Master's Thesis in Computer Science

System Design of an Intellectual Capital Management Platform Using Enterprise Java Technology vs PL/SQL

Rickard Sandström

The Royal Institute Of Technology
Kungliga Tekniska Högskolan

Examiner and supervisor at KTH	Vladimir Vlassov, Department of Microelectronics and Information Technology
Supervisor at Prohunt	Susanne Lundberg, Prohunt AB

This thesis report is the result of the course *2G1015 Master project in Teleinformatics*. This course concludes the education of academic degree
Master of Science in Computer Engineering (M.Sc.)
at the Royal Institute of Technology
(Kungliga Tekniska Högskolan) in Stockholm, Sweden.

The report and additional material is available at:
`http://www.nada.kth.se/~d96-rsa/kurser/exjobb/`

Mars 29th, 2003, Stockholm, Sweden.

Abstract

The objective of this thesis was to evaluate the distributed architecture known as J2EE (Java 2 Enterprise Edition) with emphasis on performance, scalability, flexibility and reliability on behalf of the company Prohunt. J2EE was then compared with Prohunt's existing server platform, based on Oracle's PL/SQL stored procedure language. Since Prohunt had already ported the clients from Windows applications to web based clients with Java servlets, the question was whether to also move the code on the server side of the products (known as the Intellectual Capital Management platform), thus completing the change of technical architecture.

To do this comparison, two prototypes were developed, one using PL/SQL procedures and one using JavaBeans and Borland Application Server. Several performance and scalability experiments were conducted with both prototypes and the results were then compared. Advantages and drawbacks with both architectures are discussed and considered before reaching a conclusion about which approach is the better one.

It is my conclusion that both architectures have their advantages and drawbacks, and both have different preferred areas of usage. Oracle PL/SQL is faster and less complicated when considering large database queries. Ideal applications are systems for data mining and decision support. On the other hand, if business logic is complex and less data is required to be moved, then J2EE should be considered. J2EE is slower at fetching data and requires more resources and servers, but with the right application and a great deal of thinking when creating the entity beans, it might be the right choice.

Preface

This report is the result of my thesis work at a company called Prohunt AB. The work on this thesis was made more difficult by the fact that Prohunt was declared bankrupt in mid June 2001, three months into my thesis work. This had several negative consequences: firstly, I lost virtually all supervision and help from the company. Secondly there was also a lot of confusion about the future of the company and the employees, which took a lot of time and concentration. For about a month I was all alone in the office. In the beginning of August it was decided that WM-Data would buy most of Prohunt and they came and took the equipment including the server upon which the Oracle database I was using was situated, before I had a chance to finish my evaluations and experiments.

The fact that Prohunt wasn't there for me anymore had some consequences on the subject of the thesis too. Since the planned change of technical platform for Prohunt's software product could not be realized anymore, some of the questions of the thesis (guidelines for how to move the code from PL/SQL packages to Enterprise JavaBeans for example) where not interesting anymore. Therefore some additional topics where added to the thesis by me and my supervisor Vladimir Vlassov. These more general, theoretical topics concentrated on the Application Server and its benefits.

Table Of Contents

List of Tables

List of Figures

1. Introduction

The company Prohunt has three different software products in the ICM (Intellectual Capital Management) segment: ProCompetence, ProResource and ProCareer.

The products are currently based on an architecture where the database contains both data, business logic and some of the form. This is not an optimal configuration, and using a different architecture several improvements could probably be made. Therefore, Prohunt started to investigate alternative business platforms in October 2000.

This investigation identified a number of possible improvements and resulted in a recommendation for a new architecture based on Enterprise Java Beans (EJB) and XML. The new architecture would separate data, function and form and introduce a number of other improvements.

Since Prohunt's ICM products are large, complex systems, porting the whole systems would be a very tedious task. It is believed that some parts of the products would benefit more than others from using Java instead of the rather old language PL/SQL that is currently used.

Prohunt also wants the three products to use the same system for authentication and authorization. Today these products are not integrated. As a result a user has to log into every system separately. Naturally Prohunt wants their customers to buy all three products and this integration would greatly increase the user friendliness and customer value of the combined systems.

Other benefits of the new architecture would be easier maintenance and development of the products as a result of the more modularized and multi-layered architecture.

The recommendation of the investigation was to start the porting process by constructing a new authorization system common to all three products. When completed, the next step would be to port the parts of the products that would benefit the most from it.

On Friday the 23rd of March 2001, a new project was started with the goal of designing this new authorization system, based on the proposed architecture. This project will be ongoing until the 1st of June and the thesis is supposed to work in parallel to this thesis project, exchanging information and ideas.

1.0.1 Requirements on the Reader

A reader of this thesis will have to have an intermediate knowledge of Java specifically and programming in general. He or she should also be familiar with the concepts of SQL, since it is extensively used in the thesis, and a basic understanding of distributed computing is preferred but not necessary. The thesis will not require any prior knowledge of J2EE, application servers and such, but a basic knowledge about computer science is preferred as there are some vocabulary and expressions that are presumed to be known.

1.1 Motivation

The server-side of Prohunt's existing products is for the most part implemented in Oracle PL/SQL. This is a stored procedure language that is tightly integrated into the Oracle database server and architecture, making it impossible to use database managers (DBMS) other than Oracle's. Some of Prohunt's customers use other databases and therefore Prohunt would like to be able to make their systems independent of the DBMS.

Also, some parts of the Oracle PL/SQL architecture are believed to be slow. It is assumed that Java would be more efficient in raw calculations for example. If so, maybe it would be better to port the whole systems to a Java platform instead of the Oracle dependent platform used today?

Although there are plenty of books and documentation about the J2EE platform and some about PL/SQL, no comparisons and no scientific performance benchmarks between these two architectures has been performed. To conduct a comparison and evaluation will be the biggest challenge of this thesis.

1.2 Objective of Thesis

The goal of the thesis is to evaluate the distributed architecture known as J2EE (Java 2 Enterprise Edition) with emphasis on performance, scalability, flexibility and reliability. The evaluation will include a comparison with Prohunt's existing architecture, based on Oracle PL/SQL. Prohunt's ICM platform will serve as a case study in this sense.

The purpose of the evaluation is to provide recommendations whether J2EE would be a suitable technical platform for Prohunt's ICM products or not. To answers this, we need to know if the Java Enterprise Technology is fast enough, if it scales well enough and if moving all code from PL/SQL to Java would be worth it, considering economy, time consumption and education of developers?

Another objective of the thesis was to provide guidelines on how to migrate the systems if such a migration was recommended, and find out which parts of the systems would benefit the most from this change. This objective was abandoned after Prohunt had gone bankrupt though, changing some of the perspective of the thesis to a more theoretical view.

1.3 Where the Thesis Was Done

This Master's Thesis was done at a company called Prohunt AB. Prohunt AB call themselves "the number one provider of complete solutions for development and management of the intellectual capital of organizations" [1].

Prohunt AB was founded ten years ago as Palmér System AB. This was in Linköping in 1992 and the company consisted of only three persons. In the first half decade it was just another IT consultant company but somewhere along the line a new direction was taken. The company changed name to Prohunt AB and began working in the field of competence management. A few small companies were acquired: New Start AB (competence managers), Unit Solution AB (java developers who had a selling product – ProTime) and Comenius (e-learning). Competence management was first practiced the old fashioned way with pen and paper, but since Prohunt was a firm with much knowledge in IT, they saw the need and had the competence to bring their methods to the computer age. Prohunt's Intellectual Capital Management (ICM) platform started to take form. ICM will be described below.

1.3.1 Intellectual Capital Management (ICM)

According to the Gartner Group [48] (one of USA's leading companies in business research, analysis and advisory) Intellectual Capital (or Knowledge Capital) is defined as the "*Intangible assets of an enterprise that are required to achieve business goals, including knowledge of employees; data and information about processes, experts, products, customers and competitors*" [46].

ICM is in other words the management of a company's intellectual resources.

Prohunt's ICM platform consists of three software products: ProCompetence, ProCareer and ProResource. These products are used by management, employees and human resource managers in organizations to [2]:

- increase the company's ability to attract, develop and hold on to co-workers.
- prolong the time of employment by finding individual ways of making a career within the company.
- increase coverage (share of consultants currently assigned to work) by better usage of the competences within the organization.
- gain access to the right competence in the right place at the right time.
- faster guarantee the right competence that achieves the business goals.
- gain overview of the organization's resources and demands.
- help co-workers match their competence development and goals with the company's strategic needs and future goals.

Prohunt doesn't just sell software products. They are working in accordance to a unified competence process where the customer (company) is profiled by competence consultants and the employees are trained in using the software as well as proper human resource management. By mid 2000, Prohunt was the Nordic region's leading supplier of complete solutions for ICM in organizations with customers like Telia, Swedish Match, Teracom, Cell Network, AdTranz, Posten IT and Riksskatteverket [6].

In January 2001 Prohunt had about 120 co-workers and had offices in Stockholm, Gothenburg, Kalmar and Olso. When the Swedish market began to decline, Prohunt felt the change at once. In May the staff had been cut down to 60 persons and when venture capitalists IT Provider withdrew their financial support Prohunt was declared bankrupt on the 22nd of June 2001.

In the beginning of August WM-Data Human Resource announced that they would buy the remains of Prohunt; their ICM platform, equipment and employ some of the staff.

1.4 Structure of the report

The first chapter has already laid a foundation to the rest of the thesis by giving an introduction to the problem describing the problem, motivation and objective of the thesis.

The second chapter starts with a theoretical introduction to distributed applications and architectures in general. It also describes the applications in the Prohunt ICM platform, both general specification and analyses the architecture.

Chapter three gives an overview of Oracle's PL/SQL programming language and its history, semantics and architecture.

The fourth chapter provides an overview of Sun's Java 2 Enterprise platform. It is more extensive since the Java Enterprise platform consists of many Java technologies. The chapter ends with a more in-depth description of the J2EE Application Server and the services that it provides.

In chapter five my analysis begins by describing the general architecture of the systems and describes how the two prototypes are constructed. The chapter ends with a short presentation of the software used in the thesis.

Chapter six describes the evaluation experiments. The chapter presents the method of evaluation along with the evaluation prototypes. These experiments concentrate on two properties, performance and scalability. Experiment values are presented with both raw test numbers and different graphs and plots.

The seventh chapter called "Summary and Conclusions" contains essential information; evaluation results and conclusions based on the experiments and the general comparison between the two platforms.

The last chapter presents some suggestions about what additional work could be performed in the line of this thesis. This future work falls outside of this thesis for some reason or other.

At the end of the thesis are the references and the appendices, with a glossary, complete evaluation results and source code.

2. Background

This chapter starts with an introduction to distributed computing and distributed architectures. After this general introduction it presents the case study, the distributed applications of Prohunt's ICM platform. This presentation describes both functions and specifications of Prohunt's existing software products.

2.1 Distributed Computing

Distributed computing is a term used for systems where the process of computing has been distributed across more than one physical computer. The reason for this may vary. These are the major reasons for distributed computing [30]:

- The data are distributed.
- The computation is distributed.
- The users of the application are distributed.

Distributed data

The most common reason for distributed computing is of course that the data are distributed. The Internet is based on the fact that people all around the world want to access information on computers other than their own.

Distributed Computing

The computation may be distributed when one computer does not have enough computational power for the task at hand. The most famous example of this is the SETI@home project [39] where every one connected to the Internet can download a screen saver and donate processor time to the search for intelligent life in outer space.

Distributed users

A third reason for distributed computing is that the users of the application are distributed. A popular example of this is messaging applications like ICQ (I seek you) [41] and Microsoft's MSN Messenger Service [42], which allows users around the world to communicate with each other.

2.1.1 Requirements of Distributed Systems and Applications

Systems and applications that are distributed are exposed to a different set of requirements and expectations compared to ordinary applications. Some of the requirements are of a technological nature and some are due to human expectations and conditions.

These requirements are as follows:

- Response time
- Robustness
- Scalability

Response Time

The *response time* of a system is the elapsed time from the moment the user makes some sort of input until the system indicates a response. Of course the response time should be as short as possible but the requirement of the response time differs enormously from application to application. A car simulator has to have a response time measured in milliseconds, while a user can accept a response tome of at least four to five seconds before the he gets annoyed.

Robustness

A distributed application depends heavily on many factors to run well. Since the application is divided and situated on different machines, it is highly dependent on the computer network that connects the parts. This means that a distributed application has to be *robust*, i.e. not crash if the network is down or congested, or if the connection between client and server fails for some other reason. A distributed application has to be prepared for those types of failures.

Scalability

Scalability is the concept of an application continuing to perform well while the number of concurrent users or clients increases. The response time of a scalable application should not increase unreasonable fast when the number of online users increases.

2.1.2 Distributed Architectures

There are essentially four different architectures, or paradigms, for distributed systems. These are:

- Host-Terminal [33]
- Client/server [33]
- Multi-tier [8]
- Peer-to-peer [33]

Host-Terminal

This architecture is mainly used in mainframe environments. Several dumb workstations (called terminals) are connected to a single central computer (the host), see figure 2-1. The host is responsible for all processing; the terminals are only used for input and output and perform no processing what so ever. [32]

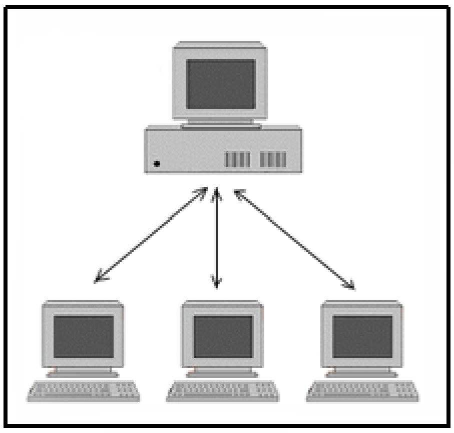


Figure 2-1. Several dumb terminals are connected to one host. All applications and data are stored and processed on the host, terminals are used exclusively for input and output.

Advantages

Terminals are very cheap since they mostly consist of a display and a keyboard.

Disadvantages

There are many disadvantages. In most cases nowadays you want to have some kind of processing on the terminal side. This architecture was very common before the breakthrough of the PC.

Client/Server

The main idea of the client/server architecture is that one or more clients request a service and the server provides this service. Servers are shared, central computers, which are dedicated to managing specific tasks for a number of clients. Clients are workstations on which users run programs and applications. Normally, clients connect to a server and request its services. The server responds to the clients according to the requests.

The characteristic of the client/server model is that both client and server is involved in the processing work. Clients rely on servers for resources, but process independently of the servers. The amount of processing work performed on the client can very, ranging from little (thin clients) to massive (fat clients). Each has of course its advantages and areas of usage. Figure 2-2 depicts a typical client/server configuration. [32]

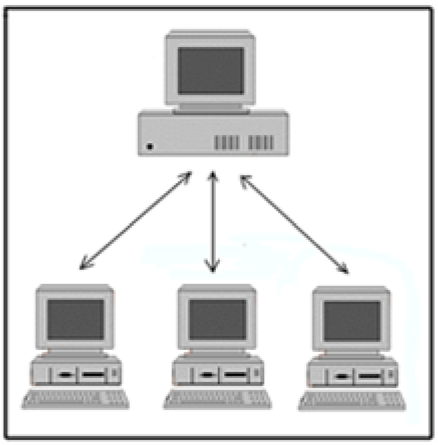


Figure 2-2. The clients connect to servers to access data or information, but are capable of functioning on their own too.

Advantages

The client has direct access to the server, which makes this a fast architecture. It is also very flexible, the client and server can be as "thin" or as "fat" as required for the specific task.

Disadvantages

If the server crashes, looses its connection or disappears of another reason, all services disappear too. This architecture is thus very dependent of the server.

Multi-tier

The multi-tiered architecture is a further development of the client/server model where data presentation, data processing and data storage has been divided into different layers or *tiers*. This division can be both logical (different tiers perform different tasks) and physical (different tiers can be situated on different machines).

A multi-tiered application can consist of a varying number of tiers, but the three basic tiers that most applications shares are the client tier, the business tier and the database tier. Figure 2-3 shows a multi-tiered architecture where the different tiers are also situated on different physical machines. Another quite common configuration is to put business tier and database tier on the same computer.

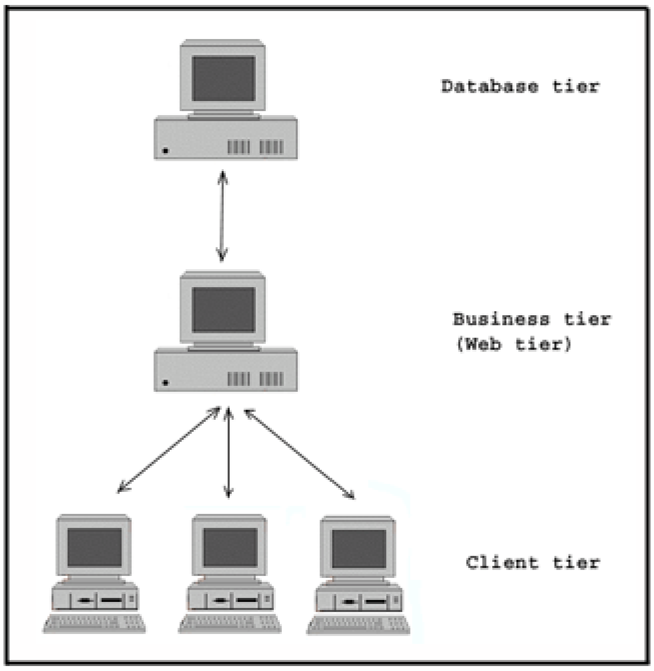


Figure 2-3. An example of a multi-tiered architecture where two machines acts as servers with different tasks. This will lead to increased network traffic but each machine will be able to process more concurrent clients.

Advantages

One of main advantages of the multi-tiered architecture is the opportunity to use third party application servers and middleware applications. These products provide easy-to-use APIs and middleware services and they are designed to make development of distributed applications easier, faster, more scalable and more fault-tolerant. The developer can focus on the business logic of the code and do not have to waste time and energy on the details behind transaction handling, security and message passing, which is taken care of by the middleware.

Another advantage is scalability. Since tiers can be distributed among several machines, computations can be divided among multiple machines or processors.

Disadvantages

Every separate tier has to communicate with the tier above and the tier below it self. This means that inter-tier communication can become a bottleneck. This increased overhead means that this architecture is more suited for complex applications where the benefits overcome the costs.

Peer-to-peer

The peer-to-peer architecture, also known as P2P, has gained a lot of interest and focus due to the success of applications such as Napster [37], Gnutella [38] and the new FastTrack technology [39]. In P2P there is no central server, all workstations are equal (or *peers*). A workstation in a peer-to-peer network is called a node, and can function as both client and/or server according to the current state of the network. The nodes in a peer-to-peer network are connected to several other nodes, as seen in figure 2-4.

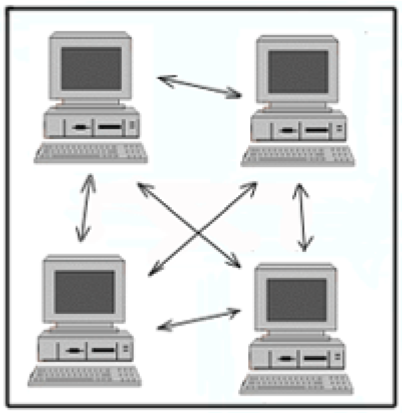


Figure 2-4. In this P2P network, every node is connected to all other nodes in the network. In large P2P networks, this is of course not possible. A node will be connected to a reasonable amount of nodes, which are connected to other nodes, thereby creating a large network.

Advantages

No central stored information. Since every node can be a server, you have access to the collected information of every node in the network. P2P is not dependent of one server, if one node disappears, another one will soon take its place.

Disadvantage

Nodes are not as stable as in a client/server environment. A node may very well disappear while you are accessing it. Requests travel from node to node until a node has the requested data. This behavior can make requests slow in the peer-to-peer architecture. P2P doesn't offer solid performance in larger installations or under heavy network traffic loads.

2.1.3 Approaches to Distributed Systems

There are a few different distributed communication technologies, or approaches, that can be used to create distributed systems. The more high-level approaches use and take advantage of the lower-level ones. These are some approaches to distributed systems:

- Lowest level: socket communication (message passing through socket connections). This is the foundation of all Internet communication.
- Remote Procedure Call (RPC) [36]. Allows applications to call procedures located on a server as if they were local procedures.

These are more advanced approaches using distributed objects:

- CORBA (Common Object Request Broker Architecture) [34] is an open, vendor-independent architecture and infrastructure that applications can use to communicate over networks. This technique can be described as a platform- and language independent version of RPCs. CORBA is developed by the Object Management Group, an association with hundreds of member companies.
- Remote Method Invocation (RMI) [28] is the Java version of RPC.
- Microsoft's Distributed Component Object Model (DCOM) [35].

Some of these distributed approaches have been used by different companies to construct middleware infrastructures. These infrastructures are provided to third party system developers to simplify, speed up and make the development of distributed enterprise systems cheaper and more robust.

