
Lessons learned from the SILENUS federated file system

Max Berger¹ and Michael Sobolewski¹

Texas Tech University `max@berger.name` `sobol@cs.ttu.edu`

Summary. The major objective of the Service Oriented Computing Environment (SORCER) is to form dynamic federations of network services that provide engineering data, applications and tools on an engineering grid with exertion-oriented programming. To meet the requirements of these services in terms of data sharing and managing in the form of data files, a corresponding federated file system, SILENUS, was developed. This system fits the SORCER philosophy of interactive exertion-oriented programming, where users create service-oriented programs and can access data files in the same way they use their local file system. This paper gives a brief overview of SORCER and then the SILENUS methodology is described. Next, we discuss SILENUS gateway, management, and data services with related disconnected and data synchronization mechanisms. We also discuss experimental results of the implemented system.

1 Introduction

In an integrated environment, all entities must first be connected, and they then must work cooperatively. Services that support concurrency through communication, team coordination, information sharing, and integration in an interactive and formerly serial product development process provide the foundation for any CE environment. Product developers need a CE programming and execution environment in which they can build programs from other developed programs, built-in tools, and persisted data describing how to perform complex design processes. Like any other services in the environment, a CE distributed file system can be structured as a collection of collaborating distributed services enabling for robust, secure, and shared vast repository of engineering enterprise data.

Ever since the introduction of a personal computer there exists the problem of managing distributed data. This data is usually stored on one host and not easily accessible for other users or from other hosts. The current trend goes away from a personal computer back to a networked one. We want to work anywhere, anytime, and of course always have the same consistent view on

the same data available. It gets even worse when it comes to collaboration in larger, and distributed engineering teams. Data files have to be accessed in real-time at different locations. Versions have to be managed in a consistent manner. There must be support for offline-use of data and for concurrency management.

Several systems exist to access data that is spread across multiple hosts. However, except for a few exceptions, all of them require manual management and knowledge of the exact data location. Very few offer features like local caching or data replication.

Under the sponsorship of the National Institute for Standards and Technology (NIST) the Federated Intelligent Product Environment (FIPER) [16][17][15] was developed (1999-2003) as one of the first service-to-service (S2S) CE computing environments. The Service-Oriented Computing Environment (SORCER) ([19], [18], [20]) builds on the top of FIPER to drastically reduce design cycle time, and time-to-market by intelligently integrating elements of the design process by providing true concurrency between design and manufacturing. The systematic and agile integration of humans with the tools, resources, and information assets of an organization is fundamental to concurrent engineering (CE).

Two years ago we introduced a novel approach to share data across multiple service providers using dedicated storage, metainformation, replication, and optimization services in the SORCER/SILENUS environment [1], a service oriented approach to distributed file systems. The access via WebDAV [6] adds to the idea of heterogeneous interactive programming, where the user through its diverse operating system interfaces can manage shared data files and folders. The same data can be accessed and updated by different service providers and authorized users can monitor data processing activities executed by the service providers involved with co-operating WebDAV user agents. Like any other services in the P2P environment, the SILENUS services are also peers in the SORCER environment.

The paper is organized as follows. Section 2 provides a brief description of the dynamic service object oriented computing; section 3 describes the SILENUS methodology; section 4 presents disconnected operation and data synchronization; section 5 describes experimental results using the NFS adapter; section 6 provides concluding remarks.

2 Service Oriented Computing

Instead of thinking of a service offered by a particular host, the current paradigm shift is towards services in the network – *the metacomputer is the grid of service providers*. In classical distributed applications, it is necessary to know exactly on which host a particular service is exposed. In most distributed file systems, for example, it is necessary to know the name of a host that a particular file is stored on. In a service-oriented (SO) environment a

service provider registers itself with a service registry. The service registry facilitates lookup of services. Once a service is found a service requester binds to the service provider and then can invoke its services. Service requesters discover a registry and then lookup a needed service. On the other hand, a provider can discover the registry and publish its own service, as depicted in Figure 1.

