Autonomic System Management in Mobile Grid Environments

Michael Messig and Andrzej Goscinski
School of Engineering and Information Technology
Deakin University
Pigdeons Road, Geelong 3217, Victoria, Australia
{messig, ang}@deakin.edu.au

Abstract

Mobile device integration in grid environments is a challenge for many researchers. Due to the transient nature of mobile devices, service management is a critical, but often overlooked area of research. We propose a distributed broker responsible for the autonomic management of grid services. The broker provides self discovery and negotiation, self configuration and self healing for SOA based mobile grids. In this paper the design and prototype implementation of the broker is presented and the importance of autonomic grid service management is shown.

Keywords: Autonomic Principles, Grid Computing, Mobile Devices

1 Introduction

The connectivity and ease of use of mobile devices has led to an explosive adoption of various mobile technologies. Wireless hot spots, broadband and 3G providers allow transient access to the World Wide Web through browsers on a variety of devices, from laptops and PDAs to smart phones. These devices, although usually equipped with limited resources (CPU, Memory), are able to serve as clients on behalf of users requesting services in Service Oriented Grid (SOG) environments.

SOGs are comprised of many systems offering numerous services in a large, heterogeneous, interconnected environment. Heterogeneity in SOGs is addressed by employing Web services, allowing clients to discover and invoke services regardless of the client or service provider's architecture. This is beneficial for mobile devices, which are Internet-enabled through wireless networks and therefore are able to communicate with Web services through the standard Hypertext Transport Protocol (HTTP) and the Simple Object Access Protocol (SOAP) (Gudgin et al., 2003). Although mobile communication is possible through these protocols, the added complexity of Web services and the impact of quality of service requirements and trust make SOGs hard to manage and susceptible to failure (Messig and Goscinski, 2005, Medeiros et al., 2003). SOGs lack vital components which aid in the ability to manage services, negotiate terms of service provision and support the evaluation of trustworthiness. These are open issues in SOG research, and therefore must be addressed to provide mobile device integration.

We have shown in (Messig and Goscinski 2005) and (Messig and Goscinski 2006) that it is possible to improve SOG environments by employing autonomic principles. Autonomic principles (Horn 2001) directly address system complexity by proposing transparent, automatic management of system components. By introducing autonomic principles in a SOG environment, the grid is able to automatically discover components of the system and learn about interactions between users and service providers. The grid is able to use this information to configure and reconfigure itself and provide fault tolerance by transparently recovering from failures. This allows the SOG to become a robust and scalable platform which provides transparency by reducing complex management of services (Horn 2001, Messig 2004).

We propose a System Management Broker (SMB) which offers autonomic mechanisms to services within a SOG. The broker provides self discovery and negotiation, self configuration and self healing to grid services by employing brokering and proxy approaches. The SMB is designed to offer a high level of interoperability and flexibility to ensure that the broker is able to be utilised by various clients, including a variety of mobile devices. To maintain scalability, the broker is distributable across various virtual organisations within the SOG. To reduce complexity, the SMB maintains full transparency, thus providing a significant improvement to grid service management.

Besides the above mentioned provision of autonomic principles to easily and transparently manage mobile grid environments and a SMB, our main contributions are as follows: first, the proposal and design of an SMB components that provide self discovery and negotiation, self configuration and self healing; second, a proof-of-concept in the form of prototype implementation and demonstration of its functionality.

The paper is structured as follows. Section 2 shows related work in SOG, autonomic systems and mobile devices for grid computing. Section 3 provides an overview of an SOG and outlines key terminology used in this paper. Section 4 provides an overview of the SMB and describes our testbed design and Section 5 discusses the testbed implementation of this design. Section 6 outlines the testing of the SMB by managing an auction service (developed using current Web service technology and standards such as WSRF (Czajkowski, et al., 2004)) invoked from a mobile device. Finally, section 7 concludes and provides an outline for future work.

2 Related Work

Mobile devices have been researched for use in areas such as ubiquitous and mobile systems (Weiser, 1991, Snoeren et al., 2001) and more recently distributed and grid systems (Migliardi et al., 2004, Phan et al., 2002). Some of these initiatives include Condor support for mobile devices (Ganzalez-Castano, et al., 2002), mobile devices for collaboration (Grabowski et al, 2006), access to Web services for mobile devices (Yang and Bouguettaya, 2006) and P2P based approaches for ad-hoc mobile networks (Baumung et al., 2006).

These systems generally address single aspects of grid computing, such as remote service invocation, and do not offer any service management, negotiation of quality of service (QoS) or trust assessment.

We have shown in (Messig and Goscinski, 2005) that autonomic service management is a feasible approach. Our previous systems however did not take into consideration negotiation of QoS and trust and also failed to address the transient nature of mobile devices as clients. Therefore we have redesigned our system with a focus on a mobile environment.

3 System Overview

This section shows introduces the SMB and how mobile device clients are provided with SMB support. SOG environments focus on sharing resources and services between distributed users, sites and organisations. Figure 1 shows an example of an SOG consisting of disparate organisations connected by fast wide area network, such as Grangenet (Grangenet 2005) or the Internet. Each organisation contains users, resources and service providers which offer services. Resources can be clusters, workstations, or any other type of resources; this is represented in Figure 1 as a generic resource. A generic resource does not necessarily have to be a compute resource, but could also include printers, storage systems, or even robotic equipment. These resources are utilised and presented to the user through Web services. The organisations offer services to clients, including a number of mobile devices, other organisations and the public which are able to consume services offered by an organisation. For example, the school of Engineering and Information Technology at Deakin University may offer services such as high performance computing which utilises cluster resources, whereas a financial institution may offer an accounting or market prediction service.

Organisations which work on similar tasks or perform coordinated resource sharing form a virtual organisation (Foster 2001). For example, in Figure 1, Deakin University might perform collaborative SOG research with Monash University and therefore these organisations may share services and resources. This arrangement forms a virtual organisation. The problem with this environment however is that there is no means of managing the services offered by the organisations. Management of services must ensure that the services remain available, are reliable and perform as required.

The complexity of this environment is also a problem; services and resources must be manually configured, managed and maintained. This requires specialised expert knowledge in grid computing.