Below, a few examples of middleware infrastructures are presented:

- Sun's Enterprise Java platform (J2EE).
- Microsoft COM+ together with Microsoft Transaction Server (MTS).
- Netscape Application Server

The J2EE platform is the only middleware architecture that is independent of the founding company. It was specified by Sun Microsystems but the specification is open. Any company can develop its own J2EE application server and sell it as long as the application server follows the J2EE specification [49]. There is currently (August 2001) at least 37 different J2EE application servers available on the market [45]. Each chooses to implement the specification differently, with differing support for features and different pricing.

However, Microsoft's MTS architecture and Netscape's application server are closed systems and cannot be edited. Changing application server would mean rewriting all code. On the J2EE platform there is plenty of opportunity to choose and change application server without to much hassle if disappointed.

2.2 Prohunt's Existing ICM Platform

Prohunt ICM is a set of products for Intellectual Capital Management. It consists of three different software products:

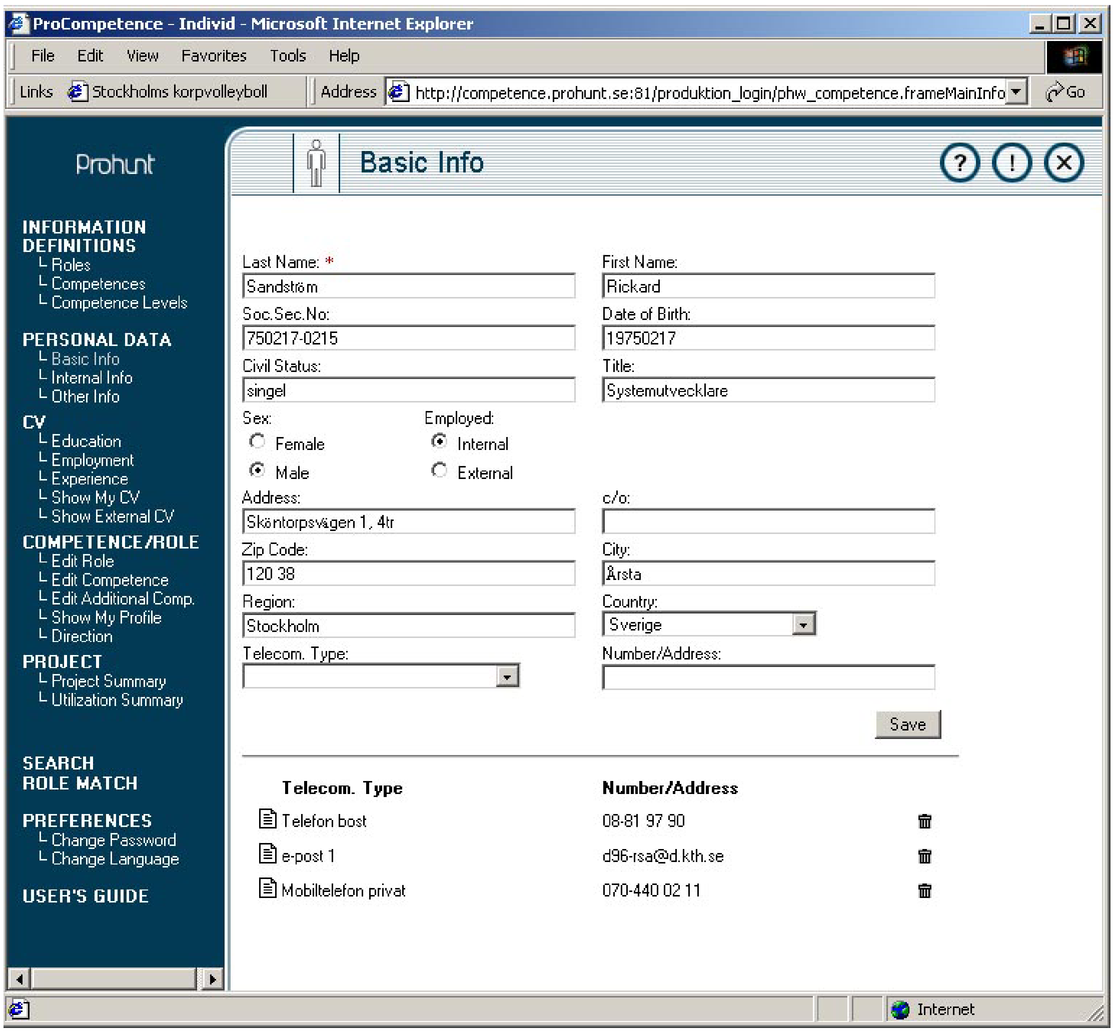
- ProCompetence – for strategic and business oriented competence support.
- ProCareer – for strategic career planning for individuals and organizations.
- ProResource – for efficient planning, manning and follow-up of projects.

The underlying server platform is fairly similar, but they differ in their client implementation. ProCompetence for example, has a traditional web client and ProCareer has a Shockwave client.

2.2.1 ProCompetence

ProCompetence is a tool for keeping track the different competences within an organization. In ProCompetence, Employees declare their competences, job position (role) and register current projects they are involved in.

Figure 2-5 shows the initial view of ProCompetence were an employee enters basic information about himself. The menu on the left shows that there are additional views for entering CVs, adding roles and competences and managing projects.



Last Name: * Sandström	First Name: Rickard
Soc.Sec.No: 750217-0215	Date of Birth: 19750217
Civil Status: singel	Title: Systemutvecklare
Sex: Female [✓ Male]	Employed: [✓ Internal] External
Address: Skäntorpsvägen 1, 4tr	c/o:
Zip Code: 120 38	City: Årsta
Region: Stockholm	Country: Sverige
Telecom. Type:	Number/Address:

Telecom. Type	Number/Address
Telefon bost	08-81 97 90
e-post 1	d96-rsa@d.kth.se
Mobiltelefon privat	070-440 02 11

Figure 2-5. ProCompetence helps employees to define their competences. This is the view that employees will first meet when starting the application.

A Competence manager will have a completely different set of options in the program. Numerous graphs and reports can show for example if the right persons are working in the right seats or you can choose a group and see the difference between existing competence and wanted competence. Figure 2-6 shows the knowledge levels of a group in the chosen competences.

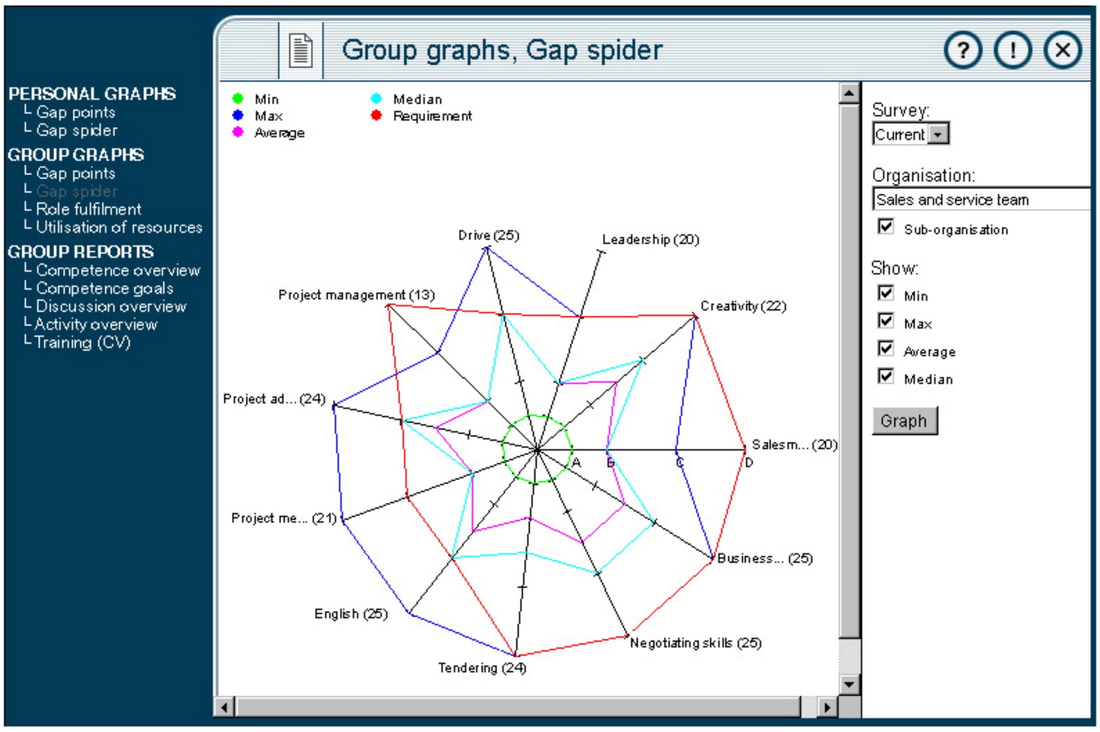
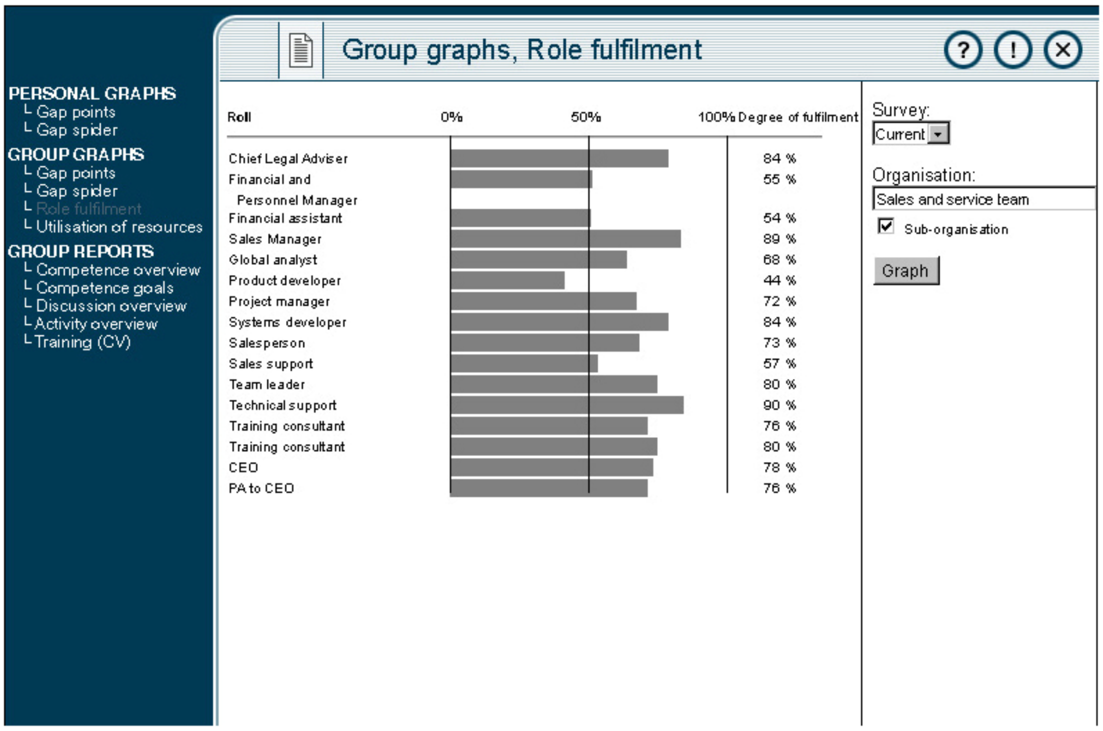


Figure 2-6. It is simple to create reports and graphs for individuals, departments and organizations. This is a gap analysis for a group where the existing competence of the group is compared with the desired competence in a spider graph. This clearly displays which competences are lacking and which are overly represented. For example, does the group include a skilled financial manager, a senior systems developer or trained sales personnel? [6]

Being web-based, ProCompetence is easy to use for all co-workers wherever they are in the world by means of either the company intranet or the Internet. With ProCompetence, employees and managers can get information about competence gaps, resource gaps, role achievement, a compilation of the organization's total overall competence and planned increases in competence. Figure 2-7 shows another way of depicting how well a group fulfills different roles.



Roll	Degree of fulfilment
Chief Legal Adviser	84 %
Financial and Personnel Manager	55 %
Financial assistant	54 %
Sales Manager	89 %
Global analyst	68 %
Product developer	44 %
Project manager	72 %
Systems developer	84 %
Salesperson	73 %
Sales support	57 %
Team leader	80 %
Technical support	90 %
Training consultant	76 %
Training consultant	80 %
CEO	78 %
PA to CEO	76 %

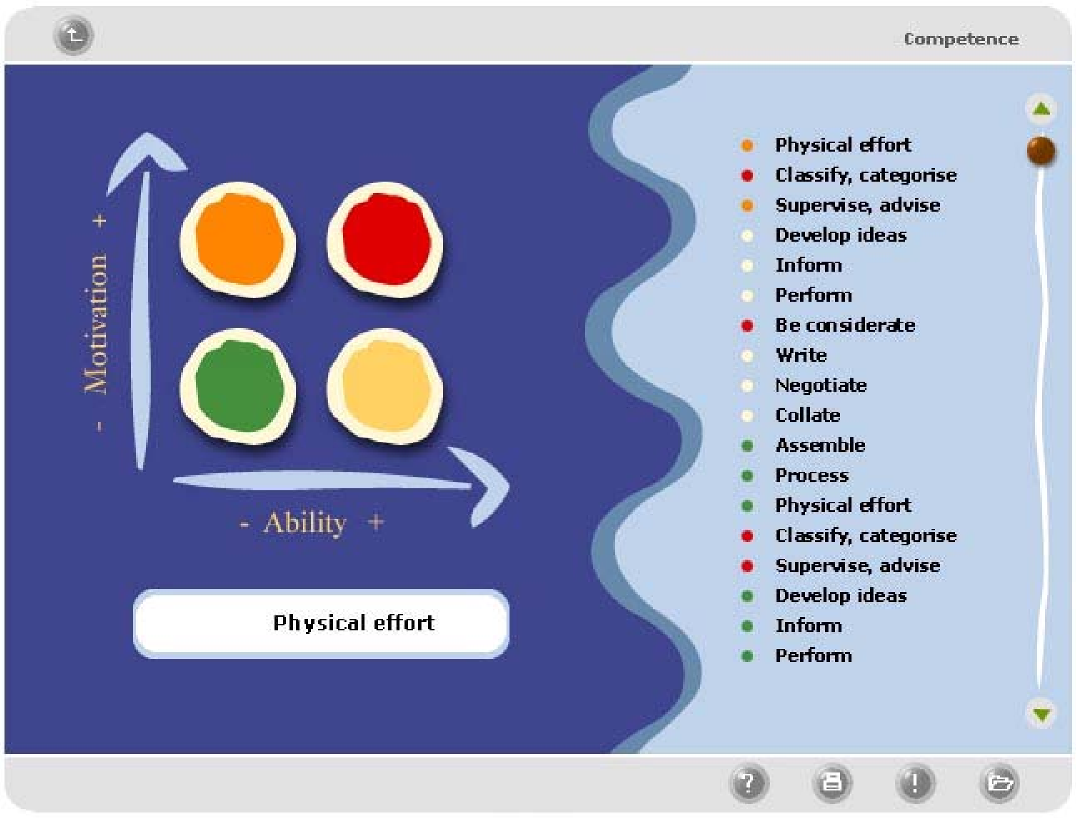
Figure 2-7. This picture shows an overview of role achievement within a group [6].

2.2.2 ProCareer

ProCareer is a career-planning tool and its purpose is to help co-workers to match their motivations, ambitions and personal goals with the company's strategic needs and future goals.

ProCareer consists of two parts, first ProCareer Inward where employees specify their motivations and ambitions, and secondly ProCareer Outward, which deals with relationships between employees and the company.

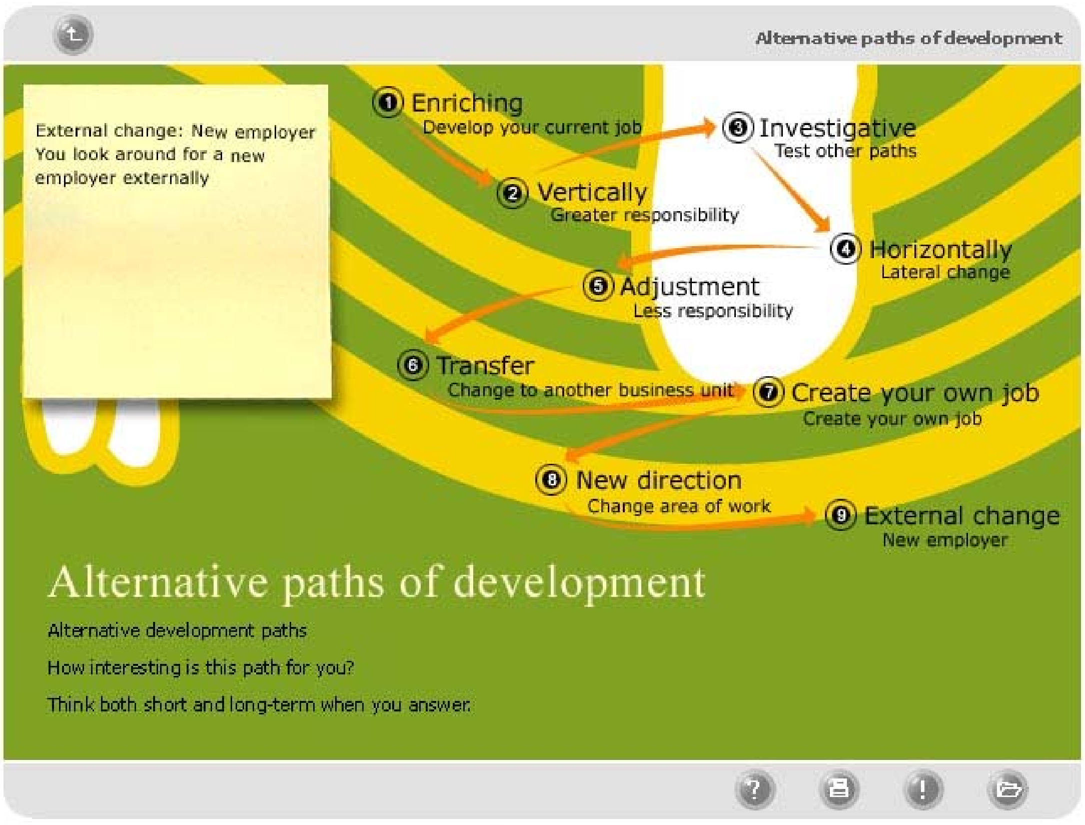
As seen in figure 2-8, ProCareer has a totally different user interface.



- Physical effort
- Classify, categorise
- Supervise, advise
- Develop ideas
- Inform
- Perform
- Be considerate
- Write
- Negotiate
- Collate
- Assemble
- Process
- Physical effort
- Classify, categorise
- Supervise, advise
- Develop ideas
- Inform
- Perform

Figure 2-8. ProCareer Inward deals with you and your personal needs. In the Practical Skills module, you decide how motivated you are to use certain skills and evaluate your capacity to use them, in order to pinpoint your Key Skills [6].

Figure 2-9 on the other hand shows the Outward part of ProCareer, where the employee chooses between different career paths within the organization. The employee ranks each path with regard to different properties.



External change: New employer
You look around for a new employer externally

Alternative paths of development

Alternative development paths

How interesting is this path for you?

Think both short and long-term when you answer.

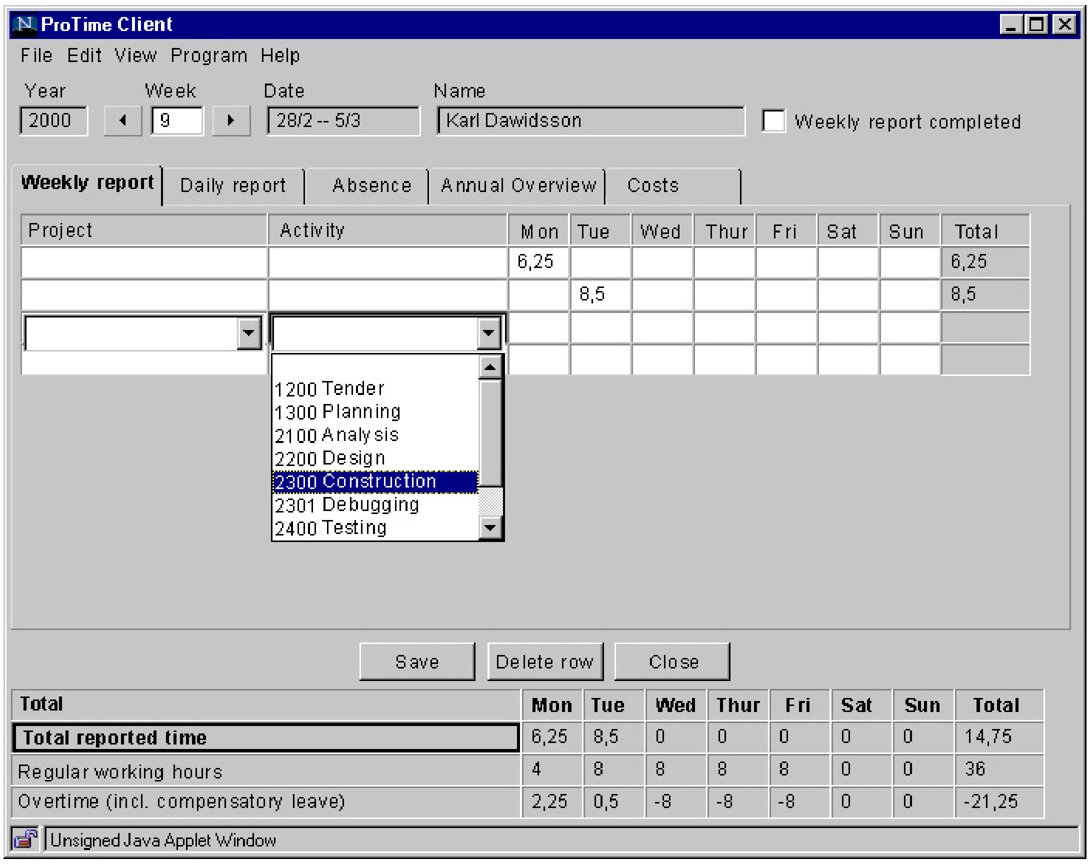
Figure 2-9. ProCareer Outward deals with the outside world and your organization. In the Alternative Development Paths module, you identify and describe both the short and long term development paths that you find most interesting within your organization [6].

