Capturing design patterns for performance issues in database-driven web applications

Osama Mabroul Khaled
The American University in Cairo AUC

The American University in Cairo
School of Science and Engineering

Capturing Design Patterns for Performance Issues in Database-Driven Web Applications

A Thesis submitted to

The Department of Computer Science

In Partial Fulfillment of the requirements for the

Degree of Masters of Computer Science

By

Osama Mabrouk Khaled

B.Sc Computer Science, AUC, February 1998

Under the Supervision of

Dr. Hoda Hosny
Dr. Amir Zeid

January 2004

The American University in Cairo

Abstract

Capturing Design Patterns for Performance Issues in Database-Driven Web Applications

By Osama Mabrouk

The Design patterns technology is a new research topic which aims at helping with communicating technical knowledge in a standard non-technical format. People coming from different technical backgrounds can share this knowledge and apply it in their own way. For example, pieces of designs could be the same for different applications but they get implemented using different programming languages. On the other hand, web applications are becoming more widely spread, especially e-commerce ones, which make light returns on investment and achieve good relations between the companies and the customers. To stabilize this relationship, a web application must have a good design. The overall design of the web application is the key issue for keeping an ongoing business. An Organization definitely loses the trust of customers if its web application goes down every now and then. This research captures design patterns that help with optimizing performance in database-driven web applications.

The research captures five design patterns which are concerned with performance optimization, application complexity and resource utilization monitoring. The Database Connection Pool, Cache, and Static Enabler design patterns introduce direct optimization solutions. The SQL Statement Template design pattern breaks down the complexity of constructing SQL statements in the application code by externalizing all the SQL statements. The Logger design patterns monitors the resource utilization and delegates performance problems to a performance handler. A pattern language is also suggested in the research to give guidelines on the efficient use of these patterns. The captured patterns handle typical recurring problems that are common to many applications. Our test results show that performance is evidently optimized when these patterns are used.

The research also introduces adjunct activities that should be performed in the design phase in order to achieve better utilization of the Cache, Logger, and SQL Statement Template patterns. These activities normally result in artifacts which are provided to the development team in order to standardize the use of the patterns. For example, the research suggests a cache and logging roadmaps to highlight the caching and logging points in the application in order to efficiently use the Cache and the Logger patterns, respectively. The SQL Statement template pattern, on the other hand, suggests preparing the needed SQL statements at the design phase and collects them in an external storage to the application.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

The author wishes to thank: -

- Dr. Hoda M. Hosny for her dedication and participation in this research.
- Dr. Amir Zeid for his advice and participation in this research.
- Dr. Aly Aly Fahmy and Dr. Sherif El Kassas for their valuable feedback and suggestions.
- Eng. Hassan Ali (IBM) for designing the core of the Static Enabler solution and for permitting me to wrap it in a design pattern format.
- My parents and my wife for providing me with the needed support.
- Vodafone Egypt represented in the Internet Development and Data Services department for allowing me to use their systems.

GLOSSARY OF TERMS

Anti-pattern. An anti-pattern is a pattern that tells one how to go from a problem to a bad situation. The bad situation is revealed by its side effects where the original problem is solved but unseen or un-anticipated problems are generated [BWMR98].

Application Server. It works mainly as a script engine that is responsible for interpreting the logic written inside the requested pages. It receives its requests from the web server.

Cache. An area of storage existing behind the scenes which is used to store copies of previously requested objects that can be quickly accessed when needed.

Cache Roadmap *. It is an activity which is done in the design phase where all the collaboration diagrams are reviewed to mark the class methods which will do caching.

Cache Roadmap Artifact *. It is a document delivered from the design phase that contains all class methods that will do caching and validity period specified for every cache point.

Database Connection. A connection is a session with a specific database where SQL statements are executed and results are returned within the context of a connection.

Database Connection Pool *. It is a pool of database connections that exists in the application memory and serves as a recycling place where connections are kept open. Database Connections that are no longer used are returned to the pool for future access.

Design Pattern. A pattern could be defined as the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts [RDZH96].

Design Pattern Template. It is a documentation structure for a design pattern where a design pattern's context, problem, and solution are defined.

Dynamic Web Application. Which is the business logic that will decide on the type, and amount of information that the user will receive. It is geared over a web server, an application server and other legacy systems may be connected to as well.

Framework. A set of cooperating classes that make up a reusable design for a specific class of software [GEHR95]. [*]

HTTP Web Server. It is a server that service files for internet browsers using the HTTP protocol.

Load Testing. It is a mechanism by which a very high load is put over an application without exceeding its maximum capacity in order to test the application behavior.

[*] Some of these terms are suggested within the scope of this research

Logger *. It is a resource monitoring solutions for the application by which its behavior is depicted through its lifetime.

Log Roadmap *. It is an activity which is done in the design phase where all the collaboration diagrams are reviewed to mark the class methods which will do logging.

Log Roadmap Artifact *. It is a document delivered from the design phase that contains all class methods that will do logging where the type of information and messages are defined.

Pattern Language. A pattern language is a collection of patterns that guide the developer in a certain domain context. It is not just a catalog of patterns; it includes design decisions and domain-specific advice to the developer [MBMT01].

RDBMS. Relational Database Management System that keeps relational data.

Reusability. Reusability is a general word that takes anything beneficial for later use. Anything could be reused as long as it meets certain conditions and measurements.

Static Web Application. Which is the business logic that will decide on the type, and amount of information that the user will receive. It is geared over a web server only.

Static Enabler *. It is a solution by which dynamic web applications can benefit from the static pages high performance without spoiling the relations among the dynamic pages and their content.

SQL Statement Template *. It is a definition method for SQL statements that are used inside an application by which SQL statements are externalized outside the application code.

SQL Statement Template Artifact *. It is document delivered from the design phase that contains all the possible SQL statement templates which can be used in the application.

* Some of these terms are suggested within the scope of this research

Chapter 1

INTRODUCTION

1.1 Overview

For many centuries now, people have been practicing their lives and along with them, as time evolved, solutions to their problems. As man gained more experience in life, he passed it on to others in order to benefit from it. Experience however varies from one person to another; everyone sees his/her own experience as the best. As time progresses, new problems emerge and people build up new experience. As people share ideas and experience with each other, they arrive at the end at what is called, the best practice. In other words, the best practice is the best solutions for certain problems in certain contexts. Problems may show up in recurring situations. Thus, as soon as a problem appears, people tend to search for previous solutions that have been tried and proven to be the best. This is what is called patterns. A pattern could be defined as the abstraction from a concrete form which keeps recurring in specific non-arbitrary contexts. [RDZH96]

A Solution is important, but designing a very good solution is much more important. People for sure gain experience from a solution, but they will rarely be able to enhance this solution without knowing how it was originally designed. Not just knowing the design of the solution, but tracing it back to its origins. This greatly helps in clarifying the picture, and may lead to discovering new solutions in completely different tracks. It does not mean that one has to always go to the primitive solutions in order to enhance an existing one. Sometimes, there are solutions that prove to be the best in their own domains. These solutions have been tried many times and there may be no need for tracing them back to their origins.

What is and isn't a pattern differs from one person to another according to his/her point of view. A pattern for one person could represent another person's primitive building block [GEHR95]. Experience and context are the key factors that determine this point of view. For example, developers that use assembly language may deal with registers to do string manipulations and build their own libraries. These libraries might have helped at one point in time to be the primitive building blocks for a programming language like C. C Developers may design patterns over these building blocks which are obviously primitive in the language. A C developer may want to design certain patterns like inheritance, encapsulation, or polymorphism. These are totally new patterns for a language like C. However, they are native to another language like Java. Hence, selecting a pattern and implementation tool is really important to show the point of view [GEHR95].

The market needs for software have been rapidly changing. Software development must cope with these very fast changes as well. One cannot start the development cycle every time from the beginning and build everything from scratch. Developers must always think of reusability whenever possible. Design patterns technology is a promising and efficient way to apply reuse and to help developers in their work. It conveys the knowledge in a high level language which makes the understanding process very easy. However, this technology is still lacking. It is always the problem of the developers who do not document their experience; a process that must be shaped in an organized way whenever possible.

Design Patterns present an interesting research field that aims to help in communicating technical knowledge in a standard non-technical format. People from different technical backgrounds can share this knowledge and apply it in their own way. For example, pieces of a design could be used in different applications and implemented using different programming languages.

On the other hand, web applications are now becoming more widely spread, especially e-commerce ones, which achieve a high returns on investment and establish good relations between the companies and their customers. To maintain this relationship, a web application must have a good design. The overall design of the web application is the keyword for keeping an ongoing business. An Organization definitely loses the trust of its customers if its web application goes down every now and then.

Web technology is a fast growing field that a large number of organizations are trying to adopt. However, these companies may make a lot of investments without getting the expected revenue. This could be because of many technical or non-technical factors. A web application is like any other application that has to be covered from all aspects especially from the development and maintenance aspects which are the main success factors. The products of the development life cycles are checked against basic factors like reusability, reliability, and complexity. Design patterns can be used to tackle such aspects. The main success factor for a web application is reliability. Reliability is the one feature that builds trust for the customers who visit an e-business website. Without trust business cannot continue.

1.2 Motivation for the Research

The Design patterns technology is a very promising area which started to gain attention since 1995. The primary motivation for this research is proper knowledge transfer, which is a common problem facing many developers. The aim is to show the importance of documenting software in a high-level language like design patterns. Developers are used to transferring knowledge verbally most of the time. They do not care so much about documenting this knowledge in a professional manner.

Documentation is considered most of the time a luxury task that need not be done at all. It is not impossible to see an application where the only reliable source of information is its source code [PAFJ96]. The verbal knowledge transfer process has major drawbacks, although it is an easy and quick process for the developers: -

1. Verbal knowledge transfer does not mean full transfer. This means that the next owner may interrupt the initial developer every now and then to ask questions about things that are not clear enough for him/her. The Software development process requires a great deal of concentration. No one likes to be interrupted every now and then.
2. Time can be wasted in trying to teach others. Verbal knowledge transfer can sometimes take more than one week. The time could be extended if there are a lot of interruptions or the audience lose their concentration for sometime.
3. Developers do not work in projects permanently, which is the nature of software development projects in large companies. Verbal knowledge transfer can continue with every new developer. This means that if knowledge is not documented, extra work must be done to gain this knowledge from the owners. What can be worse, is that knowledge may be completely lost if the owner developer disappeared for any reason.

The second motivation for this research is to raise awareness of the sense of performance for the developers regarding software in general and web applications in particular. It was found also that there is a lack in this research topic within the software community. We hope that this research adds some value for the readers. There is a bad practice that many developers seem to adopt which is: develop to achieve the functional requirements, and treat the non-functional ones, on top of which is performance, with lower priority. Performance problems can still appear even if the product passes the User Acceptance Test and after the launch of the product. At such a time, the cost of problem fixing might be really high on the business side. It is important to highlight the performance problems in the early stages of the development cycle and to try to analyze them to avoid future problems which can be difficult to trace. This early recovery approach for performance problems can save a lot of time and money.

1.3 The Addressed Problem

Web applications are used nowadays in almost all fields. There are companies that solely depend on e-business solutions for their revenue. The most common problem that can cause any web application to fail is: Usability. Web applications fail at this point either because the site is irrelevant, difficult to navigate, ugly, or slow [USPROB01]. All these problems can mark the site as unreliable. At the very top of all web applications problems is performance. Usability is indeed the factor that guarantees customers' trust. A usable web application will lead to increased customer visits which in turn translates into increased revenue.

A performance problem comes mostly from the server-side application, as will be explained in the web application chapter, because the server-side application is the variant factor in the web application. Since it is variant, some sort of control must be imposed over it. Such a control starts from the beginning of the development life cycle where design decisions taken for performance problems are handled in the early stages. Performance problems can noticeably decrease if there is a pattern language or a framework controlling them starting, at least, from the design phase.

Most research contributions concerned with performance issues are focused on tuning the ready-made applications like a web server[1], RDBMS, or an application server. Other contributions are mostly focused on suggesting guidelines for programming languages like Java, and C++. All these areas are very important without doubt. However, there could always be room for performance guidelines on the analysis and design levels in order to save time and effort, and the use of a pattern language that places performance in the first place would certainly help in that respect. The pattern language may impose a certain strategy to tackle the major problems in the web applications but it can be the base for a higher-level

[1] Appendix B shows sample configuration hints for apache web servers.

pattern language that covers performance aspects from the application point of view, e.g. application, database, web server, network, and so on.

1.4 Outcomes and Contributions

The main outcomes of this research are the 5 performance-related patterns and the performance pattern language. The aim of the research is to capture patterns that help in optimizing the performance for database-driven web applications. However, some of the patterns, like the cache and the logger, can be extended to give a wider solution for other applications.

Moreover, the documentation of these patterns is considered a valuable outcome which can be added to an existing library of patterns. Also, it is hoped that every reader would be convinced after reading this research that performance is an integral factor for any application success which must not be overlooked. In other words, every developer must have good thinking in performance issues whether he/she uses these patterns or not.

The research has also introduced a new presentation for some of the patterns by linking them to activities on the design phase. In other words, to show out the benefits of the patterns, some extra activities must be conducted on the design phase. If these activities were not performed, the patterns would be left for individual decisions which lead to diversified implementations. For example, the SQL templates shown in section 7.4 should be generated in the design phase by the same team that designed the database structure otherwise individual developers would have different approaches for constructing them. Also, a separate iteration should be conducted in the design phase to mark the logging and the caching points in the collaboration diagrams as shown in sections 7.2 and 7.5.

The research suggests some measurement metrics also for evaluating the reliability of the patterns. These measurement metrics presented in section 1.5.1 are a start point for building a standard evaluation methodology for design patterns. The measurement matrices should be useful for both the author and the audience. The author has to review his/her patterns basically against these metrics to ensure that the patterns are covered completely. The audience should be able to review more than one similar pattern and choose from among them the one that meets the measurement metrics especially if they will implement it. In other words, the audience should combine his/her common sense of the patterns selection process with such measurement metrics to come up with the best.

Our contributions in the area of design patterns may be summarized as follows: -

1. The research has come up with a new variation of the database connection pool that enables it to use the cache pattern directly and give a common identity to all the connections. It is explained in section 7.1.
2. The cache pattern is combined with a caching roadmap, an artifact which is generated from the design phase, as a guideline for the development team to insert caching points across the whole application. The pattern is presented in section 7.2.
3. The Static Enabler pattern is a new design pattern. It presents a solution to speed up the dynamic web applications by converting dynamic pages to static ones. The pattern is presented in section 7.3.
4. The SQL template is new in its approach although the externalization idea, which is presented in the pattern, is not new for many developers. It presents a simple and easy SQL statement template structure that can be linked to the application directly. The SQL template is generated from the design phase in the final format to be linked with the application directly. The pattern is presented in section 7.4.
5. The logger pattern introduces a performance handler by which problems can be analyzed. It is combined also with a logging road map that is generated from the design phase to standardize the logging format and the messages. The pattern is presented in 7.5.

1.5 The Solution Approach

As mentioned earlier, design patterns come with experience. A design pattern cannot just come by thinking about it even if it provides an innovative solution. It has to be applied and used in more than one project and proven to be successful in their context. However, to further prove the concept, the suggested design patterns have to be implemented and tested. Every design pattern must be implemented, quality assured, tested, and documented. By documenting the pattern language, a test environment, as described in Appendix D , had to be set up and an incremental testing approach was conducted on the final pattern language. Statistics were gathered during the testing process to prove the reliability of the performance optimization patterns (Database Connection Pool, Cache, and Static Enabler) as explained in section 1.5.3. It was important to use anti-patterns to show how good the suggested design patterns are. Figure 1 shows the followed solution approach plan.