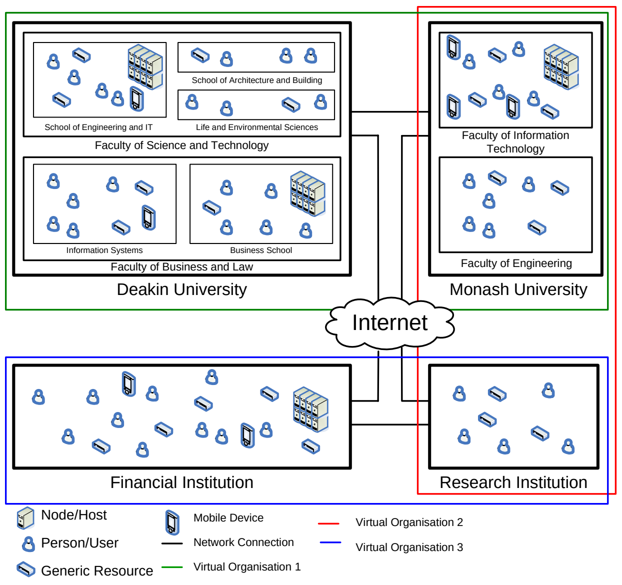


Figure 1. Virtual organisations sharing services and resources in an SOG

As seen in Figure 1, a grid environment is not necessarily focused towards Information Technologists and is not necessarily managed by experts in grid computing. Grids have been utilised by many industries outside of Information Technology and high performance computing. Industries such as the finance sector, bioinformatics, life sciences and government agencies all benefit from grid computing (IBM 2006). This requires that grid systems hide the complexity involved in discovery, configuration and the failure recovery of services. This can be addressed by providing transparent management of services and service providers and ensuring that services are able to be deployed, configured and monitored automatically (Messig and Goscinski 2006).

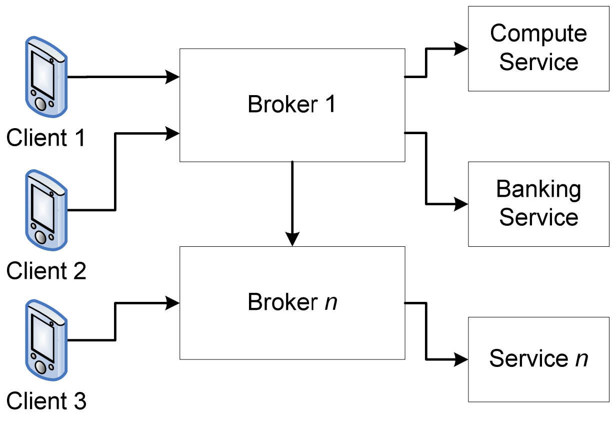


Figure 2. Interaction between clients, brokers and services

Transparency is crucial in a grid environment, and therefore, any solution that provides management must be fully transparent for the client and service providers. The incorporation of the broker in a SOG where a number of mobile clients discover and invoke services is shown in Figure 2. By taking this approach we ensure that firstly, scalability is not affected and provides interaction across many organisations, and secondly, that the SMB maintains interoperability by not being restricted to a single implementation or architecture. Also, as the broker is distributed, replication is possible which alleviates the grid from any failure of a centralised broker.

4 Testbed Design

Providing a distributed approach requires that the design of the SMB must be modular. Therefore, we break the SMB into logical partitions, each responsible for an autonomic function. Figure 3 shows the SMB divided into three autonomic managers; the self discovery and negotiation, self configuration and self healing managers. Each of these managers then contains a set of services which provide the relevant functionality.

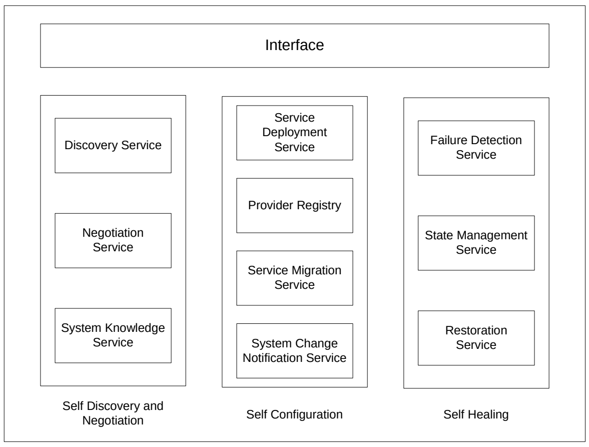


Figure 3. System components

The self discovery and negotiation manager employs discovery, negotiation and system knowledge services. The self configuration manager contains service deployment, provider registry, migration and system change notification services. Finally, the self healing manager provides failure detection, state management and restoration services. These components are detailed in the following subsections.

4.1 Self Discovery and Negotiation Services

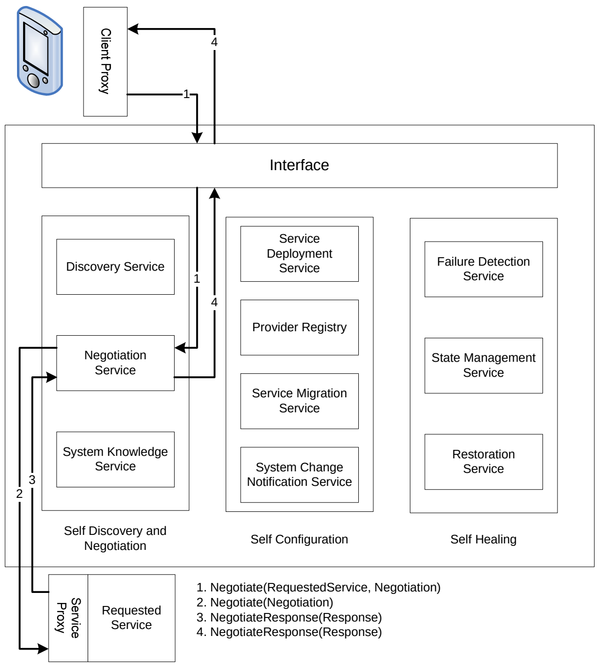
Self discovery requires actively discovering components and gathering knowledge about the grid environment. This knowledge must then be recorded so that other services such as self healing and self configuration can utilise this information. Self discovery differs from discovery; self discovery is done by the broker to discover information about the broker's environment; while discovery is done by clients to discover services. Negotiation is also part of self discovery. Once services are discovered by the broker, the broker is able to negotiate the terms and conditions (QoS) of invoking a service with the service provider and the trustworthiness of both the client and service provider.