The goal of ProCareer is to match the motivations, ambitions and goals of the employers with the strategic needs and future goals of the organization. It also helps employees to visualize ways of personal development that are unclear to them, as well as identify and profile their primary competences. The use of ProCareer within an organization is supposed to decrease the turnover of employees, increase the efficiency of leadership and teach employees to take responsibility of their own career development.

2.2.3 ProResource

ProResource is a web-based tool for planning, managing and following-up the resource situation in companies and organizations. Resources might be personnel, time, money and premises. ProResource simplifies planning and continuous follow-up for project leaders in several ways. Projects and activities can be defined to which personnel and time are then allocated. It is possible to search for employees with the right skills and knowledge needed in a particular project. Employees also use ProResource to their time reports in a module called ProTime.

ProResource allows project leaders to easily follow up the projects to make sure they are on time and that all needed resources are available. ProTime is an advanced application for employees to do time-reports. Figure 2-10 shows the main frame of ProTime. The information gathered from the employees is used in ProResource and to generate reports and graphs.



Year	Week	Date	Name	
2000	9	28/2 -- 5/3	Karl Dawidsson	☐ Weekly report completed

Project	Activity	Mon	Tue	Wed	Thur	Fri	Sat	Sun	Total
		6,25							6,25
			8,5						8,5

1200 Tender
1300 Planning
2100 Analysis
2200 Design
2300 Construction
2301 Debugging
2400 Testing

Total	Mon	Tue	Wed	Thur	Fri	Sat	Sun	Total
Total reported time	6,25	8,5	0	0	0	0	0	14,75
Regular working hours	4	8	8	8	8	0	0	36
Overtime (incl. compensatory leave)	2,25	0,5	-8	-8	-8	0	0	-21,25

Figure 2-10. Using ProTime, each employee can report the amount of time he/she has worked on a particular project. First choose the project, then the activity and then fill in how much time has been spent on it. You can also click on a tab and fill in more detailed information. Reports can then be generated on, for example, the individual and project level [6].

ProResource can generate numerous reports and other tools for project administrators. Figure 2-11 shows consultants matching a certain need and their availability.

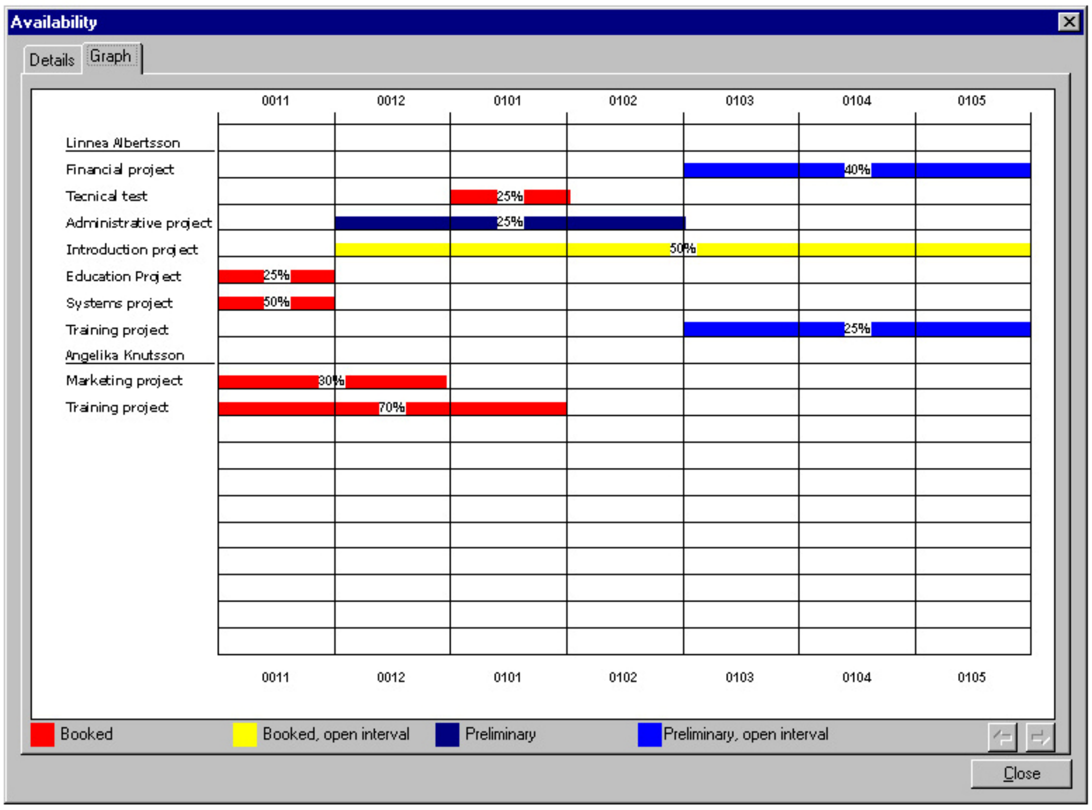


Figure 2-11. ProResource can search for consultants whose skills match a consultant profile and show their availability in graphs or as text [6].

2.2.4 Specification and Architecture

The general architecture is largely the same for all three products. They are all web based client/server solutions with Oracle Application Server running as web server and Oracle8i as database. Depending on the size of the customer company the web server and the database resides on the same machine or on separate machines. All communication between client and server is encrypted using SSL (Secure Sockets Layer). Figure 2-12 depicts the general architecture of Prohunt's products.

Client-side

As mentioned earlier, the architecture is very similar between the three products. However, there are some differences, mostly on the client-side. Both ProCompetence and ProResource have a web interface that consists of Html-pages with some additional JavaScripts, dynamically created by Java Servlets.

The ProTime module of ProResource differs from the other products because it is an advanced Java Applet instead of ordinary Html-pages. ProCareer is also different due to the Shockwave interface of the client. The differences are only in the technology of the user interface though; behind the scenes the architecture is the same.

All clients run on the browsers Internet Explorer and Netscape, version 4 or higher.

Server-side

As stated before, the server-side architecture is more uniform between the three products. On top of the server architecture are the Java servlets that dynamically create the user interface. These servlets handles all user interaction, receiving requests and returning responses. Each time a user interacts with the user interface, a request is sent to a servlet. Many servlets on many levels may be involved, but ultimately one of them calls a stored PL/SQL subprogram, waits for a result, and then propagates it up to a servlet which generates the appropriate response that is sent to the client.

Two very important servlets, *SDispatcher* and *DBLayer*, are used for every database access and they function as the glue between the Java servlets and the PL/SQL stored procedures. SDispatcher and DBLayer function as an interface between the Java servlets and the PL/SQL stored procedures. SDispatcher is responsible for verifying that the user has permission to call the requested procedure on the server. If permission is granted, SDispatcher tells DBLayer to call that procedure.

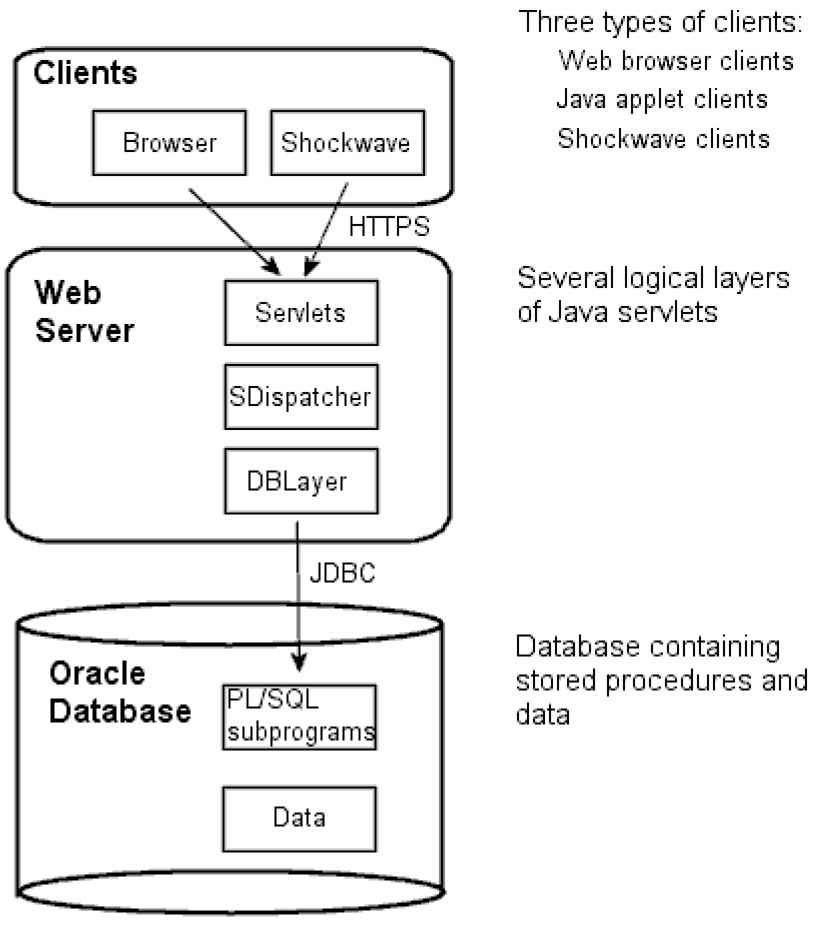


Figure 2-12. Architectural overview of the ICM platform.

DBLayer handles communication between the Java servlets located on the Web Server and the PL/SQL procedures located in the Oracle Database. DBLayer converts procedure calls, parameter data types and return values between Java and Oracle PL/SQL.

DBLayer uses Java Database Connectivity (JDBC) to access the next logical layer, the business logic implemented in PL/SQL as seen in figure 2-12.

All three products are also available in an ASP (Application Service Provider) version, enabling customers to let Prohunt take care of the operation and maintenance of the system. In the ASP solution, the product is situated on a server run by Prohunt, but the customer accesses the application as if it was run locally by the customer company. Technically there is no difference between the ASP product and the normal product, except that the servers are not connected to the customer's local network, so network traffic has to be allowed between the customers network and Prohunt's servers over the Internet. Since all network traffic is already encrypted, there is no need to alter the network protocol used.

2.2.5 Analysis

So, why did Prohunt feel the need to change technical platform? To understand this, one has to be familiar with the origin of Prohunt's applications.

Prohunt's first application was ProCompetence, which was a classic client/server application with a Windows client developed with Centura Team Developer (earlier known as SQLWindows) [7]. Centura is a 4th generation (4GL) development tool similar to Sybase's PowerBuilder, Borland's Delphi and Microsoft's Visual Basic and provides fast and easy development of Windows client applications with a graphical user interface (GUI).

ProResource was also a Centura client from the start. But at the start of the implementation of ProCareer, web-based applications were a hot topic and it was decided that all of Prohunt's product would be web-based.

To be able to both develop ProCareer *and* port ProCompetence and ProResource to the web, it was decided to keep the old server architecture to shorten development time and minimize the cost. The Centura clients were replaced by new web interfaces based on Java Server Pages. Special Java Servlets were built to connect the web interfaces to the Oracle stored procedures.

Over time, the combination of two different programming languages approaches to application development was considered a bad compromise. Prohunt wanted to fully convert the old-fashioned client/server architecture to more modern multi-tiered, platform independent, with Java on both clients and server.

Prior to the start of this thesis, Prohunt decided to remodel and rewrite the entire login and authorization sections of the applications. Today, every product is independent of the others. A user has to log into every application one by one. Prohunt wanted a user to be able to log in once for all systems. The products should all belong to the same authorization module.

All of this resulted in a decision to port all products to a distributed J2EE architecture. A figure of the proposed new architecture can be seen in appendix B.

3. PL/SQL

Prohunt's existing server platform is based on an Oracle database and the language Oracle PL/SQL (Procedural Language extensions to SQL). This extension is basically stored procedures that allow developers to add flow control, logic design and more complex behaviors onto unstructured SQL command blocks. PL/SQL also implements basic exception handling, database triggers and cursors (a data structure similar to record sets).

3.1 Background

PL/SQL was first released with Oracle version 6.0 in 1991. In the beginning, what would become PL/SQL was only a batch processing script language on the server side called SQL*Plus. SQL*Plus was very limited in functionality. For example, you could not even store procedures or functions for execution at some later time. On the client side Oracle has a tool called Oracle Forms (formerly known as SQL*Forms). SQL*Forms V3.0 incorporated the PL/SQL runtime engine for the first time on the client side, allowing developers to code their procedural logic in a natural, straightforward manner [10].

Table 3-1 shows the evolution of PL/SQL from Version 1.0 to the latest Version 9.0 and some examples of significant improvements with each release.

Version/Release	**Characteristics**
Version 1.0	Available in Oracle 6.0 and SQL*Forms version 3,
Release 1.1	Supports client-side packages and allows client-side programs to execute stored code transparently
Version 2.0	Major upgrade to version 1.0 available in Oracle Server Release 7.0. Adds support for stored procedures, functions, packages, programmer-defined records, PL/SQL tables and much more.
Release 2.1	Available with Release 7.1 of Oracle Server. Supports user-defined subtypes, enables stored functions inside SQL statements and you can now execute SQL DDL statements from within PL/SQL programs.
Release 2.2	Available with Release 7.2 of Oracle Server. Supports cursor variables for embedded PL/SQL environments such as Pro*C.
Release 2.3	Available with Release 7.3 of Oracle Server. Enhances functionality of PL/SQL tables, adds file I/O and completes the implementation of cursor variables
Version 8.0	Available with Oracle8 Release 8.0. Oracle synchronized version numbers across related products, thus the drastic change. Supports many enhancements of Oracle8, including large objects (LOBs), collections (VARRAYs and nested tables) and Oracle/AQ (the Oracle/Advanced Queueing facility)

Version 9.0	Available with Oracle9i. Many performance improvements, support for native compilation speeds up computations. Tighter integration of the PL/SQL and the SQL runtime engines. Scrolling cursors and CASE statement has been added [14].

Table 3-1. PL/SQL versions and releases [10].

PL/SQL developers are worried that Oracle will discontinue supporting PL/SQL since Oracle nowadays has a built-in Java virtual machine and native support for Java inside the server. However, this is not the case. Oracle is still developing and improving PL/SQL, for example PL/SQL is significantly faster in Oracle 8i than in 8.0, due to both internal optimizations and new features [16].

3.2 Language

PL/SQL was modeled after the programming language Ada, hence it is a high-level programming language. It incorporates many elements of procedural languages, including:

- A full range of data types
- Explicit block structures
- Conditional and sequential control statements
- Loops of various kinds
- Exception handlers for use in event-based error handling
- Constructs for modular design – functions, procedures and packages
- User-defined data types

Since PL/SQL is a procedural block language, it is quite easy for someone who has some experience of programming in other procedural languages like C/C++, Java or a similar language, to understand the structure and functionality of the code. A PL/SQL block consists of up to four different sections; the header, declarative section, execution section and the exception-handling section (see code example below). Only the execution section is mandatory, the other sections are not required. Figure 3-2 show the basic structure of a typical PL/SQL block:

```
declare
      <declarative section>
begin
      <executable commands>
   exception
      <exception handling>
end;
```

Figure 3-2. The basic structure of a PL/SQL block. Since the header is missing, this is called an anonymous block; it cannot be called by itself.

Block Header

The block header contains the name of the block and invocation information. There are three kinds of blocks; anonymous (cannot be called), procedures (doesn’t return any value) and functions (procedures that will always return a value).

Declarative Section

Variables and constants have to be declared in the declarative section before use. All SQL data types and PL/SQL data types are allowed and are handled without conversions.

Execution Section

The execution section is where the actual code is placed. The PL/SQL runtime engine will execute this code.

Exception Section

The exception section is where the code that handles exceptions to normal processing (warnings and error conditions) is placed. [10]

A simple example of a PL/SQL block, also known as a subprogram, is described in figure 3-3.

A few syntactic explanations: `--` makes the rest of the row a comment. You can also use the well know `/* comment */` and `||` is the string concatenation operator.

```
-- First comes the block header. Procedure name and argument list
-- with name and types
procedure update_cost (
   isbn_number        in number
)
is
  -- This is the declarative section where
  -- you declare local variables
  temp_cost     number;
  /* The execution section starts here */
  begin
    SELECT cost FROM db.book INTO temp_cost WHERE isbn =
      isbn_number;
    if temp_cost > 0 then
      UPDATE db.book SET cost = (temp_cost*1.2) WHERE isbn =
        isbn_number;
    else
      UPDATE db.book SET cost = 10 WHERE isbn = isbn_number;
  end if;
  COMMIT;
  -- Exception section handles all possible exceptions
  exception
    when NO_DATA_FOUND then
      INSERT INTO db.errors (CODE, MESSAGE) VALUES(99, 'ISBN ' ||
        isbn_number || ' NOT FOUND');
end;
```

Figure 3-3. A subprogram called `update_cost` *that retrieves the price of a specific book from a database. The price is stored in the variable* `temp_cost`*. If the price is larger than zero, the price in the database is updated to the old price times 1.2. If the old price of the book was zero, it is changed to 10. Last of all, the changes to the database are commited (made persistent). But in case the book doesn't exist in the database, a* `NO_DATA_FOUND`*-exception is raised and an error message is inserted into the database.*

As seen above, PL/SQL is a typed language.

3.3 Architecture

The PL/SQL runtime environment is a client/server solution. Execution of PL/SQL code can only be performed by the PL/SQL runtime engine which is only available inside the Oracle Server, or on the client side, inside a tool called Oracle Forms. PL/SQL on the client-side will not be described further, since this requires that Oracle Forms is used for client development which is not the case at Prohunt.

PL/SQL blocks are modularized into *packages* inside the Oracle Server. Packages are divided in two sections, the header and the body. The header contains declarations and the body the executable code in much the same way as C header and source files. Subprograms and variables that should be accessible outside of the package must be declared in the package header. All the source code is placed in the package body. The body is hidden from the outside, only the declarations in the package header are visible. The package header is the interface of the package to the outside.

Subprograms placed inside a package are called *stored subprograms*. If the subprogram is not declared in the header, thus not accessible from outside the package, it is called *local subprograms*. There are also *stand-alone subprograms* that are not placed within a package [9].

Below is an example of what a package header could look like:

```
package test_package
is
   -- User defined type
   type t_curRef        is  ref cursor;

   -- A test function with two IN arguments
   function test_function (
      param1      in number,
      param2      in number
   ) return number;

   -- A test procedure with three arguments, two in and one out
   procedure test_procedure (
      param1      in number,
      param2      in varchar2,
      outparam    out t_curRef
      );
end;
```

Figure 3-4. Example of a PL/SQL package header.

The functions, procedures, variables and programmer-defined types that are declared in the package header are then available from other packages.

Figure 3-5 illustrates an example the code of the package body.

```
package body test_package
is
   -- Variables declared here will be global
   -- inside the package
   number_of_tests      number;

   function test_function (
      param1      in number,
      param2      in number
   ) return number
   is
   begin
      -- Function code
      ...
   end;

   procedure test_procedure (
      param1      in number,
      param2      in varchar2,
      outparam    out t_curRef
   )
   begin
      -- Procedure code
      ...
   end;
end;
```

Figure 3-5. Example of a package body.

The PL/SQL packages are compiled and stored in the Oracle database data dictionary. Packages are schema objects, which mean that they can be referenced and invoked by any application connected to the database. When a PL/SQL subprogram is called it is loaded and passed to the PL/SQL runtime engine. The runtime engine interprets the compiled PL/SQL code line by line. The Oracle Server is capable of processing both PL/SQL blocks and SQL statements as shown in figure 3-6. [9]

Also, subprograms share memory so only one copy of the subprogram is loaded into memory for execution by multiple users [9].

