Improving the Quality of Information-Flow with the Smart Data Server[*]

Uwe Roth	Thomas Engel	Christoph Meinel
Institute of Telematics	Institute of Telematics	Institute of Telematics
Bahnhofstraße 30-32	Bahnhofstraße 30-32	Bahnhofstraße 30-32
D-54292 Trier, Germany	D-54292 Trier, Germany	D-54292 Trier, Germany

Abstract *Nowadays, we face the continually growing importance of information: it is quickly becoming our most valuable resource. The Smart Data Server improves the quality of information by combining different information-resource-pools without unnecessarily burdening the user with details. By questioning the Smart Data Server with function-requests of installed function-modules, high-level information is created. The flow of information can be improved by routing requests through networks of Smart Data Servers, each server responsible for different kinds of requests.*

Keywords: Information Retrieval; Middle-Tier Architecture; Distributed Server; Java Server; World Wide Web; Database

1 Introduction

Nowadays, we have to face the continually growing importance of information: it is turning into our most valuable resource. It is not only the question whether information is available at all which poses a problem but also where the information is stored. On the other hand, sometimes only combining data stored in different databases from different vendors can improve this data to become usable information. The access to different databases using standard HTML-browsers via CGI-scripts or servlets has become a common way, but minor changes to the database-topology may result in reprogramming these scripts. The need of an integrating system is obvious.

The aim of this paper is to present such an integrating system named the *Smart Data Server* (SDS) [8, 9]. One major feature of this system is the possibility of building up networks of SDS's to scale the information-flow, another one is the implementation of functions in independent modules which can be added and configured easily.

2 The SDS-Network Topology

Figure 1 shows various different possibilities of building up an SDS-network topology. In general, there are two different ways to access the SDS. The direct way uses a simple protocol in an XML-related language to define the function-request. The protocol and language are not dependent on any programming-language but are currently implemented in Java only.

The second way is to use an ordinary HTML-form with a specialized servlet to translate the requests and results from and to HTML. This is useful if there is no intention of using Java or applets or if the access of the SDS is impossible because of firewalls which do not allow accessing SDS-TCP/IP-ports. Additionally, the access of the SDS via servlets has many other advantages. The user-interface is standardized and well-known to the ordinary user. The creation of the page-design can be separated from the underlying connectivity by using servlets over server-side-includes.

Every SDS has its own database to store user-information or to provide storage for data that has its origin in text-based documents or in databases that are only temporarily available. In this case, the use of specialized applications is recommended to transfer this data from the text-documents or temporary databases to the SDS.

[*] As in Proceedings of the 1st International Conference on Internet Computing (IC'2000), Las Vegas, Nevada, USA, 2000

An example: Data that is available only through the Internet at different web-sites. Solely the specialized application has to be changed if the web-site's design changes [3].

To the SDS, accessing the local database which is added to the SDS as a storage volume for user-information makes no difference to remotely accessing databases from different vendors in the enterprise. Access to these databases is hidden from the internal modules. The only restriction is the acceptance of standard ANSI-SQL statements. The applied database-driver and all other connection-specific information is stored in a configuration-file and therefore easily adaptable to the ever-changing environment inside an enterprise.

platform [14] are used, everything is written in pure Java 1.1.x-code. This will be changed in order to improve the server's performance using unsynchronized container-classes instead of hashtables or vector data-structure.

Through its tree layers, the internal architecture of the SDS is quite strict. The tree layers are the Session-Layer, the Service-Layer and the Function-Layer (figure 2).

The Session-Layer is responsible for handling client requests, checking authorization for access to the server and creating time-based requests. It contains all basic functionality concerning request-handling, such as network-connections, session-handling and protocol-analysis.

The Service-Layer contains different services which can be used by the Session-Layer and the Function-Layer. There are two important modules within the Service-Layer that should be emphasized :the first one is the Datastore module. It hides all database-specific information from the modules and translates database-access-methods to standard-SQL, so any desired exchange of the underlying database is possible. Information about which database is used, which driver is necessary to access the database and about the location of the database is stored in a configuration-file so that it can be changed without recompiling the server.