4.1.1 The Discovery Service

The discovery service coordinates the self discovery of brokers, autonomic managers, service providers and services within the grid. Due to the distributed nature of grid systems and as discussed in previous sections, it is possible to have numerous SMB's within a grid environment. It is also possible that the autonomic managers within the SMB's are distributed. The discovery service actively queries discovery systems and polls any discovered broker components and queries them to ensure they are accessible before adding them to a maintained list of available broker components. A similar approach is taken with the discovery of service providers and services. The providers are then queried to ensure they are available and if so, information about the service provider and the services offered by the provider is requested. This allows the SMB to build grid wide knowledge about the services offered by service providers within the grid.

4.1.2 The Negotiation Service

The negotiation service is responsible for providing negotiation between clients and service providers and between SMBs. This includes negotiating the terms of dynamic service deployment and migration of services and also allows clients to negotiate the terms of service selection through QoS parameters.



Client Proxy
1
4
Interface
Discovery Service
Negotiation Service
System Knowledge Service
Service Deployment Service
Provider Registry
Service Migration Service
System Change Notification Service
Failure Detection Service
State Management Service
Restoration Service
Self Discovery and Negotiation
Self Configuration
Self Healing
2
3
Service Proxy
Requested Service
1. Negotiate(RequestedService, Negotiation)
2. Negotiate(Negotiation)
3. NegotiateResponse(Response)
4. NegotiateResponse(Response)

Figure 4. Proxy service negotiation

To demonstrate how a negotiation takes place, Figure 4 depicts a negotiation between a mobile client and a service provider. The mobile client's proxy invokes the SMB providing the negotiation terms on behalf of the client (Message 1). The negotiation terms which consists of a number of dynamic parameters (such as price, execution time etc.) and conforms to the draft WS-QoS specifications (Tian, 2005). The request is passed to the negotiation service which contacts the service provider's proxy (Message 2). The service proxy processes this request and decides on the outcome of the negotiation, responding with a negotiation response (Messages 3 and 4). This response may also contain a counter offer (another negotiation request). This is important as the service proxy may fulfil the client’s negotiation request however may negotiate for a higher price for fulfilling the service request. While Figure 4 shows only a single negotiation request and response, the mobile client and service provider can negotiate multiple times until the request is aborted or an agreement is met.

4.1.3 The System Knowledge Service

The system knowledge service is responsible for collecting information about the grid, including service providers, services and SMB components. The collected information is then utilised by other SMB autonomic managers to make decisions about the management, configuration, deployment and recovery of services.

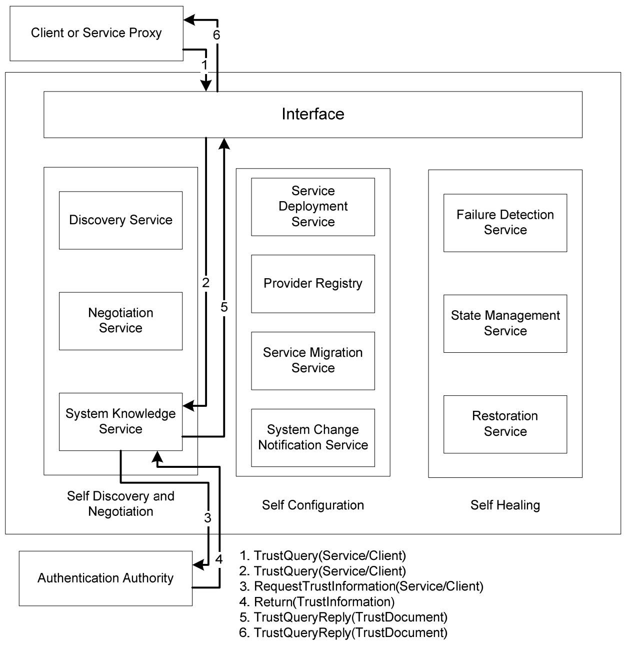

Client or Service Proxy
6
1
Interface
Discovery Service
Negotiation Service
System Knowledge Service
Self Discovery and Negotiation
2
5
Service Deployment Service
Provider Registry
Service Migration Service
System Change Notification Service
Self Configuration
Failure Detection Service
State Management Service
Restoration Service
Self Healing
3
4
Authentication Authority
1. TrustQuery(Service/Client)
2. TrustQuery(Service/Client)
3. RequestTrustInformation(Service/Client)
4. Return(TrustInformation)
5. TrustQueryReply(TrustDocument)
6. TrustQueryReply(TrustDocument)

Figure 5. Proxy trust query

The system knowledge service is also responsible for the assessment of trustworthiness of both clients and service providers. Figure 5 shows how a trust query is carried out. The assessment of trust is requested by either a client or service provider’s proxy that is passed from the SMB interface to the system knowledge service (Messages 1 and 2). The system knowledge service must then query known authentication authorities (Messages 3 and 4) to verify and gauge the trustworthiness of the subject being queried. The system knowledge service returns a trust document containing the assessment of trustworthiness back to the requestor (Messages 5 and 6). The trust assessment conforms to the WS-Trust specifications (IBM, 2005).

4.2 Self Configuration Services

Self configuration requires the ability to monitor the SOG and make changes automatically to the configuration of the grid. Service providers which are subscribed to the SMB provide feedback about changes within the grid to allow the SMB to configure and reconfigure the system to adapt to these changes. Self configuration also requires that services are able to be moved around the grid and deployed on selected service providers. To perform these operations, the system change notification, provider registry and service migration services are employed.

4.2.1 The System Change Notification Service

The system change notification service is responsible for managing the configuration and reconfiguration of the grid. The service is responsible for detecting when the grid environment changes and reconfigures the grid to deal with the change. An event based approach is used, where the broker is notified when specific events occur. For example, a service provider may be removing itself from the grid, a timer may expire or a client may have exceeded the funds allocated to use a particular service. When service providers are advertised to the SMB through the provider registry or a service is deemed to have failed due to detection by the failure detection service, the system change notification service is notified. The system change notification service is then able to determine if any reconfiguration of the system is required. This may include invoking the state, restoration or service migration services to move services from within the grid to other available, better suited hosts or even different service providers. Reconfiguration could also involve broker replication. This is done by replicating information about the grid from one broker to other brokers.