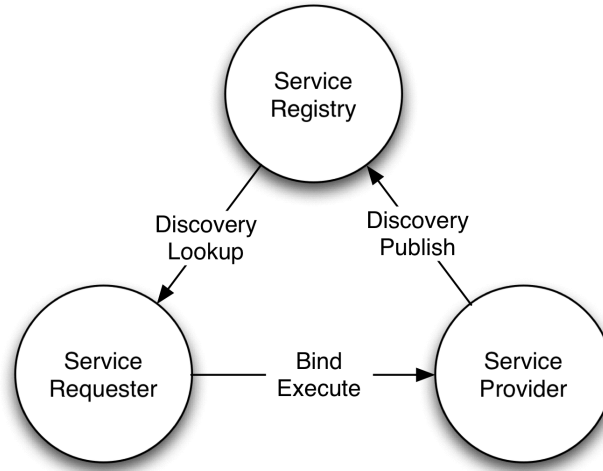


Fig. 1. Service-Oriented Architecture

In the service protocol-oriented architecture (SPOA), a communication protocol is fixed beforehand and can not be changed. Based on that protocol and a service description obtained from the service registry, the requester can bind to the service provider - create a proxy used for remote communication over the fixed protocol. In SPOA a service is usually identified by a name and/or some attributes. If a service provider registers by name, the requesters have to know the name of the service beforehand.

In the service object-oriented architecture (SOOA), a service is identified by a service type (interface) rather than its implementation, protocol. Registering services by interface has the advantage that the actual implementation can be replaced and upgraded independently from the requesters to which only interfaces have to be known. Different implementations may offer different features internally, but externally have the same behavior. This independent type-based identification allows for flexible execution of service-oriented programs in an environment with replicated services. In SOOA, a proxy—an object implementing the same service interfaces as its service provider—is registered with the registries and it is always ready for use by the requester.

Thus in SOOA a proxy is created and always owned by the service provider, but in SPOA, the requester creates and owns the proxy which has to meet requirements of the protocol the provider and requestor have agreed upon a priori. Also SOOA is protocol neutral, how an object proxy communicates with a provider is established by the contract between provider and its published proxy and defined by a proxy and provider implementation. The proxy's requester does not need to know who and how implements its interface.

In SOOA a service provider is an object that accepts remote messages from service requesters to execute an item of work. These messages are called service exertions. A task exertion is an elementary service request – a kind of elementary remote instruction (statement) executed by a service provider. A composite exertion called a job exertion is defined in terms of tasks and other jobs – a kind of network procedure executed by a service provider. The executing exertion is a SO program that is dynamically bound to all relevant and currently available service providers on the network. This collection of providers identified in runtime is called an exertion federation. This federation is also called an exertion space. While this sounds similar to the OO paradigm, it really isn't. In the OO paradigm the object space is a program itself; here the exertion space is the execution environment for the exertion that is a network OO program. This changes the programming paradigm completely. In the former case the object space is hosted by a single computer, but in the latter case the service providers are hosted by the network of computers. The overlay network of service providers is called the service provider grid and an exertion federation is called a virtual metacomputer. The metainstruction set of metacomputer consists of the method set defined by all service providers in the grid. Thus, a service-oriented program is composed of metainstructions with its own network control strategy. A service context describes the data that tasks and jobs works on. Exertion-oriented programs (metaprograms) can be created interactively [15] and allow for a dynamic federation to transparently coordinate their execution. Please note that these metacomputing concepts are defined differently in classical grid computing where a job is just an executing process for a submitted executable code.

In a federated service environment not a single service makes up the system, but the cooperation of services. A service exertion may consist of component exertions that require different types of services. Services can be broken down into small component service instead of providing one huge all-in-one service. These smaller component services then can be distributed among different hosts to allow for reusability, scalability, reliability, and load balancing.

Instead of applying these metacomputing concepts to compute services only, they can, and should, also be applied to data services as well. Once a file is submitted to the network it should stay there. It should never disappear just because a few nodes or the network segment goes down. Also, it should not matter what client node the file is requested from. With the SILENUS distributed file system in place, SORCER also provides transparent, reliable, and scalable file-based data services complementing the existing compute services.

2.1 Dynamic service object oriented computing environmet: SORCER

SORCER is a federated service-to-service (S2S) metacomputing environment that treats service providers as network objects with a well defined semantics of dynamic service object oriented architecture (DSOOA) based on the FIPER methodology [16][17][15]).