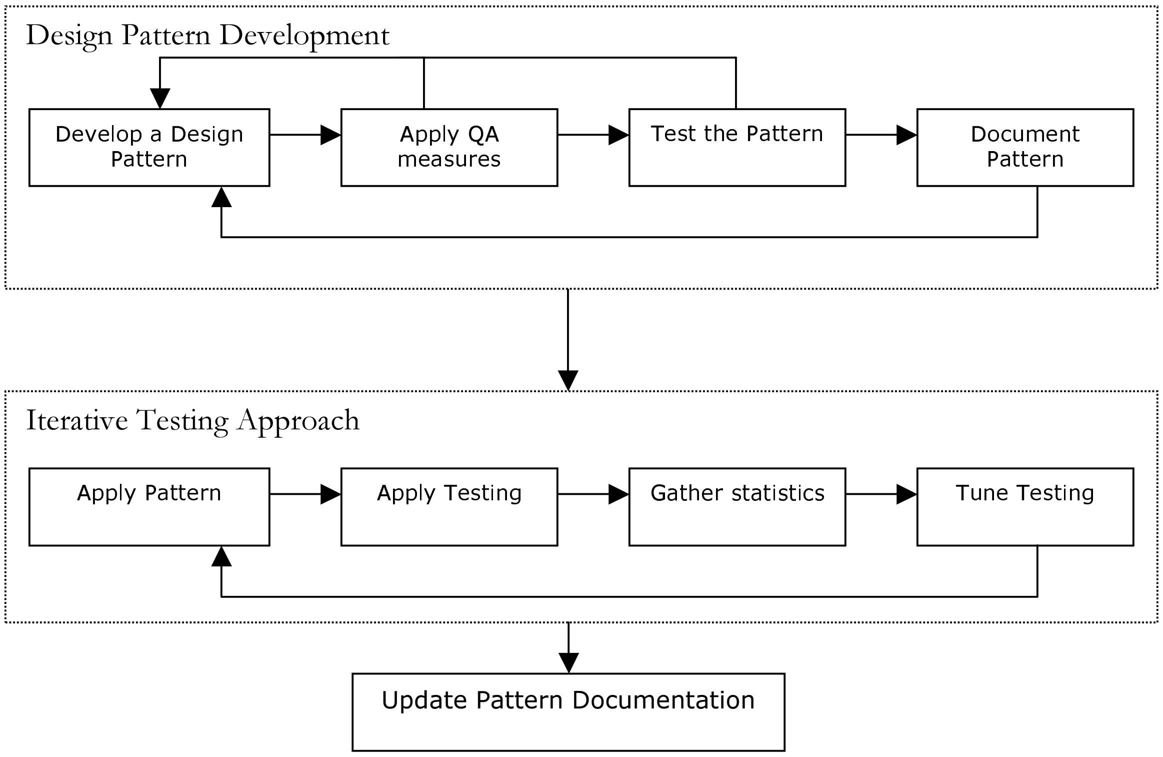


Figure 1: Solution Approach

1.5.1 Design Patterns Quality Validation

Some kind of quality metric had to be applied on every design pattern to validate its quality. This was very useful since it gave an indication of how good or bad the suggested design is. There are metrics available for every phase of the development cycle. However, we were interested mainly in applying metrics on the design phase. This was necessary since some of the design patterns were not directly tested against performance. For example, the Logger and the SQL Statement Template patterns could not be directly measured against performance criteria. Table 1 shows the qualitative design metrics which were applied, whenever possible, on the design patterns.

A Design pattern is validated against the following criteria: -

- **Encapsulation and Abstraction [LD93]:** Each pattern must describe a well defined-problem and its solution in a certain context. It abstracts also the domain knowledge and experience

- **Openness and Variability [LD93]:** Each pattern should be open for extension or enhancement. It should be recognized also in a variant number of implementations regardless of the programming language or the platform used.
- **Equilibrium [LD93]:** Each pattern should show some sort of balance between its pros and the cons. It is not practical to see a design pattern that has more side effects than benefits.
- **Minimality [CC02]:** The pattern must be able to utilize the existing patterns, if possible, in order not to re-implement the wheel.

Table 1 shows the suggested Metrics for design patterns validation criteria.

Characteristic	**Metrics**
Encapsulation	• Does the design pattern have a clear interface that does not expose private information? • Does the design pattern include only the needed classes ? • How will the design pattern interface be affected if implementation is changed?
Abstraction	• Will the design pattern work elegantly if put in a similar context?
Openness	• Can a design pattern be extended to expose more functionality?
Variability	• Is it recognized by more than one programming language? • Can it be implemented on more than one platform? • Does the design pattern show features biased to a certain programming language?
Equilibrium	• What are the benefits shown by the design pattern? • What are the side effects shown by the design pattern?
Minimality	• Does the pattern use other patterns? • Is it recognized by other patterns?

Table 1 Design Pattern Metrics

1.5.2 Pattern Template

Pattern documentation starts after pattern testing to capture the design pattern. Documenting a design pattern is the first step to build a pattern language.

The following template was used to document a design pattern [GEHR95] [LT01]: -

Context:

- The general situation in which the pattern applies

Problem:

- Describing the main difficulty in which the pattern applies.

Forces:

- The issues or concerns to consider when solving the problem. If ignored, then the pattern may be invalidated.

Solution:

- The recommended way to solve the problem in the given context —'to balance the forces'.

Consequences

- Side effects or tradeoffs of the design. It is important to mention them in order to evaluate different design patterns

Antipatterns: (Optional)

- Solutions that are inferior or do not work in this context.

Related patterns: (Optional)

- Patterns which are similar to this pattern.

References:

1.5.3 Testing

In order to give a clear picture of the pattern behavior, implementation is provided to facilitate the testing process. Every design pattern has gone through a testing process in which its behavior was verified. The whole design pattern was then passed into what we can call a pattern-testing phase. It is like component testing but not strictly the same as the component has definite and clear usage and is treated as a black box. However, a design pattern is a white box that needs to work in a certain context and possibly with the help of other patterns.

An incremental testing approach for load testing was conducted in this research. The Connection Pool, the Cache, and the Static Enabler patterns were introduced one by one and measurements recorded. The web application was run under a simultaneous load of virtual users for a certain period of time. The virtual users are the number of connections opened to the web application. During the testing period, measurement values were taken.

It is important to clarify here that the load testing conducted in this research was to prove that web applications that implement performance optimization techniques achieve better performance. The aim is to introduce to the developers solutions that can help him/her in his/her work. Hence, implementation of the solutions is not important from the testing point of view as it could vary from one person to another.

The main measurement factors that were recorded are: -

1. **Latency:** the minimum time required to get any form of response, even if the work to be done is nonexistent.
2. **Response Time**: the average time needed from the server to serve one client.
3. **Throughput:** the number of successful requests per minute.
4. **Reliability**: The ratio of failed request to the successful requests. It gives an indication of the system stability. All other parameters, like maximum client connections or maximum requests per connection, are not considered in this testing. They will be set to very high values so as not to affect the testing results.
5. **Server Memory Utilization:** design patterns like the Cache, Database Connection Pool, and Static Enabler have a direct impact on the memory

usage. This impact must be checked and reported clearly on the pattern language documentation as the tradeoffs of the design pattern.

6. **Server CPU Utilization:** indicates how the server can be busy serving clients according to server load. It is assumed that the machine is ideally used by only the web services.
7. **Server I/O Waiting:** The average I/O waiting on the server machine

1.6 The Suggested Patterns

All the suggested patterns can solve problems that may occur during the development phase of a large number of the database-driven web applications development. They adhere the most critical performance problems that face developers. They are briefly described below but their detailed designs are presented in Chapter 7.

1.6.1 Database Connection Pool

It is the concept of maintaining established connections between the application and the database so that no delay time is suffered when accessing the database every time information is retrieved. Database connection is a very demanding activity since it has to allocate resources like memory, communication, and a user security context. It is very common to see database connections that take up to two seconds [BH99].

Database driven web applications can suffer severely if database connection pooling is not implemented. It is one of the basic infrastructure mechanisms that any database driven web application must implement. There is a connection pool manager that guarantees proper use of connections, like specifying a minimum, and a maximum number of connections. However, misuse of the connection pool can lead to a dramatic leakage and may cause the web application to crash.

The concept is not new. It has been utilized in many contexts. However, it is very useful in web applications where concurrent access is normal. For web applications that have very high hits, the connection pool plays a major role. It must be an integral part of any database driven web application.

1.6.2 Cache

Database Queries may take a considerable time to finish and the query result might be small and not expected to change in the near future. It is not abnormal to see queries that take more than one minute to finish. Queries can take more than one minute on very large databases. It also happens that tuning the database, like creating indices, may be space consuming. Web Applications sometimes interact with such databases. So, web applications must have techniques to overcome the database delay.

A web application can use different caching techniques to reduce latency, minimize network traffic, and consequently reduce timeouts. Caching, as defined by Brown [BB02] is *"an area of storage existing behind the scenes which is used to store copies of previously requested objects that can be quickly accessed when needed."* It improves the performance and makes the web applications more reliable [BB02]. Caching is not a new technique; it has been used in many software fields and on different aspects. There is special caching hardware, like cache proxies, which cache the static web content. There is caching even on the client machines so that not to download the static content, like images, every time. Clients become satisfied with the quick response which increases their trust on the web application.

Server-side caching is a performance enhancement technique that is used in many web and non-web applications. It brings the expensive results to the application memory. Ideally, a database query result is cached. A caching solution must handle the object expiration to ensure consistency. When the application invokes the query it looks up first in the cache before accessing the database. If looked-up objects are found and are not expired, then they are returned back to the client without passing through the database bottlenecks to establish a connection and run the query. The pattern abstracts the different caching solutions and leaves the implementation to the developer.

1.6.3 Logger

Logging is a technical solution to write, most of the time, certain informative statements in an external media, like plain files, to record system problems or track system behavior. Logging is a practice that is usually ignored until the development phase, usually because it may delay the project. It is usually done to debug the system behavior. Many developers do not conform to a certain logging standard. This leads at the end to messy log files that contain a lot of messages from many sources. This unprofessional development practice happens mostly because of the over-trust in the final product, especially if it is well tested. Unprofessional developers do not anticipate future runtime problems. This is why they do not plan for them.

Moreover, performance problems are the least logged issues. These issues are usually closed in the development phase, as thought by the developers. Concentration is mainly on logging functional behavior. For example, database queries can be logged, but their time consumption is not logged which leads definitely to performance ignorance of the system.

The purpose of this design pattern is mainly to introduce a logging design pattern technique which targets database querying in the first place in order to build more robust designs. Once this pattern is applied on the design, it becomes a standard in the development that no one can ignore.

1.6.4 SQL Statement Template

A key problem with database-driven web applications is that queries are mostly considered as part of the code. Queries may be structured and very complex. Coupling code and database queries may be needed because queries are considered part of the business logic. Separation has been presented but in the form of a code layer to access the database. This separation is mainly important to map relational databases to object-oriented software. However, bundling the queries inside the software code has the following main defects: -

1. Tuning any database query requires re-compilation of the code which may be time consuming. It happens that very simple keywords are needed in the old database queries. This happens a lot if new constructs are introduced into the database query language or a standard modification across all the queries is needed.
2. Software programmers play the roles of both developers and database experts which deviate their focus from application development.
3. Database queries are not easily maintained by experts.
4. Database queries decrease readability and understandability.

Externalizing database queries can be introduced to database-driven web applications and at the same time control the change. Queries can be written in flat files with some special tags to replace the dynamic constructs. So, database queries and applications are not 100% decoupled. Programmers and database experts can still work in parallel without interrupting each other. The query result interface can be agreed upon in the early stage of the development and the final queries may be integrated later on.

1.6.5 Static Enabler

The Internet was a major breakthrough in the field of information technology. The main benefit from this technology was information sharing. As part of this evolution, people started to post their information through websites. A website is a designed location on the Internet that posts information about a certain subject. Static websites emerged at the advent of this technology. A website normally contains a mixture of different media objects, like images, flashes, and text. The only backend application needed to serve a static website is a web server.

Static websites are sufficient for small business where information is not expected to change quickly. However, if the business size gets bigger, the need for dynamic content becomes mandatory. The main issues that a dynamic website is needed for are: -

1. When Content management for a static website is difficult.
2. When A static website does not integrate with legacy systems. A dynamic website is mainly needed to achieve an interactive communication with the website users and the existing backend.
3. Legacy systems may have tons of information that can take a very long time to be manually posted on a static website.
4. Periodic and automatic change of the content can not be supplied using a static website
5. Logical and consistent relations of the different parts of a website are better represented by a content management pool, e.g. a database.

On the other hand, a static website has the following benefits over a dynamic website: -

1. It responds much more quickly.
2. It is more robust and reliable.
3. It has fewer problems.
4. Implementation is done more quickly than for a dynamic website which can have many complexities in the backend.

Some applications provide content management tools that build the required content and publish it as static pages. The limitation here is that large database systems can consume a huge amount of disk space if they are published as static pages. Some other people prefer to work with dynamic websites and accept their limitations for the sake of other benefits as mentioned above.

However, a dynamic website can benefit in a number of ways if there are static pages posting dynamic content: -

1. Highly accessed dynamic pages can be converted to static ones to introduce a faster response, while it still keeps the dynamic page for other references and saves at the same time disk space.
2. The decision can be taken to convert all the dynamic pages to static or just leave the dynamic ones according to the capabilities of the working environment.
3. Some backend systems can be temporarily shutdown if their counterpart dynamic pages are converted to static.

1.7 Thesis Organization

The document is organized in 9 chapters. Chapter 1 presents the roadmap of the thesis. It presents the suggested design patterns that are captured to tackle performance issues and argues for them. It discusses the different testing and QA methods that were used to verify the design patterns impact on the testing environment.

Chapters 2 through 6 cover the background literature of the Design Patterns and web applications. Chapter 2 discusses the Reusability obstacles and the different methods that can be used to extract reusable components. Chapter 3 is a briefing on the history of Design Patterns and related technologies. It also discusses how to capture, describe, and select design patterns. Chapter 3 complements the picture of design patterns by presenting pattern languages and anti-patterns, some framework examples, and shows how they are integrated with design patterns. Chapter 4 discusses the web technology, shows an ideal architecture for a database-driven web application, and discusses the performance issues related to a web application. Chapter 5 presents some quality assurance techniques that could be used to verify design patterns and identifies testing methods that may be considered for testing web applications. The background provided here emphasizes on design patterns as a design technique that can be used in many areas, emphasizes on web applications performance issues, and at the same time paves the road to show how design patterns and web applications can be tested. Chapter 6 cites some of the related work that has been published by some researchers in the area of design patterns and web applications.

Chapter 7 presents the five patterns, which are the core of the research, and gives guidelines for their usage. Chapter 8 gives an analysis of the patterns according to the quality assurance measurements and provides further analysis according to the testing results. Chapter 9 concludes the research and highlights some future directions for future.

Chapter 2

REUSABILITY

The Software Engineering field first started to emerge in 1968 when the NATO Software Engineering Conference first took place. The conference focused primarily on the *Software Crisis,* which was the first time for this term to be used. Software reuse was the main candidate to overcome this *crisis.* Some reusable component libraries were proposed and used effectively in numerical computation, I/O conversion, text processing, and dynamic storage allocation. Twenty-three years later, software reuse was still recognized by computer scientists although it failed to achieve its goal of becoming a standard practice for software construction [KC92].

Reusability is a general word that takes anything beneficial for later use. Anything could be reused as long as it meets certain conditions and measurements. A piece of documentation could be reused, classes and objects could be reused, and design could be reused too [BG94]. Software systems could be built from predefined, interchangeable common software components as they share a lot of similarities. It has been shown, in general, that software systems share about 60 percent commonality. This percentage could reach up to 90 percent in the domain specific projects [MC95]. Usually, when one starts to build a design, he/she tends to concentrate on the work domain on hand. This approach to a large degree helps in finding the appropriate classes and objects needed for a new system. However, another round must be carried out to figure out the general purpose, common, and domain specific objects.

General-purpose objects are usually found in already existing libraries. However, if they are not found then, one should start designing them. Actually, one may be wasting some time in the beginning in order to build something generic, but this effort pays off in the future when other applications make use of such libraries. Common classes and objects are not for general-purpose use, but they are used inside a working system. They are to some degree tied to the work domain, and generally used internally. The maximum one can wish for is that the design is built out of general-purpose pieces, and possibly some common classes. However, there must be some domain-specific classes and objects to solve the problem on hand.

This is all well talking about designs and how to classify objects. However, in order to keep the design robust, future changes must be anticipated [GEHR95]. A design that does not anticipate future changes may suffer severe modifications in the design and implementation as well. Here comes the role of design patterns. They help in changing different aspects of the system independently from each other.

As Booch says "In successful projects, we have encountered reuse factors as high as 70% (meaning that almost three-fourths of the software in the system was taken intact from some other source) and as low as 0%" [BG94]. This shows how projects could be finished very quickly if they strive for reusable components. The reusable components could be anything useful as explained before. Design patterns are among these reusable components, since they are components of design. However, identifying a design pattern is not usually done, just to meet project schedule times. It is very important to know the obstacles in practicing reuse and how to overcome them, and then develop standard techniques to search for and find reusable items.

2.1 Reuse Obstacles

It is really difficult to argue against reusability. However, there are many obstacles that prevent us from making full use of it. Table 2 shows that most obstacles come from non-technical perspectives [MC95].