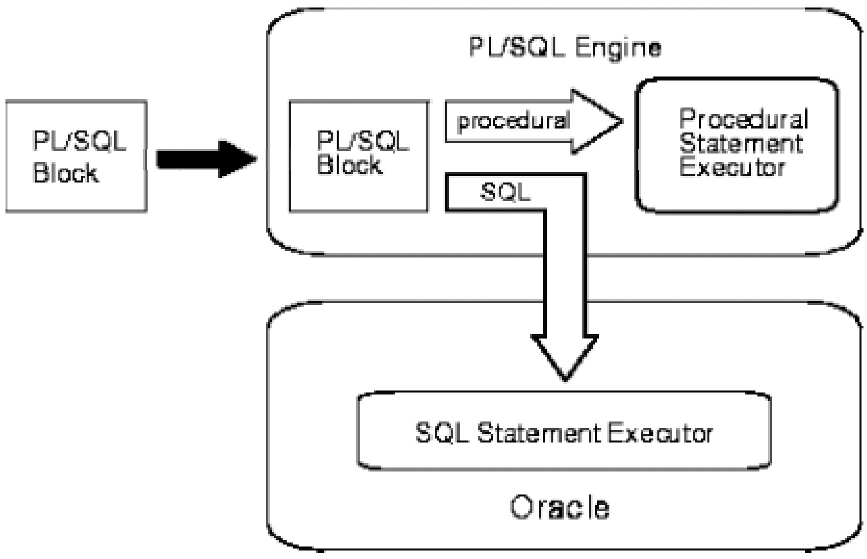


Figure 3-6. The PL/SQL Engine executes both PL/SQL blocks and SQL statements [9].

Figure 3-6 depicts the PL/SQL engine in Oracle8. The integration of the PL/SQL runtime engine and the SQL Statement Executor has been further developed in Oracle9i [14].

3.4 PL/SQL Summary

Since PL/SQL is executed in the Oracle environment, near the data both physically and logically, it is optimized for handling large amounts of data at the same time. But calculations and procedural logic are not believed to be as fast. The main drawback though, is the inflexibility of the system. Data and code are both stored in the database, making it harder to separate the two and the level of data abstraction is lower than in Java. For example, there are very little support for object-oriented concepts like encapsulation, information hiding and inheritance.

Another disadvantage with PL/SQL is that developers are bound to the Oracle platform. This may not be a problem as long as the Oracle database is used. However, if another database is used, the PL/SQL code cannot be used and has to be ported to, or rewritten for that database.

4. Enterprise Java Technologies (J2EE)

The Java 2 Platform Enterprise Edition [18] (also known as J2EE) is a collection of technologies, all using Java. Every technology or API fills its function inside J2EE. However, some are also available outside of J2EE, such as JDBC. Below are the technologies that belong to J2EE:

- Enterprise JavaBeans (EJB) [18]
- JavaServer Pages (JSP) [20]
- Java Servlets [21]
- Java Naming and Directory Interface (JNDI) [23]
- Java Database Connectivity (JDBC) [21]
- Java Transaction API (JTA) and Java Transaction Service (JTS) [24]
- Java Message Service (JMS) [25]
- J2EE Connector Architecture [25]
- A subset of CORBA (Common Object Request Broker Architecture) known as RMI/IDL and RMI over IIOP [27]
- The Extensible Markup Language (XML) [29]
- ECperf [30]

4.1 Distributed Multi-tiered Platform

The J2EE platform is a distributed, multi-tiered platform [1]. The fundamental set up of the different tiers depicted in figure 4-1 and also presented below.

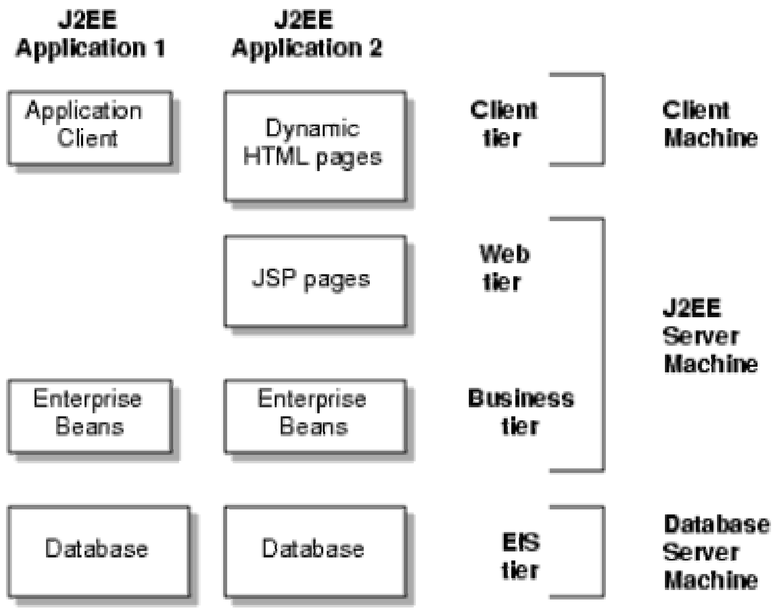


Figure 4-1. The J2EE distributed, multi-tiered application model [1].

Enterprise Information System Tier

Bottommost there is a database where all information is stored. The tier that contains the data is called EIS, enterprise information system.

Business Tier

On top of the EIS is the business tier, which contains most of the logic of the application. Calculations and processing are performed in this tier. In a J2EE application, this tier is located in an application server on the J2EE server machine.

Web Tier

The web tier is not mandatory. Its existence depends on the type of client application. If an Applet or stand-alone application is used as the client, the web tier is not necessary. But if the client consists of dynamic Html-pages shown in the client's browser, the web tier is essential. The web tier consists of Java Server Pages or Java Servlets (called *web components*). Web components are Java code that is executed on the web server and dynamically create and return static Html-pages to the client's browser, allowing the resulting web pages to depend on user input and server state.

Client Tier

The set up of the client tier can vary, depending on the application. The client can be a stand-alone Java application, a Java Applet, dynamic Html-pages returned from the Web Tier to a client browser or a Shockwave application.

The multi-tiered nature of the technology separates data (database tier), function (business tier) and presentation (client tier for standalone applications and client tier together with the web tier for web applications).

4.3 J2EE Components

J2EE Applications consists of *components*. A component is a self-contained functional software unit that communicates with other components via well-defined interfaces. They are written in ordinary Java and are fully reusable. The J2EE specification defines the following components [1]:

- Client applications and applets are client components.
- Java Servlets and JavaServer Pages (JSP) are web components.
- Enterprise JavaBeans (EJB) are business components.

Client Components

A client component can either be a standalone application, a web browser or a java applet. The clients in a J2EE application are so called *thin clients*; they are basically just a user interface of the underlying business application. Most computations are hidden in the web and business tiers on the J2EE servers. Client components belong to the client tier.

Web Components

J2EE web components can be either Servlets or JSP pages. Servlets are Java classes with embedded Html-code that receive HTTP-requests and produces HTTP-responses. Java Server Pages on the other hand are Html-code with embedded Java code. Both are complied (Servlets are compiled explicitly by the developer while JSPs are implicitly compiled by the web server when invocated for the first time) and run on the web server, which is part of the web tier.
Java Server Pages resembles Microsoft's Active Server Pages (ASP) in the way they

are constructed. Java Server Pages are considered easier and faster to develop than Servlets because of the more hands-on approach. JSP may be preferred when the dynamic element of the Html-code is small. For complex applications though, Servlets are the way to go.

Business Components

The business tier is the heart and soul of a J2EE application; this is where the actual data handling and processing take place. The business components are called Enterprise JavaBeans (EJB). There are two essentially two kinds of beans, *session beans* that contain the business logic and *entity beans* that represent persistent data stored in a database and hold the actual data. These components are deployed in J2EE containers inside an application server. The containers provide many services like naming service lookups, component life-cycle-handling, transaction-handling, security issues and load balancing, allowing the developer to focus on the actual business logics of the application.

4.3 Enterprise JavaBeans

As stated earlier, the business components are called Enterprise JavaBeans. An Enterprise JavaBean is similar to a Java class in that it is a combined set of code with methods. However, it is more than a simple class. To be an Enterprise JavaBean, a component has to comply with a massive set of rules known as the J2EE Specifications [49]. These specifications are quite extensive and set up rules that make a JavaBean ready to be inserted into any J2EE Application Server, independently of the vendor, as long as the Application Server complies to the specification.

A JavaBean has to implement several things such as a *Home* and *Remote Interface*, some mandatory methods, it must have a *JNDI* name (described in chapter 4.5.2 Services Provided by the Application Server), a *deployment descriptor* and some more. A JavaBean thus consists of several different files, which are then packaged (compiled) in a JAR-file [55] (Java Archivefile) and deployed in a J2EE container.

Session Beans

A session bean represents a single client inside a J2EE server. An instance of the session bean is created when a client creates it for the first time and it is removed to garbage collection when the same client invokes the remove methods.

There are two kinds of session beans; *stateful* and *stateless* session beans. A stateful session bean contains data about the client and holds a state between invocations. The stateful bean can only have one single client and is associated to the same client during its lifetime. This state is only retained for as long as the session bean is alive. The state is lost, when the bean is removed by the client.

A stateless session bean on the other hand does not contain any state or client-specific information *between* invocations of its methods. All stateless session beans are thus equal except during method invocation, allowing the EJB container to assign any instance to any client. This fact can be used for better performance and scalability. Because stateless session beans can support multiple clients, they can offer better scalability for applications that require large numbers of clients. Normally, an application requires fewer stateless session beans than stateful session beans to support the same number of clients. [8]

Performance may also be better generally for stateless session beans. The EJB container may write a stateful session bean out to secondary storage at times. On the other hand, stateless session beans are never written out to secondary storage [8]. Also, the equality of stateless session beans allows the concept of pooling. The EJB container can keep a pool of instantiated stateless session beans which then can be assigned to a client if needed. Both these issues increases performance for stateless session beans, and they should be used instead of stateful session beans whenever possible.

Entity Beans

Entity beans are different to session beans in several ways. Entity beans are *persistent*, allow *shared access* and must have *primary keys*.

Entity beans represent entities (data), stored in some persistent storage mechanism, usually a database. Persistent data is data that continues to exist even after you shut down the database server or the applications the data belongs to.

Entity beans compose yet another logical layer between the computational logic (session beans) and the database. Depending on usage, entity beans may slow down an application. The extra layer means extra overhead. But when the data record has been read from the database to the entity bean, all processing of that data record can be performed on the application server without having to read or write the database. Entity beans may boost performance if the data records are read and updated frequently, but requires much primary memory for storage.

The persistence of an entity bean can either be *container-managed* or *bean-managed*. The difference lies in how the persistence are managed, or handled. The container-managed entity beans are easier to develop since database access is handled by the container and no explicit database code has to be written. Container-managed persistence can often be slower than the bean-managed persistence where the developer writes the code for database access, with the opportunity to optimize the code for the specific application.

The shared access means that any instance of an entity bean can be accessed by any number of clients since there is only one instance for a specific set of data (one bank account = one entity bean). This means that the concept of transactions is crucial for entity beans, making it impossible for two clients to update the same entity bean at the same time. Fortunately, the EJB container will handle the transaction management after the developer has specified the transaction attributes in the beans deployment descriptor.

Since there is only one entity bean for each database entity, an entity bean has to contain a unique object identifier – a primary key. The primary key is used to find entities, or records, in a database just like in a relational database.

Home and Remote Interfaces

The business components (enterprise beans) must follow a certain, well-defined template (the J2EE Specification) to be deployable. The Home and Remote Interfaces contain declarations of the invokable methods of a bean. These interfaces expose the methods of a bean to a container, and thereby to the outside world. When a client has obtained a reference to one of these interfaces, the client is able to call the bean's methods for execution.

The *Home Interface* of a bean defines the mandatory methods of the bean specified by the J2EE Specification. This includes methods for creating and destroying instances of the bean. These mandatory methods differ a little between session beans and entity beans, but most of are involved in managing the state of the bean.

The *Remote Interface* on the other hand defines the business methods of the bean, extending the set of methods that can be invoked by the container. The business methods of the bean are methods added by the developer and these methods are the essence of the application.

Figure 4-2 shows how the home and remote interfaces make it possible for clients to invoke methods of the enterprise beans even though the client and the business components aren't located on the same physical machine.

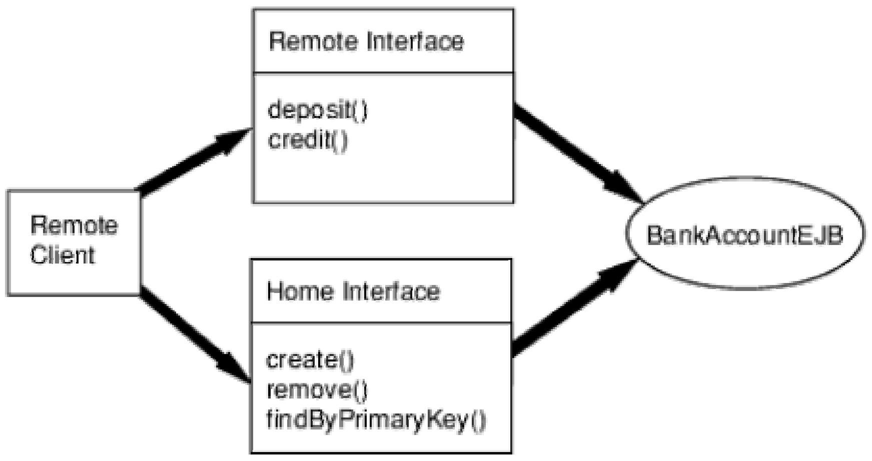


Figure 4-2. Remote clients cannot invoke methods of a bean directly. The Home and Remote Interfaces presents a window for the client to the bean.

4.4 JDBC

Java DataBase Connectivity (JDBC) is a technology for accessing databases from Java applications. The JDBC API allows a developer to read, write, update and delete data stored in relational databases from his Java methods via the SQL language. JDBC is the Java equivalent of Microsoft's ODBC API [50], which is used for similar tasks on the Windows platform.

JDBC is not a J2EE specific technology, it can and is used by all Java programs to access databases. As described above, the database access is an automated service in container-managed entity beans. Programmers are not required to know the details of JDBC, it is handled in the background by the J2EE containers and the bean implementation contains no JDBC code or SQL operations. However, if the developer decides to use bean-managed entity beans, he or she is in fact choosing to write the JDBC code himself. In some cases it can also be advantageous to let session beans access the database directly without any entity beans in-between, and in these cases too, JDBC has to be explicitly used for the database operations.

The JDBC API consists of two parts [8], an application-level interface used by the application components to access a database. This interface is what the programmer uses in his applications. The other part is a service provider interface used to create a JDBC driver for a specific database management system (DBMS). Each DBMS like Oracle, Micrsoft SQLServer or MySQL implements its own JDBC driver that programmers can invoke from their applications, using the application-level interface.

4.5 The Application Server

The main advantage of the J2EE technology is the services that it automatically provides. Development of multi-tiered applications would be much more difficult and time consuming without theses services. These kinds of applications involve lots of communication between layers, transaction handling, state management, multithreading, resource management and other complex matters. All of this is handled automatic by the J2EE application server, which lets the developer concentrate on the specific details of the real business problems.

4.5.1 EJB Containers

For enterprise components to take advantage of these services, they have to be inserted, or *deployed*, into J2EE *containers*. The components themselves are platform independent and the containers function as the interface between components and the low-level, platform-specific functionality that supports the components.

As described earlier, the components are packaged in JAR-files, which are deployed in the container. This JAR-file also includes a *deployment-descriptor*. This deployment-descriptor is an XML-file that contains container settings, telling the container which services the components should have access to, and customizing the behavior of these services. However, some services are non-configurable and cannot be customized like that. The exact appearance of the deployment-descriptor differs in-between Application Servers.

Listed below are some of the services provided by the EJB containers:

- JNDI lookup services provide a unified interface to multiple naming and directory services. Using this, application components can find and access other components and naming and directory services.
- The J2EE remote connectivity model manages low-level communications between clients and enterprise beans. After creation of an enterprise bean in the application server, a client can invoke the bean's methods as if they were local.
- The J2EE transaction model lets you specify which methods make up a single transaction so all methods in one transaction are treated as a single unit, also called an *atomic action*.
- The J2EE containers are also responsible for managing the life-cycles of components, creating, removing and idling the state of the components.

- Unless bean-managed entity beans are used, the container will also handle data persistence, reading and writing the database for the components.
- The J2EE security model makes it possible to configure a web component or enterprise bean so only authorized users can access the systems resources.

Figure 4-3 depicts the relationships between the J2EE application server, the containers and the components. It also shows the different containers for enterprise components, web components, client components and applet components.

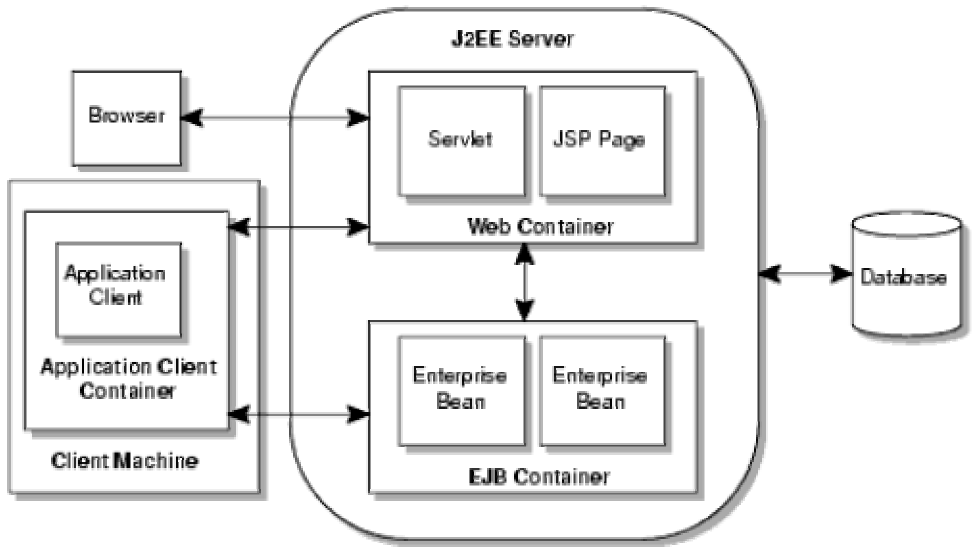


Figure 4-3. The J2EE Application Server holds EJB containers in which the EJB components are deployed [1].

4.5.2 Services Provided by the Application Server

Below follows a presentation of the services provided by the containers inside the J2EE application server.

Naming Services

The client cannot directly instantiate an enterprise bean or call one of its methods, it will have to ask the EJB container within the J2EE server to do that. Java Naming and Directory Service (JNDI) is used by the client to find the home interface of the bean. A naming service binds a name to an object (not necessarily a Java object) and a directory service connects attributes to that object. In this case the client asks the JNDI service for a home interface using its JNDI name, and is given a reference to that home interface in return. This is called a JNDI *lookup*.

Remote Connectivity

When the client has a reference to the home interface of the bean, it can tell the container to invoke the bean's methods via the reference that the client has obtained. This J2EE remote connectivity model uses RMI over IIOP (Internet InterORB Protocol) and is an advanced form of Remote Procedure Calls (RPCs) and very similar to CORBA with messages passed between a RMI client and a RMI server. In fact, IIOP is CORBA-compliant.

Transaction Model

A transaction is an atomic action and should be treated as one single unit. If the whole transaction could not be executed without errors, none of its results should be stored. The J2EE transaction model lets the developer choose transaction mode for methods and then the container will manage so that transactions will be executed as atomic units. In case of an exception, the container will handle the rollback automatically. There are several transaction modes, or attributes, deciding whether the method should require being part of an existing transaction, begin a new transaction or not be part of any transaction and modifications of these.

Component life-cycles

Enterprise components inside an EJB container have different *states* during their lifetime. The containers manage all components' changes between states, called the components *life-cycle*. Different components have different life-cycles; the available states of a stateful session bean differ from those of a stateless session bean or an entity bean.

Stateful session beans have three different states. First, it is *non-existing*. A client then initiates the life-cycle by invoking the `create` method. The EJB container instantiates the bean, bringing it to the *ready* state, waiting for its business methods to be called. Now, two things can happen: If the container needs to free some memory and the session bean hasn't been used for some time, the container can decide to *passivate* the bean by moving it from memory to secondary storage. When the session bean is needed again, it is simple activated into ready state again by restoring it from secondary storage to memory again. The other possibility from the ready state is that the client chooses to terminate the session bean by calling its `remove` method, leaving it in the does not exist-state once again.

Stateless session bean can never be passivated, therefore they have only two states: non-existent and ready.

The life-cycle of an entity bean is different compared to those of a session bean. It starts as non-existing and is moved by the container to the *pooled* state when an instance is created. When in the pooled state, the instance is not associated with any particular EJB object identity, all instances are identical and possible to assign to any client. In the pooled state, an entity bean can also be written to secondary storage by the container. A pooled entity bean can either be created by a client or activated by the container, both actions puts it in the ready state. There are also two paths from the ready state back to the pooled state. First, a client may invoke the `remove` method or the container may passivate it. A client cannot move an entity bean to the non-existing state at the end of its life-cycle, only the container may do that.

As described here, the container manages all the life-cycle changes of Enterprise JavaBeans.

Data persistence

As mentioned earlier, there are two kinds of data persistence management strategies for entity beans, there is the bean-managed data persistence where the developer is responsible for data persistence and there is container-managed data persistence.