As shown in Figure 1, SOA consists of three primary entities: providers, requestors, and registries. The provider is responsible for deploying the service on the network, publishing its service to one or more registries, and allowing requestors to bind and execute the service. Providers advertise their availability on the network; registries intercept these announcements and add their services. The requester looks up services by sending queries to registries and making selections from the available service interface. Queries generally contain search criteria related to the type and quality of service. Registries facilitate searching by storing the proxy object of the service and making it available to requestors. Requesters and providers can use either unicast or multicast discovery/join protocols [8] to publish services on the network. Please note the important distinction between SORCER's services and SPOA like web services or grid services (e.g., Globus). In DSOOA, a proxy – an object implementing the same service interfaces as its service provider – is registered with the registries and it is always ready for use by the requestor. Thus, in DSOOA, the proxy is published as the active surrogate object by the service provider and can be found dynamically with no a priori knowledge about the location of registries. In SPOA, by contrast, a passive service description is registered (e.g., an XML document in WSDL, or in IDL for CORBA); the requester then has to generate the proxy based on the fixed communication protocol and a service description obtained from a registry in a priori known location to the requester. This is referred to as a bind operation. This operation is not needed in DSOOA since the requester already holds the active surrogate object obtained from a registry. The binding operation has to obey a protocol established by SPOA (e.g., the SOAP or IIOP protocol cannot be changed). Thus, web services and grid services cannot change the communication protocol between requestors and providers while DSOOA is protocol neutral. In the DSOOA, how an object proxy communicates with a provider is established by the contract between provider and its published proxy and defined by a proxy and provider implementation. The proxy's requester does not need to know who and how implements its interface. So-called smart proxies are capable of processing locally on the requester side by themselves; they can also communicate with multiple providers on the network regardless of who originally registered the proxy. Thus, different parts of the smart proxy interface can be implemented by separate providers on the network; communication protocols may also vary.

Each SORCER provider offers services to other peers on the object-oriented overlay network. These services are exposed indirectly by methods in

well-known public remote interfaces and considered as elementary (tasks) or compound (jobs) statements of DSOOA [16]. Requesters do not need to know the exact location of a provider beforehand, they can find it dynamically by discovering service registries (lookup services) and then looking up a needed service implementing required service types.

A SORCER program can be created interactively [15] or programmatically (using SORCER APIs) and their execution can be monitored and debugged in the overlay service network [19]. Service providers do not have mutual associations prior to the execution of an exertion; they come together dynamically (federate) for all component tasks and jobs in the exertion.

Each specialized provider in the federation executes either tasks. A job is coordinated by a Jobber (or subclass of it) - one of SORCER infrastructure services [16]. However, a job can be sent to any peer. A peer that is not a Jobber type is responsible for forwarding the job to one of available existing jobbers in the SORCER grid and returning results to the requester. Thus implicitly, any peer can handle any job or task. Once the job execution is complete, the federation dissolves and the providers disperse and seek other exertions to join. Also, SORCER supports a traditional approach to grid computing - like in Condor [22] and Globus [4] style. Here, instead of exertions being executed by services providing a business logic for requested exertions, the business logic comes from the service requester's executable programs that seeks compute resources on the network provided by grid services. These services in the SORCER environment are as follows: Grider and Jobber for traditional grid job submission; Caller and Tasker for task execution [10]. Callers execute conventional programs via a system call as described in a service context of the submitted task, while taskers download a mobile code from requesters to process a submitted context accordingly. In either case the business logic comes from requesters, it is pushed to callers by requesters or pulled by taskers from requesters.

To integrate applications and tools on a business-to-business (B2B) grid with shared engineering data, the File Store Service (FSS) [18] was developed as a core service in FIPER/SORCER. The value of FSS is enhanced when both web-based user agents and service providers can readily share the content in a seamless fashion. The FSS framework fits the SORCER philosophy of grid interactive exertion-based programming, where users create distributed programs using exclusively interactive user agents. However FFS does not provide the S2S flexibility with separate specialized and collaborating service providers for file storage, replication, and meta information that are presented in this paper.

3 SILENUS methodology

SILENUS is based on a dynamic service object oriented architecture. As such, it consists of individual service objects, which, when combined, provide the

SILENUS functionality. These components can broadly be categorized into gateway components, data services, and management services. Figure 2 gives an overview of the SILENUS architectural components.