4.2.2 The Provider Registry

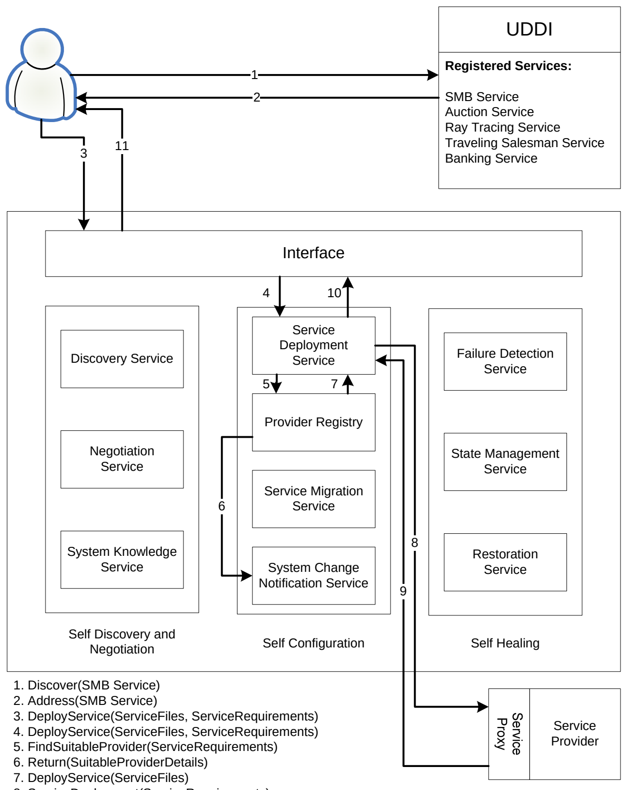
The provider registry is responsible for storing information about service providers and hosts which have been advertised to the SMB. The information stored about the service provider and the hosts available by the service provider is used when selecting a suitable candidate for deploying, restoring or migrating services. The provider registry must take the metrics of the system into account and can use mechanisms such as global scheduling algorithms to decide which provider is the best suited to deploy, restore or migrate a service.

4.2.3 The Service Migration Service

The service migration service is responsible for coordinating the migration of a selected service from one host at a service provider to another host at the same service provider, or a host at an alternative service provider. The migration service aggregates all the required information about a service prior to migration and moves this information to the destination. This requires coordination with the negotiation service, state management service, restoration service, provider registry and system knowledge service. When a service requires migration, the migration service must find a suitable host as a candidate for the migration. If the service is to be migrated to another service provider, the migration service must negotiate with the destination provider to reach an agreement for the service migration. This ensures that the destination provider agrees to the migration of the service, essentially giving permission for the migration service to coordinate service migration.

4.2.3 The Service Deployment Service

The service deployment service coordinates the deployment of a service on a host. An author of a service, or user authorised to deploy a service within the SOG, submits the service to be deployed to the service deployment service. The SMB invokes functionality of the provider registry to select the most appropriate service provider for deployment of the service. The service deployment service keeps a record of all services which it has deployed and is also responsible for registering the deployed service with a discovery service. This allows the service to later be replicated or relocated to another service provider within the grid. The service deployment service is also responsible for providing information about the deployment of services to the system knowledge service.



1. Discover(SMB Service)
2. Address(SMB Service)
3. DeployService(ServiceFiles, ServiceRequirements)
4. DeployService(ServiceFiles, ServiceRequirements)
5. FindSuitableProvider(ServiceRequirements)
6. Return(SuitableProviderDetails)
7. DeployService(ServiceFiles)
8. ServiceDeployment(ServiceRequirements)
9. Return(Success)
10. Return(Success)
11. Return(Success)
12. Return(Success)

Figure 6. Dynamic Service Deployment

The dynamic deployment of a service is demonstrated in Figure 6. Firstly, the user deploying the service discovers an available SMB through a known discovery service, such as UDDI, (Messages 1 and 2). The user then invokes the SMB and provides the relevant files for the service and a description of the service's requirements (Messages 3 and 4). The deployment service finds a suitable provider to deploy the service (Messages 5 and 6) and then attempts to deploy the service by invoking the destination's service proxy (Message 7) and if the deployment is successful, the success messages are returned to through the SMB services back to the user.

4.3 Self Healing Services

Providing self healing mechanisms for heterogeneous grid services requires several mechanisms to coordinate the identification of failed services, manage service state information and restore service availability within the grid. The failure detection, state and restoration services are responsible for providing these mechanisms. These services also coordinate with the self discovery and self configuration managers as well as the client and service proxies to allow failed grid services to be identified and restored.

4.3.1 The Failure Detection Service

The SMB must be able to detect when action is required due to the failure of a service or service provider within the grid. This requires a failure detection mechanism which, when a failure is detected, must act on behalf of the affected services. This also requires being able to transparently recover from the failure and provide the user of the failed service with a restored version of the service being used. The failure detection service is responsible for monitoring services within the grid and alerting associated autonomic managers when a failure is detected.

Failure detection is achieved through several subscription mechanisms. A heartbeat mechanism requires subscribed services and resources to notify the broker of their existence at given intervals; this notification service is provided by the SMB service proxy on the service provider. The SMB is also able to poll services to determine whether the service should be deemed failed. The failure detection service uses a retry mechanism to detect a failure and applies a back off timeout period to ensure the failure is not transient. By providing a mechanism to detect failures, the grid is able to attempt to automatically recover or relocate the failed services transparently.

4.3.2 The State Management Service

The state management service is responsible for two operations, storing the state of a service and providing the stored state to the restoration service when a service must be restored. The state management service must be able to store the state of a service from information provided about the service by the service proxy in stable storage. The state of the service must be described in a standardised format, such as a well defined serializable schema, allowing the state manager to process and store the state of services from different heterogeneous systems. The service proxy located on the service provider is responsible for delivering the state information to the broker in this format. A proxy is employed on the client side which is responsible for delivering state information in this format to the state management service.

4.3.3 The Restoration Service

The restoration service is responsible for coordinating the restoration of a failed service. The restoration service must communicate with the failure detection service, state management service and the service deployment service to recreate a failed service or move a service by interrupting its execution on the current host and recreating the service at another location. A suitable destination for the restoration of the failed service is determined by the service deployment service which invokes the provider registry to execute this decision. The restoration service coordinates the retrieval of any available state information from the state service and supplies this to the service deployment service.

4.4 Proxies

There are two proxies that are required in an environment managed by a SMB. Each service provider that is to be managed by the broker requires the service proxy which is responsible for communication between the broker and a Web service and each client that is to use SMB service requires a client proxy.

