Pers Ubiquit Comput (2006) 10: 21–27
DOI 10.1007/s00779-005-0036-5

ORIGINAL ARTICLE

Martin Modahl · Bikash Agarwalla · T. Scott Saponas
Gregory Abowd · Umakishore Ramachandran

UbiqStack: a taxonomy for a ubiquitous computing software stack

Received: 10 July 2004 / Accepted: 17 November 2004 / Published online: 19 August 2005

Abstract This paper describes a taxonomy for a ubiquitous computing software stack called *UbiqStack*. Through the lens of the UbiqStack taxonomy we survey a variety of subsystems designed to be the building blocks from which sophisticated infrastructures for ubiquitous computing are assembled. Our experience shows that many of these building blocks fit neatly into one of the five UbiqStack categories, each containing functionally-equivalent components. Effectively identifying the best-fit ''Lego pieces'', which in turn determines the composite functionality of the resulting infrastructure, is critical. The selection process, however, is impeded by the lack of convention for labeling these classes of building blocks. The lack of clarity with respect to what ready-made subsystems are available within each class often results in naive re-implementation of ready-made components, monolithic and clumsy implementations, and implementations that impose non-standard interfaces onto the applications above. This paper describes the UbiqStack classes of subsystems and explores each in light of the experience gained over 2 years of active development of both ubiquitous computing applications and software infrastructures for their deployment.

M. Modahl (✉) · B. Agarwalla · G. Abowd · U. Ramachandran
College of Computing, Georgia Institute of Technology,
801 Atlantic Drive NW, Atlanta, GA, 30332-0280 USA
E-mail: mmodahl@gmail.com
E-mail: bikash@cc.gatech.edu
E-mail: abowd@cc.gatech.edu
E-mail: rama@cc.gatech.edu

T. S. Saponas
Department of Computer Science, University of Washington,
Box 352350, Seattle, WA, 98125-2350 USA
E-mail: ssaponas@cs.washington.edu

Present address: M. Modahl
Marsdorferstr 36, 50858 Cologne, Germany

1 Introduction

A ubiquitous computing (UbiComp) environment entails (1) an extensive and complex computer architecture deployment, (2) sophisticated conduits for dynamic data flow and control across this architecture, and (3) a judicious external interface facilitating user interaction with the system. A ubiquitous infrastructure must act very much like a telephone patch board i.e., a software layer where various assets are ''hooked'' into and where logical and physical connections between these assets are instantiated. Assets, in the broadest sense, refer to physical devices, bandwidth, processors, as well as services made available through the patch board. This analogy suggests a standard and uniform, but not necessarily central, interface. It is only through this middleware layer that the vast array of disparate assets can be brought together to create the type of rich ubiquitous applications we are excited about.

A necessary precondition for deployment of a UbiComp application is a distributed system composed of (1) computational resources including network connected sensors and actuators, and (2) a mechanism to generalize patterns of interaction with these resources. The reusable corpora of middleware can speed the second step, but in our own experience, we have often overextended or mis-used existing UbiComp middleware subsystems. A standard language for describing middleware subsystems or a simple taxonomy for classifying subsystems could aid in the identification of potential candidates for a middleware deployment.

Different distributed applications, each using middleware for intra-application communication, have very diverse needs. Similarly, an individual distributed application might, at times, require very different things of middleware subsystems. In these cases, multiple, complementary middleware subsystems might be used simultaneously across a single computer architecture deployment to provide the most effective communication tools to all applications. Here the middleware tax-

onomy must also provide aid in choosing what multiple middleware subsystems might be deployed simultaneously to the greatest effect.

To arrive at a reasonable taxonomy describing UbiComp infrastructure and to understand the possible infrastructure interactions, we begin by examining some current and potential UbiComp infrastructures such as UPnP, MediaBroker, and the GRID. By partitioning these varied middleware subsystems into categories/equivalence classes, it becomes possible to reason about entire infrastructures piecemeal. Thus a whole infrastructure may be evaluated in terms of its constituent parts, streamlining the evaluation process and the infrastructure's subsequent refinement. This evaluation allows isolation of individual infrastructure system faults as well as isolation of security concerns.