When a container-managed entity bean is deployed in a container, the container is responsible for all database access. SQL statements are generated automatically by the container, and are hidden from the developer. For example, when a client creates an entity bean, the container automatically generates a SQL `insert` statement. The code of the entity bean includes no SQL calls. As a result, the entity bean is completely database independent and portable though out all J2EE servers.

Security and Authentication

The J2EE security model handles two vital concepts, *authentication* and *authorization*. Authentication is the mechanism that verifies a user's identity. Mutual authentication will clarify the identification of both parties, the client and the component. A component can also allow connections with no authentication (anonymous login). A client can be identified as belonging to three different J2EE authentication concepts; *user*, *realm* and *group*.

A J2EE user is similar to an operating system user where one user represents one person. A realm is a collection of users that are controlled by the same authentication policy. A group is a category of users, classified by common traits such as job title or customer profile.

Authorization is the mechanism that gives some users access to certain components and some users are not allowed access. Users, realms and groups can be given *roles*. The developer can declare *method permission* when deploying a method which determines which roles are allowed to invoke that method.

Clustering

Clustering is a technique that provides an infrastructure with high availability and scalability. A *cluster* is a group of Application Servers, usually running on different physical machines that transparently run the J2EE application as if they were a single application server.

Instead of having one machine with one application server, serving thousands of concurrent clients, multiple application servers share the workload. If a server goes down, there are others to handle incoming client calls (high availability). A group of servers can definitely support more users than a single one (increased scalability). And developers do not have to write specific code to do this; it is taken care of by the application server. [31]

Resource pooling

To *pool* a resource is to reuse a resource. The J2EE containers support the reuse of sockets and database connections, this is called *connection pooling*. Communication sockets and database connections are kept in a pool. When a client needs a socket, the container will assign a free socket from the pool to the client. When the client has finished using the socket, it is returned to the pool, ready to be used by another client. This service increases the scalability of a system. [32]

Connection pooling, just like many other container services, is taken care of the container behind the scenes, meaning that the bean code is oblivious to it.

Other Services

The J2EE container provides yet more services not listed here as well as access to the J2EE platform APIs (see chapter 4. Enterprise Java Technologies (J2EE)).

All of the examples above are available in EJB version 1.1. There is a new version called 2.0 [58] on the way (autumn 2001), which handles yet more services like relationship management in entity beans for example.

4.6 J2EE Summary

There are several advantages with the J2EE technology, but there are also drawbacks. J2EE is becoming more and more one of the most talked about technologies for developing robust enterprise applications. It was introduced by Sun in 1999 and now there are many rivaling application servers. The application server contains the J2EE containers where you put your enterprise beans, the Java business components. The most well-known application servers are BEA WebLogic [52], IBM WebSphere [54], Oracle Application Server [53] and Borland AppServer [51] (formerly known as Inprise AppServer).

The main advantage of using the J2EE technology is shortened development time [1]. Many time-consuming and complicated details of developing multi-tiered applications are hidden from the developer and others are made easier. The main drawback can be performance. But it may not have to be. Developers can either choose to let the container hide all database access from him with the risk of it being slow, or code the database access calls himself, with more control over it.

The J2EE technology is extensive; ranging from database access through JDBC (Java DataBase Connectivity), remote procedure calls with RMI (Remote Method Invocation), naming services through JNDI (Java Naming and Directory Interface), different deployable components like Session Beans (both stateful and stateless), Entity Beans (container-managed and bean-managed) and dynamic web pages generated with Java Servlets and Java Server Pages. J2EE also uses XML (eXtensible Markup Language) quite extensively.

5. Design and Development of Prototypes

This chapter describes the developed prototypes to evaluate the two architectures and will present the design, architecture and behavior of the prototypes. The chapter ends with a brief introduction to some of the software tools used to create these prototypes.

The intention was to implement a specific part of one of Prohunt's products as a prototype. This part was going to be implemented on both platforms and then evaluation tests were going to be performed on both prototypes. But with the bankruptcy, this plan had to be abandoned. Instead, two prototypes were developed, performing the same actions, or server requests. These requests are designed to be easily compared between platforms.

5.1 General Architecture

The basic architectures of the prototypes are displayed in figures 5-1 and 5-2. Figure 5-1 depicts the PL/SQL prototype and figure 5-2 depicts the J2EE prototype.

5.1.1 PL/SQL Prototype Overview

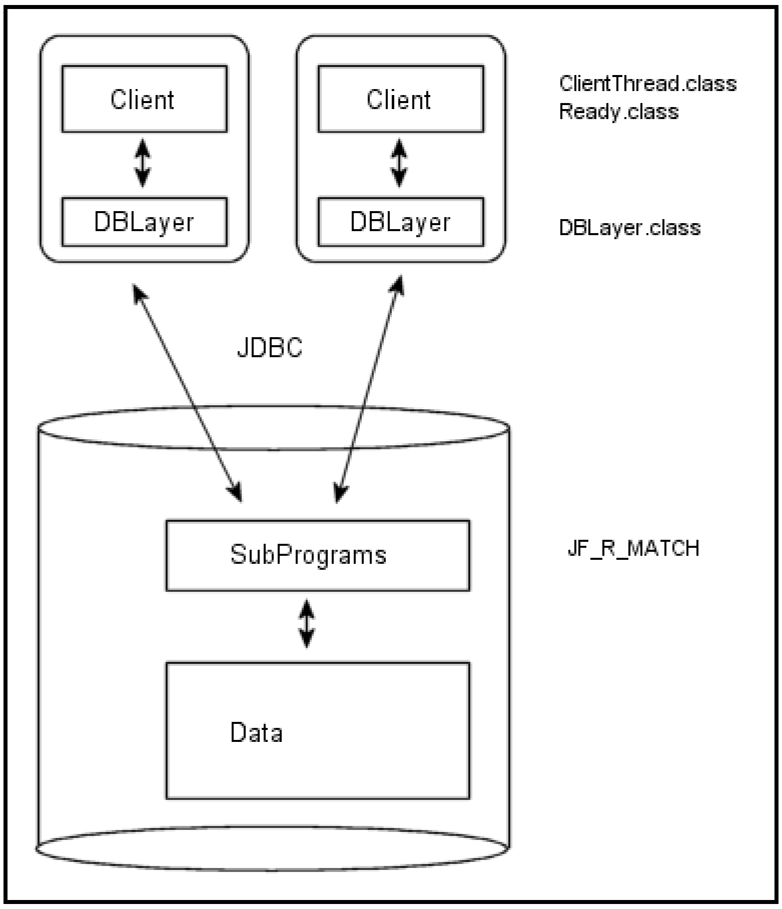


Figure 5-1. Basic architecture of the PL/SQL evaluation prototype.

The client application was developed in Java for both prototypes. The reason for this was that the timing in the evaluation experiments described in chapter 6, is performed on the client side, and to be sure that the response times (see chapter 6) are calculated the same way, the clients were made as similar as possible. Unfortunately, this meant that an extra layer, the DBLayer class described in section 2.2.4 Specification and Architecture, was needed on the client side in the PL/SQL prototype.

The server side architecture is very simple in the PL/SQL case. Both subprograms and data are stored in the Oracle database. The subprograms reside in package `JFP_R_MATCH` and consist of only three subprograms, one function and two procedures. The connection between client and server is handled by Oracle's thin JDBC driver, version 1.11. The subprograms are invoked and executed in the database and has fast access to the data.

5.1.2 J2EE Prototype Overview

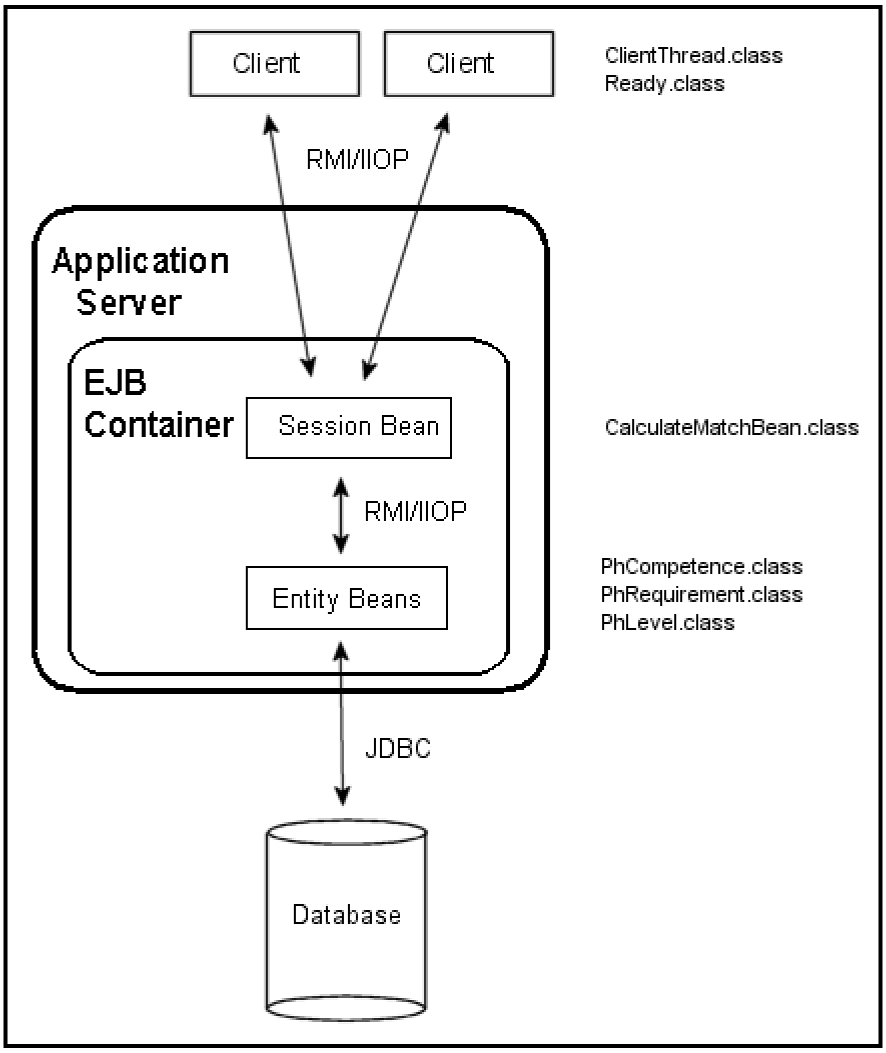


Figure 5-2. Overview of the J2EE prototype architecture.

The architecture overview in figure 5-2 does not seem much more complicated than the PL/SQL figure, but the inside of the business objects (EJBs) are in fact more complicated and powerful. There are no differences on the client side except when it comes to invoking the procedures on the server side. The J2EE client doesn't need the DBLayer; instead it has to obtain a reference to the home interface of the session bean by using a JNDI lookup, and then create a reference to the session bean itself. When this is done the client is ready to call methods on the server. RMI over IIOP is used as protocol here, unlike the PL/SQL prototype which uses JDBC.

The Application Server has different containers for different applications/modules. The container of the prototype holds four enterprise beans; one session bean and three entity beans. The clients invoke the appropriate method on the session bean which in turn calls methods on different entity beans. Session beans also use RMI over IIOP to access entity beans. It is the entity beans that access the database using Oracle's thin JDBC driver 1.11.

5.2 The Prototypes

An important part of the thesis was to implement two software applications, one using PL/SQL and one the Java 2 Platform, Enterprise Edition. These applications are identical in function but different in implementation. One reason for the implementation of these experimental *prototypes* was to perform several evaluation experiments, aiming to compare the two approaches. The development of these prototypes was also in itself a step of the evaluation since I learned differences in complexity, usability and such features of the two architecture platforms.

5.2.1 Server Calls – "Requests"

One of the main objectives of the thesis was to compare the performance and scalability of the platforms. The evaluation method and experiments are described more extensively in chapter 6.1 Evaluation method, but something has to be said about *request types* at this time. To measure performance and scalability, we measure how many transactions, or requests, that can be performed under a certain time frame and under a certain condition. In the evaluation prototypes, the same kinds of server requests have been implemented for both platforms, to be able to compare them. These requests are of different types depending on how much computations has to be performed on the server and the amount of data that is returned to the requesting client. The requests are of three types plus a hybrid; light weight requests, middle weight requests, heavy weight requests and a mix of them all called mixed weight requests.

5.2.2 PL/SQL Application

The PL/SQL prototype consists of the following classes and subprograms:

The client application consists of three Java classes;

- `TestAppPLSQL.class` is the evaluation class that initializes a specified number of clients (ClientThread instances) and measures the time needed by the architecture to perform the chosen requests. This class is described in chapter 6.1.3 Client Applications Used For Experiments.
- `ClientThread.class`. The main class of the client application, which connects to the server and sends requests to the server. The class uses DBLayer to call PL/SQL subprograms and is also detailed in chapter 6.1.3 Client Applications Used For Experiments.
- `Ready.class` is a helper class that notifies TestAppPLSQL when a client has received a response from the server and finished executing. It is basically just a structure that counts how many clients has returned ok and how many has returned with an exception. When the sum of those variables is equal to the number of initialized clients, the evaluation run is completed.

- `DBLayer.class` is another helper class needed by the Java client to connect to the Oracle database and invoke PL/SQL subprograms. Described in chapter 2.2.4 Specification and Architecture.

The business logic on the server side is implemented with only one PL/SQL package called `JFP_R_MATCH` which contains four subprograms, each one corresponding to each of the four requests types:

- `Light_Request`
- `Middle_Request`
- `Heavy_Request`
- `Mixed_Requests`

The package header seen in figure 5-3 includes the user defined data type `t_curref` and declarations for the three subprograms.

```
package JFP_R_MATCH
is
   type t_curRef   is ref cursor;

   -- Start of thesis code
   function light_request(
      input1 in number,
      input2 in number
   ) return number;

   procedure middle_request(
      input1 in number,
      input2 in number,
      output out t_curRef
   );

   procedure heavy_request(
      input1 in number,
      input2 in number,
      output out t_curRef
   );
   -- End of thesis code
end;
```

Figure 5-3. The package header must include declarations of all functions and procedures that are going to be accessible from outside of the package. Internal procedures do not have to be declared.

As seen in figure 5-3, there is no subprogram for the mixed requests, it is implemented in the evaluation client, and uses the other three server request types. The subprograms themselves are located in the package body (figure 5-4). The contents of the different requests are described in detail in chapter 6.1.2 Experiment Suite.

```
Package Body JFP_R_MATCH
is
   type t_tabId is table of number index by binary_integer;
```

```

-- first the light weight request
function light_request(
   input1 in number,
   input2 in number
) return number
is
   res     number;
begin
   SELECT COUNT(*) INTO res FROM ph_competence;

   return res;
end;

-- middle weight request
procedure middle_request(
   input1 in number,
   input2 in number,
   output out t_curRef
)
is
begin
   open output for
      SELECT com_id FROM ph_competence;
end;

-- heavy weight request
procedure heavy_request(
   input1 in number,
   input2 in number,
   output out t_curRef
)
is
   l_curRef    t_curRef;
   i           number;
   l_tabIndId  t_tabId;
   res         number;
begin
   open l_curRef for
      SELECT ind_id FROM ph_xref_ind_com_lvl WHERE com_id < 10;
      i := 1;
      loop
         fetch l_curRef into l_tabIndId(i);
         exit when l_curRef%NOTFOUND; --exit loop when empty
         i := i + 1;
      end loop;
   close l_curRef;

   i := l_tabIndId.first;
   loop
      exit when i is null;
      res := l_tabIndId(i) * 10 / 22;
      i := l_tabIndId.next(i);
   end loop;

```

```
      open output for
         SELECT ind_id FROM ph_individual WHERE ind_id < 10000 AND
            ind_id > 10;
   end;
end;
```

Figure 5-4. The code of the PL/SQL subprograms that contains the requests.

5.2.3 J2EE Application

The underlying architecture of the J2EE prototype is more complicated than the architecture of the PL/SQL prototype as stated earlier. However, the Java code is not so complicated, the complex behavior of name lookups, managing transactions and in between layer calls is all handled automatically.

Client modules:

- `TestAppJava.class.` Almost identical to the corresponding class in the PL/SQL prototype. The class initializes clients and measures the execution time.
- `ClientThread.class`. Behaves similar to the PL/SQL class with the same name. The difference is that this client obtains a reference to a session bean (called `CalculateMatch`, see figure 5-5) and invokes the business method that corresponds to the wanted request.
- `Ready.class`. Exactly the same class as in the PL/SQL prototype.

Server modules:

- `CalculateMatch` session bean.
- `LightWeight` entity bean.
- `MiddleWeight` entity bean.
- `HeavyWeight` entity bean.

These classes were compiled into a JAR-file called EJBPrototype, which subsequently was deployed in the container EJBContainer on the J2EE server.

Before the client can interact with a session bean on the server, the client has to have a reference to the session bean. Figure 5-5 shows how this is accomplished by using JNDI (Java Naming and Directory Service) to do a JNDI lookup. This gives access to the beans home interface, which is used to call the `create()` method of the bean. `Create()` returns a reference to an instance of the bean on the server, ready to accept calls.

```
try {
   javax.naming.Context ctx = new
      javax.naming.InitialContext();
   CalculateMatchHome home = (CalculateMatchHome)
      javax.rmi.PortableRemoteObject.narrow(ctx.lookup(
      "CalculateMatch"), CalculateMatchHome.class);
   sbean = home.create();
} catch (Exception e) {
   System.out.println("Error trying to get session bean");
   e.printStackTrace();
}
```
Figure 5-5. This Java code snippet shows how to obtain a reference to a session bean. This code is located in the initialization section, init() method, of the client.

5.2.4 Summary

Two prototypes were developed. They both perform similar requests and return the same end result. But the prototypes reach this result in very different ways. The PL/SQL prototype was faster and easier to develop. It is only a matter of writing an appropriate subprogram and calling it from the client. In the J2EE prototype, JavaBeans have to be written, assigned JNDI names and deployed into containers together with deployment descriptors. However, Java is a more capable programming language than PL/SQL, with more functions and capabilities. It is also a more living language that evolves and grows, and there are massive resources on the Internet.

5.3 Tools used

Below, the most important software tools used when developing the prototypes and conducting the evaluation experiments are described.

5.3.1 Oracle

Oracle is one of the most frequently used Database Management Systems (DBMS) in the world. For time-critical enterprise solutions, Oracle is the DBMS to use. Prohunt uses Oracle8 as DBMS in all their software products. As stated earlier, PL/SQL is a Oracle-specific programming language.

5.3.2 Borland Application Server

The Borland Application Server was formerly known as the Inprise Application Server (IAS) before Borland decided to change back to their old name. Borland's Application Server is still known as IAS though. The current version 4.51 (summer 2001) is tightly integrated with Borland JBuilder Enterprise. For example, it is possible to automatically deploy the enterprise beans in the EJB Containers of IAS from JBuilder, using the J2EE Deployment Wizard. This is very fast and easy compared to manual deployment which includes manual editing of the EJB Deployment Descriptor, an XML file I have described briefly earlier.

5.3.3 Borland JBuilder Enterprise

I have been using version 4 of Borland's development tool for Java. Borland JBuilder is a sophisticated Integrated Development Environment (IDE) that includes many quick wizards and templates for developing, packaging and deploying Enterprise JavaBeans.

5.3.4 SQL Navigator from Quest Software

SQL Navigator is a nice application for manipulating the contents of an Oracle database, both data and code. Navigator connects to the Oracle server and provides a GUI to the content of the database.

6. Evaluation and Results

Evaluating the two architectures wasn't an easy task. The experiments had to be re-run several times, before it could be considered a fair evaluation. Even so, a precise comparison is hard to achieve, due to many factors. Efforts have been made to make this evaluation as fair as possible.

6.1 Evaluation method

The performance of an application depends on many different factors, including raw speed of the machine/machines used (processor speed, number of processors, OS used etc), amount of primary memory (RAM), how well the database is configured with indexes and cache, how populated the database is, the number of concurrent requests to the database (called *workload* of database) and of what type those requests are.

To be able to compare two applications, or in this case architectures, the evaluator needs to keep most of these properties constant while varying the properties that are being compared. A serious comparison must include several evaluation steps (called evaluation experiments), in which critical properties are changed.

All experiments were run on the same computers. The database was also the same and it was already very well populated with indexes for all essential tables, the database should not be a bottleneck in the evaluation. These experiments were then performed with both the PL/SQL evaluation prototype and the J2EE evaluation prototype. The properties that vary throughout the experiments are the types of requests and the number of concurrent users running the application and thus accessing the database at the same time.

6.1.1 Request Types

In an evaluation, it is important to decide how and what to measure. One can measure response time (the time between a request is sent until a response is received) or throughput (requests per second) [44]. These measures depend heavily on the type of requests. The type of a request is measured by its *weight*. A request can be light or heavy weight depending on how much resource it requires from the server or the client. A light weight request will take a short time for the server to process and the response is small counted in bytes. A heavy weight request can contain several queries to the database; some calculations and the response itself can be quite large.

Performance is often measured as transactions per second or response time for a transaction (time that elapses from the beginning of a transaction until it is completed). In this evaluation, each request makes up one transaction, so request per second is the same as transaction per second in this evaluation. In other more complicated systems, one single transaction often consists of several requests to the database. Some may be light weight and some heavy weight, depending on the transaction.