4.4.1 The Service Proxy

The service proxy is required on each host which hosts services that utilise the SMB, allowing the host to transparently interact with the broker. The service proxy can be developed in any language which supports standard Web service interactions, however must conform to the standard SMB interface. This ensures that valid SMB interface invocations are made.

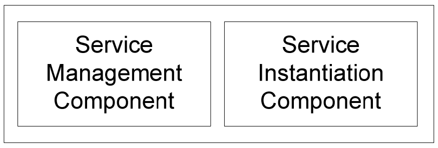


Figure 7. The Service Proxy

The service proxy contains two components, the service management component and the service instantiation component as seen in Figure 7. The service management component is responsible for discovering and subscribing to SMBs by utilising discovery services, identifying services running on the service provider and transmitting the state of the services to the SMB. The service instantiation component of the service proxy is responsible for instantiating a service on the given host. The service's details are acquired from the SMB, including the service's binary files, requirements and if available, the current state of the service. The instantiation component then reconstructs the service on the service provider and notifies the broker that the service has been instantiated. The broker notifies the system knowledge service about the service and its instantiation on the service provider.

4.4.2 The Client Proxy

The client proxy is located on the Web service's client and is responsible for transparently interacting with the SMB on behalf of the client. The client proxy takes requests from the Web service client (such as a mobile device) and transparently invokes functionality of the SMB. The client proxy assists with discovering SMB managed services as well as reporting failed services. The client proxy is compiled with client, such as the mobile device's client code to allow SMB interaction from the mobile device.

4.5 Summary of the Testbed Design

The components which make up the SMB are designed to provide transparency when utilising SMB services as well as the ability to distribute individual SMB components. By separating functionality into a set of sub services, it is possible to distribute these services across different systems within an SOG environment. The following sections explore the testbed implementation of the SMB and demonstrate the effectiveness of autonomic service management in a mobile device SOG environment.

5 Testbed Implementation

To demonstrate that our claim is sound we have implemented the testbed design of the SMB. This section reports on the implementation of both the SMB and the testing of the broker using an auction service in the .NET environment. We selected an auction service for our experimental study because it could be one of the most commonly consumed services (by business and ordinary people) via a mobile device. The client for the auction service is a PDA connected via a wireless network within the SOG.

5.1 Prototype Implementation of the SMB

A prototype implementation of the SMB was developed and tested in the .NET environment and is written as a set of C# Web Services. A single Web service is provided as the interface to the subset of SMB Web services. This allows the SMB to provide a single standard interface to services which utilise the SMB and is written using WSDL. Clients of the SMB written in any language are able to discover the interface to the broker and invoke methods provided by the broker. There are several methods which are implemented as part of the prototype, of these the most important methods include:

SaveState() – This method is utilised by the service proxy to save the state of an auction. The method takes the state of the auction service as an argument in the format of an XML document and saves it to stable storage.

RestoreState() – The RestoreState() method is used by the ReconstructService() method to restore the state of the service. This method reads the state of the service saved by the SaveState() method from disk and restores the state of the service, this includes any WSRF resources which exist (such as the bid value of the auction service). If there is no state saved for the service, this method returns an error.

ReconstructService() – This method firstly invokes the RestoreState() method to restore the state of the service, if this is successful the new address of the service is provided to the calling method, otherwise an error is returned.

ServiceRequest() – The ServiceRequest() method is invoked by the client proxy. This method is responsible for firstly determining whether the service has actually failed, by attempting to invoke the service. If this is the case, the ReconstructService() method is then invoked and provided no errors have occurred, the new address of the service returned by this method to the client.

Two prototype proxies are also implemented, the client proxy and the service proxy. The client proxy is written as a C# class and is compiled with the client application. The proxy includes a method which takes the client’s invocation of a Web service, such as invoking the bid() method of the auction service, and actually invokes the Web service. This is done to ensure that if there is no response from the service, the client proxy is able to invoke the ReconstructService() of the SMB. If the ReconstructService() method is invoked, the client proxy is then aware of the new address of the service if it has been restored by the SMB.

The implementation of the service proxy is also developed as a C# class. This proxy provides the SMB with the state of the service in the format of an XML document when a method is invoked by a client and data is changed, such as the bid value of the auction service. When the state of the service changes, the state is provided to the SMB by invoking the SaveState() method.

5.2 Implementation of the Auction Service

The auction service is implemented using the WSRF.NET implementation of the Web Services Resource Framework. WSRF.NET is developed for the Microsoft Visual Studio .NET environment and uses the IIS application server to execute Web services. WSRF.NET extends the Web services model offered by Microsoft .NET languages by providing classes, methods and attributes for the WSRF in the .Net development environment (Wasson and Humphrey, 2005). The WSRF.NET implementation uses the Microsoft SQL Server database server to store data related resources, for example, variables, structures and classes which are serializable and converted into XML then stored in the database. To allow WSRF resources to be used in a Microsoft .Net Web service environment, several attributes are used to identify resources and methods which are used to create the stateful resources.

Two resources are declared for the auction service, the item description and the current bid variable which holds the value for the current highest bid. The resources are attributed with the [Resource] attribute to signify that it is a WSRF resource as specified by the WSRF.NET implementation (Wasson and Humphrey, 2005). The auction service implements several methods which are required by the language and the WSRF.NET implementation. These methods form the constructor for the auction service as well as some initialization methods. The auction service contains several methods to create an auction and check the bid on current auctions. The most critical method which is publicly accessible is the bid method:

```
public bool bid(int clientBid)
```

The bid method is responsible for returning true or false based on the clients bid. Within the bid method however, the auction service must access the WSRF resource. This is done by using the get {} and set {} attributes to retrieve and store the values of the resource to and from stateful storage. The interaction with the underlying database is transparent and handled by the WSRF.NET implementation.

To allow the client application to exploit the WSRF.NET auction service, it requires the ability to get resource properties from the auction service, create end point references and access the auction service’s bid method via SOAP. The WSRF.NET implementation allows the majority of this functionality to remain hidden from the developer. The client application is written in C# using the Microsoft Compact Framework, which includes libraries and code to develop applications for mobile devices. The client application implements a function, itembid() which is responsible for connecting to the Web service, placing a bid on behalf of the client and printing the result of the bid. The itembid() function is responsible for connecting to the auction service, calling the auction service’s bid function and returning the result of the bid.