- No understanding of what to reuse and how to create it for reuse
- No planning for reuse
- Reuse confined to an individual or within one system
- Limited to code-level reuse and reuse-in-the-small
- No management involvement or support
- No or negative reuse incentives
- Reuse practiced only in the coding phase
- Cost of reuse is too high
- Reuse is contrary to current software culture
- Nothing to reuse; absence of a library of reusable components
- Too expensive and too dangerous to adapt existing components for reuse in another system
- Inability to recognize what has high potential for reuse
- No reuse activities defined as part of the software life cycle process or methodologies
- Reuse is not supported by current software tools
- Management not convinced of value of reuse
- Software professionals not trained in reuse and do not want to practice reuse

Table 2 Obstacles for Reusability

Some software organizations think that they have to practice object-oriented technology in order to practice reuse. Actually, it works the opposite way. Reuse must be practiced first in order to make use of object technology. Although, object-oriented technology presents very useful concepts like inheritance, encapsulation, etc, a non-expert designer or developer can still build a spaghetti solution using it. Actually, reuse can be practiced in object-oriented technology and in other traditional software environments. It is the organization that takes the decision on how to go about Reuse programs [MC95].

Another important obstacle is the recognition of reuse as a separate discipline. Many software developers think that reuse is too obvious to make plans for [MC95]. They practice it on daily basis in their work. However, they did not recognize that they do it on individual basis, group basis, or even project basis. These individual activities can be applied in one project and they might show up again in a future project doing the same sort of actions. What organizations need is to have a plan for reuse that monitors all these activities in a smooth way so that they do not interrupt work progress. When these scattered reuse activities are collected, organized, and indexed in libraries, other groups and future projects can benefit from them. It will then become a practice not to start any project before checking out the available library and search for a solution that can be used directly or at least enhanced or modified.

In addition to the above, management in some organizations might not be convinced with reuse [MC95]. Usually, time-to-market and cost are the primary factors that control any software development. Reusable solutions usually take more time to develop and test, despite the fact that they make a big difference in implementing libraries that can be used in the current and the future projects. Direct and non-reusable solutions are usually built very quickly. Management usually considers current business needs, they do not look into the future investments. So, the first thing to ask when considering reusable solutions is that if it is going to delay the project then do not use it.

This leads us to a basic question: why is reuse not considered in project planning? Actually, the customer cares only about the final product. Such an internal process does not mean anything to him. However, any project plan must take some factor of delay in its initial phases due to reuse. On the other hand, risk potential will decrease, as reuse will be one of the major factors to reduce it. Management has to be convinced with such plans as they introduce real benefit to the workflow.

To summarize, most of the obstacles, if not all of them, are not technical. These obstacles are divided into the following major categories [SD96]: -

1. **Organizational**: where there is an additional effort to catalog, archive, and retrieve reusable components.
2. **Economic**: where reuse may cost a lot of money and delay to some projects.
3. **Political**: such as not wanting to share components with others or cannot use free components.
4. **Psychological factors:** such as not wanting to use components that are not built in-house.

Reuse could be achieved on a narrow scope but it is very difficult to distribute among all the development community.

2.2 Extracting Reusable Components

Reusability can be an expensive operation if it is not well planned. Software must be evaluated to know the data and process components that should be created as reusable components. The future projects that could be created from reusable components have to be identified and the portions that have high potential of reusability must be highlighted. Moreover, business divisions that heavily practice reusability can be identified and given more focus on their systems [MC94].

The second technique for capturing reusable components is domain analysis. The objective of domain analysis is to capture and model the information that describes a domain for the purpose of gaining a better understanding of the domain and/or to use this information to develop future systems for the domain from sharable reusable components such as generic architectures and processes [MC94]. Systems that support a certain domain, or a portion of a domain, are a good area to perform domain analysis. For example, billing systems for telecommunications companies could be very similar and hence worth building a system that uses common components.

Domain analysis can be conducted either as a bottom-up or a top-down approach. The Bottom-up approach works by studying the old or current systems to get the common components. It is a reverse engineering exercise but focuses only on common components. The Bottom-up approach gives only half of the picture. In other words, extracting common components from existing systems, make them available most of the time for future versions of these systems. To complete the picture, a top-down process is conducted as well to figure out components that can be used in future systems. Hence, common components can be studied to discover a generic version [MC94].

C h a p t e r 3

DESIGN PATTERNS AND RELATED TECHNOLOGIES

3.1 History of Design Patterns

Design Patterns were first introduced in architecture engineering. Alexander [CA79] in 1979 introduced the concept in his book, *The Timeless Way of Building.* His definition of design pattern reflects his engineering thinking point of view. He says, "Each pattern is a three part rule, which expresses a relation between a certain context, a problem, and a solution." By configuring context, problem, and solution with forces linking them together, they make their pattern [CA79]. He gives another definition "'Each pattern describes a problem which occurs over and over again in our environment, and then describes the core of the solution to that problem, in such a way that you can use this solution a million times over, without ever doing it the same way twice". Although he wrote his book for architecture engineering, yet it became much more clear that its effect was found useful in software engineering as well.

Later, in 1987, *Kent Beck* and *Ward Cunningham* published a technical paper describing how they used Alexander's concepts of patterns to accelerate the development of user interface in one of their projects [BKCW87]. They proposed five patterns: -

1. Window Per Task
2. Few Panes Per Window
3. Standard Panes
4. Short Menus
5. Nouns and Verbs

They gave these patterns to a specialist team that were writing technical specification for a system. The good thing about these patterns, as they said, was that the team after only one day presented reasonable interfaces. The pattern language they presented detached the technical team from full knowledge of the programming language they were using, SmallTalk. It showed a success that made them decide to write more patterns. They reported that they finished 10 patterns, sketched out 20-30 more, and expected to have up to 100-150 other patterns

[BKCW87].

Meanwhile, Erich Gamma was preparing for his PhD which was on capturing patterns. He consolidated his ideas with some others Richard Hel, Ralph Johnson and John Vlissides. The concept of design pattern started to become popular when these four first published their famous book, *Design Patterns Elements of Reusable Object-Oriented Software* [GEHR95]. The book is one of the most famous books in object oriented design and is considered a bible for almost all researchers in this field. Design patterns do not come with just thinking, they need real experience. However, designers, usually, don't do a good effort in recording their experiences for others to use. [GEHR95]

3.2 Capturing Design Patterns

It is really difficult to capture a design pattern. Although, novel designs could be created from scratch, a design pattern has to come from experiencing a design and proving that it is worth using with other projects. A novel piece of design could be very successful in one application but it may fail in another. So, a design pattern will not be captured unless it is used in more than one project inside the same domain or other domains. These patterns need to be documented for future use.

A practice that is really good in building a pattern is to make review sessions for a team of developers who are working in the same phase of a project, analysis, design, implementation, or testing and to consolidate their work into a common design if similarities are found across their work. The technical team leader, for example, could review the teamwork pieces and build, with the help of the team, a new design that is generic enough to capture all the team work commonalities. Afterwards, every member of the team has to adapt the new design. The teamwork review helps greatly in building reusable patterns that could be used at least in the limited scope of one project and increases the possibility of using them in other projects.

3.3 Describing Design Patterns

Is it difficult to describe a design pattern? An expert software designer might say, No, it is not difficult. There are graphical notations that can tell exactly what a piece of design is doing. However, are the graphical notations enough? How can one tell if this design pattern is best suited for an application without knowing the tradeoffs, design decisions, or the alternatives that the developer encountered? It is more important to know such information before deciding to use a design pattern. So, the graphical notations are not enough, literal description is important in this case. But, the graphical notations could support the literal description too [GEHR95].

The following are the basic elements that are needed to describe any design pattern [GEHR95]: -

1. **Pattern name**: Naming a pattern is one of the most difficult tasks of describing a pattern. A good pattern name will describe, to a large degree, its intent.
2. **Problem:** When to apply the pattern. This section should encompass the problem and its context. One may add other pieces of information that add value to this section like class or object structure, conditions that must be met before applying the pattern, or other design problems.
3. **Solution:** The elements from which a solution is composed. It must not to be an absolute solution, however; it should be abstract enough to be applied in different situations. The actual design should be left to the developer
4. **Consequences:** The results and the trade-offs from applying a design pattern. Although, they might not be noticed after applying a design pattern, consequences are important in order to evaluate different design patterns. The results and trade-offs may address different areas, like design flexibility, extensibility, portability, or programming language specific issues.

3.4 Selecting a design pattern for use

There are no well-known search engines for design patterns. One has to strive to find good patterns for his/her design. It is not mandatory that every project has to be built from design patterns; however, a user can select certain sections of his/her past designs and merge them with patterns. These sections could be fully or partially built over patterns. This kind of reusability ensures: -

1. The high possibility of a rapid and successful overall architecture of a system.
2. The existence of documentation that is already available on the used pattern.

So, to start selecting a pattern, one has to start first by narrowing down his/her needs and comparing different alternatives. One may follow these guidelines to select a design pattern (Figure 2) [GEHR95]: -

1. Know exactly what you are trying to find. Developers may be tempted to search in the patterns catalogs that are related by name to their work. This kind of search may succeed and may not. So, try first to categorize your work and put related words ahead and then search for design patterns related to these categories. For example, developers who are designing web applications may face a problem of building or designing heavy weight pages. The developer's search concern may go first to design patterns of HTML pages. However, his/her solution could be related to performance issues.
2. Once you defined some design patterns, start comparing their problems and solutions to the one you have. One may accept a design pattern that partially solves a problem just because it is promising for better enhancement of the system rather than a pattern that solves the problem but does not consider variable changes [GEHR95].
3. Compare the consequences of the selected patterns. Review the consequences of each pattern and build your selection criteria as you go over the trade-offs.

Finding a good design pattern to work with could be difficult to the extent that the developer will take a decision to build the design from scratch. However, a limited amount of time could be allowed for searching for design patterns at the beginning of each project. This effort, even if it does not come up with selected patterns, will help the developer in the way he/she thinks and add more to his/her knowledge.

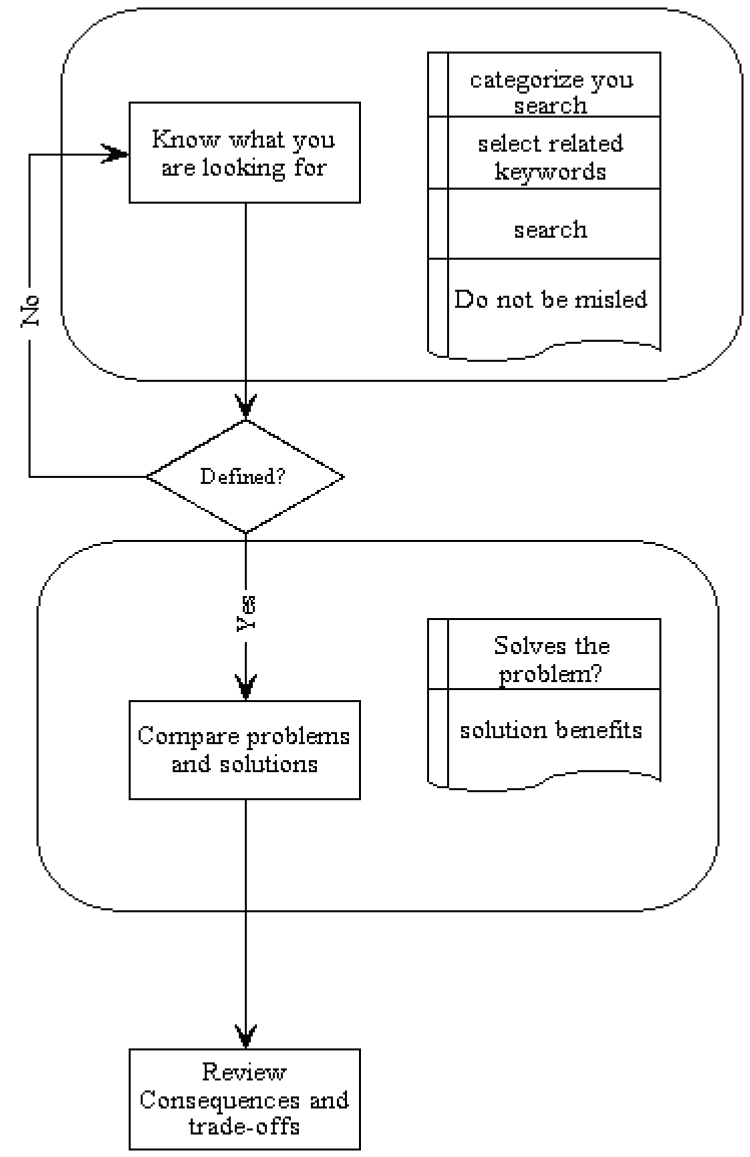


Figure 2 Selecting a design pattern

3.5 An Example of A Design Pattern

Design patterns fall under three categories, *Creational, Structural, and Behavioral.* Creational and Behavioral patterns describe objects at runtime. Structural patterns describe object structure at compile time, *Class Declaration* [GEHR95]. For example, virtual constructors in C++ are an example of a creational pattern where construction is determined at runtime. The appropriate object constructor will be called according to the object type. Inheritance is an example of a structural pattern. Class inheritance forces some methods and attributes to the child class. A good example is the *Object* class in Java. The *Object* class declares some methods that any class declared must inherit from. For example all classes in Java inherit *toString()* method that returns a string representation of an object, *equals(Object)* that indicates whether some other object is equal to this one [JAVA1.3].

Behavioral patterns are the most commonly used ones in object-oriented design. They actually can embody the structural and creational patterns in their implementation. Here is a very good example of one of the design patterns that is used to break down the difficult algorithms into small pieces of organized and structured object-oriented code. The pattern captures a certain inherited feature in some of the sorting algorithms and builds object-oriented patterns that consolidate their behavior. These algorithms that depend on the divide-and-conquer algorithm start by splitting into sub-arrays, sorting, then joining (Figure 3). On top of these algorithms are quick sort and merge sort. The insertion sort could be viewed as a special case of merge sort where splitting is done with one element at every pass [NDWS01].

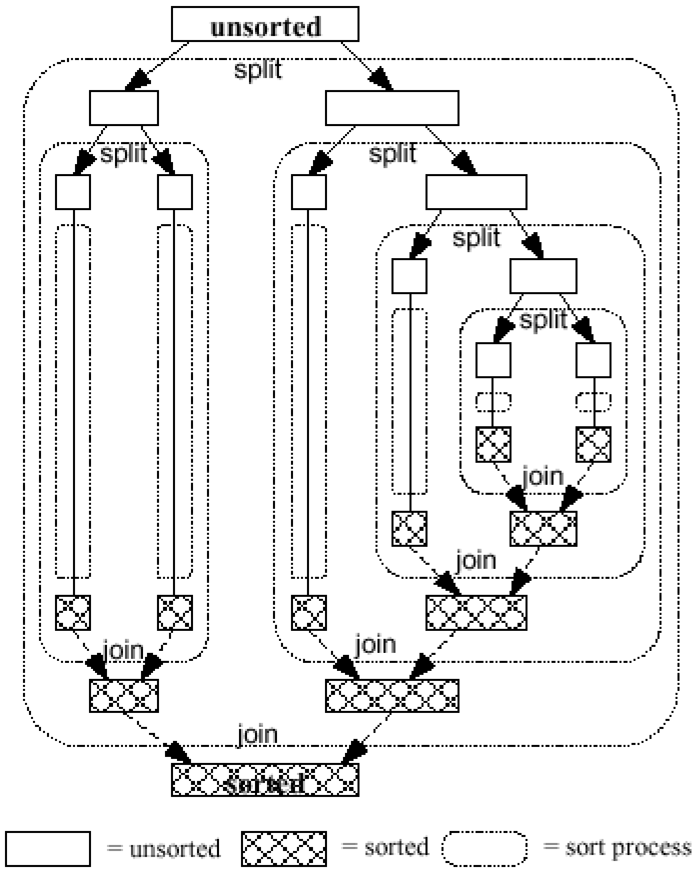


Figure 3 Hypothetical Sort Recursion Tree [NDWS01]

The main design decision taken here is that: -

1. All the studied algorithms share the same concept of splitting, sorting, and then merging.
2. The sorted data does not carry its sorting criteria. It is the responsibility of the user to provide this information through *AOrder* object.

The pattern represented here is composed of two classes. ASorter that abstracts the behavior of the sorting algorithms and AOrder that carries the ordering criteria for the sorting. Both of them could be overridden to give concrete examples. For example, quick sort can inherit from ASorter to give the needed functionality for quick sort [NDWS01].

An added value to this pattern is that it allows the user who wants to monitor sorting to add graphics animation without changing the code. This is intrinsically the nature of well-designed object-oriented design. So, a decorator pattern can extend the functionality of one of the sorting algorithms to show how sorting is done. Another monitoring pattern can extend AOrder to measure performance. Measurement is dependent mainly on counting the number of comparisons (Figure 4) [NDWS01].