In addition, standard subsystem classes can improve development of standard intra-infrastructure, system-to-system interaction patterns. With a clearer picture of what infrastructure capabilities exist and where they are located, developers are better positioned to leverage these capabilities in their applications. Once the application requirements have been identified, developers may be able to pick existing infrastructure pieces to develop the application quickly and effectively.

This paper proposes UbiqStack: a five-class infrastructure taxonomy based on the orthogonal functionalities of most commonly occurring subsystems. These include *registration and discovery*, *service and subscription*, *data storage and streaming*, *computation sharing* and *context management*.

Figure 1 shows a visualization of our UbiqStack corresponding to top to bottom interaction between the equivalence classes. The user application is provided with the APIs for interacting with services, streams and storage locations, and process migration/distribution. Context management services and ontologies are also common among all applications.

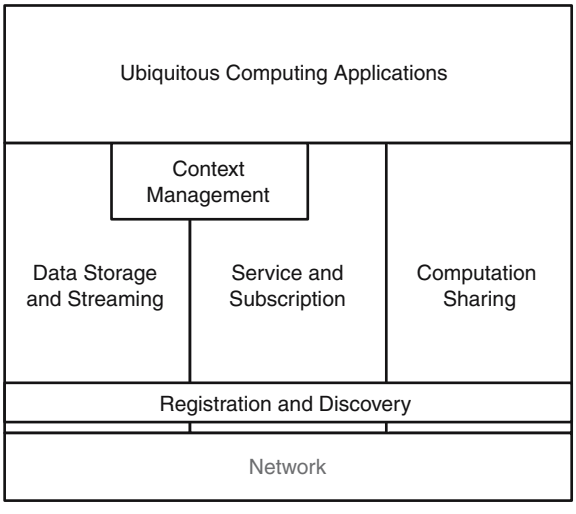

Fig. 1 OSI application layer with added UbiqStack infrastructure component classes

This paper continues in Sect. 2 discussing the imagined computer architecture deployment for UbiComp as well as some initial application deployment experience using middleware to bridge between application and hardware deployment. Section 3 details the taxonomy classes of UbiqStack. Section 4 discusses securing a ubiquitous computing infrastructure in UbiqStack terms. We conclude in Sect. 5.

2 Examining existing infrastructure components

The intrinsic nature of common UbiComp applications builds on connecting distributed nodes into a overarching application. This application might distribute itself onto multiple computers as a common Internet messaging client is deployed on multiple computers throughout the world, while each individual client connects into a larger whole that is a global messaging application. An application might also run in a single location but request and access resources available throughout its environment. As an example, a UbiComp television set-top box might request both a program stream from the cable company and display permission on an autonomous television display. The set-top box could then provide advanced television watching functionality as it routed the stream from the cable company to the television.

A common infrastructure built to support UbiComp applications must then provide the ability for applications to use computation and data resources throughout the environment. Previously we discussed UbiComp infrastructure as the combination of deployed computer hardware and the middleware stitching it together. Before we discuss the middleware subsystems running across the hardware, we will discuss the hardware deployment paradigm we are working from and our previous experiences in building applications on top of hardware deployed according to this paradigm.

2.1 A deployment paradigm

Striving toward the deployment of UbiComp applications in smart spaces, we have been developing both applications and infrastructures with a simple paradigm in mind. This paradigm is constructed of simple, modular, moderately-capable computing devices being connected directly to sensors or sinks throughout a space. In this paradigm each computer can proxy access to its directly connected devices over its network interfaces. Transient devices and stationary devices are connected together abstracting over connection media to form connected graphs of computational devices. Each device then is a node in the graph.