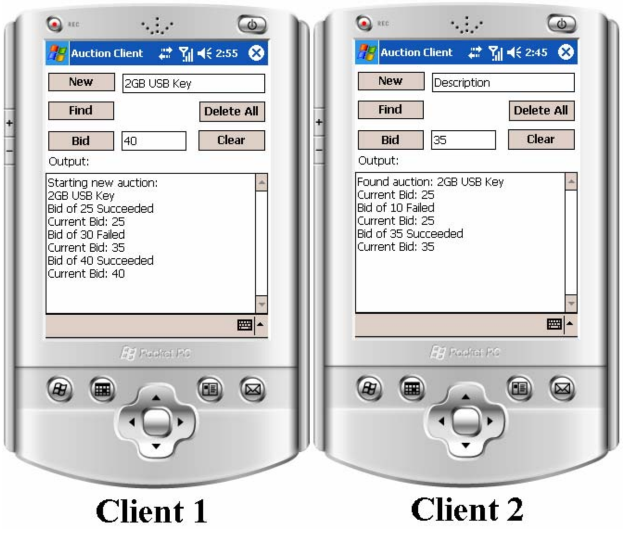
The development of the auction service provides a service with which to test the SMB. To do this, the auction service must also include a SMB service proxy and the auction service client application must include a SMB client proxy. These are simply included with the respective applications during the compilation of each application. The auction service is used in the experimental study of the SMB in the following sections.

6 Testing the Auction Service

The experimental study to test and provide a proof of concept of the SMB using an auction service is done in several steps. Firstly, the auction service is tested to ensure that the service operates as required. The auction service maintains state through the use of the Web Services Resource Framework. This test is then used to show the logical operation of the SMB and how the service will react when a failure in the system occurs, where the auction service becomes unavailable. The prototype of the SMB is then used to show the restoration of the auction service.

To test the implementation of the auction service several auctions were set up for client bids. The testing environment on the service provider’s side consisted of one PC, hosting the SMB and the auction service. This represents a service provider. The testing environment also includes two PDAs as the clients. This test involved two client applications on the two PDAs accessing the auction service in succession.

As can be seen in Figure 8, the two clients are run on two Microsoft Compact Framework Emulators (Emulators were used in the figures purely to show the results clearly; these tests were in fact carried out on PDAs). Client 1 initially starts a new auction and performs a bid of 25. The client application invokes the auction service with the itembid() which passes through the client proxy to the auction service. The auction service then compares this with the current bid (which is initialised to 0) and returns the boolean value of “true” as the bid was successful. Client 2 then attempts to firstly bid 10, which fails as client 1 has previously bid 25, and subsequently bids 35, which is successful. Client 1 then attempts to bid with a value of 30, which is unsuccessful due to Client 2's bid, and therefore bids with a value of 40. Each of these bids is an individual invocation of the auction service, showing the ability for the auction service to maintain the state of the current bid across interactions with different clients.



2GB USB Key
40

```
Starting new auction:
2GB USB Key
Bid of 25 Succeeded
Current Bid: 25
Bid of 30 Failed
Current Bid: 35
Bid of 40 Succeeded
Current Bid: 40
```

Description
35

```
Found auction: 2GB USB Key
Current Bid: 25
Bid of 10 Failed
Current Bid: 25
Bid of 35 Succeeded
Current Bid: 35
```

Figure 8. Testing the Auction Service

If there is no service responsible for managing the auction service, such as a broker, there is no guarantee that the auction service will process the client's bid, for example, if a failure occurs and the auction service becomes unavailable. This means that if a client bids on an item and the auction service becomes unavailable, the client may not be able to successfully complete the bid, even though it is in fact a valid bid. The introduction of an SMB in the following section is able to provide this support to the auction service.

7 Experimental Study of the SMB

By introducing the SMB into the SOG environment, it is possible to provide autonomic principles to the auction service transparently. To test the SMB, a failure of the auction service is simulated.

7.1 Failure of the Auction Service

The auction service must first be deployed on a suitable host by utilising the service deployment manager. An SMB service proxy must exist on this host, and a client proxy must be utilised by the mobile client. The client proxy is responsible for discovering an SMB managed auction service and subscribing to the service through the SMB interface.

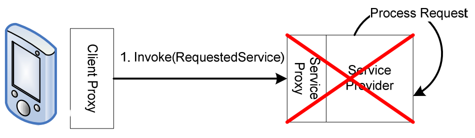


Figure 9. A Web Service Fails

We depict a typical failure, seen in Figure 9. The mobile client invokes a service. The service begins processing the request and fails. The service is now no longer available. The SMB is then able to step in and restore this service, however must first determine that a failure has occurred. The client's proxy is responsible for requesting the service, alerting the SMB that the service has failed. This is shown in Figure 10 and takes place as follows.

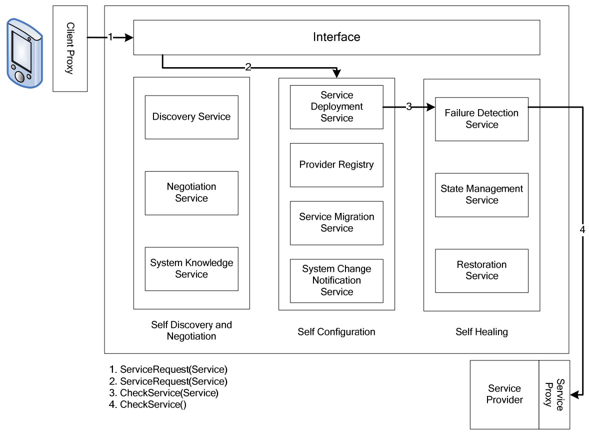


Figure 10. SMB Detection of a Failure

The request for the service is passed through the client's proxy to the SMB (Message 1). The SMB passes the request to the service deployment service (Message 2) where a request is made to the failure detection service to check the availability of the service (Message 3). The failure detection service requests a check on the auction service (Message 4) but the failed service does not respond and a timeout occurs. The SMB is now aware of the failed service and is able restore the service.

7.2 Restoring the Failed Auction Service

Once the SMB is aware of the failure, the broker must decide where to restore the service. If the original host is still available the service may be restored on the same host, or select a different host may be chosen which meets the service's requirements. Deciding on the most appropriate host to restore the service is the decision of the restoration service and subsequently the service deployment service. Restoring a service is seen in Figure 11.