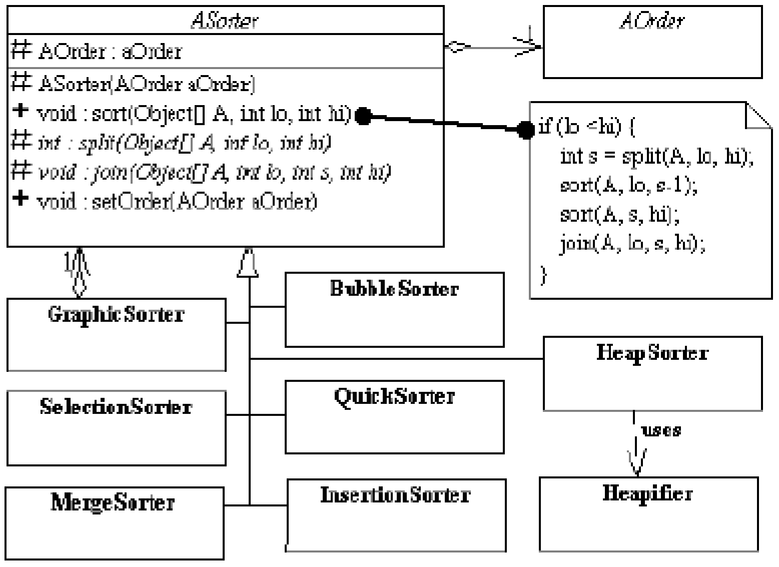


Figure 4 The Template Method Pattern for Sorting [NDWS01]

3.6 Pattern Language

A pattern language is a collection of patterns that guide the developer in a certain domain context. It is not just a catalog of patterns; it includes design decisions and domain-specific advice to the developer. The choice of the appropriate pattern will be determined from the advice of the preceding pattern [MBMT01]. It is very important to note that a pattern language is not just a collection of isolated design patterns, they are heterogeneous pieces of design that give, at the end, a good solution for a certain problem. In other words, if a design pattern is selected for implementation, then succeeding design patterns in the tree (Figure 5) are the only candidates for future implementation. Figure 4 is a conceptual view of the pattern language which represents a directed tree of patterns.

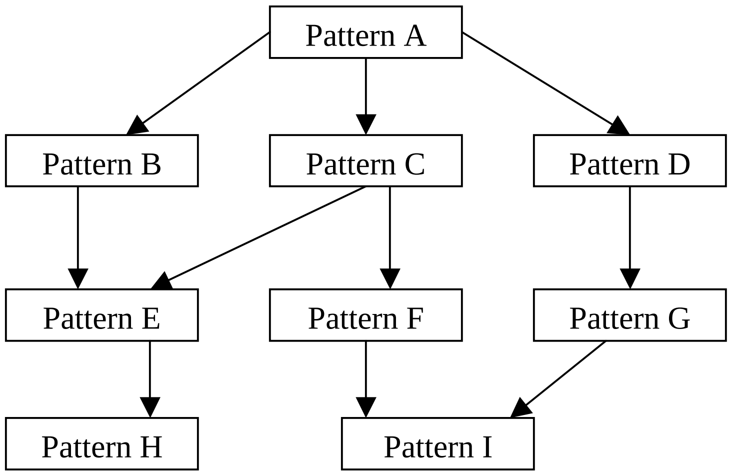


Figure 5 A conceptual view of a Pattern Language

An example for a pattern language is the one proposed for parallel application programming [MBMT01]. The pattern language is structured into four main categories that will execute in sequence. The top-level patterns help in finding the components that can run concurrently. The lower-level patterns help the programmer to use parallel algorithms to express concurrency. The lowest levels are concerned with parallel environment's primitives [MBMT01].

As mentioned before, the top-level is concerned with finding the components that will run in parallel. This is supposed to be a higher-level decision and will be removed from the lower level implementation. The path starts by organizing the problem information on hand. The next step is to decompose the problem either by task (TaskDecomposition) or by data (DataDecomposition) or by both (Figure 6). The designer may keep oscillating between them until a final decomposition is made. The next step helps the designer to 1) group tasks together whose elements will run concurrently 2) order the group of tasks according to their data or temporal dependency on each other, and 3) decide on how data will be shared among tasks. The last pattern consolidates the designer's use of the previous patterns and makes sure that they are efficiently used [MBMT01].

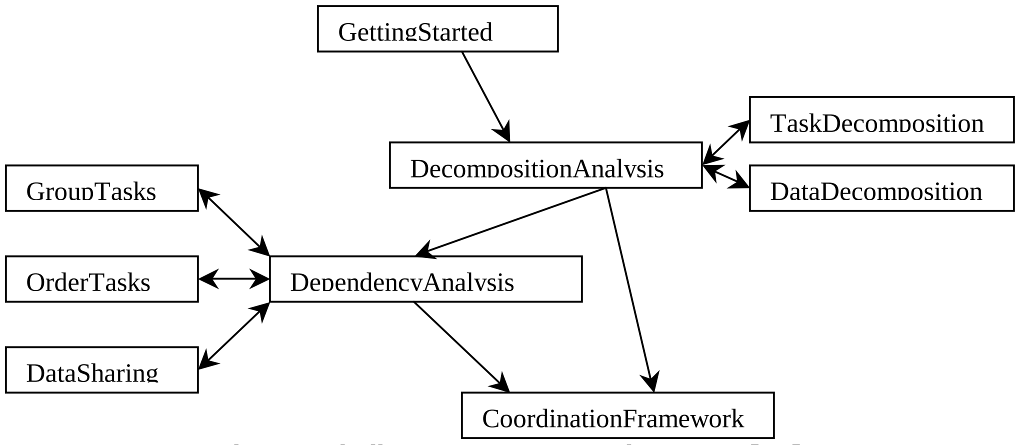


Figure 6 Finding Concurrency Design Space [24]

3.7 Anti-Patterns

An anti-pattern is a pattern that tells one how to go from a problem to a bad situation. The bad situation is revealed by its side effects where the original problem is solved but unseen or un-anticipated problems are generated. They may be tiny problems, but they will need time and effort to solve and will disturb the focus of the developer from his original problem [BWMR98].

Have we ever faced such problems? Yes, and yet we always tend to think of our solutions as the best. We may categorize anti-patterns as: -

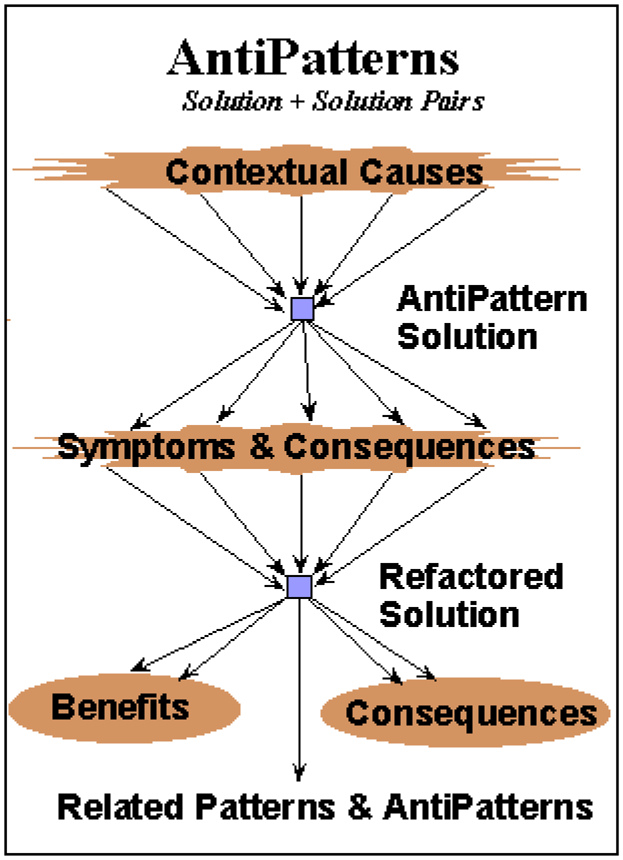


Figure 7 Anti-Patterns [BWMR98]

- Those that tell a bad solution to a problem that ends into a bad situation.
- Those that tell how to get out from a bad situation into a better solution.

So, if a pattern is defined as "best practice", then an anti-pattern represents a "lesson learned" [AB00]. It is really important to record the anti-pattern practice and show its counterpart pattern that proves to work well. Actually it could be more than one pattern that helps at the end in getting out of the problem. Figure 7 shows how an anti-pattern can converge to a better solution. From a practical experience, developers tend to fall in the trap of anti-pattern of the first category when they mainly: -

- Focus only on the context on hand and not try to generalize a pattern.
- Do not anticipate future changes.
- Do not propose their solutions for others to criticize.

So, why is it worth knowing an anti-pattern? Like design patterns, anti-patterns provide a general template where the problem, symptoms, solutions, and consequences are described clearly. This means that software developers will have a common vocabulary that they can use to talk about software defects. As the knowledge base of anti-patterns grows up, the software industry's common problems get clearly identified and shared among the development community. Since an anti-pattern is forced by a certain problem, it could be shared among a number of developers who have to communicate the problem with each other [BWMR98].

Let's look into an example that represents the first class of anti-patterns "Those that tell a bad solution to a problem that ends into a bad situation." Following that, a solution will be given to see how to get out of the bad situation into a better solution, the second class of anti-patterns. Note that the purpose of presenting the anti-pattern example is not to give an ideal solution to a problem.

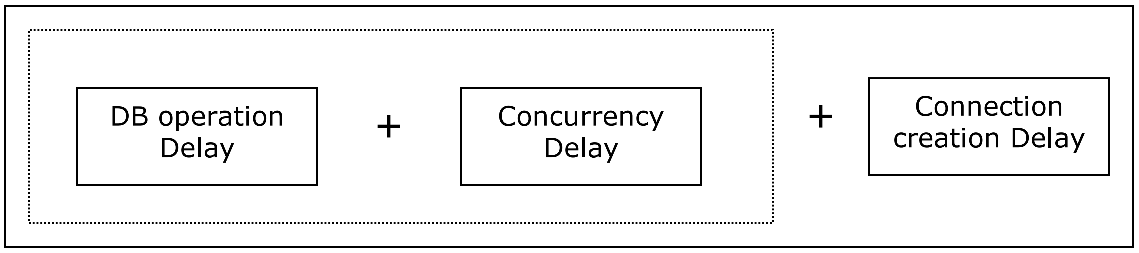


Figure 8 Time Delay for a small application

However, the purpose is to get close to the picture and try to understand it. So, the use of the anti-pattern template will not be thorough. It is enough to describe the problem, the bad solution, and how to get out of the bad situation. Moreover, the example here will be useful later on when we talk about connection pooling.

Consider an application that is dependent on a database. The application uses only a single connection to do all the database operations. The connection is never closed until the application exits. Every operation is taking a considerable amount of time. There is no problem if there is only one user who is using the whole application. However, when more than one user use the application, a time out problem occurs. This is normal since there is only one connection that could serve one operation at a time and every operation takes a considerable amount of time. The developer who designed the application thought of fixing the problem by creating a new connection for every request, the same way it is done for the single connection!

It is really a bad solution. Now there will not only be a delay from the concurrency and the operation time, but another overhead time will be added from the creation of a new connection (Figure 8). Actually this solution has a very little benefit in comparison to its side effects. The developer thought of solving the time out problem, which is mainly due to concurrency, by introducing new connections every time there is a request. This solution will serve the requests for a limited period of time then the database connections will be consumed and the time out problem will appear again. That is because the user repeated the solution of opening a connection and releasing it while the application exists. Moreover, memory consumption will be very high on the database side and on the application side.

Is there a solution to this bad solution? The smart developer said, "How about creating a request for every user". Now, the database connections will be left free for a considerable amount of time and they may not be totally consumed. However, the developer still does not have control over the stability of the system. First, the number of users could be at one time higher than what is expected. Second, the database might be configured with a number of connections less than what is required. Also, the memory consumption might touch the dangerous edge.

The developer is still in a dilemma. "Yes, I should close the opened connections". This for sure will solve the time out problem due to the consumption of the database connections and so the memory usage will go a little bit down. However, the connection creation delay will still be introduced.

This iterative thinking could lead the developer at the end to maintain a connection pool. This solution has many benefits. First, time out due to the consumption of the database connections will not occur. Second, memory will always be under control, since it is only a limited number of connections deteremined from the start of the application. Third, the user will not suffer the connection creation delay. Fourth, the concurrency delay will be controlled to a large extent since the users will be competing over a pool of opened connections, not a single connection.

To sum up, developers usually start with solutions that do not represent the best practice, especially if these solutions are new to them. The experience really makes a big difference in their decision. Experience would make one developer try only two or three iterations to arrive at the best solution, if it is not reached from the first time; and would make another developer reach for his/her solution all at once and not need to go over it at all. Experienced developers, however, will normally tend to compare alternatives to select the best for their problems.

3.8 Frameworks

A Framework is "a set of cooperating classes that make up a reusable design for a specific class of software" [GEHR95]. A framework is always geared into a specific language where it could be reused directly to dictate the architecture of your application. Although the framework forces a certain architecture, it relieves the developer from the main design decisions that are already managed in the framework.

Because the applications built over the frameworks are fully dependent on them, they are also sensitive to changes of the general design. Actually, here comes the main advantage of the OOD, where many of the design decisions could be hidden from the developer. The developer will implement the interfaces proposed by the framework to do his/her special work.

3.8.1 How Frameworks solve the reusability problem

The framework gives the developers certain guidelines to implement certain methods and declare certain variables so that the framework could call them. Such inflexibility limits the developers' creativity, but it lets the developer concentrate on the problem on hand [GEHR95]. This is why a developer chooses a framework. It is in contrast to the toolkits concept where there are ready-made classes and objects that could be called to achieve certain functionality.

Frameworks are mainly focused on Object Oriented Technology. They Enforce flexibility, where you can override certain behaviors, or configure default behaviors. Moreover, certain design decisions could be deferred to the application developer where he/she can put his/her solution with the support of the framework.

3.8.2 Examples of Frameworks

One example of frameworks is the SanFrancisco framework developed in IBM labs [CJCB00]. This framework is one of the examples where design patterns are used as an effective element. The development team believed from the beginning that the framework will not solve all problems, so they provided the application developers with a hierarchy of solutions from which they can choose the ones to work with. The team developed a design pattern which supports directed design decisions, providing the ability to:

- Configure a default behavior for a domain algorithm.
- Override that default behavior on an object-by-object basis.
- Make performance vs. flexibility tradeoff decisions on an object-by-object basis.

Another example is DIWB (Distributed Interactive Web-site Builder) Framework [EESW00]. The framework itself is built over two other frameworks: *WebObjects* and *OpenStep*. The first framework is specialized in web application development. The second one is focused on data management. The framework uses a different approach, or if we can say design pattern, for development. The project team decided to build it over 5-model architectures instead of the famous MVC model. The 5 proposed modules are: -

- **Presentation layer:** to prepare data for display on the web browser.
- **UI Components:** to provide UI components for display by the Presentation layer
- **Business Logic:** to provide the services that could be called by the UI Components in the presentation module.
- **Data Management:** to provide a data management system away from the database

- **System Infrastructure:** to provide low-level administration of hardware, network, and operating system.

They argue that this model is better since it decouples the business layer from the presentation and system layers.

Another framework that has been used in commercial domains for some time is *Expresso. Expresso* is a project which aims to build a framework for web applications that are database driven. It is based on the **MVC (Model-View-Controller)** architecture. It provides a lot of basic functionalities that are always needed like security, login, database connectivity, caching, etc… As a powerful framework it integrates its model with S*tratus*, which is an open source framework, to build its **controller**. Its **Model** can integrate with any standard database. The view can utilize presentations systems such as Java Server Pages, Velocity templates, and XSLT [JCORP].

Expresso, like any other enterprise framework, allows the developer to gain a lot of benefits since he/she will focus on the business logic without having to go into building code from scratch. This is very useful and gives the development company a lot of benefits like [JCORP]:

- A Shared Standards Based
- Lower Costs
- Simplify Development
- Faster Delivery and Time to Market
- Promote Consistency through Code Reuse
- Promote Quality
- Provides Vendor Independence
- Greater flexibility for customization to requirements
- Improved Web-based collaboration

3.8.3 How Design Patterns integrate with Frameworks

The problem that always faces developers when designing Frameworks is the separation between the static and dynamic parts. In other words, the developers would always want to make a framework as generic as possible, but this can never happen. There has to be a static part that the framework is fully dependent on. The problem is that framework developers may not be able to satisfy the different business needs where dynamic and static parts are exchangeable.