Each of these distributed nodes will provide simple building block resources that applications might query and use. This way, an application might require a

camera stream or a speaker, query for devices containing that service in the area requested, then request and use the resource. Each of these resources as well as the infrastructures exposing these resources to the outside world are managed as modular building blocks.

2.2 Development case study

Our software intercom is an example of an application built on existing hardware and middleware, deployed with the above paradigm in mind, in the Aware Home at Georgia Tech [1]. Using consumer electronics computers, microphones, speakers, and our middleware—MediaBroker [2]—this software intercom allows users throughout a home to communicate through walls and floors as if they were in the same room.

In accordance with the paradigm described above, applications expose resources which other applications use. The user interacts with an intercom interface application which controls a set of modular audio applications deployed throughout the Aware Home. The user interface application projects a control-information resource which outputs current state information while the audio applications provide audio resources which stream captured audio. While running, the interface application queries the state of the audio resources and actuates end-to-end connections through its single control resource. In turn each audio application listens to the interface application's control resource for direction.

Our initial deployment, built entirely on MediaBroker, relied on one-way data streams to transfer audio information between audio inputs and outputs between audio access applications while using the same one-way data streams to distribute control information from the interface application to the audio applications. The nature of these one-way, relatively static, connections between controller and controlled made the deployment of the intercom much less dynamic than anticipated.

Recently, the intercom has been redesigned to use MediaBroker as an infrastructure for low latency audio data streams and UPnP for distributing discrete control messages between a set of control applications and the audio applications they control.

In this case, taking advantage of UPnP for low-bandwidth message passing allows us to apply MediaBroker to the problem it better solves—moving audio data from sources to sinks according to data requests—while offloading control message handling to UPnP. UPnP has better facilities for creating the interface application's control resource and for allowing the many audio access applications to better listen to multiple interface applications at once.

Using the strength of each subsystem and the scaffolding each offers, the job of programming the audio application and the control application is greatly simplified. Using the standard interfaces provided by UPnP and MediaBroker cleanly, the programmer can concentrate on the individual concerns of each application and not how to translate those concerns into something the middleware can do. With a well-defined taxonomy with which to understand UbiComp middleware subsystems, implementation of this intercom would begin with brief modeling of all intra-application and inter-application interactions based on the capabilities of each middleware subsystem categories' abilities.

3 Subsystem category overview

In the brief case study above, system deployment and application development were made simpler through leveraging an existing technology in the context of a novel problem. This anecdotal evidence is in fact exemplary of the benefits derived through clean separation of subsystems. As we have seen, the delineation and choosing of subsystems along functional boundaries enables us to better reason about the infrastructure as a whole as well as to improve the quality of a specific application. Each of these existing subsystems filled a specific role in the development of the application and in the larger infrastructure deployment.

Similar in functionality to UPnP [3], most WebServices implementations provide the ability to make remote calls to procedure located in other applications on other hosts. We argue, then, that UPnP and WebServices provide interchangeable functionality, which means that they belong to the same category of subsystems, namely Service and Subscription, and might be fairly compared to each other when selecting the best-fit subsystem for the job at hand. In turn, their current limitations may serve as a driver for development of a more robust replacement.

In turn, the high-bandwidth, low-latency streaming facilities offered by MediaBroker is what earns its place in a different category of subsystems, namely Data Storage and Streaming. Other subsystems exist to accomplish similar tasks: GnuStream [4] and D-Stampede [5], while still other subsystems provide shared, network accessible, data stores: Coda [6] and T-Spaces [7].

3.1 Registration and discovery

At the heart of UbiqStack, registration and discovery facilitates connection of all other UbiqStack components. Communication between applications running on top of the distributed nodes of a UbiComp infrastructure begins with discovery of available resources. Once different applications connect, they can share descriptions of their capabilities and begin to collaborate on higher order tasks. The semantics of discovering distributed resources (whatever they might be) varies along several independent dimensions. One is whether the resource is actively published/advertised, the other is where the resource is marked globally accessible. Another dimension is the level of flexibility in querying.

