127] Visual Workflo: FileNet. URL <http://www.filenet.com/>.
- [128] World Wide Web Consortium (W3C). URL <http://www.w3c.org/>.
- [129] C. A. Waldspurger, T. Hogg, B. A. Huberman, J. O. Kephart, and W. S. Stornetta. Spawn: A distributed computational economy. *Software Engineering*, 18(2):103–117, 1992.
- [130] Y. Wang and J. Liu. Macroscopic model of agent-based load balancing on grids. In *Proceedings of the 2nd international joint conference on Autonomous Agents and Multi-Agent Systems*, pages 804–811, 2003.
- [131] Websphere MQ Workflow. URL <http://www-3.ibm.com/software/integration/wmqwf/>.
- [132] WfMC. URL <http://www.wfmc.org/>.
- [133] G. J. Wickler. *Using Expressive and Flexible Action Representations to Reason about Capabilities for Intelligent Agent Cooperation*. PhD thesis, University of Edinburgh, 1999.
- [134] T. Winograd. *Language as a Cognitive Process*. Addison-Wesley, MA, 1983.
- [135] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. Analyzing market-based resource allocation strategies for the computational Grid. *The International Journal of High Performance Computing Applications*, 15(3):258–281, Fall 2001.
- [136] R. Wolski, J. S. Plank, J. Brevik, and T. Bryan. G-commerce: Market formulations controlling resource allocation on the computational grid. In *Proceedings of the 15th International Parallel and Distributed Processing Symposium (IPDPS 2001)*, pages 46–63, 2001.
- [137] M. Wooldridge. Agent-based software engineering. *IEE Proceedings - Software Engineering*, 144(1):26–37, 1997.

- [138] M. J. Wooldridge and N. R. Jennings. Intelligent Agents: Theory and Practice. *Knowledge Engineering Review*, 10(2):115–152, June 1995.
- [139] WSFL. URL <http://www.ebpml.org/wsfl.htm>.
- [140] XLang. URL <http://www.ebpml.org/clang.htm>.
- [141] H. Yu, X. Bai, and D. C. Marinescu. Workflow management and resource discovery for an intelligent grid. *Parallel Computing*, 31(7):797–811, July 2005.
- [142] H. Yu, X. Bai, G. Wang, Y. Ji, and D. C. Marinescu. Metainformation and workflow management for solving complex problems in grid environment. In *Proceedings of the 18th International Parallel and Distributed Processing Symposium (IPDPS 2004)*, Santa Fe, New Mexico, April 2004.
- [143] M. zur Muehlen. Evaluation of workflow management systems using meta models. In *Proceedings of the 32nd Hawaii International Conference on System Sciences*, pages 1–11, 1999.