The role of design patterns appears here. Use the design patterns to decouple the static parts from the dynamic ones [SDC96]. Some of the most useful patterns emerge and describe frameworks. Since frameworks are language dependent, design patterns are used to abstract high-level architectures. The usefulness comes from reusing these pieces of patterns inside the system or in any other system. In other words, design patterns could be used to describe frameworks in a language-independent manner, and frameworks are used to consolidate different design patterns in a language specific manner. Neither design patterns nor frameworks precede each other. One could start with a framework then use the design pattern methodology to describe it. Or a framework could be built around design patterns that solve problems in a specific domain [GEHR95].

If a framework needs more knowledge and training in order to master its techniques, then the design patterns are the way to understand it. Frameworks are in between the toolkits and design patterns. They are more abstract and flexible than the toolkits, but more concrete and easier to reuse than design patterns [JR97]. Many developers do not consolidate the understanding of a certain design until they see it implemented. So, if you want to understand a certain design pattern, check its implementation. It is important to notice that design patterns usually achieve success in frameworks and a pattern language could be extracted from a framework design.

Chapter 4

WEB APPLICATIONS

4.1 A Brief History of the Internet

The origins of the internet goes back to 1962 when J.C.R. Licklider of MIT started to propose the idea of "Galactic Network" in a series of memos. He described his *Galactic Network* as a set of interconnected computers where every one can access any program or file from any site. Licklider convinced two of his colleagues in DARPA (Defense Advanced Research Projects Agency), as he became the first head of the computer research program, as well as Lawrence G. Roberts from MIT with the importance and real need of this network [LBCV03].

The real network communication researching started when Leonard Kleinrock at MIT published his paper on packet switching in July 1961. The packet switching idea was revolutionary as circuit switching was the primary switching method available at that time. Roberts, working with Thomas Merril, created the first wide area computer network ever built when they connected two computers, one in Massachusetts and the other one in California. The experiment was very successful as both computers were able to share programs and files remotely. However, it was found that the dial-up telephone line connection that was used in the experiment was not efficient in connecting both computers. Roberts was totally convinced that packet switching would be able serve as an efficient way of communication [LBCV03].

ARPANET was the first network to be designed over packet switching in 1966. With the help of other researchers in England, ARPANET upgraded its line speed from 2.4 kbps to 50 kbps. A team from Bolt Beranek and Newman (BBN) started to develop the first version of packet switches called Interface Message Processors (IMP's) in December 1968. The first host computer in ARPANET was installed in the Network Measurement Center at the University of California Los Angeles (UCLA) accompanied with the installation of the first IMP in September 1969. The second node was installed in Stanford Research Institute (SRI). By the end of 1969, four computers were connected, as two more were added from UC Santa Barbara and the University of Utah. These universities were also running researches on the network infrastructure and utilization, which is still a topic of research until today. By 1972, more sets of computers were added to complete the physical architecture of the ARPANET. The first Host-to-Host protocol was implemented too, called Network Control Protocol (NCP). By that time, users were able to start developing applications [LBCV03].

In 1972, BBN introduced the first application, electronic mail that is installed on ARPANET. The software was developed to let the developers on ARPANET communicate easily. It was capable of sending and reading messages. This utility was expanded to list, selectively read, file, forward, and respond to messages. Email, which is the largest usage Internet application, took off [LBCV03].

ARPANET was the seed for the modern Internet as known today. However, the internet had a much more different and diversified structure of the ARPANET. As ARPANET was a reliable network that has no interference with other network, there was no error detection built on the communication protocol used (NCP). Error detection was primarily needed to allow different networks, which might have different structures, to communicate efficiently. So, Kahan decided to build a new protocol suitable for the Internet which eventually was called Transmission Control Protocol/Internet Protocol (TCP/IP) [LBCV03].

The widespread development of PCs, LANs, and workstations from the beginning of the 1980’s followed by the development of Domain Name Systems (DNS) and later followed with the development of different routing algorithms, made people think of transferring its usage to the host software, especially operating systems. Thus the Unix operating system was first adapted to support TCP/IP which is implemented now in all the well known operating systems [LBCV03].

4.2 Web Application Development Lifecycle

The development of a web application involves contribution from many fields. In addition to the technical perspectives, it includes managerial, organizational, most probably artistic, and may be social contribution. When there are many contributors, there are many dilemmas. An application could span more than 3 months just because the management sees that artistic graphics are not satisfactory or colors are not that tasty. In addition to the technical issues and problems, a web application may last for more than one year for average size projects [FP99].

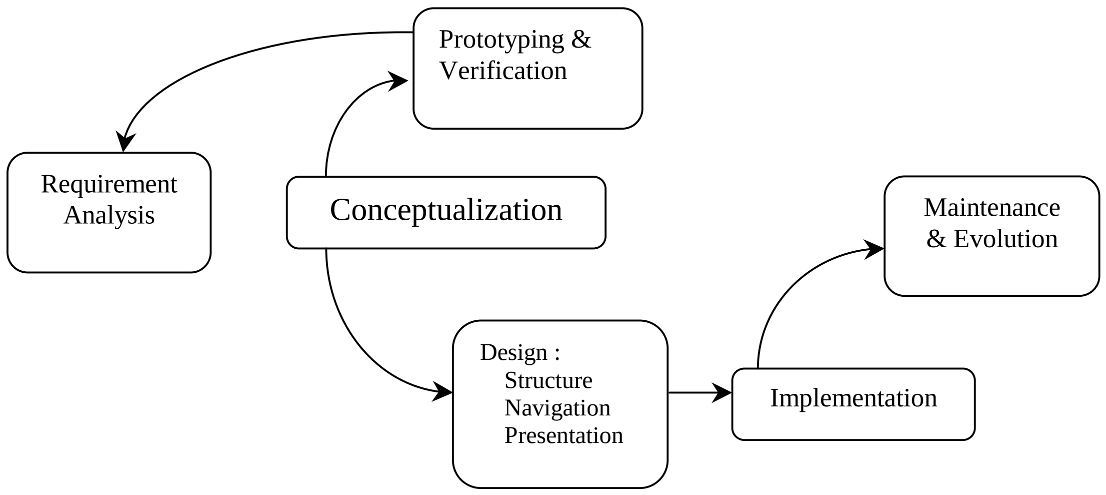


Figure 9 The Life Cycle of a web Application [FP99]

The diagram in Figure 9 shows the most commonly used architecture for a web application. As noted the iteration of requirements analysis, prototyping, and conceptualization may repeat several times until the customer is satisfied and concepts are consolidated [FP99].

In addition to the normal work in the Requirements Analysis phase where users are identified, the nature of information is defined, and the customer view of the application is established, a web application adds more to it. A system analyst should identify the type of audience that will interact with the web application and their expected behavior with more than one use case. He/She may target more than one output device (PC, Palm, Mobile, etc) [FP99].

Conceptualization is primarily different from that in normal information systems. The system analyst has to identify the objects and relations recognized by the customer regardless of their representation on the system. This does not mean that the notations that the developer has become used to will change, he/she will still continue to use the same notations to build his/her solution [FP99].

Prototyping and validation makes the user visualize the GUI of the application. This is a simplified version of the web application which is built very quickly to capture user requirements for the interface. A Prototype can be built over a small context of data to get a general understanding of the web application. However, it could also be a very good method to agree with the customer on the final product appearance and functionality [FP99].

Design is responsible for transforming concepts into lower-level representation. It prepares for the implementation, where it defines the architecture of the whole application. The architecture of the database is defined, representation classes and functions of the code are designed, without going into details, and the GUI is finalized, regardless of the content. Ideally, the phase should be completely and accurately designed before going into the implementation phase. However, it is almost impossible to build a final design from the first time, it will most probably change in the implementation phase [FP99].

Implementation will use the delivery from the design phase with samples of contents, provided from domain experts, to build the final product using (e.g. HTML, Java, ActiveX, CGI). From now on, developers will speak most of the time about lower-levels of detail. One of the major risks that may make a design change in this phase is the introduction of content. Actual content may greatly impact the database, and the GUI, as well as the code [FP99].

Evolution and maintenance is a normal process phase that is always practiced after every product. Users may change their requirements again, bugs may appear in areas that were not carefully tested, or performance enhancements have to be applied. However, this phase should not last too long. A good web application should remain stable for a good period of time before applying new changes [FP99].

4.3 Analyzing A Web Application Performance

4.3.1 Quantitative Analysis Cycle

As web applications are becoming an interaction media between the customers and the companies, performance becomes a very important factor that defines customer experience. To ensure a web application performance, it is required not to focus only on the application but also on the customer connectivity. The following is an 8-steps Quantitative Analysis Cycle of an E-Business Site [GRSA01]: -

1. **E-Business Site Architecture**: where the participating components of software, hardware, and networks are specified.
2. **Measure E-Business Site**: Performance measurement is needed from different perspectives. It has to be measured internally (application side) and externally (customer side).
3. **Characterize Customer Behavior**: Build profiles for customer classes that interact with the site. This allows the site owner to detect the resources needed by every class.

4. **Characterize site workload**: Every class of users place different load on the site. To quantify workload, resource demands, visit frequency, site navigation, and any other load parameters that need to be described precisely.
5. **Develop Performance Models**: Performance Models will help in detecting site performance under different variant changes of configuration, application, network connectivity, or work load levels.
6. **Obtain Performance Parameters**: These parameters are used as direct input to the performance models. Some of the parameters can be directly measured like the I/O time, CPU speed, etc … Other parameters need to be estimated like the bandwidth between remote client and the site.
7. **Forecast Workload Evolution**: Anticipating the future workload is a very good practice. It allows the site to avoid dramatic failures due to unknown problems especially on hot events where workload is expected to be very high. For example, a telephone company may expect very high workload every month due to bill release.
8. **Predict E-Business Site Performance**: If performance models are capable of detecting future site workload, changes in the site architecture may fail this model. So, architecture changes could be evaluated to detect the most cost-effective way to satisfy future workload.

4.3.2 A Response Time Reference Model

A Response Time Reference Model describes the download process of one website page. It is composed of eight stages. Every stage is a component time. The summation of all the component times is the response time for a website page. Two stages are not used in this model as they are not participating directly in the response time of website page request [LCGR00]: -

1. **Establishing Internet Connection**: which is mainly the responsibility of the user to get into a network that is connected to the Internet.
2. **Starting a Web Browser**: It is also the responsibility of the user to get a web browser started.

The reference model's response time components are [LCGR00]: -

1. **DNS Lookup:** Where domain names are translated into their target IP (Internet Protocol) addresses. The Browser sends a request to the Internet

Service Provider's Domain Name System (DNS) to translate domain names into their target IP addresses.

2. **TCP Connection:** After a domain name is resolved, a TCP request is sent to the web server to establish a connection with it.
3. **Server Processing:** After the connection is established, a request is sent to the web server to get the request page from the user. If the request is sent to get a static web page then the web server serves it directly. In case of dynamic pages, additional processing is needed to generate the page. The additional processing may include other applications like application server or database.
4. **Redirection:** It happens when the requested page is not fetched, but a replacement page is sent to the browser. This process could be repeated several times until the final response is sent to the browser. In other words the eight steps that we are talking about will be performed every redirection time.
5. **Base Page Download:** After the page is fetched it is sent to the browser.
6. **Content Download:** The browser then examines the fetched page to check for the embedded graphics or objects. A request for every object is sent to the web server in order to download it.
7. **Page Rendering:** The client browser may take additional time to render the requested page.
8. **User Interaction:** The user will then take some time to consume the content of the page and then decide on what to do next.

Table 3 shows the knowledge areas which one needs to gain in order to fully understand the performance model [LCGR00].

Stage	Domain Knowledge needed to optimize performance
DNS Lookup	DNS/BIND
TCP Connection	Networking, TCP/IP, Internet connectivity
Redirection	Web site implementation, HTTP
Server Processing	Web page generation: Systems design, Systems integration, Web servers, Application servers, Database Servers, Middleware
Base Page Download	Web page construction using HTML, scripting
Content Download	Graphics, content distribution networks
Page Rendering	Web browser operation, Client OS tuning
User Interaction	Human factors, User interface design using HTML

Table 3 Knowledge Needed to optimize performance in every stage

4.3.3 Hardware & Networking

A web server uses TCP/IP connectivity to transfer HTTP requests that come from the client. TCP/IP connection could be viewed as a black box where a request comes in and a reply goes out. However, the process contains more details on the networking and hardware level which are as follows [PFDL96]: -

1. A TCP/IP connection is created according to a certain request received from a client.
2. The server CPU is interrupted by the network hardware to serve the new request.
3. A new http process is created to serve the new request.
4. The http process accesses the secondary storage and retrieves the requested data from the requested file.
5. The response data is sent through the TCP/IP stack where it is sent as packets.
6. The http process runs the network device driver to pass the reply to the network hardware. The http process will remain active until the reply is sent completely.
7. The requested data is placed on the local network. Bandwidth, routers, and network links contribute in the response time that is seen by the client.

As seen from the above scenario, CPU, Memory, secondary storage, and network bandwidth are the main bottlenecks. CPU could have many other tasks to do other than those related with the http process. So, 100% CPU utilization will limit the http connection rate. If there is no limit on the number of http processes created, more memory may be needed. However, it is not a big problem as computer memories can be very large nowadays even for personal computers. The secondary storage may represent some delay if there is a problem with the requested file I/O. However, it is not likely to happen as the I/O operation usually exceeds the network bandwidth. File-cache solutions do solve, to a large degree, the I/O bottleneck [PFDL96].

Hardware redundancy can be a good solution for heavy loaded web applications. Hardware redundancy means that if a web server is located on one machine, another web server could coexist to share the responsibility with the current one. Hardware redundancy can be on the machine level where more CPUs could be added, if possible. Or it can be a complete load balancing solution where more than one machine can share responsibility. Sharing responsibility could be done on a web server, application server, or database server. However, hardware redundancy is not always the good thing to do. The web application revenue has to be measured against the new hardware. If a web site is receiving only 100 requests per day, then it may not need to have a load balancing solution. In other words, an accurate assessment has to be done before deciding on using hardware redundancy.

4.3.4 The Server Side

A web application will not run without both the client and the server. The user usually writes down the website URL and waits for the results. The web application is more than just a web server with static HTML pages. It can contain a logic that many parties will share until it goes on duty. The common architecture for a web application is composed of: -

1. **HTTP Web Server**: which is responsible for receiving client requests and responding back with the requested information.
2. **Application Server**: which is responsible for hosting the web application and managing the requests that come from the web server.
3. **Web Application**: which is the business logic that will decide on the type, and amount of information that the user will receive.
4. **RDBMS**: where information is stored to be retrieved later by the web application.

Other components could be used with the application server as needed (

Figure 10). For example, the web application could utilize a reporting server that fetches information for the database and generate HTML reports with charts. A search engine could be utilized as well to search in a file system or in a database. The developer can introduce other components as needed. Every added component can contribute to the efficiency of the web application and to the latency of the web application as well.

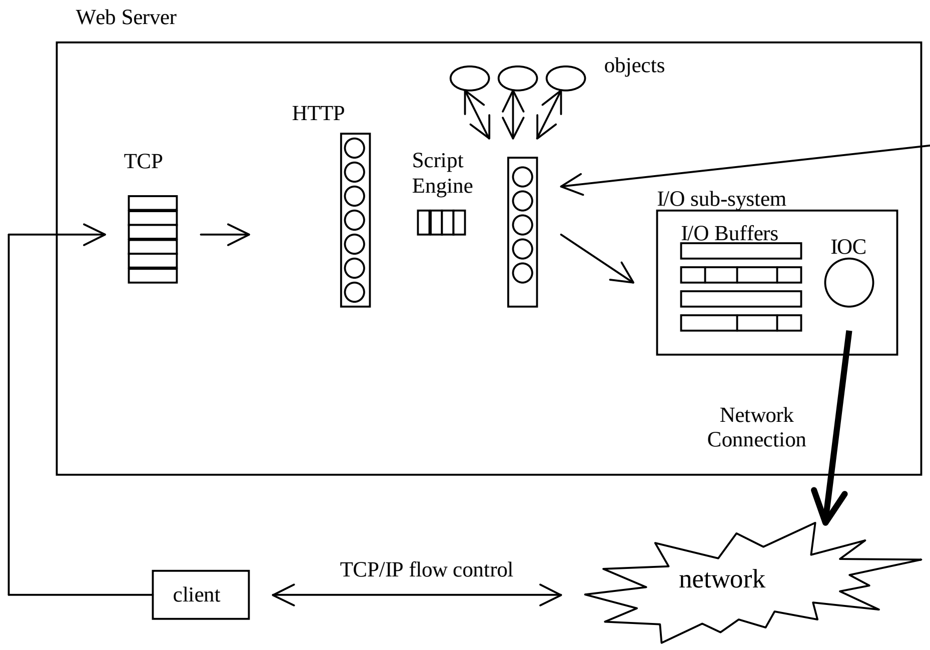


Figure 10 Model of the transaction flows within the Server Side [MREW00]

4.3.4.1 HTTP Web Server

The main job of the HTTP web server is to respond to user requests that are usually in HTTP format. The web server is ideally a file server that responds through a predefined protocol (HTTP Protocol). Whatever the request is, the web server will contact other participating servers to build up the page, in case it has dynamic content, or fetch it directly in case it is static. Each HTTP request proceeds through four successive phases: TCP Connection, HTTP Processing, SE (Script Engine) Processing, and Network I/O Processing [HREW00]. SE Processing is optional if there is no dynamic content.