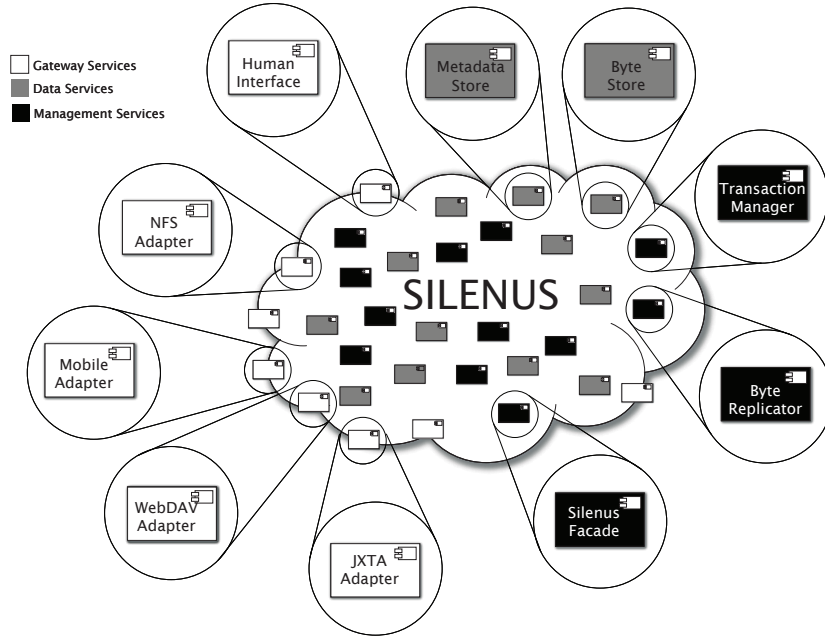


Fig. 2. Component services of the SILENUS architecture

To store data in the SILENUS file system, the following assumptions about the data are made:

- File metadata is relatively small. Therefore there is no a problem to replicate the file metadata.
- File content is relatively large. Therefore files should be replicated for reliability, but not onto every system.
- Management data, e.g., proxies to needed services and transactions, can be handled autonomically. That type of data does not have to be known to requesters explicitly, but it can be dynamically discovered in runtime in the SILENUS environment.

3.1 Gateway services

The Service Interface, NFS Adapter, Mobile Adapter, WebDAV [6] Adapter, and JXTA [9] Adapter are client modules. Each one of them serves a particular type of client. The ones given here are just examples, adapters could

be written for any other existing file storage solution. The service interface (ServiceUI) [14] provides support for file storage and management through a proprietary user interface attached to a service provider. It provides access to the extra features, which are not available through the other interfaces: advanced features such as manual migration, number of replicas, log-file viewing, and others. The service interface should only be needed for these extra features and can be ignored by most users. The WebDAV Adapter provides support for operating systems that have support for WebDAV, such as Windows, Mac OS X, and newer UNIX systems. It provides support for existing applications. This gives current operating systems the possibility to use the SILENUS file storage directly with no need to install any additional support. The NFS adapter provides support for older UNIX systems that do not have WebDAV preinstalled. A JXTA adapter provides support for the JXTA content management interface. These adapters are just examples of various mappings from SILENUS to existing systems, other adapters can be developed as well.

3.2 Management services

The SILENUS Facade and Transaction Manager provide coordination services. To make the client modules even smaller, the coordination between the client modules and the providing services is sourced out to the SILENUS Facade. Facades are gateways to the SILENUS file storage for both user agents and service providers. Each Facade provides a dynamic entry point to the underlying SILENUS file metadata and content storage services. It takes care of transactional semantics between file content and meta information storage. The facade provides support for discovering the relevant services that participate in a requester's file upload/download federations. A Transaction Manager is used for ensuring two-phase commit transactional semantics for file uploads that involves at least metadata store and byte store services. The Transaction Manager used in SILENUS is a Jini [3] standard service for handling transactions in a distributed environment.

The Byte Replicator and other optimizer services provide support for automatic computing. In a classical data storage solution, an administrator has to manually move and distribute files among different servers. In SILENUS, this is done by optimizer services. These services will analyze the current network conditions and make decisions on where to store files, where to keep replicas, and even when to startup and shutdown services needed and underutilized services. Each optimizer service is a separate component, allowing the SILENUS administrator to choose exactly which kinds and how many optimizer services to run on the network. One example of these services is the Byte Replicator service. This provider makes sure that uploaded files are replicated among different byte store nodes to provide required redundancy. Optimizer services can request log information from the storage providers, and can automatically initiate replication and migration. Optimizers can detect usage patterns and make sure that the files are available to the user in the most preferred location

possible. Also optimizers can detect non-responding providers and automatically replicate all files that were stored on it. The Replicator also ensures that all storage servers have the latest version of the persisted files.