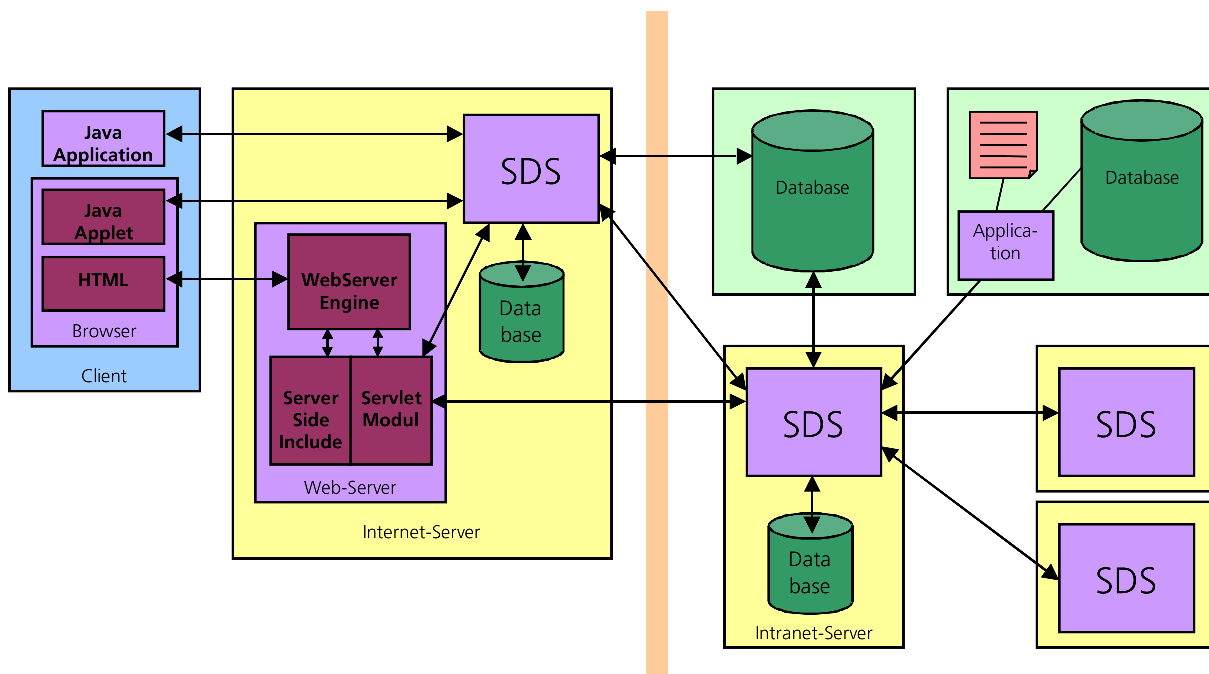

Figure 1: SDS-Network

If a firewall is involved, the use of two SDS's is reasonable: one SDS can be settled on each side of the firewall. Through encrypted requests, the Internet-SDS only acts as a gateway to the Intranet-SDS.

3 The Internal Structure of the SDS

The SDS is a server written in 100% pure Java [4, 10]. This allows changes to the underlying platform. Currently, no features of the Java2