The failure detection service invokes the ReportFailure method on the restoration service to reconstruct the failed service (Message 1). The restoration service checks whether the state of the failed service has been recorded with the state management service (Message 2). If the service proxy associated with the failed service had updated the current state of the service with the broker before failure, the state exists and is returned to the restoration service (Message 3). The restoration service then is able to invoke the ReconstructServiceRequest on the service deployment service. The name of the failed service, and the state of the failed service, if it exists, are supplied with this invocation (Message 4). To determine where the failed service is to be restored, the service deployment service invokes FindSuitableHost on the provider registry (Message 5) to select a suitable host for deployment and returns the suitable host to the service deployment service (Message 6).

The service is then deployed on the selected host by invoking the DeployService method of the SMB service proxy on the selected host (Message 7). If successful an "Ok" message is returned from host's SMB proxy (Message 8). Once the deployment of the restored service is successful, the SMB returns the new address of the service (Message 9) and the service is restored.

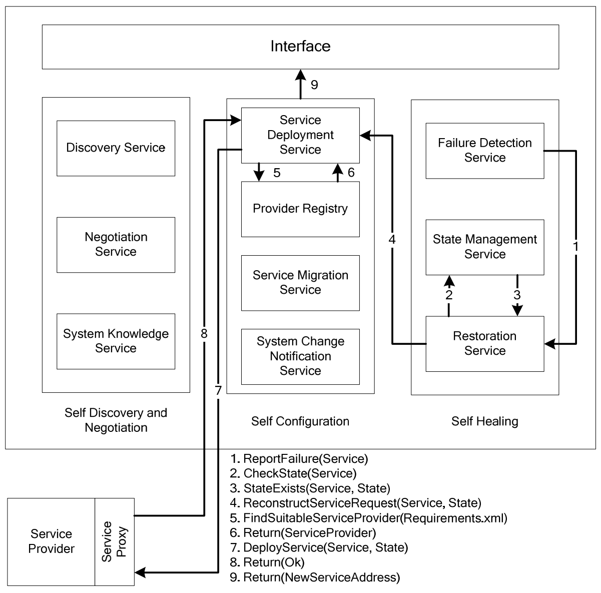
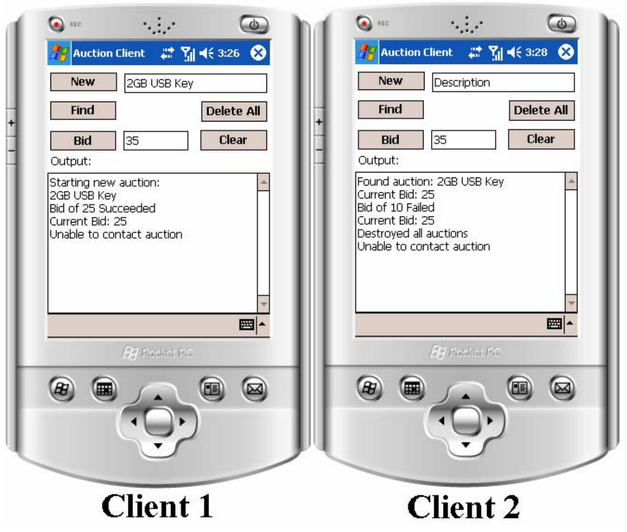

Interface
9
Discovery Service
Negotiation Service
System Knowledge Service
Self Discovery and Negotiation
Service Deployment Service
5
6
Provider Registry
Service Migration Service
System Change Notification Service
Self Configuration
8
7
4
Failure Detection Service
State Management Service
2
3
Restoration Service
1
Self Healing
Service Provider
Service Proxy
1. ReportFailure(Service)
2. CheckState(Service)
3. StateExists(Service, State)
4. ReconstructServiceRequest(Service, State)
5. FindSuitableServiceProvider(Requirements.xml)
6. Return(ServiceProvider)
7. DeployService(Service, State)
8. Return(Ok)
9. Return(NewServiceAddress)

Figure 11. Restoration of an SMB Service

7.3 Testing the SMB

The prototype SMB and auction service are once again deployed on the relevant hosts. The tests are performed with two clients who are bidding on an item at an auction initiated by one of the clients. The bidding on the item is conducted in a manner similar to that shown in the previous section; however a failure is simulated during the auction bid. Two tests are run, in the first test no SMB support is provided, while the second test utilises the SMB to provide autonomic support for the auction service. These tests are shown in Figure 12 and Figure 13 respectively.



2GB USB Key
35
Output:

```
Starting new auction:
2GB USB Key
Bid of 25 Succeeded
Current Bid: 25
Unable to contact auction
```

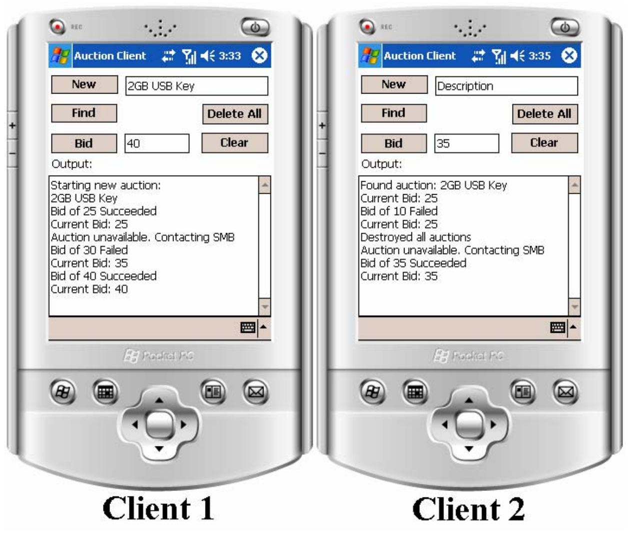
Description
35
Output:

```
Found auction: 2GB USB Key
Current Bid: 25
Bid of 10 Failed
Current Bid: 25
Destroyed all auctions
Unable to contact auction
```

Figure 12. Testing the auction service without SMB support

The first test shows the initialisation of the auction and several bids on the item. During the bidding by Client 2, the auction service and WSRF resources are destroyed. Client 2 attempts to conduct a bid on the item after the failure occurs, however is unable to complete the request due to the destruction of the service. Subsequently, Client 1 also attempts to bid on the item; however the invocation of the auction service once again fails.