3.3 Data services

The Byte Store provides functionality for creating and retrieving file content. The Byte Store does not provide file attribute storage. It does, however, provide support for retrieving attributes that are derived from the file data. Such attributes include file size and checksums. These can be used to verify the integrity of file contents. The Byte Store provides fast access to the files stored on the provider's host. Stored files are usually encrypted, but can be stored unencrypted for performance reasons.

The Metadata Store provides functionality to create, list, and traverse directories. It also provides functionality to retrieve the file data location. File metadata is all the information that is either included in the actual file data or that can be derived from the file data, such as file name, creation date, file type, type of encryption, and others. As a matter of fact, the file storage location, the file name, and even the directory a file is in are nothing different than just three file attributes. This allows all these attributes to be handled in a standard way persisted in the Byte Store's embedded relational database. Multiple versions of one file may exist in the database for recovery purpose.

4 Disconnected operation and data synchronization

Metadata stores are synchronized while connected. All metadata stores contain the same information. Should a metadata store be disconnected while its information changes, it will be resynchronized when it is connected back to the other metadata stores.

Changes in file metadata occur every time the file content stays the same, but new information for the file is available. This does not only refer to the classical file metadata, such as file owner, or file name, but to all information stored in the metadata store. It includes information such as the directory a file is in or the location of the file content.

Each change in the file metadata triggers an event that needs to be sent to the other metadata stores. Since the metadata stores should contain the same information, it is necessary to synchronize them continuously.

Unlike the metadata stores, the byte stores are not synchronized. File data (content) is much larger than file metadata. Would the file data be replicated on every node the storage capacity would be filled very quickly. It is the job of the optimizer services to provide appropriate file data replication.

Changes in file content are handled through auto versioning. Every time a file is saved, a new version of that file is created. The old versions will stay intact. Instead of an actual change in file data, the change operation now

becomes two operations: a change in file metadata, and the upload of new file content. The change in file metadata is handled with the same metadata synchronization process as for the regular metadata change. The same upload process as for a new file handles the change in new file content.

5 Experimental results

The SILENUS system was designed and implemented as part of a dissertation research at Texas Tech University [2]. Over the course of three years, the system has been designed, refined, and implemented. The core services, some management services, and some gateway services have been implemented and deployed in the SORCER Lab environment. The NFS adapter was used to test the SILENUS framework performance.

What	0 KB	10 KB	1MB	100MB
Disk to disk (local)	0.0 sec	0.0 sec	0.0 sec	0.7 sec
Disk to SILENUS	0.2 sec	1.6 sec	1.7 sec	22.8 sec
SILENUS to disk	0.0 sec	0.1 sec	0.2 sec	16.6 sec

Fig. 3. Data collected for SILENUS performance using the NFS adapter in a 100 MBit network.

Figure 3 shows that the performance of the SILENUS system is not so much dependent on the actual file size but rather on the number of requests. Creating an empty file is almost instant, but it still requires a file metadata creation. Retrieving an empty file is instant, as there is no file content to retrieve. For small files, the time for creating the file is about 2 seconds, not really dependent on the file size. Retrieving a file is much faster: no transaction is needed and no modifications are done. For a large file, the actual network performance shows up as indicated in Figure 3. Without any overhead, a 100 MB file could be transferred in about 9.3 seconds. For file upload, the SILENUS system reaches 40% of the maximum network performance. For file download this increases to 56% of the maximal network performance. Given the overhead of locating the file, transferring it from a byte store to the NFS adapter, and through the NFS protocol to the local host these values are very satisfying.

6 Conclusions

This paper highlights the issues involved in designing and implementing federated file systems and demonstrates the feasibility of such deployment in CE federated environments. The presented SILENUS architecture shares the

attributes of grid systems, P2P systems, dynamic service object oriented programming, and inheriting the security provided by Java/Jini security services. It is modularized into a collection of core distributed providers with multiple remote Facades. Facades supply with a uniform access points via their smart proxies available dynamically to file requesters. A Facade smart proxy encapsulates inner proxies to federating providers accessed directly (P2P) by file requesters.

Core SILENUS services have been successfully deployed as SORCER services along with WebDAV and NFS adapters. The SILENUS file system scales very well with a virtual disk space adjusted as needed by the corresponding number of required byte store providers and the appropriate number of needed metadata stores to satisfy the needs of current users and service requesters.