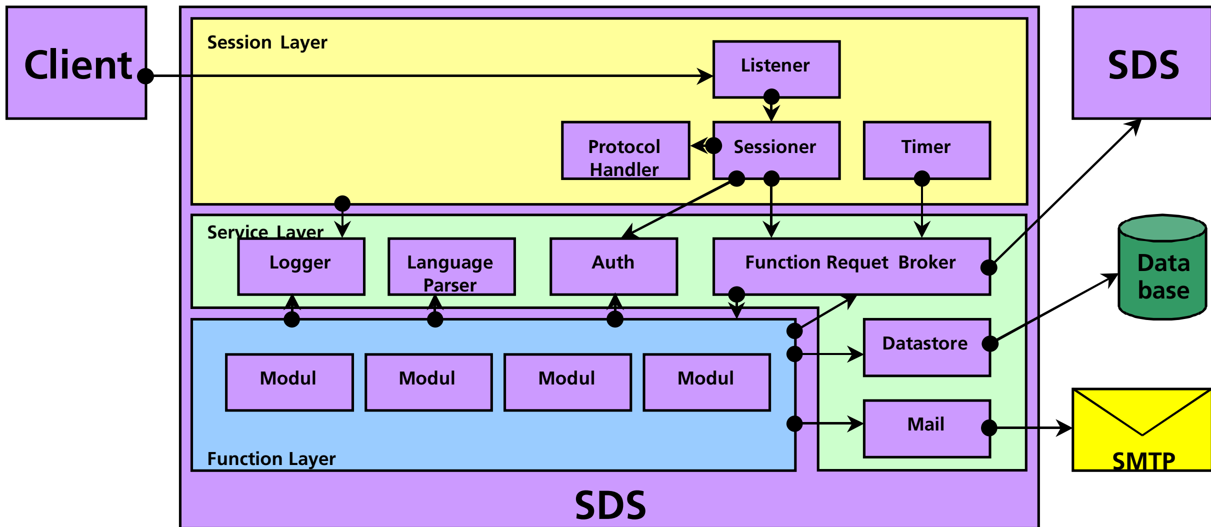

Figure 2: Internal Structure of the SDS

The second important module is the Function Request Broker. This module is responsible for directing the user-requests to the appropriate modules. To build up networks of SDS's there are three ways to require the Function Request Broker to provide a function (figure 3): if a module is configured as an internal module, the local classes' methods are called; if a module is configured as an external module, the Function Request Broker routes the request to a predefined second SDS. The request itself is not parsed at this stage of processing and may be encrypted and unreadable for the routing SDS. Regarding external calls the first SDS acts like a client for the second SDS.

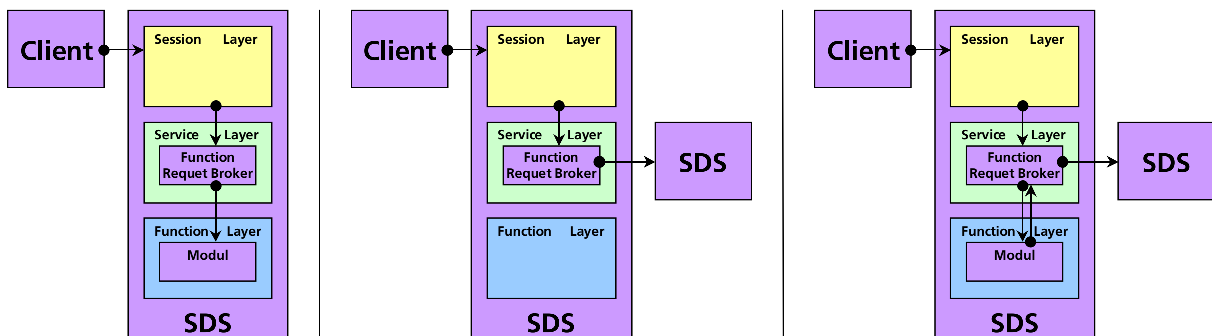

Figure 3a: Internal Call   Figure 3b: External Call   Figure 3c: Internal/External Call

A third way to access a module and its function is the combination of both methods. First, a function and its module are called internally. The function accesses databases or does different calculations and may recognize that it is not able to fulfill the request by itself. In this case, the function builds up a module-request and directs it towards the Function Request Broker. If the name of the needed module is identical with the name of the calling module, the Function Request Broker notices this and then treats the request like an external call.

The definition whether a module is internal, external or both is static at that specific moment and thus has to be made in the server's configuration-file.

Not only requests from clients and other SDS's may be handled by the Function Request Broker: the Timer Module is able to produce requests within the Session Layer which are triggered by a timer. These requests can be compared with cron-jobs on Unix-systems. The

great difference to client-requests is the fact that a response is neither generated nor expected., only the function's side-effects are relevant. The regular clean-up of a database every night or the replication of a database may serve as an example for a timer-based request.

4 Conclusion: How the SDS Improves the Information-Flow

The last two sections have thoroughly discussed the SDS's internal structure and the possibilities of building up networks of SDS's. There are numerous reasons why this concept improves the information-flow. First of all, it is not necessary for a client to know which SDS is handling a request or whether it is routed through a large network of SDS's. The client is only connected to one SDS and this SDS returns all results.

It is possible for different SDS's to be involved, depending on the databases that are associated with each SDS. The deployment of specialized SDS's is possible, where each SDS is responsible for different kinds of databases equipped with different security levels. For security reasons, some SDS's may by definition only be allowed to access an internal database. Only few users may be allowed to access these SDS Therefore, the same request may can result in different responses depending on the rights of the requesting user.