The second test utilises the prototype implementation of the SMB. An SMB service proxy is implemented and deployed with the auction service and similarly an SMB client proxy is attached to the each client. During the initialisation of the auction service, the prototype SMB is notified about the existence of the auction service. Initially, Client 1 begins a new auction and once the request is complete, Client 1 bids on the item. The service proxy informs the SMB that a change in the service's state has occurred by invoking the SaveState() method and supplying the values of the auction description and the current bid. The SMB stores the state of the service in stable storage. Client 2 performs several bid attempts before the auction service is destroyed.



2GB USB Key
40
Output:

```
Starting new auction:
2GB USB Key
Bid of 25 Succeeded
Current Bid: 25
Auction unavailable. Contacting SMB
Bid of 30 Failed
Current Bid: 35
Bid of 40 Succeeded
Current Bid: 40
```

Description
35
Output:

```
Found auction: 2GB USB Key
Current Bid: 25
Bid of 10 Failed
Current Bid: 25
Destroyed all auctions
Auction unavailable. Contacting SMB
Bid of 35 Succeeded
Current Bid: 35
```

Figure 13. Testing the auction service with SMB support

Once the auction service is destroyed, the client attempts to invoke the bid method of the service. The SMB client proxy realises that the auction service has failed due to the unsuccessful invocation of the service and displays some debugging information informing the user of the failure. This information is displayed for the purpose of this report and is not necessary in a production environment so that full transparency is maintained. The SMB client proxy invokes the ServiceRequest() method of the SMB. The SMB verifies that the service has failed and invokes the ReconstructService() method where the state of the auction service is retrieved from stable storage and used to reconstruct the service. The new address of the service is provided to the SMB client proxy. Client 2 is then able to continue bidding on the auction item. Subsequently Client 1 also detects the failure of the service and is informed by the SMB of the service's new location. Client 1 is then able to bid on the auction item.

The tests performed on the SMB show that the availability and reliability of the auction is able to be improved by deploying the service in an environment which provides service management. The SMB is able to identify a failed service and provide mechanisms to support the restoration of the service with only a short delay during bidding.

8 Conclusion

We have shown in this paper a need for transparent autonomic management in service oriented grids. The development of an auction service has shown through a simulated failure that utilising Web services and standards such as WSRF is not enough to guarantee a reliable grid environment. However this problem can be solved by introducing an autonomic broker.

We have demonstrated the design and implementation of a System Management Broker which is capable of transparently providing self discovery and negotiation, self configuration and self healing in a service oriented grid environment. By utilising the SMB, the auction service is able to be transparently restored when a failure occurs improving the reliability and availability of the service. The auction service was used as a simple test of the broker and only highlights the initial benefits of the SMB. There are many other applications, where the SMB will benefit service oriented grids, for example compute services, banking services, medical diagnosis services and hospital services; however we have shown that the provision of the autonomic broker ultimately supports clients and service providers.

Future work on the SMB includes deploying the broker in a production grid environment and utilising the broker to manage many other services. This will show the effectiveness of the broker.

9 References

Czajkowski, K., et al., The WS-Resource Framework Version 1.0, 2004.

Horn, P., Autonomic Computing: IBM's Perspective on the State of Information Technology, 2001.

Medeiros, R., et al., Faults in Grids: Why are they so bad and what can be done about it? in Proceedings of the Fourth International Workshop on Grid Computing, pp 18, 2003.

Messig, M., Grids and Globus and Where to Now? Technical Report, Deakin University, TR C4/11, 2004.

Messig, M, and A. Goscinski, Self Healing and Self Configuration in a WSRF Grid Environment, in Proceedings of the 6th International Conference on Algorithms and Architectures for Parallel Processing, ICA3PP, Springer-Verlag, LNCS 3719, pp. 149 – 158, 2005.

Wasson, G. and M. Humphrey. Exploiting WSRF and WSRF.NET for Remote Job Execution in Grid Environments, in Proceedings of the International Conference of Parallel and Distributed Systems, Denver, Colorado, 2005.

I. Foster, C. Kesselman, and S. Tuecke, 2001, 'The anatomy of the Grid: enabling scalable virtual organizations. International Journal of Supercomputer Applications', 15(3):200—222.

Grangenet, http://www.grangenet.net/, last accessed June, 2006.

IBM, 2006, 'Grid Solutions by Industry', IBM Corporation, http://www-1.ibm.com/grid/solutions/by_industry/index.shtml, last accessed June 2006.

IBM 2005, 'Web Services Trust Language (WS-Trust) draft specification', http://www-128.ibm.com/developerworks/library/specification/ws-trust/, Feb 2005, last accessed May 2006.

Messig, M and A. Goscinski, 2006 'Self-Discovery, Self-Configuration and Self-Healing in Service Oriented Grid Environments', submitted to Future Generation Computer Systems, The International Journal of Grid Computing, 2006.

Gudgin, M., et al. (2003), 'SOAP Version 1.2 Part 1: Messaging Framework', http://www.w3.org/TR/soap12-part1/, last accessed June 2006.

Weiser, M. "The Computer for the Twenty-First Century", Scientific American, September, 1991

Snoeren, A. C., Balakrishnan, H., and Kaashoek, M. F., "Reconsidering Internet Mobility". 8th Workshop on Hot Topics in Operating Systems, 2001.

Migliardi, M., et al., "Mobile Interfaces to Computational, Data, and Service Grid Systems". Mobile Computing and Communications Review, Vol 6, No 4, October 2004

Phan, T., et al., "Challeng: Integrating Mobile Wireless Devices Into the Computational Grid". Proc. of the 8th ACM Intl. Conf. on Mobile Computing and Networking (MobiCom), Sept. 25-27, 2002.

Yang, X., and Bouguettaya, A., "Efficient Access to Wireless Web Services". Proceedings of the 7th International Conference on Mobile Data Management (MDM'06), 2006.

Grabowski, P., et al., "Context Sensitive Mobile Access to Grid Environments and VO Workspaces". Proceedings of the 7th International Conference on Mobile Data Management (MDM'06), 2006.

Baumung, P., Penz, S., Klein, M., "P2P-Based Semantic Service Management in Mobile Ad-hoc Networks". Proceedings of the 7th International Conference on Mobile Data Management (MDM'06), 2006.

Ganzalez-Castano, F., et al., "Condor Grid Computing from Mobile Handheld Devices"., Mobile Computing Review, Vol 6 No 2, 2002.

Tian, M., "QoS Integration in Web services with the WS-QoS framework", PhD thesis, Freien Universitat, Berlin, 2005.