This evaluation measures the response time of the requests. The evaluation is divided into several experiments. Each experiment except the Mixed Weights Requests was conducted three times on each architecture and the best result (lowest response time) was used in the graphs and tables. For the Mixed Weights Request the average response time of the three experiment runs were calculated and used in the graphs and tables. The reason for this is that the Mixed Weights Request depend so much on randomness that one run can vary greatly from another. The results of the Mixed Weights Request should not be compared between platform for this reason. It was included into the evaluation to see how well the platforms responded to varying requests that wouldn't be cached by the machines, as 100 exactly alike requests would be.

The complete result series are available in appendix C.

6.1.2 Experiment Suite

To be able to see differences in behavior between the architectures, it is necessary to run more tests than one. An application or architecture might be better suited to handle server calculations than sending large amounts of data between tiers or vice versa for example. For this reason, an *experiment suite* containing four different evaluation experiments was created:

1. Light Weight Request
2. Middle Weight Request
3. Heavy Weight Request
4. Mixed Weights Request

The experiment suite was first executed on each platform using only one client, these are the *performance experiments*. The performance experiments measures the response times on the two platforms without any concurrent users. It is the following *scalability experiments* that measure how the two platforms perform when the number of users grows.

The plan was to evaluate scalability using the following series of concurrent users: 1, 10, 50, 100, 500, 1000, … until the application wouldn't take any more. In practice, this turned out to be impossible. The Java Virtual Machine JVM running the evaluation prototype crashed with `OutOfMemory`-exceptions before reaching these counts. These problems are described in chapter 6.1.8 Problems during Experiments.

It can often be hard to decide the weight of a request. A request can be heavy on the server (lots of calculations and processing) but light in the response (return only a single integer variable for example). Or it can be light on the server and heavy on the client. I have tried to keep this experiment suite balanced, with request weights that are compatible with the names of the request. A more technical description of the four request types follows below.

Light Weight Request

The Light Weight Request will perform a SQL `"count(*)"`query on a table consisting of 4403 rows. The result ("4403") is returned as a number (8 bytes in Oracle) to the middle-tier and is then sent to the client.

Middle Weight Request

This request performs a SQL `"SELECT com_id FROM ph_competence"`. This table consists of 4403 rows. Com_id is of type number (8 bytes) so the size of the returned result is 4403*8 = 35224 bytes, about 35kb. The result is first returned to middle-tier and then to the client.

Heavy Weight Request

The Heavy Weight Request is a little more complex than the first two, it performs several operations. First, the query “SELECT ind_id FROM ph_xref_ind_com_lvl where com_id < 10” is executed. This table contains 100.176 rows of which 2344 matches the query. The result (2344*8 = 18752 bytes) is returned to the middle-tier where the ids are stored in an array. Next, we loop through each of these ids and perform some simple calculations on them. Then another search is executed which returns 14.930*8 = 119.400 bytes. This result is also returned back to the client.

Mixed Weights Request

The Mixed Weights Requests are intended to simulate real-life load more accurately. Normally 50 users don’t send the exact same request at the same time and then disconnect. This request uses the light, middle and heavy weight requests to create a mixture of requests. It will execute ten requests of random weights with a random pause between requests. Statistically the distribution between the different sorts of requests is 60% Light Weight Requests, 30% Middle Weight Requests and 10% Heavy Weight Requests.

The randomness should also lessen the chance that the requests are cached by the database manager and the middle-tier. Caching of the Heavy Weight Request could lower the response time when 100 requests are executed simultaneously. This would hopefully not be the case with the Mixed Weights Request.

Table 6-1 illustrates the different types of requests.

Request Weight	**Description**	**Request size (PL/SQL)**	**Request size (J2EE)**	**Response size (kb)**
Light	Count(*)	≈ 60 bytes		8 bytes
Middle	Select all id’s	≈ 60 bytes		35 kb
Heavy	Both calculations and a big result set	≈ 60 bytes		119kb
Mixed	Random	≈ 600 bytes		Differs.

Table 6-1. The size of requests and more important, the size of the responses.

The anatomy and size of the requests depends on the platform. In PL/SQL (first values, the request to the server is really just a PL/SQL subprogram call, it is the name of the subprogram and value of the parameters that is being sent from client to server encapsulated in a JDBC call. The string (about 60 bytes) sent from the client looks like this:

```
“? = Call JFP_R_MATCH.light_request(input = 0, input2 = 1)”
```

In J2EE, the client request is an invocation of a session bean method, i.e. a Java RMI call encapsulated inside the IIOP protocol. In reality this later becomes a new request between the J2EE server and the Oracle database, containing a SQL statement.

6.1.3 Client Applications Used For Experiments

The evaluation itself was performed using an *Evaluation Client Application*. This client application is written in Java for both platform evaluations (it is not the Evaluation Client Application that is supposed to be performance evaluated, it is the underlying server architecture. The Evaluation Client Application has two main classes, `TestApplication` and `ClientThread`. `TestApplication` contains `main()` and takes two application parameters, number of users to simulate (`noofusers`) and what type of requests to execute (`request_type`). `runTests()` is the method that executes the experiments itself, see figure 6-2.

First, the desired number of clients (`ClientThread`) are created (row 9) and put in a `vector`. Then the start time is noted (row 16) before going into a loop that calls each clients `start()`-method (row 20) which itself will invoke the `run()`-method of the `ClientThread` class described below. After the loop the program is waiting for all clients to receive a response from the server. The application has one `ok_count` and one `error_count`. Each client will update one of these variables according to result of the call to the server. When the `ok_count` plus the `except_count` is equal to the number of started clients, each client will have received its response and the while-loop at row 32 is ended. The end time is noted and the response time returned.

```
 1 public long runTests() {
 2   Vector clients = new Vector(noofclients);
 3   long startTime;
 4   long endTime;
 5
 6   // Create n clients
 7   System.out.println("Create and initialize clients...");
 8   for (int i=0; i < noofusers; i++) {
 9      clients.addElement(new
10        ClientThread(ready,request_type));
11   }
12
13   System.out.println("Initialization of clients took " +
14      (endTime - startTime));
15   System.out.println("Start the test...");
16   startTime = System.currentTimeMillis();
17
18   // Start each client
19   for (int i=0; i < noofusers; i++) {
20      ((ClientThread) clients.elementAt(i)).start();
21      try{
22         this.sleep(sleepTime);
23      } catch (InterruptedException e) {
24         System.out.println("Error (this.sleep): ");
25         e.printStackTrace();
26      }
27   }
28
```

```
  // Wait until all clients have received a response,
  // both the ones who returned OK and those with
  // exceptions
  while (!((ready.ok_count + ready.except_count)==
        noofusers)){
     try{
        this.sleep(50);
     } catch (InterruptedException e) {
        System.out.println("Error (this.sleep): ");
        e.printStackTrace();
     }
  }
  endTime = System.currentTimeMillis();

  return endTime - startTime;
}
```
Figure 6-2. The runTests()-method of class TestApplication. Initializes, starts and times all clients.

When the experiment suite was run the first times, the response times were very high, even for one user and the light request. It was found that the creation and initialization (setting up JDBC connection/retrieving a reference to the session bean) of the threads that simulate concurrent clients (the `ClientThread` class) took a lot of time. Since this has nothing to do with the response time of the request to the server, the start of the timing was moved to after the creation and initialization of these threads as seen in the code.

The `TestApplication` class does not differ for the two platforms. The `ClientThread` class on the other hand does, since it is responsible for setting up a connection with the server. In the PL/SQL version, the `ClientThread` class is where each client connects to the Oracle Server and calls the appropriate PL/SQL subprogram. The `run()` method listed in figure 6-3 is invoked when `ClientThread.start()` is called.

The instance variable `request_type` (initialized when the class was instantiated), determines which PL/SQL procedure should be called starting on row 8. For light, middle and heavy weight request, the right procedure is called using the helper class DbLayer (see chapter 2.2.4 Specification and Architecture) with previously determined arguments, and when the request has been processed, the result is returned to the client. In the mixed weights request, it is a little more complicated. Ten requests of random weight are performed, with random pause in-between calls. Finally the variable `ready.ok_count` is increased (row 60) if the response was successfully received, but if an exception was thrown, the variable `ready.except_count` is increased (row 68).

```
1 // PL/SQL-version of ClientThread.run()
2 public void run() {
3    String res;
4    Vector vres = null;
5
6    // Start the DB call
```

```
  try {
     switch (request_type) {
     case 1:
        // Light weight request
        res = DbLayer.callFunction(
           "JFP_R_MATCH.light_request", args);
        break;
     case 2:
        // Middle weight request
        vres = DbLayer.callProcedure(
           "JFP_R_MATCH.middle_request", args);
        break;
     case 3:
        // Heavy weight request
        vres = DbLayer.callProcedure(
           "JFP_R_MATCH.heavy_request", args);
        break;
     case 4:
        // Mixed requests
        int random_number;

        // Execute 10 random requests.
        // 60% - light weight requests
        // 30% - middle weight requests
        // 10% - heavy weight requests
        for(int i=0; i < 10; i++) {
           // Random number between 0-9
           random_number = random.nextInt(10);

           if (random_number < 6)
              res = DbLayer.callFunction(
                 "JFP_R_MATCH.light_request", args);
           else if (random_number < 9)
              vres = DbLayer.callProcedure(
                 "JFP_R_MATCH.middle_request", args);
           else
              vres = DbLayer.callProcedure(
                 "JFP_R_MATCH.heavy_request", args);

           // Random number between 10-110
           random_number = 10 + random.nextInt(101);

           // Wait for 10-110 ms until next request
           try {
              sleep(random_number);
           } catch (InterruptedException e) {
              System.out.println("Sleep error: ");
```

```
               e.printStackTrace();
            }
         }
         break;
      }
      // Client finished OK, increase count
      ready.ok_count++;
   } catch (Exception e) {
      if (ready.except_count == 0) {
         System.out.println("PL/SQL client generated error
            while accesing database.");
         e.printStackTrace();
      }
      // Client finished with exceptions
      ready.except_count++;
   }
}
```

Figure 6-3. The run()-method of PL/SQL's ClientThread class. Performs the request that has been choosen.

The `run()`-method in the J2EE version of `ClientThread` is very similar to the PL/SQL version as seen in figure 6-4. The only difference is that instead of calling the database directly, the client calls the session bean containing the code of the requests. Earlier in the code, when the client was created and initialized, the client queried the container using a JNDI lookup, and obtained a reference to the session bean through the home interface of the bean.

Now that the client has a reference to the session bean, it is easy to invoke the business methods (different request types) of the session bean (rows 9, 13, 17 and 32). The rest of the method works exactly as in the PL/SQL case.

```
 1 public void run() {
 2    int res = 0;
 3    Vector vres = new Vector();
 4
 5    try {
 6       switch (request_type) {
 7       case 1:
 8          // Light weight request called
 9          res = sbean.light_request(arg1, arg2);
10          break;
11       case 2:
12          // Middle weight request called
13          vres = sbean.middle_request(arg1, arg2);
14          break;
15       case 3:
16          // Heavy weight request called
17          vres = sbean.heavy_request(arg1, arg2);
18          break;
```

```
      case 4:
         // Mixed requests
         int random_number;

         // Execute 10 random requests.
         // 60% - light weight requests
         // 30% - middle weight requests
         // 10% - heavy weight requests
         for(int i=0; i < 10; i++) {
            // Random number between 0-9
            random_number = random.nextInt(10);

            if (random_number < 6)
               res = sbean.light_request(arg1, arg2);
            else if (random_number < 9)
               vres = sbean.middle_request(arg1, arg2);
            else
               vres = sbean.heavy_request(arg1, arg2);

            // Random number between 10-110
            random_number = 10 + random.nextInt(101);

            // Wait for 10-110 ms until next request
            try {
               sleep(random_number);
            } catch (InterruptedException e) {
               System.out.println("Error (sleep): ");
               e.printStackTrace();
            }
         }
         break;
      }
      // Client finished OK, update finished count
      ready.ok_count++;
   }
   catch (Exception e) {
      System.out.print("J2EE client generated error while
         accesing database.");
      e.printStackTrace();
      // Client finished with exceptions
      ready.except_count++;
   }
}
```
Figure 6-4. The run()-method of J2EE's ClientThread class.

The sbean variable is the reference to the session bean on the application server.

6.1.4 Java Experiments Second Run

These evaluation experiments were conducted at two separate occasions. Only two computers where used the first time for both PL/SQL and J2EE (called J2EE from now on) experiments. Computer 1 was used as dedicated database server in all experiments. This means that Computer 2 was used as both Application Server *and* for running the client application. In the performance experiments, with only one concurrent client, this was not a problem. But in the scalability experiments this meant that the machine was running around 100 client threads concurrently with the application server, a *very* heavy load. The Java Runtime Environment (JRE) also consumes lots of memory. All of this makes the results of the first J2EE run (J2EE 1) *unbalanced* compared to the PL/SQL experiments where the server and clients ran at different machines. The extra heavy load affected the results, both client threads and (more noticeably for the evaluation) the application server suffered.

A new J2EE experiment run (J2EE 2) was conducted for this reason, using three machines and thus separating the clients from the servers. This would turn out to improve the results of the scalability experiments on the J2EE platform, and more concurrent clients were supported. The results from J2EE 1 should be taken with a pinch of salt, but have been included for comparison reasons and to emphasize the impact of uneven benchmarks.

6.1.5 Hardware and Software

These evaluation experiments were performed using first two, and then three computers. The setup of the machines as well as the roles the machines had during the different experiment run can be seen in table 6-5 below.

	Computer 1	*Computer 2*	*Computer 3*
PL/SQL Role	Database Server	Running clients	Not used
J2EE 1 Role	Database Server	Clients + Application Server	Not used
J2EE 2 Role	Database Server	Running clients	Application Server
Machine	HP Vectra VL	HP Kayak XA	HP Kayak
Processor	Pentium III-500 MHz	Pentium III-550 MHz	Pentium III-733 MHz
RAM	128 Mb	256 Mb	256 Mb
OS	MS Windows NT4 Workstation Service Pack 4	Microsoft Windows 2000	Microsoft Windows 2000
Database	Oracle8 version 8.1.5	-	-
Application Server	-	Borland Application Server 4.51	Borland Application Server 4.51

Table 6-5: The configuration of the machines involved in the evaluation experiments.

Computer 1 was used as database server in all experiments. Computer 2 ran the clients, except in J2EE 1 were it ran both clients and application server. Computer 3 was not used at all until the last run (J2EE 2), were it functioned as application server. The machines communicate through a 10 Mbps LAN without any interfering traffic at the evaluation time.

6.1.6 Performance Experiments

In all results tables below, the best (minimum time) result is used for all experiments except the Mixed Requests experiment, where the average time is calculated instead.

I started with PL/SQL to see what it had to offer, see table 6-6 for the results Remember that the performance experiments only simulates one user.

PL/SQL

Experiment	**Run1**	**Run2**	**Run3**	**Min/Average**
Light Request	891	911	891	891
Middle Request	1562	1552	1573	1552
Heavy Request	2294	2323	2323	2294
Mixed Requests	7541	5007	5227	Avg: 5925

Table 6-6. The results of the performance experiments on the PL/SQL platform. Each experiment was run three times and an average or minimum was calculated. All results are response times measured in milliseconds.

After PL/SQL, it was time for J2EE. First, the experiments were run with the Application Server and the Evaluation Client Application residing on the same machine. Table 6-7 presents the results.

J2EE 1

Experiment	**Run1**	**Run2**	**Run3**	**Min/Average**
Light Request	50	50	60	50
Middle Request	3184	3175	3595	3175
Heavy Request	6980	6880	6950	6880
Mixed Requests	27760	16965	30625	Avg: 25117

Table 6-7. These are the results of the first performance experiments on the J2EE platform. The experiments were conducted the same way as for the PL/SQL platform. All results are response time in milliseconds.

Later, the J2EE experiments were rerun with the Evaluation Client Application separated from the Application Server. Below are the results from that run.

J2EE 2

Experiment	Run1	Run2	Run3	Min/Average
Light Request	50	50	60	50
Middle Request	3125	3090	3080	3080
Heavy Request	6910	6810	6820	6810
Mixed Requests	15860	21140	24665	Avg: 20555

Table 6-8. These are the results of the second performance experiments on the J2EE platform. This time, three machines where used, separating the application server from the clients.

Comparison

As can be see in figure 6-9, PL/SQL has a clear advantage over J2EE in all request types except the light weight requests. With only one concurrent user, the difference between J2EE 1 and J2EE 2 is small.

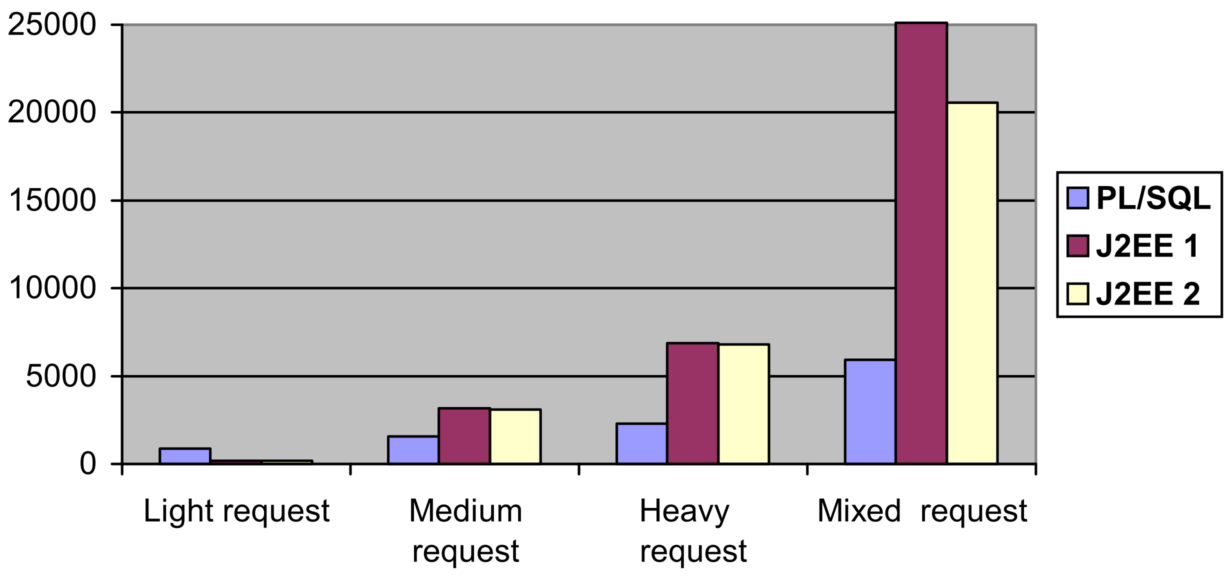


Figure 6-9. Results of the performance experiments. The times used in the plot are the minimum response time, except for the mixed request, where the averages are used0.

6.1.7 Scalability Experiments

After the performance experiments were done, the scalability experiments were conducted. The requests are exactly the same as in the performance experiments but now the evaluation prototype will simulate up to 1.000 concurrent users (or as many as the platform can handle). One thing that should be remembered is that the number of users in the experiments doesn't really reflect the behavior of the systems in reality with that number of users. In reality, if a system has 50 active users, only a few of them send requests at the same time. The evaluation prototype sends all requests (almost) simultaneously. It is generally considered that ten concurrent users in a simulation experiment counts as 30-40 persons in real life [44].

My opinion, based on how users interact with the products in Prohunt's ICM platform, is that this number is even higher in Prohunt's case. Users often have to think about their decisions before continuing with the programs, which creates long pauses between server calls. Therefore, it is fair to guess that 10 concurrent users in the experiments would compare to about 50 in real life

Only the minimum/average results will be shown in the following tables and plots, because of the quantity of experiments. Complete experiment results are available in Appendix C. Some experiments were not able to return a result, those results are marked as not available (N/A). The reason for this is described in chapter 6.1.8 Problems during Experiments.

Table 6-10 contains all results from all scalability experiments conducted on the PL/SQL platform.

PL/SQL

Users	**Light Request**	**Middle Request**	**Heavy Request**	**Mixed Requests**
1	881	1522	2304	4466
10	1643	3956	8342	15379
50	5758	15482	38856	64032
100	10815	32327	76820	204958
200	20759	69380	151708	N/A
300	112221	N/A	N/A	N/A

Table 6-10. The scalability experiments on the PL/SQL platform. The best results are used, except for the Mixed Requests where the average response time is calculated. Unsuccessful experiments are shown as N/A.

Below are the results of the scalability experiments using the J2EE prototype for the first time (J2EE 1). Some experiments were never performed with 5 users, they are marked with '-'.

J2EE 1

Users	**Light Request**	**Middle Request**	**Heavy Request**	**Mixed Requests**
1	50	3045	6970	17028
5	-	22142	-	-
10	160	55179	70271	N/A
50	711	N/A	N/A	N/A
100	10024	N/A	N/A	N/A
200	20078	N/A	N/A	N/A

Table 6-11. The first scalability experiments on the J2EE platform. All results show response time in milliseconds. Experiments that has not been performed are marked as '-'.

The last experiment run uses three computers on the J2EE prototype.