Myriad languages, protocols and frameworks exist for the discovery of hosts on the network, applications running on hosts, and services and resources provided by those applications. Common to the WebServices and UPnP world are protocols like SSDP [8], UDDI [9], and WSDL [10]. These allow robust and systematic description and discovery of services. Jini provides significant functionality in the area of service discovery as well as actual resource migration once it is discovered. Many lightweight services like DNS [11] and LDAP provide a more static registration mechanism suitable for resources with considerably longer lifetimes. Specialized facilities like BSDP, which is tied to Bluetooth protocol stack, are also popular.

Evaluating registration and discovery subsystems based on their capabilities must take into account efficiency and extensibility. Some subsystems will require strict and complex registration as well as manual discovery and iterative search of what is available. Other subsystems will provide functionality to automate much of this.

3.2 Service and subscription

Service and subscription, the primary pillar of UbiqStack, encompasses the passing of discrete messages to accomplish distributed service registration and access. For example, an application might offer a resource capable of converting US zip codes to map images [12]. Applications across the network can then access this resource to accomplish this. Applications might also want to know when some event happens and they might subscribe to be notified of some change. Both service and subscription rely on discrete messages being reliably passed between applications.

According to the paradigm discussed above, UbiComp applications will consist of many smaller parts that will require intra-application communication for synchronization. The discrete information exchange provided by a service and subscription subsystem allows distributed application entities to keep state between themselves and provides the necessary abstractions for setting state on adjacent nodes and being updated with their states when other nodes' states change.

The service and subscription component must provide an extensible application programming interface. The component must also be able to scale with respect to the number of applications using the subsystem and the size of data being transferred between applications. The speed with which remote services are called and with which subscription updates are propagated is also integral to the evaluation.

A service and subscription subsystem we have discussed previously in this paper, UPnP, is really only mediocre at message passing and has limited functionality allowing extensibility. UPnP's scalability is also limited to a single network subnet without resorting to complex bridging technology. Context Toolkit [13] provides a service and subscription component as well as some basic context reasoning ability but it has many of the same limitations of UPnP in its inefficiency. WebService implementations vary greatly, but many provide very efficient messaging and fairly extensible application programming interfaces.

3.3 Data storage and streaming

Beyond the simple message passing described above, many UbiComp applications, especially those geared toward multimedia capture and access, require significant, ordered data transfer. The second UbiqStack pillar, data storage and streaming, allows distributed applications to share large amounts of data in a structured manner. For multimedia streaming, structure might emerge in the ordering of video frame delivery, while other applications might require the structure necessary to address, request and change a single value in very large data structure. A data storage and streaming component must be capable of handling significant structured data movement between distributed nodes. This data movement might involve streaming multimedia from a media provider to a media consumer or it might encompass more complex requirements for data archival and retrieval.

Streaming middleware subsystems like MediaBroker [2], GnuStream [4], Indiva [14] and D-Stampede [5] allow distributed nodes to ship streams of data to each other, while other subsystems like serverless file systems [15] and Coda [6] provide distributed access to common file systems. Infrastructures like T-Spaces [7] or JavaSpaces [16] provide more generic shareable network memory. All of these classes provide access to large amounts of data in a structured manner.

Measuring the performance of a data storage and streaming subsystem might be difficult if we only take into account efficiency at providing quality of service. Many data storage and streaming subsystems provide varied types of functionality including different delivery or data set management schemes.

3.4 Computation sharing

The third UbiqStack pillar, computation sharing allows a running application to make use of remote computational resources on demand. For example, a surveillance application running on a sensor network may decide to offload complicated vision tracking components onto more capable high performance computing resources (HPC) strategically placed throughout the ubiquitous infrastructure. Furthermore, in order to accommodate mobile clients to deliver required quality of service, such

computation must be allowed to move from one HPC resource to another. In order to support such unfettered, yet highly-specialized access to computing cycles, a mechanism for discovering unclaimed CPU time and dynamically scheduling computations on the HPC resources is essential.