The system handles very well several types of network and computer outages by utilizing the presented disconnected operation and data synchronization mechanisms. It provides a number of user agents including a zero-install file browser (service UI) attached to the SILENUS Facade. This file browser with file upload and download functions is combined with an HTML editor and multiple viewers for documents in HTML, RTF, and PDF formats. Also a simpler version of SILENUS file browser is available for smart MIDP phones.

References

1. Max Berger and Michael Sobolewski. Silenus - a federated service-oriented approach to distributed file systems. In *Next Generation Concurrent Engineering* [7].
2. Maximilian Berger. *SILENUS – A Service Oriented Approach to Distributed File Systems*. PhD dissertation, Texas Tech University, Department of Computer Science, December 2006.
3. W. Keith Edwards. *Core Jini*. P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2001.
4. Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid services for distributed system integration. *Computer*, 35(6):37–46, June 2002.
5. Sanjay Goel, Shashishekara S. Talya, and Michael Sobolewski. Preliminary design using distributed service-based computing, 2005.
6. Y. Goland, E. Whitehead, A. Faizi, S. Carter, and D. Jensen. HTTP Extensions for Distributed Authoring – WEBDAV. RFC 2518 (Proposed Standard), February 1999.
7. ISPE. *Next Generation Concurrent Engineering*. Omnipress, 2005.
8. Jini. Jini Discovery & Join Specification, March 2007. <http://java.sun.com/products/jini/2.1/doc/specs/html/discovery-spec.html>.
9. JXTA. Project JXTA, March 2007. <http://www.jxta.org/>.
10. Vivek Khurana, Max Berger, and Michael Sobolewski. A federated grid environment with replication services. In *Next Generation Concurrent Engineering* [7].
11. M. Lapinski and Michael Sobolewski. Managing notifications in a federated S2S environment. *International Journal of Concurrent Engineering: Research & Applications*, 11:17–25, 2003.

12. P.J. Rohl P.J., R.M. Kolonay, R.K. Irani, M. Sobolewski M., and K. Kao. A federated intelligent product environment. In *8th AIAA/USAF/NASA/ISSMO Symposium on Multidisciplinary Analysis and Optimization*, Long Beach, CA, sep 2000.
13. R.M. Kolonay R.M., M. Sobolewski, R. Tappeta, M. Paradis, and S. Burton. Network-centric MAO environment. In *The Society for Modeling and Simulation International, 2002 Western Multiconference*, San Antonio, Texas, 2002.
14. ServiceUI Project. The ServiceUI project, March 2007. Retrieved from <http://www.artima.com/jini/serviceui/>.
15. Michael Sobolewski and R. Kolonay. Federated grid computing with interactive service-oriented programming. *International Journal of Concurrent Engineering: Research & Applications*, 14(1):55–66, 2006.
16. Michael Sobolewski. Federated P2P services in CE environments. In *Advances in Concurrent Engineering*, pages 13–22. A.A. Balkema Publishers, 2002.
17. Michael Sobolewski. FIPER: The federated S2S environment. In *JavaOne, Sun's 2002 Worldwide Java Developer Conference*, San Francisco, 2002. <http://sorcer.cs.ttu.edu/publications/papers/2420.pdf>.
18. Michael Sobolewski, Sekhar Soorianarayanan, and Ravi-Kiran Malladi-Venkata. Service-oriented file sharing. In *CIIT conference (Communications, Internet and Information Technology)*, pages 633–639, Scottsdale, AZ, November 2003. ACTA Press.
19. Sekhar Soorianarayanan and Michael Sobolewski. Monitoring federated services in CE. In *Concurrent Engineering: The Worldwide Engineering Grid*, pages 89–95. Tsinghua Press and Springer Verlag, 2004.
20. SORCER. Laboratory for Service-Oriented Computing Environment, March 2007. <http://sorcer.cs.ttu.edu/>.
21. Sun Microsystems. Jini architecture specification, 2001. <http://www.sun.com/software/jini/specs/jini1.2html/jini-title.html>.
22. Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the Grid. In Fran Berman, Anthony J.G. Hey, and Geoffrey Fox, editors, *Grid Computing: Making The Global Infrastructure a Reality*. John Wiley, 2003.
23. S. Zhao and Michael Sobolewski. Context model sharing in the FIPER environment. In *8th Int. Conference on Concurrent Engineering: Research and Applications*, Anaheim, CA, 2001.