TCP connection is a two-way connection established between the client and the server. It consists of a TCP Listen Queue and server daemon that picks a connection to serve the client. The HTTP sub-system is a process that listens to a predefined port. It receives the client requests through the TCP connection. HTTP sub-system consists of a Listen Queue and a number of threads that coordinate the processing performed by the worker threads. An SE sub-system consists of Listen Queue and a number of handlers that are responsible for interpreting the scripts (e.g. C++, Java, Perl, etc…) and may interact with other backend systems like a database. The HTTP sub-system will send the formatted data as received to the client. The I/O sub-system consists of a number of I/O buffers, I/O Controller, and the connection from the server to the network. This sub-system is highly dependent on the operating system [HREW00].

Tuning the HTTP web server could be of great help in enhancing the performance of a website. There are a different number of HTTP servers that have different configuration settings. However, they almost share the same concepts. Apache is the most famous HTTP web server that we can take as an example. See Appendix B for some performance tuning techniques.

4.3.4.2 Application Server

Web applications started with static pages to show data that are static by nature. However, there was a lot of information that was important to show up and at the same time must be kept in its database format. Data was not the only motive behind the application servers; there was also the logic that was not supported by the static pages. There is also the desk space that might not be able to consume a lot of data. In other words, a lot of complexities were behind building the application servers.

The application server is installed with the web server as a plug-in. It works mainly as a script engine that is responsible for interpreting the logic written inside the requested pages. There are many script technologies like CGI, JSP (Java Server Pages), ASP (Active Server Pages). The script is supposed to do some logic, connecting to database, or calling third party software. The final product of any script is HTML, or any kind of text that the browsers can render. The application server is mainly a container for web applications. It has a Listen-Queue that receives requests from the web server then it dispatches it to the proper web application. The transaction flows related to the SE depends on the object scope and the threading model [MREW00].

4.3.4.3 Web Application

A web application like any other application is built because there is a business driver. The application passes through all the normal application phases of collecting requirements, analysis, design, implementation, and testing. The web application could be built over a database or other infrastructure. The unique feature of the web application is that it has to always receive HTTP requests and its objects can be accessed by multiple threads.

Objects can be single-threaded or multi-threaded. A single-threaded object can only be accessed through its methods by a single dedicated thread. If another thread needs to access the object, then it has to request it from the owner thread. In contrast, multi-threaded objects need not have an owner thread. The object can be accessed by multiple threads concurrently. There is no guarantee as to which thread can execute which method, and no guarantee which thread will run first. A Multiple-threaded model can lead to improper resource utilization if it is not handled carefully [MREW00].

Objects can be also classified by their scopes. Objects can live during the scope of the application. In other words, it is a global object that can be used by all the requesting threads. Only a single instance of a thread can exist during the application lifetime. The object can also exist during the scope of the transaction. This means that every transaction can create more than one object that may do different activities at the same time. Objects can also live during a user session. For example, if a web application allows its users logon to do some activities, then logon information (e.g. user name, permissions, and so on) can be instantiated and kept during this session until the user logs out [MREW00].

4.3.4.4 RDBMS

Almost all modern web applications are dependent on RDBMS (Relational Database Management System). The amount of information to represent in an e-business website with the relations that are required to impose on it cannot be easily represented without a RDBMS. As the business of a company gets larger and larger, the need for RDBMS increases as well. It then can greatly affect the performance of the website.

The need for database tuning becomes very clear at this stage. A lot of techniques are very well known for Database Administrators by which they can enhance the database performance regarding the given capabilities of the hardware. The goal for the tuner is to eliminate the bottlenecks, minimize the hard disk access, and guarantee low response time. The tuner can modify table design, select new indices, rearrange transactions, tamper with the operating system, or utilize any other technique that enhances the performance [SHASHA96].

Chapter 5

SOFTWARE QUALITY AND TESTING

Any quality software system has to pass through software quality check and testing scenarios before its release for public use. Software quality is the guidance on how to design and code to improve program understandability, adaptability and decrease faults [BJAR02]. All software development stages have quality measurement factors that must be met from the first stage of collecting requirements up until the maintenance phase where measurements such as reliability, complexity, and reusability have to be considered. These factors have criteria that are measured like completeness, correctness, and size [LKZS00]. Testing is the checking process that guarantees that the software meets its specification and the needs of the customer [SI95]. It goes beyond validation and verification to checking system behavior under stress and concurrent use. Testing scenarios have strategies that should be conducted to capture all the errors in all the development phases.

This chapter is meant to familiarize the reader with the software design quality and testing methods used in Chapter 8. Chapter 8 contains analysis information about the suggested design patterns that the reader would be better prepared for after reading this chapter. It draws the reader’s attention so that he/she can decide on the validity of the suggested design patterns.

5.1 Software Design Quality

Building software is similar to constructing a building. If the building is constructed from good material and has a good architecture, it will remain even if there is an earthquake. We cannot accept any fault in a construction, since this may lead to death of thousands of people. The case is similar with software design, in which external functionality is achieved by internal design. However, it is almost the trend of most of the developers not to have a good design as long the work is finished up and the system specifications are satisfied. Software that has a good and robust architecture works better, costs less, matches user needs, has fewer bugs, runs faster, is easier to fix, and has a longer life span [CC02].

The good Software has the following properties [CC02]: -

1. **Cooperation**: Software must be able to work successfully with the surrounding environment and adapt itself with the available resources. For example, good software must live with storage areas that could be shared with others. This means, for example, that the log files that are generated during its lifetime must be removed after a certain period of time or backed-up somewhere.
2. **Appropriate form:** Software should not have fixed rules. They have appropriate rules that could be used according to the context. For example, global variables, as we were taught, must not be used. However, they might be the appropriate choice for a certain application.
3. **System minimality:** A well-designed software must be able to utilize the existing libraries. It is the responsibility of the developer to understand the existing systems in order to make use of its capabilities.
4. **Component singularity:** The developer has to decide on the software component's job and it should not be overloaded with unnecessary functionality. For example, the component that is doing I/O should not involve other functionality like graphics utilities.
5. **Functional locality:** Where related items are placed together. This makes sense and makes the development work much easier since the development team will share the same terminology. Bug fixing is very easy as well since code replacement will be in one place. For example, a Java project may contain one package that contains all I/O classes together. Upgrading the I/O classes means in most of the time replacing the old package with the new one.
6. **Readability:** This means that the pieces of design have to be understandable and very clear. This maps directly also to the code where code must be clear and readable. Overall, this lets other developers to maintain the code. Comments have to be written whenever possible to clarify the vague parts.
7. **Simplicity:** Simplicity is the art of finding simple solutions for complex problems. Simple software design is much easier to maintain and modify. Simple software have fewer bugs, run faster, are smaller in size, and are easier to fix when broken.

The design process is the model that captures the domain problem and reflects customer's requirements. It is the conceptual solution that works as a base for implementation. Design, with analysis, is considered the most important phase of the software development lifecycle. Experience shows that a fault in the design phase can cost thousands compared to a fault in the implementation phase. A well designed software makes maintenance, enhancement, and support replacement an easy process [LKZS00].

The most important factors influencing object-oriented software design are Reliability, Complexity, and Reusability. Reliability is the most important factor, as it is not accepted to deliver software that fails on regular basis. It is a measure of how well users think it provides the services that they require [SI95]. Complexity is measured by the personal experience and the design method used. It is always emphasized that simplicity must be taken as a design goal. It is already inherited in object-oriented design as implementation almost directly maps the definitions on design concepts on the implementation language. Reusability may not be a measurement factor in small projects. However, large projects that reuse modules or objects are regarded of a better quality [LKZS00].

The main criteria by which reliability is measured are correctness and completeness. This means that a design is correctly and completely capturing and representing the user's requirements. There are other important criteria like [RUP]:

1. **Coupling**: which is the measurement of the strength of interconnection among the system's elements
2. **Cohesion**: this means that every single component has to carry out only its responsibilities and they should be very well defined.
3. **Primitiveness**: if methods can be constructed from existing methods then this is not primitiveness.
4. **Volatility**: how frequently the design can change.
5. **Size**: It is always an indication, even if it is not an accurate indication, of the complexity and reliability of the system. For example, depth and width of class hierarchy, total number of classes, number of classes newly developed, number of classes re-used, number of attributes in each class, encapsulation of data in the class, number of methods in the class, reusability of operations, and length of operations in the class [LKZS00].

5.2 Testing

The main objective of testing is to find errors in a certain system. The successful test is the one that finds the errors that have not been noticed at all. The developer has to do his best to build tests that uncover errors systematically in a minimum amount of time. Testing assures that software intended functionality are met as they are designed, performance requirements are met, and assure the system quality and reliability [TEST01].

Testing is divided into black box and white box testing. Black box testing is usually conducted when there is a component with certain specifications. It is enough to verify that the specifications are followed up strictly. White box testing is usually done to verify that a component is conforming to a design. Systems that have a small number of lines of code can give a very huge number of test cases that cannot all be checked [TEST01].

The testing process consists of three main stages: Component Testing, Integration Testing, and User Testing. All systems must pass through all these stages except for small ones. They are further divided into five stages as shown in Figure 11. Other testing stages may be repeated if defects are discovered in later stages. It's an iterative process where information is feed back from later stages to earlier ones [SI95].

Unit testing is concerned with each component separately. Each component is tested separately without other system's components. Module testing stage makes sure that a module, which is consistent of independent components or abstract data types, is working according to its specifications without other parts of the system. Sub-systems is consistent of collections of modules that compose the sub-system interface. The most common problems in sub-system testing appear from sub-system interface mismatch since every sub-system could be independently designed and implemented. System testing is mainly concerned with testing the integration of the different sub-systems and system components. System testing has to confirm that functional and non-functional requirements are met. The final stage comes at the user acceptance testing where users provide real data, validate user requirements, and exercise system's performance [SI95].

An object-oriented application has a testing model where every unit is tested separately. When integrated with others they have to be tested again. The system starts with simple components that are tested separately. Simple components are integrated with each other to form modules. Modules are integrated to form sub-systems. Sub-systems are finally integrated to form the complete system. The level of integration is less visible in object-oriented applications. Data and methods are formed to build an object. Testing an object corresponds to unit testing in figure 10. Module testing is equivalent to testing a component in object-oriented where a component could be composed of more than one object [SI95].

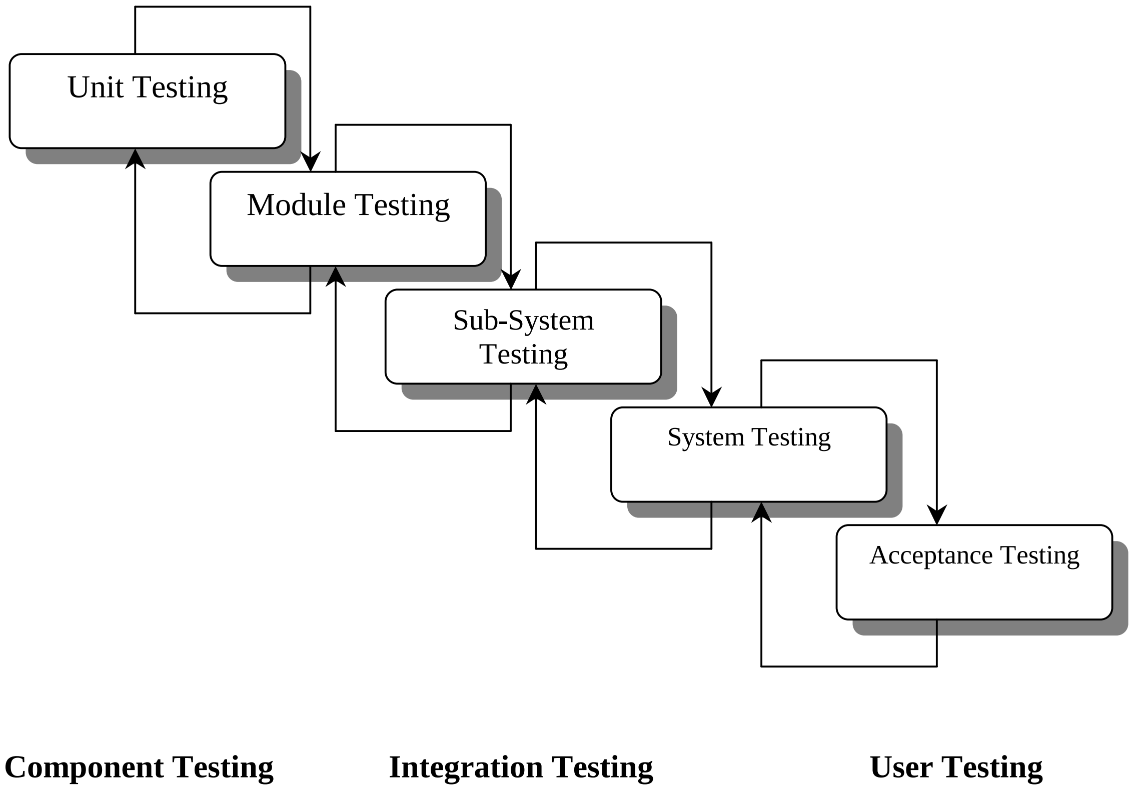


Figure 11 The testing process [SI95]

5.2.1 Testing for Reliability

Reliability is the main factor by which a system can get its users' acceptance. A user can accept an inaccurate result from the system, but he can never live with his system always unavailable. Performance and stress testing cover these aspects efficiently and discover unseen errors early enough.

Performance testing covers all aspects of software by which a system can fail under abnormal circumstances. The user's main concern is to measure throughput, stimulus-response time, or the availability of the system. There are a number of goals that the developer has to assess when doing performance testing [VFWE96]: -

1. Test cases are designed for performance testing not for functional testing.
2. The matrices needed to assess the comprehensiveness of the test use case have to be defined.

3. The matrices needed to assess the effectiveness of different test strategies.
4. Define the relation between test strategies in order to tell how one strategy could be better than another.
5. Testing has to be done on more than one hardware platform or program version when applicable.

There are other performance factors that could be measured like machine CPU, disk I/O access, database access rate, and so on. Note that it is sometimes impossible to satisfy all the resource requests. For example, there might be a request to minimize the hard disk space and minimize the memory usage as well, which may not be applied in most of the cases [VFWE96].

In order to do good performance testing, performance requirements have to be gathered from the users. There should be a very clear specification document that clears out all the performance tests that should be conducted. For example, the document may include simultaneous access, load testing, availability under load testing, expected average response time, and so on. However, such requirements are not usually gathered and if they are found, it may be very difficult to fulfill some of them [VFWE96].

Stress testing extends the normal load testing beyond the maximum until the system fails to ensure 'fail-soft' concept and show up hidden defects. Fail-soft concept ensures that the system will not cause data corruption or unexpected loss of user services if the system fails under abnormal circumstances of unexpected events. Moreover, stress testing may show defects that might not appear in normal circumstances. Although, It could be argued that these cases could rarely occur, it is important to know what stress testing will come up with. A major defect that stress testing could show it early is memory leak. An application may perform perfectly under normal circumstances or if it is most of the time running in a certain path. A memory leak could occur on the long run, which can cause of course data loss. Stress testing could show this defect very early [SI95].

5.2.2 Testing Strategies

A testing strategy is a general approach for testing rather than a method of composing certain tests for experimented systems. The most applicable ones during the lifetime of a project are [SI95]:

1. Top-down testing: where testing is started from most abstract components to the bottom.
2. Bottom-up testing: where testing starts from the smaller components to the top.
3. Thread testing: which is used for systems that execute transactions on multiple processes or threads.

Whatever testing strategy is adopted, it preferred to do testing incrementally through sub-system then system testing.