Many tools for enabling distribution of compute-intensive tasks have emerged over the years. The scientific computing community is largely responsible for this variety. Regrettably, the focus has often been on batch processing, which alone is insufficient to support rich ubiquitous computing applications we envision. Only recently, the grid computing [17, 18] community has started to focus on applications that require capabilities beyond batch processing. Grid computing allows controlled sharing of computation resources across many different physical organizations in order to deliver a quality of service. Open grid service infrastructure (OGSI) [18] has extended WebServices by providing standard mechanism for creating, naming and discovering grid services, providing virtualization through location transparency, and supporting integration with underlying native platform facilities. The Globus Toolkit [19] built on OGSI model, provides the middleware support needed to build grid computing applications.

MPI [20], PVM [21] are some of the other tools that can be used for resource sharing within the set of connected computers. These tools provide libraries for building scientific computing applications and running them on multiple computers at once. However, the capabilities of these tools don't extend very well for streaming applications. Moreover, they don't provide capabilities for dynamically asking additional resources or dynamic joining of computational entities.

Facilities for computation sharing must be dynamic enough to provide for application, thread or even function migration across multiple distributed entities. Applications might want to dynamically move through the environment in order to physically follow a user. Applications might also simply want to spread computationally intensive processes to more capable elements in the environment. The tools for computation sharing should also support dynamism in terms of the resources being allocated, the computation being performed as well as the mobility of the user.

3.5 Context management

A context management subsystem defines a common language to describe the context of an environment and also defines a language to discuss described context state. This middleware subsystem is built on both the service and subscription components as well as the data sharing and streaming components as it will commonly differentiate streaming sensor data into discrete values. While this differentiation will happen at the ubiquitous application level, the differentiation process can be represented inside the context state.

Much of UbiComp relies on having access to a consistent view of the world so that applications might correctly actuate themselves according to the environment and those in the environment. This common context management subsystem component allows applications to keep current on the state of the world by communicating with each other in a language common to all possible applications in the space.

Important in the realm of context management, synonym and unique naming management have been studied fairly thoroughly by multiple projects under the SemanticWeb [22] umbrella and also by projects like OntoWeb [23] and Context ToolKit [13].

4 Compartmentalizing security with UbiqStack

The difficulty in deploying ubiquitous computing infrastructure components to enable application level communication is compounded by the need to provide secure apparati for this communication. Envisioned ubiquitous computing applications require capture, transmission and analysis of extremely sensitive information. With video cameras capturing our every move, large data sets describing what each of us is doing, and computers ready and willing to help less capable computing devices with calculations, system security becomes paramount. Following from the traditional "CIA model" [24] of ensuring confidentiality, integrity and availability, a secure ubiquitous computing infrastructure would demand that all communication remained secure and authentic.

Similarly, to how the UbiqStack taxonomy allows description of complex infrastructures in terms of components, we believe security is best described as a per component responsibility. While an orthogonal set of security policy standards will enable each infrastructure component to consult access control lists or key servers in a standard way, each infrastructure component requires unique security procedures which an over-arching security layer cannot handle. Each infrastructure component will implement a common security policy in some verifiable way that is meaningful to that component, but such that the composition of all the components will still retain the properties of the model and access control policy.

4.1 Questioning security

The UbiqStack component categories each require that specific security concerns be addressed. Here we examine hypothetical security questions and analyze why these questions are unique to each of the UbiqStack categories.

4.1.1 Registration and discovery: can I know if a service or resource exists?

An infrastructure component providing secure registration and discovery scaffolding must address the very

specific needs of applications attempting to register and later discover a service or resource online. Often simply acknowledging that you are interested in knowing something can reveal details best kept private: ''Does anyone know where I can refill my prescription for heart medication?''