J2EE 2

Users	**Light Request**	**Middle Request**	**Heavy Request**	**Mixed Requests**
1	50	3022	6720	15802
5	-	-	25112	36874
10	155	12795	48868	86953
50	699	24685	121798	N/A
100	8221	86111	305210	N/A
200	17215	226978	N/A	N/A
300	95152	N/A	N/A	N/A

Table 6-12. The second scalability experiments on the J2EE platform.

Comparison

All scalability experiments have been performed. We compare the results for each request type, starting with the light weight request. Figure 6-13 shows the comparison based on the light request. The colored bars for both J2EE experiments have been enlarged for one and ten users; otherwise they wouldn't be seen at all. No result is available from J2EE 1 at 300 users. PL/SQL falls behind in all results in the light weight request experiments.

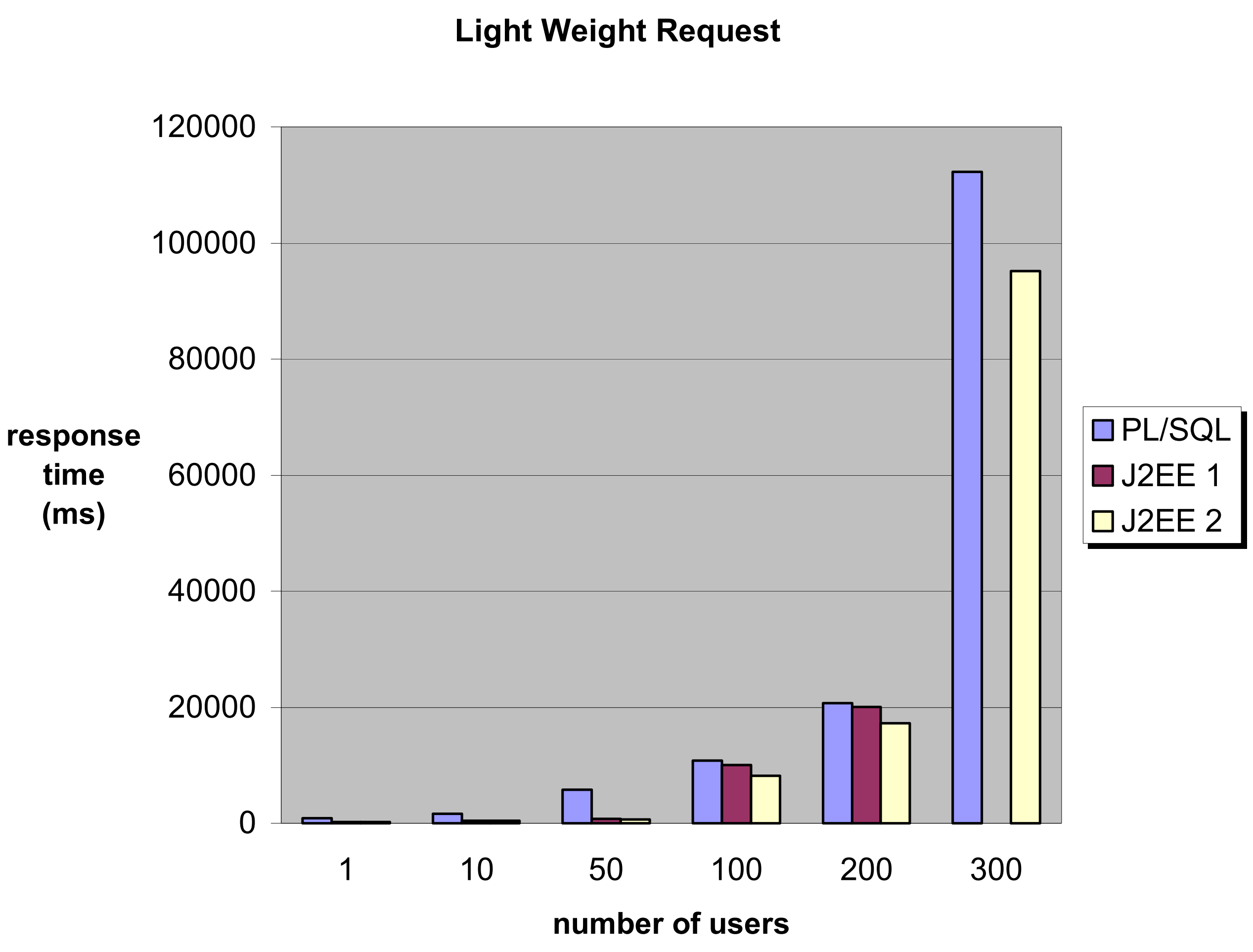


Figure 6-13. Comparison between platforms for the light weight request. J2EE1 is unable to deliver a result for 300 concurrent users.

With the requests that follow the light weight request it becomes apparent that it was untenable to have application server and evaluation prototype situated on the same computer. J2EE cannot deliver a result with more than ten concurrent users, due to lack of primary memory. When the application server has a dedicated machine (J2EE 2), the results improve significant as seen in figure 6-14. But already now, PL/SQL is displaying much better results than any of the J2EE experiment runs. PL/SQL shows a linear increase on response time, whereas the response times of J2EE increase exponentially.

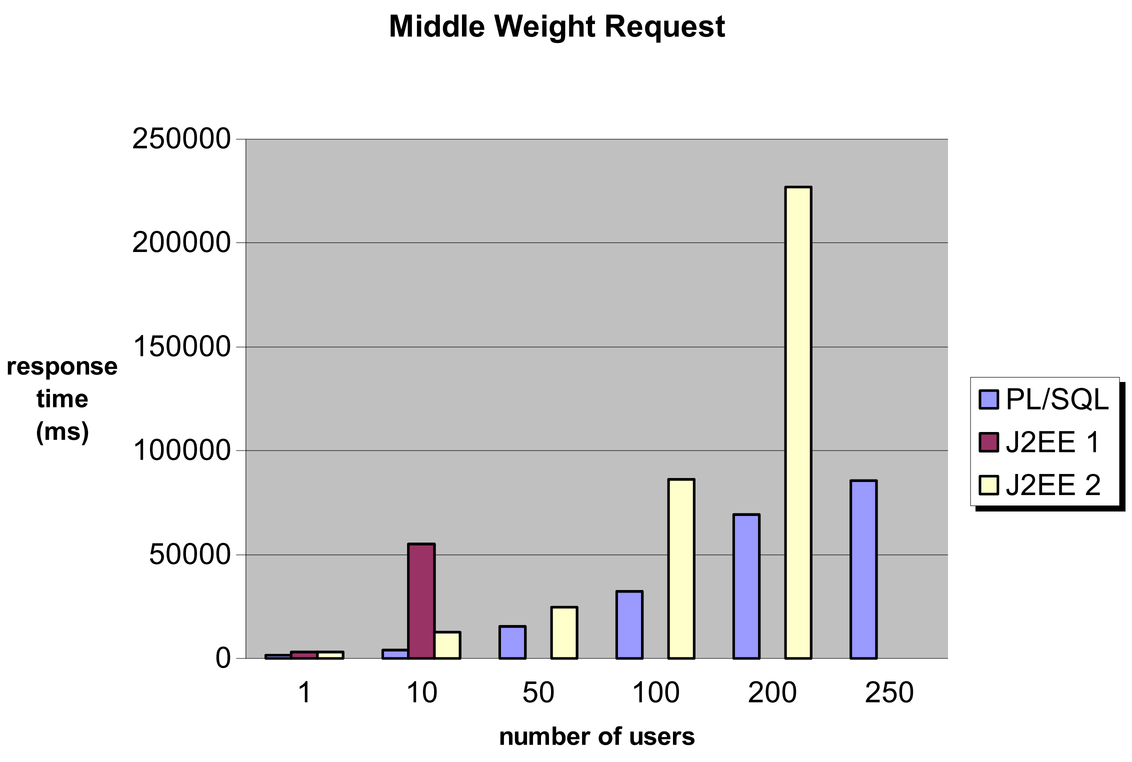


Figure 6-14. Comparison between platforms for the middle weight request. J2EE1 is unable to deliver results with more than ten concurrent users.

The results of the heavy weight request are similar to those of the middle weight request. J2EE1 is unable to deliver a result when the number of clients exceeds ten. Figure 6-15 shows that PL/SQL is the only platform to handle 200 clients sending

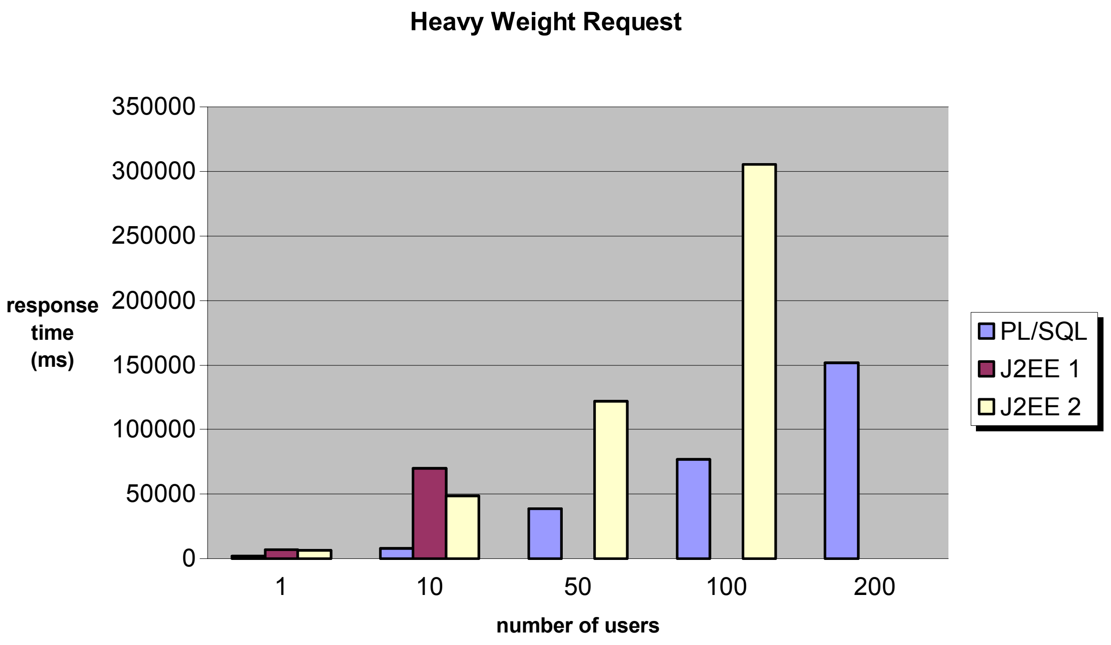


Figure 6-15. Comparison plot for the heavy weight request.

heavy weight requests.

The results of the mixed weights requests should not be taken so seriously, but the graph is included none the less (figure 6-16). PL/SQL has a lower average response time for 100 clients than J2EE2 has for 50 clients. J2EE1 does only have results for one and two clients. Only one value is shown in the graph though, since only experiments that were conducted on all three evaluation platforms are presented in the graphs.

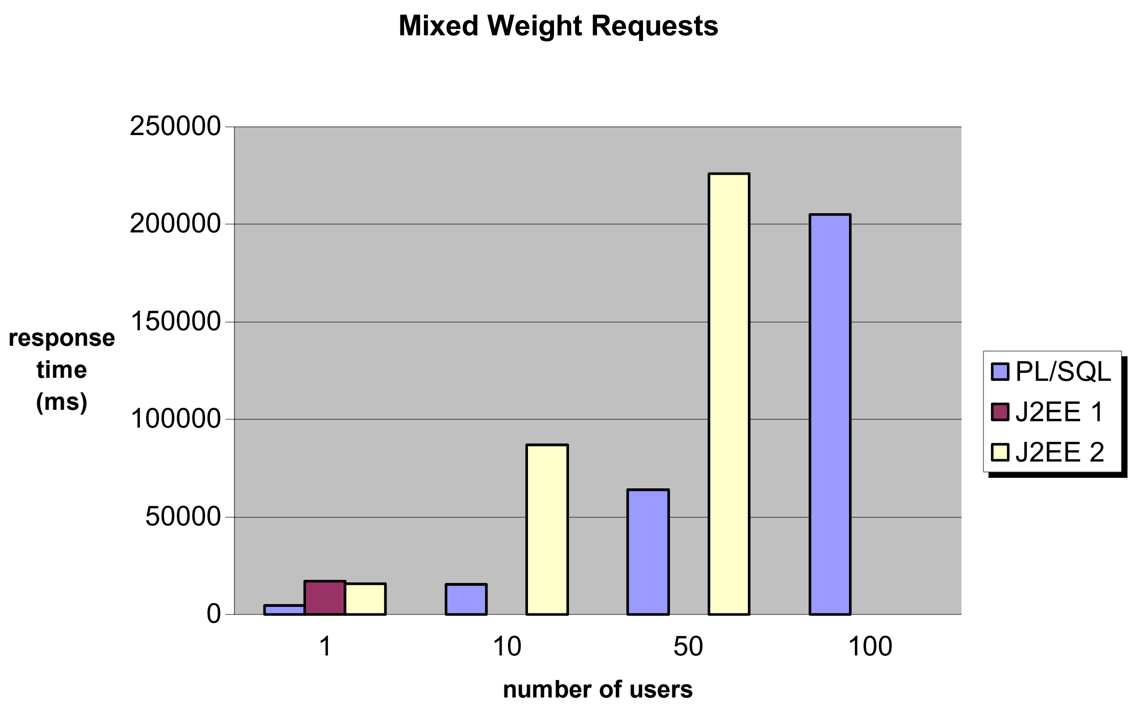


Figure 6-16. Comparison of the results from the mixed weight requests.

Clustering

Clustering (see chapter 4.4.2 Services) is a method to boost performance on the J2EE platform when performance is declining due to heavy load on the application server from many users. Another evaluation run on the J2EE platform had been planned, this time using clustering. But the fact that Prohunt went bankrupt left me without the necessary resources (more computers) so this experiment had to be canceled. It is my belief that clustering would have made a difference when the number of users increased beyond 50 if not sooner.

6.1.8 Problems during Experiments

Several problems were encountered during the course of the experiments. One major problem occurred when the company WM-Data retrieved the equipment I was using, including the Oracle Server that was used in the experiments. The experiments were not finished yet by then. I was given the chance to rerun the Java experiments at one occasion, but more time would have been needed for better results.

The experiments had to be rerun several times. At the first occasion, everything when fine up to 50 concurrent clients. Then the following error was thrown every time:

```
Java.sql.SQLException: ORA-00020: maximum number of processes
(50) exceeded
```

The Oracle server was configured to not accept more than 50 concurrent connections. I didn't have time to correct this before the server was taken away. But I was given access to the servers for one day, when I was able to remove the connection limit and run the whole experiment suit for both platforms. These were the PL/SQL and J2EE1 experiment runs.

The results were far from satisfying, as many of the scalability experiments for Java crashed with `OutOfMemoryExceptions,` both on the clients and the server. It was obvious that a third, dedicated application server machine was needed. And eventually I was given a last chance to run the J2EE experiments, J2EE 2.

The solution of emulating many concurrent clients on one single machine has its downsides. The software cannot provide enough network connections at one time. The evaluation client application generated a network error, `”Network Adapter Could Not Establish Connection”`, at several occasions when the number of concurrent client threads rose near 100. By adding a short random delay between client thread instantiations, most of these errors could be avoided.

If the experiments could have been performed yet another time, the first thing to do would be to reserve more memory for the Java Virtual Machine on both application server but most important on the machine running the clients. This can easily be done with command prompt switches when starting the application. The second thing would be to use clustering as mentioned earlier.

7. Summary and Conclusions

The objective was to compare two different approaches to developing distributed applications, Sun's Enterprise Java Technology (J2EE) and Oracle's PL/SQL architecture. But the objective was also to give guidelines to the company Prohunt about which products to port from PL/SQL to J2EE, how to perform this porting and what the gains would be. When Prohunt went bankrupt in the middle of the thesis, these objectives had to be altered. The focus of the thesis was moved to the comparison and a more in-depth look at the architectures itself.

To do this, it was necessary to study Prohunt's products ProCompetence, ProCareer and ProResource, and of course the PL/SQL architecture that was new to me. Enterprise JavaBeans and Application Servers were also a new experience and development of small applications for learning was slow at first. The evaluation copy of Borland JBuilder expired and there was a delay before Prohunt supplied me with a real licence.

Before Prohunt went bankrupt I started to implement a J2EE version of a small part of one of Prohunt's products. The goal was that the J2EE-version would perform this specific task faster and better that the actual working PL/SQL-version included in the product. But as a consequence of the bankruptcy, this prototype was abandoned and instead the implementation of two evaluation prototypes was started. These would do exactly the same using the two architectures, the difference was that the operations was no longer part of a real application but rather different request of different weights, invented by myself to evaluate performance and scalability.

The evaluation was performed, but not without problems. For example, it proved to be difficult to do a fair comparison. The simulation of several concurrent clients used up much of the computers memory and in the J2EE case, this together with the application server made the computer crash. The comparison consisted of a set of experiments for evaluating performance and scalability, but also comparisons of flexibility, usefulness and such things.

After having implemented prototypes using both architectures, after having performed all these evaluation experiments and after doing a general comparison of the differences between PL/SQL and J2EE, the following conclusions and opinions was reached.

7.1 PL/SQL vs J2EE: Architecture

PL/SQL is a two-tiered client/server architecture while J2EE is multi-tiered. This allows J2EE to be divided onto several machines for increased scalability, but multiple tiers also means more complexity and communication in-between tiers with higher response times as a result. Network communication is a bottle neck compared to CPU calculations.

PL/SQL is highly integrated with the Oracle database server, and developers are forced to use Oracle as DBMS if they want to use PL/SQL, which means that there is no platform independence. J2EE is built to be independent, both of platform, database manager, application server and development tools. The integration of PL/SQL and Oracle has an advantage though; it makes database access extremely fast.

J2EE was constructed to aid developers when developing distributed multi-tiered applications. The containers provides many services that let the developer concentrate on the business issues of the application, and takes care of security, transactions and such in the background. PL/SQL is not that sophisticated, but on the other hand it is much simpler in its design, so these issues are not that complicated to implement anyway.

Java is a fully object oriented language with all the benefits and drawbacks that comes with that. PL/SQL is a stored procedure script-language, a subset of a procedural language.

7.2 PL/SQL vs J2EE: Performance

Looking at the raw data from the performance experiments, it seems like J2EE is more than 17 times faster than PL/SQL on light weight requests, while being increasingly slower when the weight of the requests increases. These results don't reflect the reality; there are reasons that explain these results.

The answer for the unreasonably long response time for PL/SQL in light weight requests lies in the extra layer in the PL/SQL evaluation client. The client is implemented in Java and it uses an extra layer, or tier, when calling the PL/SQL subprograms. The extra tier is a module called DBLayer. It is a Java class that converts Java calls into PL/SQL calls, and the returned results from PL/SQL data types to Java data types. This extra computation takes extra time of course. And this extra time is most noticeable when the rest of the request time is small, in the light weight requests.

If it wasn't for this extra processing, I believe that PL/SQL would win all performance experiments, for several reasons:

- J2EE uses more tiers with more calls in-between themselves.
- There is room for much more optimizing of the code than I have done.
- In J2EE 1, application server and client are executed on the same physical machine, which degrades performance.
- In J2EE 2, three machines are used, which means more network traffic and extra processing time because there are two server machines that has to communicate in-between. The Oracle machine in the PL/SQL prototype functions as both processing server and database server, data travels thousands of times faster inside a machine than between machines.
- All record sets that are returned from the database has to be converted twice for the J2EE application, and only once in the PL/SQL case (when the result is returned to the Java evaluation client).

Considering all of this, I conclude that applications using lots of database accesses are executed faster in PL/SQL than in J2EE. The advantages of J2EE lie in other areas.

7.3 PL/SQL vs J2EE Scalability

The results of the scalability results show that the PL/SQL platform can handle lots of users considerably better than the J2EE platform. Better response times and more concurrent users before the system stops to function and starts to deliver error messages.

In support of J2EE, it should be mentioned that there are some extenuating circumstances considering scalability too. The results improved considerably when a third machine was used to run the application server dedicated. J2EE consumes a lot of memory. It didn't take too long before the Java clients started to report `java.lang.OutOfMemory` exceptions. Both the application server as well as the clients would benefit from more primary memory, or at least that the available memory should be reserved for the JVM. If each client were run from its own machine, I believe that the scalability results would improve as the number of concurrent clients increase.

Another way to improve scalability and boost performance as the number of concurrent users increases on the J2EE platform would be to set up a cluster of application servers as described earlier. Since I never got the chance to try this, it is difficult to predict how much difference this would have done.

Despite this, the conclusion can only be that PL/SQL scales better than J2EE.

7.4 Usage Comparison

There are differences about working with the two languages and architectures.

First, it must be said that there are countless different development tools for Java. Some are differently advanced text editors, some are Integrated Development Environments (IDEs) with both editors, debuggers and extra programming tools, and there are some even more advanced development tools which are integrated with an application server and has specialized "wizards" for developing all kinds of different beans, components and for deploying them into containers. The application I used, Borland JBuilder 4 Enterprise Edition [56] is of the last kind with wizards for everything and a tight integration with Borland Application Server [51].

There are also different development tools for PL/SQL if not as many as for Java. I have been using one called SQL Navigator from Quest Software [57]. It allows development, debugging, execution and insertion into the database.