Top-down testing is executed through verifying high-level interfaces then breaking down and implementing the lower-level components and testing them. It's important to note that top-down testing is coupled with top-down programming where testing is always done with no hard separation. The main advantage of top-down testing is that it ensures a very early limited version of the system. This demonstrates the feasibility of the system to the management. Validation as distinct from verification will be available also to the users where feedback could be captured in early stages of development. However, strict top-down testing is not applicable because it is really difficult to simulate all the interfaces. It also requires from the user to feed the system with valid inputs and the system must simulate valid replies. It may be also very difficult to simulate very complex routines with diversified return values. This requires a very good knowledge of the final results and their calculations which may not be available in the early stages. Object-oriented systems can adopt this strategy on the level of components, but it cannot be applied to systems that are composed of objects which are usually building up the system from bottom to top [SI95].

On the other hand, bottom-up testing works first on the level of the fundamental components and then build the lower level components to make up the higher levels then testing is repeated over the new ones. Testing drivers, which simulate the environment behavior, have to be always implemented to verify the components specifications. Combining top-bottom development with bottom-up testing requires almost the whole system implemented before testing can begin. Architectural problems are most likely to appear and code rewriting may be required which leads to staring testing from the beginning. However, bottom-up testing is more feasible than top-down testing which almost impractical to conduct especially for object-oriented systems [SI95].

Thread testing is an event-based approach where there are events that trigger actions. This strategy can be only used after processes or objects are integrated and individually tested. It is very difficult to identify all the possible scenarios of inputs and expected outputs. However, the most exercised ones must be tested very carefully. Thread testing should start first by identifying as many threads as possible with the same inputs or with different inputs as shown in Figure 12. Testing then starts by examining one thread at a time with one input. The next step is to test the same thread with concurrent inputs. Later, a versified number of inputs is applied to the system with different threads. Note that this is not a stress testing, this testing is intended to tackle the system behavior with concurrent users. It is important to note that web applications are inheritably thread-based systems [SI95].

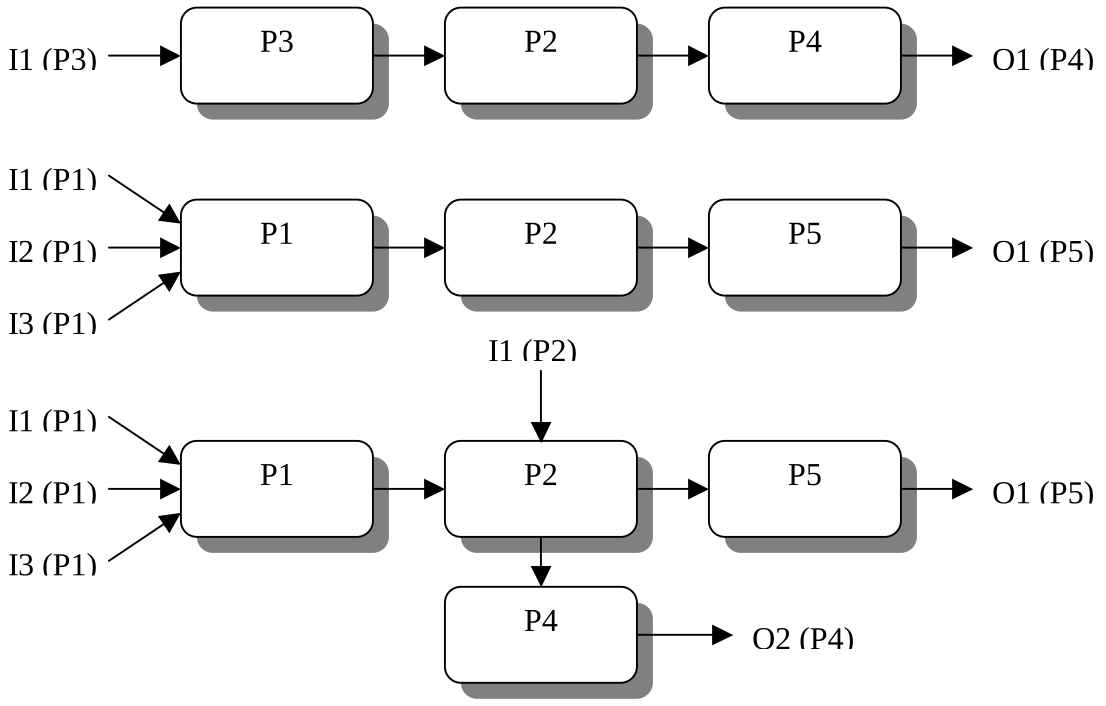


Figure 12 Thread testing [SI95]

Chapter 6

RELATED WORK

This chapter paves the way for the suggested design patterns which will be presented in the next chapter. It discusses some of the offered solutions that are related to these patterns. The focus here is on solutions, which do not have to be design patterns, but do tackle problems for web applications in general and performance problems in particular. They could be solutions that discuss the problems of the suggested design patterns from a different perspective. The purpose is not to compare others' solutions to the suggested design patterns but rather to give an overview of related research work and contributions and to provide a wider context of understandability.

6.1 Active Query Caching for Database Web Servers

The work on Active Query Caching [LQNJ99] presents a solution for caching dynamic content on proxies. They mainly retrieve database queries through a Java Applet. Web caching proxies are the main solution for enhancing web applications performance nowadays. However, they cannot handle dynamic content since they cache only static files. The collaboration scheme goes as follows between the active proxy, an experimental proxy, and the web server [LQNJ99]: -

1. The web server passes simple query processing to the *active proxy* through the *query applet.*
2. The *query applet* can answer queries whose results are a subset of other cached queries.

Figure 13 shows the architecture of the solution which involves a client, a proxy, and a server. The client sends a query request through the front end which is sent to the proxy as an HTTP request. The proxy checks if the URL has a query applet to invoke; otherwise, forwards the request to the web server. The web server translates the HTTP request to an SQL query and sends the result back to the proxy as an XML file. The query result may be sent associated with a query applet to the proxy. If the same query is called again, the proxy will return its result using the query applet. Other restrictive queries may use existing cached results if possible otherwise they are forwarded to the web server to be processed [LQNJ99].

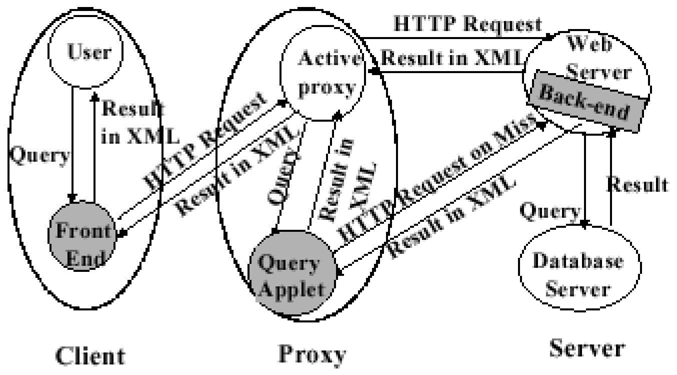


Figure 13 Active Query Caching Architecture [51]

6.2 A Pattern Language for Content Conversion and Generation on the Web

The research work of Vogel and Zdun [VOZU02] presents a pattern language targeting software architects that build highly dynamic, personalized, and content-centric web applications. The research helps in building a web application that can serve clients in a consistent and efficient manner. The pattern language consists of seven patterns which handle content representation and its accessibility [VOZU02].

There are some patterns intended to separate the roles of the web developers. So, the web design, developer, and content provider can work in parallel. Figure 14 shows an overview of the pattern language and its interaction. They are briefly described in the following points [VOZU02]: -

1. **GENERIC CONTENT FORMAT:** Content provided from different sources of information, like RDBMS or legacy systems, should be represented in an application-independent standard format like XML. CONTENT CONVERTER pattern is used to convert to the standard format. This generic format will be used on the web application before processing.
2. **PUBLISHER AND GATHERER:** It is a central point of management where content conversion is triggered, content consistency is verified, lookup in the cache, and other management tasks are provided. It is an abstraction layer by which different protocols and platforms can be used.
3. **CONTENT FORMAT BUILDER:** It solves the problem of building content in different formats and reuses the same code. It provides a common layer by which content format builders abide without hard coding.
4. **CONTENT FORMAT TEMPLATE:** It provides the content editor with a

simple way to easily add content without going into technical knowledge. There are special template tags that the content editor can replace.

5. **FRAGMENTS:** Tackle the development of the web pages by splitting them into fragments. It is the same concept of Divide-and-Conquer. For example, a page can be divided into a header, a left navigation menu, a body, and a footer. Further fragments can be introduced inside every fragment.
6. **CONTENT CONVERTER:** It provides a mechanism to convert the content from one format to another or update the content according to the change rules. Every format has a tailored converter that has input, conversion/update, and output processes.
7. **CONTENT CACHE:** Makes the dynamic content highly available by providing it inside a central cache. An invalidation mechanism is provided to control the lifetime of the cached objects.

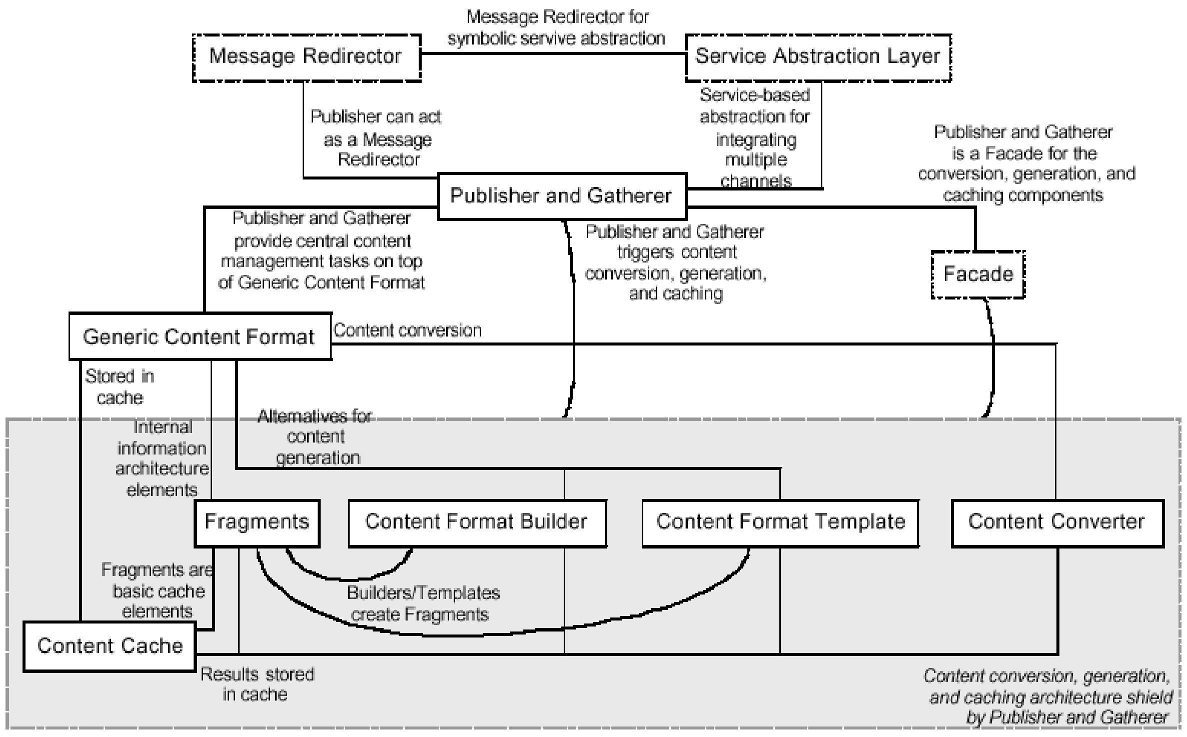

Message Redirector for symbolic servive abstraction
Message Redirector
Service Abstraction Layer
Publisher can act as a Message Redirector
Service-based abstraction for integrating multiple channels
Publisher and Gatherer is a Facade for the conversion, generation, and caching components
Publisher and Gatherer
Publisher and Gatherer provide central content management tasks on top of Generic Content Format
Publisher and Gatherer triggers content conversion, generation, and caching
Facade
Generic Content Format
Content conversion
Stored in cache
Internal information architecture elements
Alternatives for content generation
Fragments
Content Format Builder
Content Format Template
Content Converter
Fragments are basic cache elements
Builders/Templates create Fragments
Content Cache
Results stored in cache
Content conversion, generation, and caching architecture shield by Publisher and Gatherer

Figure 14 An Overview of the Pattern Language for Content Conversion and Generation on the Web [VOZU02]

6.3 Patterns for Web Applications

Abstract Form, Active View, Bus Class, and Active Proxy are four design patterns suggested for web applications by Bonura, Culmone, and Merelli [BDCR02]. They suggested these patterns to improve the effectiveness and efficiency of the development process. They target dynamic web applications where the business logic layer generates dynamic content from the back-end systems and sends it to the front-end (browser) [BDCR02].

The Abstract Form pattern is designed for web pages which carry information essential for creation of the next page. Abstract Form Pattern shows a clear communication between a *form*, front-end, and a *servlet,* business-logic. The pattern shows that a servlet can create a form and the form can have a connection to a servlet. Parameter checking is also covered in this pattern. Formal rules are used to control parameters and their check [BDCR02].

The Active View pattern makes use of the Observer pattern suggested by the four Gangs [GEHR95]. The Active View pattern is built over a *subservlet* (A Subject Servlet) and an *observlet (Observer Servlet).* There can be more than one *observlet* watching over a *subservlet.* It gives a method to update an active view when its content is changed. The business logic layer sends a signal to the front-end to update its view. The back-end contains *triggers* that notify the business-layer for any changes to let it update its views [BDCR02].

The Bus Class pattern is concerned with the use of the distributed objects in a connectionless protocol like HTTP. It helps the developers to handle distributed objects in the business logic layer without losing the connection with the front-end layer. An ID parameter is used to recognize the clients. This pattern is built on the pattern “State” suggested by “the Gang of Four” where it can be extended to have objects with dynamic properties [BDCR02].

The Active Proxy pattern brings the concept of the static proxy to the dynamic context. Because static websites can be cached on the static proxy level, users do not suffer from performance problems. Almost all the business logic that the proxy has to do is to check the validity of the cached page against the one on the server. The goal of the active proxy is to provide the page saved on the proxy with a state management solution pattern. The purpose is to refresh the page depending on its state using the refresh pattern.

6.4 Meta-Patterns

The research done by Moisés Daniel Díaz Toledano [TM02] recognizes some techniques that minimize the complexity of the design patterns. The researcher introduces some patterns, which are called meta-patterns, that aim to reduce the complexity of patterns. It tackles a lot of issues like encapsulation, standardization, and decoupling complexity by introducing indirection to functions, classes, components, subsystems, etc. The main meta-patterns introduced here are Meta-Pattern Encapsulator, Meta-Pattern Adder, and Meta-Pattern Decoupler [TM02].

Meta-pattern Encapsulator checks for a distributed functionality among different classes and simplifies the design with a new role to a certain class. Meta-Pattern Adder modifies or adds certain functionality in a class by introducing a new intermediate inheriting class that encapsulate this functionality. The Met-Pattern Decoupler breaks down a complex functionality in a certain class by introducing an intermediate class that contributes with the original one to serve the required functionality. These general meta-patterns are recognized in more specific meta-patterns like Validator, Persistor, Set, and Logger [TM02].

Validator meta-pattern breaks down a complex validation process by introducing an intermediate class that does only validation for the given parameters. Persistor meta-pattern makes persistent data, like files, available in a more flexible way. It does so by introducing a decoupler class called *Persistor.* Set meta-patterns add a new role *Set-Mapper* which gives information about a collection of objects, which are generally of the same type. Logger meta-pattern encapsulate all the logging functionality that are distributed among different classes in one class. This allows the system to register information about its behaviour in a very standard way [TM02].

6.5 Capacity Planning for e-Business

Although Cranmore et al’s [CAUJ02] research deals with capacity planning and does not handle patterns in software development, their solution seems to help greatly in enhancing the performance. It also shows how to control capacity so that performance solutions do not become a burden. Three patterns are suggested in their work, namely: Build-in Redundancy, Avoid Overbuild and Share Resources Carefully. There are also three *patlets,* in order to avoid improper implementation of the patterns, which support the main patterns. Figure 15 shows a roadmap for the patterns, in solid boxes, and the *patlets*, in the dotted boxes [CAUJ02].