4.1.2 Service and subscription: can I call this service?

Where the registration and discovery infrastructure component had the specific duty to ensure only legitimate parties could communicate, the service and subscription infrastructure component must protect all communications related to the discrete message interchange of remote service calls and subscription delivery. Beyond securing the messages passed between applications, a service and subscription component must ensure that all messages are legitimate and form legitimate entities.

4.1.3 Data storage and streaming: can I see this stream or access this store?

While similar to the question for the service and subscription component, it is inherently different because of the semantics of the data being traded/addressed. Where a service might be able to authenticate or exchange keys for each message traded back and forth, the sheer volume traded by data streaming and storage infrastructure components require different access rights management. The data volume also means that multiple consumers will probably be consuming the same data where the service and subscription data can be encrypted and tailored specifically for each caller.

4.1.4 Computation sharing: can I run this code here?

Establishing trust between applications running a computation sharing infrastructure component is possibly the most critical in terms of security for ubiquitous computing. Here alone in the UbiqStack taxonomy is actual, possibly destructive, logic being moved between applications. Secure computation sharing components must ensure that logic is verified for safety, and appropriately corralled to prevent malicious code from causing destruction.

4.1.5 Context management: can I know where someone is?

The context management infrastructure component must provide security that is no longer specifically technical. With the UbiqStack concepts in mind, we expect an application to consult a secure context management resource only after finding it with a secure registration and discovery component and establishing a connection over a secure service and subscription infrastructure component. Here a technically secure connection has been establishing between querier and queried and what remains to be established is a socially secure connection. The context management UbiqStack component must again use common access control lists and public key server to verify that personal and social facts are accessible.

4.2 Building security

This discussion of security relies on the separation of each infrastructure component's security concerns. Allowing each component to manage its own security will step toward portable, extensible systems. The redundancy of security might also allow for better technical security. In a similar manner, modern wireless and web technologies use compartmentalized security to establish secure wireless communication (via WEP or other) before establishing a secure application layer SSL channel to perform what would finally be called a secure credit card transaction [25]. Here the SSL communication channel secures the application layer communication while the WEP encryption secures network packet transmission. The two security apparati are unaware of each other, but as each does it's best to secure the communication, the system is more resilient.

With UbiqStack component category security, we first expect a device to establish some secure network connection. With network connection in place, the device would proceed to authenticate to registration and discovery peers where self-registration and peer discovery would occur. After discovering an interesting multimedia stream, the device would authenticate to the owner/carrier of the multimedia stream before the stream would be accessible. Through compartmentalizing security within each UbiqStack component, secure applications can be built on secure infrastructure components.

5 Conclusion

In this paper we have presented a comprehensive UbiComp middleware taxonomy, UbiqStack, to ease discussion and facilitate the process of building both ubiquitous middleware subsystems as well as ubiquitous applications. We also discussed where in UbiqStack existing middleware subsystems fit and how security questions are best addressed by each individual subsystem. In our experience, the sampling of industrial and academic experience and the combination of knowledge across groups allows better middleware construction and this paper hopes to further these interactions through the creation of a common language for middleware discussion. We are not entirely sure about the feasibility of creating complex infrastructures by picking and choosing subsystems from each taxonomy category, but we look to examine it. Much research is still to be

done at the intersections of these UbiqStack categories to explain the gaps that exist and also to examine their interactions. Research can also be directed to survey entire UbiqStack middleware categories; learning from the state of the art and designing and building new class-complete infrastructures.

Acknowledgements We would like to acknowledge Ilya Bagrak, Phillip Hutto and Matthew Wolenetz for their help reviewing and revising this paper. The work has been funded in part by an NSF ITR grant CCR-01-21638, the Yamacraw project of the State of Georgia, the Georgia Tech Broadband Institute, and the PURA award from Georgia Tech.