While combining data of different database-resources, the SDS can create additional information that would not be available if the databases could be accessed only separately. It is not obvious - from the client's point of view - which data packets have their origin in databases and which have been calculated and combined as a direct answer to the request. For performance-reasons, some data may have been pre-calculated at night to improve the server's speed . A prediction module has been developed (but not added to the framework yet) to figure out which request might soon be asked and which efforts have to be made to pre-calculate the data. A cost-benefit calculation is performed to ensure the efficiency of such a prediction [5, 6, 7]

During the lifetime of the SDS, the databases can be changed or replaced. This information is entirely hidden from all clients. A function-module itself has no "idea" of databases or SQL-statements, so the replacement or change is even hidden from the modules.

If data is inserted to the SDS by specialized applications that are able to transfer data from different locations to the SDS (e.g. from Internet-sites, local text-documents, temporarily available databases), only those specialized applications may have to be replaced because of a database-source change. The SDS itself remains unchanged.

5 Related Work

The need for an integrating platform has been proclaimed by software-developers for a long time. The time it takes to build applications that fulfill the requirements within an heterogeneous enterprise has to be reduced. It has been recognized that only some aspects of developing such an application focus on the main solution of the problem, most time is used to place the application inside the enterprise – this resembles reinventing the wheel anew each time.

A related concept with the aim of fulfilling the need for an integrating platform is the Enterprise Java Beans Platform [12, 13]. It covers many aspects that are useful in an enterprise's environment. One of the SDS's major advantages is its functional approach, which may seem a little obscure at the beginning but which has the advantage of keeping the overhead of requesting very small. A similar argumentation applies for CORBA [15, 16, 17] and DCOM [1]. In fact, the call to a distributed object is very different from the call to a distributed function: it is not necessary to keep track of problematic tasks such as garbage-collection of objects or dead objects. RMI [11], on the other hand, is a Java-specific solution, but the client's programming-language should not necessarily have to be Java. The protocol to access the SDS is not dependent on any programming language.

6 References

[1] Brown, Nat; *Distributed Component Object Model Protocol -- DCOM/1.0*, Microsoft Corporation; http://msdn.microsoft.com/library/specs/distributedcomponentobjectmodelprotocoldcom10.htm; 1997

[2] Dickmann, A.; *Two-Tier Versus Three-Tier*

Apps. InformationWeek 533, 13/95, 74-80

[3] Duan, Nick N. *Distributed database access in a corporate environment using Java;* Computer Network and ISDN Systems 28 (1996), 1149-1156

[4] Farley, Jim. *JAVA - Distributed Computing*, O'Reilly & Associates; 1998

[5] Haffner, Roth, Engel, Meinel; *A Semi-Random Prediction Scenario for User Requests*; As in Proceedings of the Asia Pacific International Web Conference, APWEB`99, Hong Kong, 9/1999.

[6] Haffner, Roth, Engel, Meinel; *Modeling Time and Document Aging for Request Prediction - One Step further*, As in Proceedings of the Symposium on Applied Computing, ACM SAC2000, Como, Italy, 2000

[7] Haffner, Roth, Engel, Meinel; *Optimizing Requests for the Smart Data Server*; As in Proceedings of the Conference on Applied Informatics, IASTED AI2000, Innsbruck, Austria, 2000

[8] Roth, Haffner, Engel, Meinel; *The Smart Data Server: A New Kind of Middle-Tier*; As in Proceedings of the Conference on Internet and Multimedia Systems and Applications , IASTED IMSA`99, Nassau, Bahamas, 10/1999

[9] Roth, Haffner, Engel, Meinel; *An Approach to Distributed Functionality - The Smart Data Server*; As in Proceedings of the World Conference on the WWW and Internet, AACE WebNet`99, Honolulu, Hawaii, 10/1999

[10] Sridharan, Prashant; *Advanced JAVA networking*, Prentice Hall; 1997

[11] Sun Microsystems; *Java Remote Method Invocation Specification*, Revision 1.50, JDK 1.2, Sun Microsystems; ftp://ftp.javasoft.com/docs/jdk1.2/rmi-spec-JDK1.2.pdf; 1998

[12] Sun Microsystems (1999); *Enterprise Java Beans Developers guide*; 1999

[13] Sun Microsystems (1999); *Enterprise Java Beans Specification*, v1.1; 1999

[14] Sun Microsystems*; Java 2 Platform*; http://java.sun.com/jdk

[15] OMG; *The OMG's site for CORBA and UML Success Stories*; http://www.corba.org

[16] OMG; *Object Management Group Home Page*; http://www.omg.org

[17] OMG; *The Common Object Request Broker Architecture and Specification*; http://www.infosys.tuwien.ac.at/Research/Corba/OMG/cover.htm