Java as a language is more complete and powerful than PL/SQL. Since many developers are familiar with Java, there is often a lower threshold to use Java than to use PL/SQL. But for a developer that has already used another stored procedure language before, it should not be a problem to get started with PL/SQL. The architecture of J2EE is a very complex architecture though, with several APIs, specifications to follow and technologies to learn. It is not so hard to make a working distributed application, but to *really* learn J2EE and its optimizations is another question. PL/SQL is not very complex to learn, but there are features that you miss in the language.

J2EE has a more flexible and structured way of doing things. When working with large, complex applications, object oriented programming can be helpful to keep things manageable. PL/SQL has no such modulation, but it could be enough to put similar subprograms in the same package. The PL/SQL architecture will leave you with thousands of procedures though, so a working system for naming procedures is essential.

Java är lättare, snabbare, mer flexibelt osv

7.5 PL/SQL vs J2EE: Summary

As a summary one can say that PL/SQL are more effective and scalable, at least until you come to a certain point. Java though, may be the future. There are numerous ongoing projects using J2EE, but the technology is new and it is too early to say if it will be the technology of the future. Many experts says that that is the case, but there are beginning to emerge some opposition, saying that the architecture is too slow and complex for large distributed application systems.

My opinion is that you should probably choose J2EE if you already have Java competence and you are starting the project for scratch. In Prohunt's case, they had a working PL/SQL environment and there was no reason to change it. The cost in both time and money would have been great, and there are no guarantees that the end result would have been faster than the old architecture. The time would have been better spent in optimizing the existing code. One place to start might have been to reduce the number of calls for authorization reasons, for example.

8. Future Work

The first thing to do would be to expand the performance and scalability evaluations with clustered J2EE containers. I am confident that the scalability of the J2EE platform would improve significant if the J2EE containers would be clustered onto many separate machines.

Another improvement concerning the evaluations has to do with the clients. I developed a client software for simulating many clients concurrently accessing my evaluation prototypes on both architectures. These concurrent clients are in fact run as separate Java threads on the same physical machine. This is both CPU and memory consuming, and probably had some influence on the evaluation results. In a more accurate evaluation, clients would be spread out on several machines. Maybe not hundreds, but a few, each one simulating about ten clients.

Since Prohunt has gone bankrupt, this future work would only be of academical interest though.

References

[1] Prohunt – Företagspresentation, Information Sheet, Prohunt AB, 2000.

[2] Prohunt – ICM Platform, Information Sheet, Prohunt AB, 2000.

[3] Prohunt – ProCompetence, Information Sheet, Prohunt AB, 2000.

[4] Prohunt – ProCareer, Information Sheet, Prohunt AB, 2000.

[5] Prohunt – ProResource, Information Sheet, Prohunt AB, 2000.

[6] Prohunt's home page, Prohunt AB, 2001. URL: http://www.prohunt.se

[7] Gupta's Centura Team Developer. URL: http://www.centurasoft.com/products/ctd/Default.asp

[8] The J2EE Tutorial – A Beginner's Guide to Developing Enterprise Applications on the Java 2 Platform, Enterprise Edition SDK. Sun Microsystems, 2000. URL: http://java.sun.com/j2ee/tutorial/index.html

[9] PL/SQL User's Guide and Reference Release 8.0, Oracle TechNet documentation, 1997. URL: http://technet.oracle.com/doc/server.804/a58236/toc.htm

[10] Oracle PL/SQL Programming, Steven Feuerstein, O'Reilly & Associates Inc. 2nd Edition, 1997.

[11] Borland/Inprise Application Server, Inprise Corporation. Educational folder from course about J2EE development with JBuilder & Inprise Application Server (IAS).

[12] Borland JBuilder 4 – Enterprise Application Developer's Guide, Inprise Corporation, 2000.

[13] Oracle PL/SQL Release 8.0 – Feature Fact Sheet, Oracle TechNet documentation, 1997. URL: http://technet.oracle.com/products/oracle8/htdocs/xsqlffs3.htm

[14] Oracle9i New Features Summary – An Oracle White Paper, Mark Townsend, Oracle Corporation, 2001. URL: http://technet.oracle.com/products/oracle9i/pdf/9i_new_features.pdf

[15] PL/SQL Tutorial and Reference, Michael M. URL: http://saturn.cs.unp.ac.za/~michaelm/oracledocs/www.spnc.demon.co.uk/ora_sql/sqlmain2.htm

[16] Is Java better (or faster) than PL/SQL?, Oracle JServer FAQ, Oracle TechNet documentation, 1999. URL: http://technet.oracle.com/products/oracle8i/htdocs/jserver_faq/815faq0005.html

[17] EJB Development Using JBuilder 4 and Inprise Application Server 4.1, Todd Spurling, Inprise Corporation, 2001. URL: http://community.borland.com/article/images/26007/ejbdevelopusing.pdf

[18] Java 2 Platform, Enterprise Edition, Sun Microsystems, 2001. URL: http://java.sun.com/j2ee

[19] Enterprise JavaBeans Technology, Sun Microsystems, 2001. URL: http://java.sun.com/products/ejb

[20] JavaServer Pages, Dynamically Generated Web Content, Sun Microsystems, 2001. URL: http://java.sun.com/products/jsp

[21] Java Servlet Technology, The Power Behind The Server, Sun Microsystems, 2001. URL: http://java.sun.com/products/servlets

[22] JDCB Data Access API, Sun Microsystems, 2001. URL: http://java.sun.com/products/jdbc

[23] Java Naming and Directory Interface (JNDI), Sun Microsystems, 2001. URL: http://java.sun.com/products/jndi

[24] Transactions and Java Technology, Sun Microsystems, 2001. URL: http://java.sun.com/j2ee/transactions.html

[25] Java Message Service API, Sun Microsystems, 2001. URL: http://java.sun.com/products/jms/index.html

[26] J2EE Connector Architecture, Sun Microsystems, 2001. URL: http://java.sun.com/j2ee/connector

[27] CORBA Technology and the Java Platform, Sun Microsystems, 2001. URL: http://java.sun.com/j2ee/corba

[28] Java Remote Method Invocation (RMI), Sun Microsystems, 2001. URL: http://java.sun.com/products/jdk/rmi/index.html

[29] Java Technology and XML, Sun Microsystems, 2001. URL: http://java.sun.com/products/xml

[30] ECperf Version 1.0, Sun Microsystems, 2001. http://java.sun.com/j2ee/ecperf/

[31] J2EE Clustering, part 2, Abraham Kang, JavaWorld, 2001. URL: http://www.javaworld.com/javaworld/jw-08-2001/jw-0803-extremescale2_p.html

[32] Mastering Enterprise JavaBeans (Second Edition), E. Roman, S. Amber and T. Jewel. The Middleware Company, 2002. Also available online: http://www.theserverside.com/books/masteringEJB/downloadbook.jsp

[33] Client-Server Architecture: Bringing Order To The Bramble Bush, E. Steele and T. Scharbach, Steele Scharbach Associates L.L.C., 2000. URL: http://www.ssa-lawtech.com/wp/wp3-5.htm

[34] CORBA Basics, Object Management Group, Inc., 2001. URL: http://www.omg.org/gettingstarted/corbafaq.htm

[35] DCOM Technical Overview, Microsoft Corporation, 1996. URL: http://msdn.microsoft.com/library/backgrnd/html/msdn_dcomtec.htm

[36] RPC: Remote Procedure Call Protocol Specification Version 2 (RFC 1831), R. Srinivasan, Network Working Group, 1995.
URL: http://www.freesoft.org/CIE/RFC/1831/index.htm

[37] How Napster Works, Jeff Tyson, HowStuffWorks.com, 2000. URL: http://www.howstuffworks.com/napster.htm

[38] Gnutella homepage, Wego Systems, 2001. URL: http://gnutella.wego.com/

[39] FastTrack Technology, FastTrack, 2001. URL: http://www.fasttrack.nu

[40] SETI@Home: Search For Extraterrestrial Intelligens At Home. Seti@home, 2001. URL: http://setiathome.ssl.berkeley.edu/

[41] What is ICQ?, ICQ Inc., 2001. URL: http://www.icq.com/products/whatisicq.html

[42] MSN Messenger Service, Microsoft Corporation, 2001. URL: http://messenger.msn.com/default.asp?mlcid=US

[43] EJB with CORBA vs. COM+ with DCOM, Master's Thesis in Computer Science, Fredrik Jansson and Margareta Zetterquist, KTH, 2001. URL: http://www.it.kth.se/~vlad/edu/exjobb/Janson/thesis.pdf

[44] Tips on Performance Testing and Optimization, Floyd Marinescu, The Middleware Company, 2000. URL: http://www.theserverside.com/resources/article.jsp?l=Tips-On-Performance-Testing-And-Optimization

[45] Application Server Comparison Matrix, Flashline.com, 2001. URL: http://www.flashline.com/Components/appservermatrix.jsp

[46] Glossary of the Gartner Group, Gartner Inc, 2001. URL: http://www4.gartner.com/6_help/glossary/GlossaryI.jsp

[47] ProCompetence White Paper, Kenneth Hedlund, Prohunt, 2001.

[48] The Gartner Group, Gartner Inc, 2001. URL: http://www.gartner.com

[49] Java 2 Platform Enterprise Edition Specification v1.3, Bill Shannon. Sun Microsystems, 2001. URL: http://java.sun.com/j2ee/j2ee-1_3-fr-spec.pdf

[50] Microsoft ODBC home, Microsoft Corp., 2001, URL: http://www.microsoft.com/data/odbc/

[51] Borland AppServer, Borland Software Corp., 2001. URL: http://www.borland.com/besappserver/previous/index.html

[52] BEA WebLogic Server. BEA Systems, 2001. URL: http://www.bea.com/products/weblogic/server/index.shtml

[53] Oracle Application Server, Oracle Corp., 2001. URL: http://www.oracle.com/ip/deploy/ias/index.html

[54] IBM WebSphere Application Server, IBM Corp., 2001. URL: http://www-3.ibm.com/software/webservers/appserv/

[55] Trail: JAR Files, Alan Sommerer, 2001. URL: http://java.sun.com/docs/books/tutorial/jar/index.html/

[56] JBuilder Enterprise Edition, Borland Software Corp., 2001. URL: http://www.borland.com/jbuilder/enterprise/index.html

[57] SQL Navigator for Oracle by Quest Software, Ashutosh Gaur, 2001. URL: http://www.orafaq.net/tools/quest/sql_navigator.htm

[58] Enterprise JavaBeans Technology Downloads and Specification, Sun Microsystems, 2001. URL: http://java.sun.com/products/ejb/docs.html

[59]

Other Resources Used But Not Referenced

- Fundamentals of Java Security, MageLang Insitute, 1998. URL: http://developer.java.sun.com/developer/onlineTraining/Security/Fundamentals/Security.html

Appendix

A. Glossary

COM+	Microsoft's Component Object Model
CORBA	Common Object Request Broker Architecture
CPU	Central Processing Unit
DCOM	Distributed Component Object Model (Microsoft)
DBMS	DataBase Management System
DDL	Data Definition Language
DNA	former name of the Microsoft middleware platform
EJB	Enterprise JavaBeans
GUI	Graphical User Interface
HTML	HyperText Markup Language
ICM	Intellectual Capital Management
IDL	CORBA Interface Definition Language
IIOP	Internet InterORB Protocol
J2EE	Java 2 Platform Enterprise Edition or Enterprise Java
j2ee	the actual implementation of the J2EE architecture
JAAS	Java Authentication and Authorization Service
JAR	Java Archive
JDBC	Java Database Connectivity API
JMS	Java Message Service
JNDI	Java Naming and Directory Service
JRE	Java Runtime Environment
JSP	JavaServer Pages
JTA	Java Transaction API
JTS	Java Transaction Service
JVM	Java Virtual Machine
MTS	Microsoft Transaction Server
ODBC	Open Database Connectivity
ORB	Object Request Broker
PL/SQL	Programming Language/Structured Query Language
RFC	Request For Comments
RMI	Java Remote Method Invocation
SQL	Structured Query Language
SSL	Secure Sockets Layer
XML	Extensible Markup Language

B. Proposed new architecture for the ICM platform

This is the proposed new architecture that was the result of Prohunt's own investigation conducted prior to the start of this thesis.

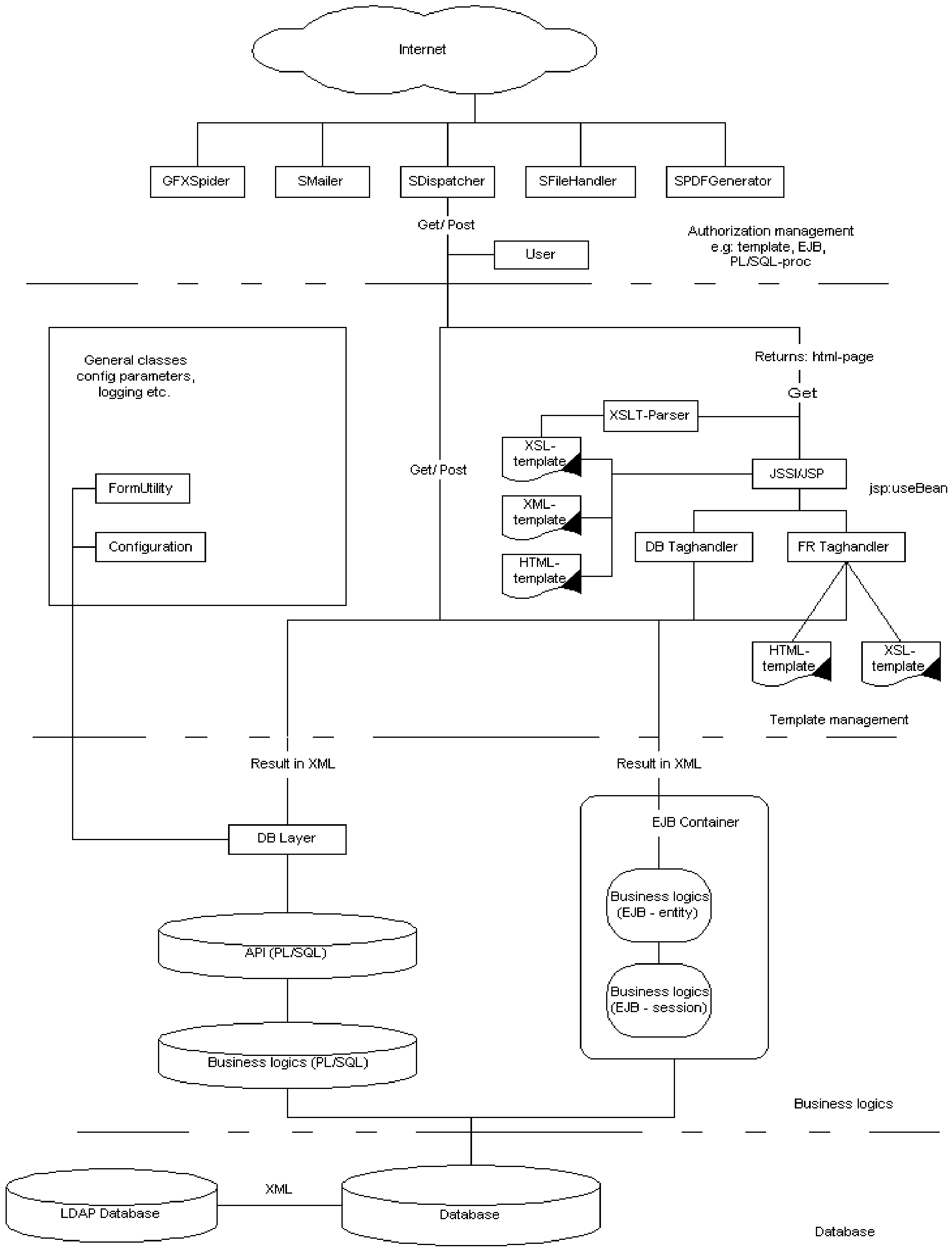


Figure B-1: This is a figure of the new architecture proposed by the project group investigating new architectures at Prohunt.

C. Complete Results From The experiments

These are the complete results of the evaluation experiments; this time presented each platform for itself.

C.1 PL/SQL

Results from the Performance Experiments

PL/SQL - Performance

Experiment	Run1	Run2	Run3	Min/Average
Light Request	891	911	891	891
Middle Request	1562	1552	1573	1552
Heavy Request	2294	2323	2323	2294
Mixed Requests	7541	5007	5227	Avg: 5925

Table C-1. PL/SQL Performance experiment results.

Results from the Scalability Experiments

PL/SQL - Light Weight Request

Users	Run1	Run2	Run3	Minimum
1	881	882	892	881
10	1693	1743	1643	1643
50	5758	5779	5769	5758
100	10855	10815	10816	10815
200	20912	20759	21191	20759
300	112221	113012	112555	112221

Table C-2. PL/SQL Light Weight Request results.

PL/SQL – Middle Weight Request

Users	Run1	Run2	Run3	Minimum
1	1522	1602	1522	1522
10	3956	4005	4076	3956
50	15482	15702	15683	15482
100	32767	32327	32486	32327
200	70256	69380	70114	69380
250	85633	85721	86018	85633

Table C-3. PL/SQL Middle Weight Request results.

PL/SQL – Heavy Weight Request

Users	**Run1**	**Run2**	**Run3**	**Minimum**
1	2334	2333	2304	2304
10	8382	8442	8342	8342
50	39747	38946	38856	38856
100	76820	79404	78483	76820
200	153765	152982	151708	151708

Table C-4. PL/SQL Heavy Weight Request results.

PL/SQL – Mixed Weights Requests

Users	**Run1**	**Run2**	**Run3**	**Average**
1	3915	4005	5478	4466
10	11657	18046	16433	15379
50	75189	74748	42160	64032
100	224393	172878	217603	204958

Table C-5. PL/SQL Mixed Weights Requests results.

C.2 J2EE 1

Below are the results of the first J2EE evaluation experiments.

Results from the Performance Experiments

J2EE 1 - Performance

Experiment	**Run1**	**Run2**	**Run3**	**Min/Average**
Light Request	50	50	60	50
Middle Request	3184	3175	3595	3175
Heavy Request	6980	6880	6950	6880
Mixed Requests	27760	16965	30625	Avg: 25117

Table C-6. J2EE 1 Performance experiment results.

Results from the Scalability Experiments

J2EE 1 - Light Weight Request

Users	Run1	Run2	Run3	Minimum
1	50	50	60	50
10	160	190	170	160
50	851	711	812	711
100	10024	10035	10044	10024
200	20186	20298	20078	20078

Table C-7. J2EE 1 Light Weight Request results.

J2EE 1 - Middle Weight Request

Users	Run1	Run2	Run3	Minimum
1	3234	3214	3045	3045
2	6245	6189	6179	6179
3	10586	10724	11087	10586
5	23112	22674	22142	22142
10	58164	59997	55179	55179

Table C-8. J2EE 1 Middle Weight Request results.

J2EE 1 - Heavy Weight Request

Users	Run1	Run2	Run3	Minimum
1	6970	7661	6980	6970
2	21134	21110	22467	21110
10	78985	73345	70271	70271

Table C-9. J2EE 1 Heavy Weight Request results.

J2EE 1 - Mixed Weights Requests

Users	Run1	Run2	Run3	Average
1	14120	16745	20219	17028
2	16364	40658	36327	31116

Table C-10. J2EE 1 Mixed Weights Requests results.

C.3 J2EE 2

Below are the results of the second J2EE evaluation experiments.

Results from the Performance Experiments

J2EE 2 - Performance

Experiment	Run1	Run2	Run3	Min/Average
Light Request	50	50	60	50
Middle Request	3125	3090	3080	3080
Heavy Request	6910	6810	6820	6810
Mixed Requests	15860	21140	24665	Avg: 20555

Table C-11. J2EE 2 Performance experiment results.

Results from the Scalability Experiments

J2EE 2 - Light Weight Request

Users	Run1	Run2	Run3	Minimum
1	50	60	50	50
10	180	155	160	155
50	789	734	699	699
100	8576	8221	8350	8221
200	17568	17215	17236	17215
300	95152	96222	95478	95152

Table C-12. J2EE 2 Light Weight Request results.

J2EE 2 - Middle Weight Request

Users	Run1	Run2	Run3	Minimum
1	3156	3075	3022	3022
10	13184	12862	12795	12795
50	25802	24685	24706	24685
100	86845	86312	86111	86111
200	227265	226978	227514	226978

Table C-13. J2EE 2 Middle Weight Request results.

J2EE 2 - Heavy Weight Request

Users	**Run1**	**Run2**	**Run3**	**Minimum**
1	6870	7224	6720	6720
5	25425	25254	25112	25112
10	49146	48868	49233	48868
50	122569	122017	121798	121798
100	305210	306963	305415	305210

Table C-14. J2EE 2 Heavy Weight Request results.

J2EE 2 - Mixed Weights Requests

Users	**Run1**	**Run2**	**Run3**	**Average**
1	14896	12657	19852	15802
5	38521	29845	42258	36874
10	95252	89120	76488	86953
50	237784	198265	242311	226120

Table C-15. J2EE 2 Mixed Weights Requests results.