References

1. Kidd CD, Orr R, Abowd GD, Atkeson CG, Essa IA, MacIntyre B, Mynatt ED, Starner T, Newstetter W (1999) The aware home: a living laboratory for ubiquitous computing research. In: Cooperative buildings, pp 191–198
2. Modahl M, Bagrak I, Wolenetz M, Hutto P, Ramachandran U (2004) MediaBroker: an architecture for pervasive computing. In: Proceedings of the 2nd IEEE international conference on pervasive computing and communications
3. Contributing Members of the UPnP(TM) Forum: Welcome to the UPnP(TM) Forum (2003) http://www.upnp.org (March 1st, 2004)
4. Jiang X, Dong Y, Xu D, Bhargava B (2003) GnuStream: a p2p media streaming prototype. In: Proceedings of IEEE international conference on multimedia and expo
5. Adhikari S, Paul A, Ramachandran U (2002) D-Stampede: distributed programming system for ubiquitous computing. In: Proceedings of the 22nd international conference on distributed computing systems (ICDCS), Vienna
6. Satyanarayanan M, Kistler JJ, Kumar P, Okasaki ME, Siegel EH, Steere DC (1990) Coda: a highly available file system for a distributed workstation environment. IEEE Trans Comput 39:447–459
7. Lehman TJ, McLaughry SW, Wycko P (1999) T Spaces: the next wave. In: HICSS
8. Goland YY, Cai T, Leach P, Gu Y (1999) IETF Internet draft: simple service discovery protocol/1.0 oct
9. Bellwood et al (2002) UDDI Version 2.03 Data Structure Reference
10. Gudgin M, Lewis A, Schlimmer J (2004) W3C working draft: Web services description language (wsdl) version 2.0 part 2: message exchange patterns
11. Mockapetris PV (1989) RFC 1101: DNS encoding of network names and other types
12. Terraserver.Com, Inc.: TerraServer.com (2004) http://www.terraserver.com (March 1st 2004)
13. Salber D, Dey AK, Abowd GD (1999) The context Toolkit: aiding the development of context-enabled applications. In: CHI, pp 434–441
14. Ooi WT, Pletcher P, Rowe LA (2002) INDIVA: middleware for managing a distributed media environment. (BMRC Technical Note)
15. Anderson T, Dahlin M, Neefe J, Patterson D, Roselli D, Wang R (1995) Serverless network file systems. In: Proceedings of the 15th symposium on operating system principles, Copper Mountain Resort, Colorado, pp 109–126
16. Sun Microsystems: JavaSpace Specification (1998)
17. Foster I, Kesselman C, Tuecke S (2001) The anatomy of the grid: enabling scalable virtual organizations. In: Lecture Notes in Computer Science 2150
18. Foster I, Kesselman C, Nick J, Tuecke S (2002) The physiology of the grid: an open grid services architecture for distributed systems integration. Technical report, The Globus Project
19. Foster I, Kesselman C (1997) Globus: a metacomputing infrastructure toolkit. Int J Supercomput Appl High Performance Comput 11:115–128
20. Forum MPI (1994) MPI: a message-passing interface standard. UT-CS-94-230 MPI Forum
21. Geist A, Beguelin A, Dongarra J, Jiang W, Manchek R, Sunderam V (1994) PVM parallel virtual machine, a user's guide and tutorial for networked parallel computing. MIT Press, Cambridge
22. Calvanese D, Giacomo GD, Lenzerini M (2001) A framework for ontology integration. In: The first semantic Web working symposium, pp 303–316
23. Spyns P, Oberle D, Volz R, Zheng J, Jarrar M, Sure Y, Studer R, Meersman R (2002) OntoWeb—a semantic web community portal. In: Fourth international conference on practical aspects of knowledge management (PAKM)
24. Parker DB (2002) Computer security handbook. Wiley, New York
25. Zhou L, Haas ZJ (1999) Securing ad hoc networks. IEEE Netw 13:24–30