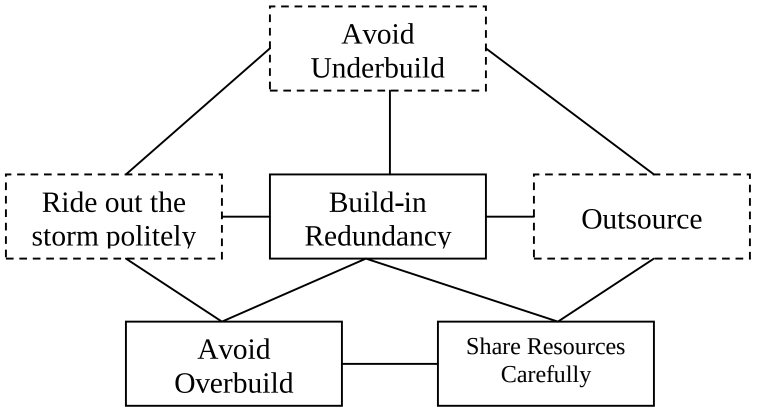


Figure 15 Capacity Planning Roadmap [56]

Build-in Redundancy proposes the idea of having redundant capacity of hardware or software. This idea of redundant capacity increases the e-Business site availability and enhances performance. The decision to have a redundant capacity must be carefully taken according to the costs and the benefits. So, if there is an e-Business site which survives with three servers, then an extra server is considered an unneeded *overbuild*. However, if benefits surpass costs, then redundancy is needed [CAUJ02].

6.6 Conclusion

As shown from the above research paper discussions, there are always solutions for performance in one way or another. The first three research papers mentioned in 6.1, 6.2, and 6.3 provide a cache solution which is a direct performance optimisation mechanism. The fourth research paper mentioned in 6.4 provides meta-patterns for patterns to break down their complexities. It also provides an indirect performance optimization solution by structuring the design patterns in a robust way. The fifth research paper mentioned in section 6.5 provides a set of capacity planning patterns to optimize performance through hardware redundancy.

From all of these research efforts, it is shown that patterns play an important role in guiding the developers through the different optimization mechanisms. The research in this thesis groups performance optimization, and complexity breakdown solutions together with a resource monitoring solution to build up a pattern language focusing on performance issues. This pattern language provides solutions for the most common performance problems in database-driven web applications.

Chapter 7

THE PERFORMANCE PATTERNS

Web applications are by nature based on multi-threaded architectures. This enables them to respond quickly and efficiently to the hundreds or may be thousands of concurrent user requests. The database driven web applications have more complex architectures because of the inherited database management system complexity. Although, the complexity is decoupled in every layer (web server, application server, database), the web application has to always commit itself to a certain response time, show stability, and be available all the time.

Because of the inherited multi-threaded nature of the web applications, there is always competition over system resources and it becomes very difficult sometimes to trace any resulting problems. There are always concurrent user requests for dynamic web pages, which in turn trigger requests to the subsequent database access layer. Through the path to the database, there is a competition over resources such as database connections, or database objects. If the web application is not ready with good solutions such as the ones mentioned in the up coming sections, the system would always be in chaos.

Some of the patterns outlined below wrap solutions that have been implemented in a wide range of previous software and hardware systems. They are introduced in a pattern format and concentrate on performance as the main problem to be solved. Some of the contributions of this research have been added either as enhancements to existing patterns or as completely new patterns as explained in section 1.4 (Outcomes and Contributions).

7.1 Pattern 1: Database Connection Pool

7.1.1 Context

The Database Connection Pool (DCP) solves the problem of database connection acquisition and release through recycling.

7.1.2 Problem

Creating a database connection is very expensive since the process requires resources that consume time and memory. A connection is a session with a specific database where SQL statements are executed and results are returned within the context of a connection [JAVA1.3]. In database driven applications, creating a database connection is a very demanding activity especially those ones that are designed for multi-access. The problem here is that only one user can acquire a database connection to retrieve information from the database. If users kept creating database connections, there will soon be memory overflow which would eventually cause system instability.

Creating a database connection for every request is too simple a solution and it can never succeed with the applications that have concurrent access. Another naïve solution is to keep a created database connection for every application session. However, applications that have thousands of users, like E-Commerce applications, will have severe memory leak problems that cause system instability which will ultimately get the applications down since there is no management over the database connections. The problem here is the expensive acquisition and release of database connections which have to be done by the database driven applications but they also have to be stable and responsive.

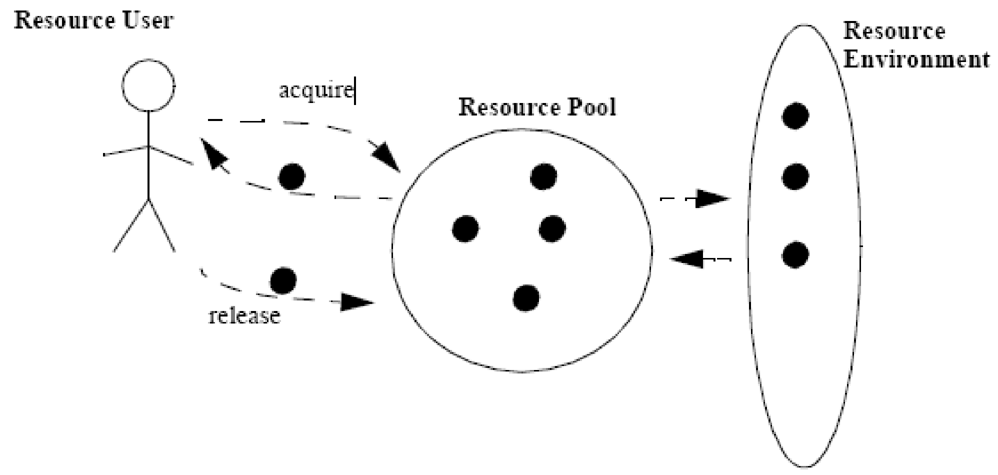


Figure 16 Resource Management [57]

7.1.3 Forces

The following issues must be resolved to solve the pooling problem [KMJP02]: -

- *Performance*: where wasted CPU cycles should be avoided.
- *Predictability*: resource acquisition must be predictable even if it is not predictable from the resource environment.
- *Simplicity*: the solution should be simple to minimize application complexity.
- *Stability*: the solution should minimize the system instability.

7.1.4 Solution

Figure 16 shows that the solution is to have a pool of connections to save users from the tedious acquisition and release process. This pool exists in the application memory and serves as a recycling place where connections that are no longer used are returned to the pool for future access. However, since database connections consume a large space in memory, a good strategy has to be utilized for connections acquisition and eviction.

Database connection acquisition defines how connections are acquired from the database management system and stored in memory for use. The level of acquisition can be lazy, eager, or partial. The lazy acquisition delays the expensive database connection acquisition until the connection is demanded and then it stores it in memory [KM01]. The eager acquisition prepares all the database connections before they get acquired [KM02]. The partial acquisition implements eager acquisition at one stage to ensure quick satisfaction for connection requests and then it starts lazy acquisition to minimize memory consumption until there is a need for it

[JPKM02]. The optimization criteria should be decided according to the system available resources of CPU, memory, and availability of other resources.

Software resources are precious because they are limited, even if there are several of them. The resource users can be more than the number of resources which may result in unstable situations if not handled properly. Eviction discusses how and when to release resources. The problem is that systems may keep acquiring resources without releasing them which may cause system instability [JP01].

However, not all resources should be released, as some of them may be expensive to re-acquire [JP01]. There are some issues that must be taken into consideration if connection eviction is decided like the frequency of using a connection, CPU utilization, and available memory size. Moreover, the user should not suffer from the selected solution.

Eviction uses a certain strategy like Last Recently Used (LRU) or Least Frequently Used (LFU) to control the life cycle of the resource. The resources that are frequently used or recently used are marked by the application. The evictor, which can run periodically or in demand, releases only those resources that are not marked and the other ones continue to stay in memory for later use [JP01].

A better approach to implement the database connection pool is to consider it as a variation of the cache pattern. Actually the main difference between the cache and the pool is that the cache has items with unique identities and may be of different types but the pool has items of no identities and of the same type. So to retrieve a cached item, the unique identity has to be provided but the connection pool returns an item directly without providing any identity. So, think of the database connection pool as a cache with only one identity. So, if you provide that common identity to the pool, it would return a connection all the time since all of them have the same identity.

Figure 17 shows the class structure of the pattern. It shows that the database connection pool extends the *Pooling* Pattern [KMJP02] and uses other patterns like eviction, and strategy.

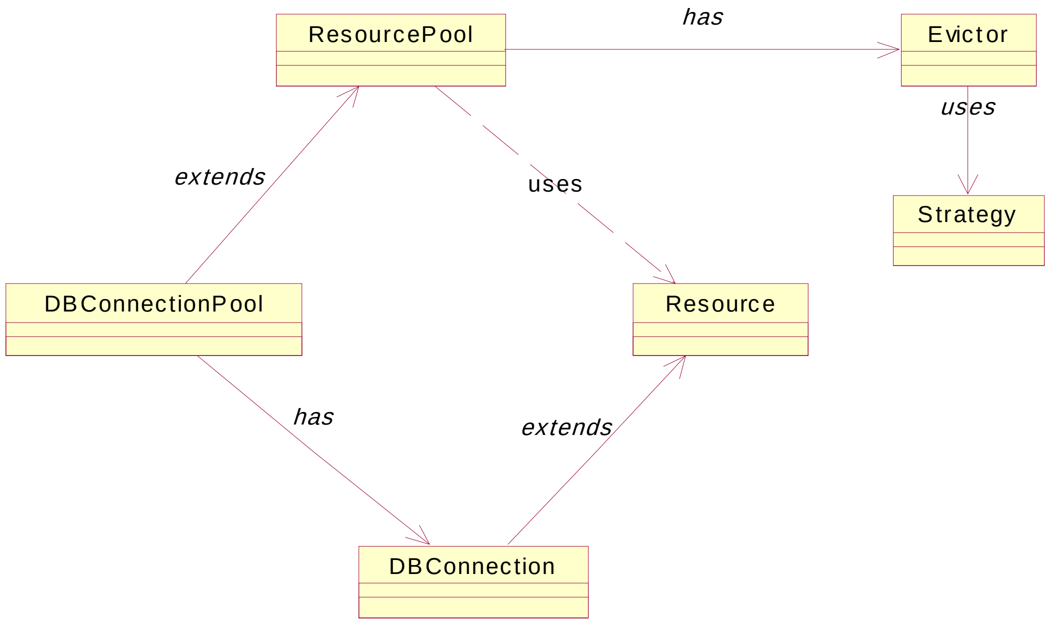


Figure 17 Database Connection Pooling

Figure 18 shows the user interaction with the database connection pool. As shown from the diagram a new connection is not established until the database connection pool is checked for free ones.

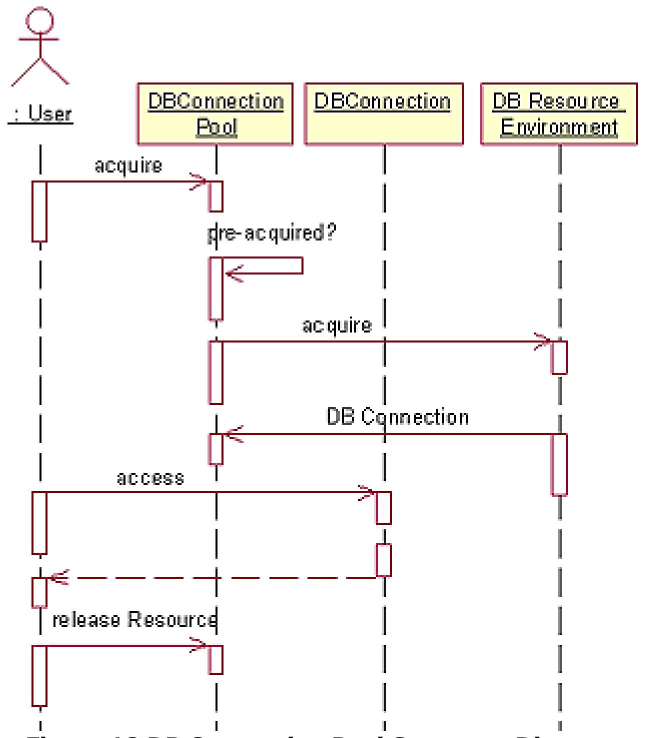


Figure 18 DB Connection Pool Sequence Diagram

7.1.5 Consequences

- *Performance*: Establishing a connection to a database represents the major delay to any request. Any one who has worked with a database will definitely appreciate having a connection pool in his application because the response time will definitely be much better. Although there is a management time overhead for acquiring and releasing connections, it is acceptable by the systems if the pool is tuned correctly.
- *Stability [KMJP02]*: The applications that implement connection pooling experience a large degree of stability because acquisition and eviction solutions maintain the resource environment and ensure that there is no leakage in resources.
- *Predictability* [KMJP02]: A database request has fairly good and known response time if the database connection is served from the pool. On the other hand requests that constantly have to acquire connections from the database have long and unknown response times.
- *Deadlock*: deadlock situations may be reached because of race conditions if the connections are not properly acquired and released. A race condition can happen if all connections are not released and a thread acquires a connection and the pool cannot allocate one.

7.1.6 References

- *Data Sources*: Some application servers, like IBM WebSphere Application Server, implements the database connection pool using EJB technology.
- *Thread Pools*: Systems like web servers and application servers may need to satisfy hundreds or thousands of concurrent user requests. In order to maintain a good response time, they implement a pool of threads to handle the concurrent user requests instead of creating and releasing a thread for every request.
- *Caching*: Caching is exactly like the database connection pooling from the basic concepts. However, object identity is more important in caching unlike the database connection pooling.

7.2 Pattern 2: Cache

7.2.1 Context

Systems that need to bring objects to the memory of the server (Server-side caching) to optimize for performance can use this pattern. This pattern applies to database-driven applications where queries are the primary key player in retrieving information.

7.2.2 Problem

Database Queries may take a considerable amount of time to come up with a result and the query result might be small and may not be expected to change in the near future. It is not abnormal to see queries that take more than one minute to complete. Queries can take more than one minute on very large databases. One minute delay combined with other delay factors is considered too much time for a web application. It happens also that tuning the database, like creating indices, may be space consuming for very large database. Creating indices solution may not be recommended for all database objects. Web Applications sometimes interact with such databases. So, web applications must have techniques to overcome the database delay.

7.2.3 Forces

- *Performance*: The performance problem is a key problem that cache can greatly reduce. It should minimize the role of the database and network by saving the database object in the server memory which is a very fast storage area to optimize performance.
- *Reliability*: Systems that have different components cannot guarantee the same service level. For example, a web application may consist of a web server, an application server, and a database. Every one of these components can have a different response time due to many other internal factors. If you consider a database query request trip, you will find that it passes through the web application, network connection, and the database itself. Caching should minimize the contribution of the network and the database; hence, the system must show more stability.

- *Complexity* [KMJP03]: The pattern should not complicate the resource acquisition and release process. Moreover, implementation must maintain the existing code uniformity and not change its business logic.

7.2.4 Solution

A web application can use different caching techniques to reduce latency, minimize network traffic, and consequently reduce timeouts. Caching, as defined by Brown [BB02] is *"an area of storage existing behind the scenes which is used to store copies of previously requested objects that can be quickly accessed when needed."* It increases the performance and makes the web applications more reliable [BB02]. Caching is not a new technique; it has been used in many software fields and on different aspects. There is special caching hardware, like cache proxies, which cache the static web content. There is caching even on the client machines so that one need not download the static content, like images, every time. Clients become satisfied with the quick response which increases their trust on the web application.

Server-side caching is a performance enhancement technique that is used in many web and non-web applications. It brings the expensive results to the application memory. Ideally, a database query result is cached. A caching solution must handle the object expiration to ensure consistency. When the application invokes the query it looks up first in the cache before accessing the database. If looked-up objects are found and have not expired, then they are returned back to the client without passing through the database bottlenecks to establish a connection and run the query. The pattern will try to abstract different caching solutions and leave the implementation to the developer.

The Cache pattern is similar to the Database Connection Pool pattern in terms of resource acquisition and release. However, caching objects always have a unique identity by which they can be accessed. The Database connection Pool does not assign an identity to the objects [KMJP03]. Also, cached objects can be shared among multiple users. On the other hand, the Database Connection Pool does not allow sharing.

The cache consists mainly of entries and an evictor, as shown in Figure 19. The cache entries are environment resources that are usually expensive to acquire. The evictor is responsible for periodically removing the entries that have not been used for a long time [KMJP03]. It can be configured to use different strategies like

LRU (Last Recently Used), or LFU (Least Frequently Used). The strategy pattern mentioned by the four Gangs can be used here. By referring to Figure 20, one would notice that the cache does not always return directly to the user. The cache entry might have expired or may not be there which leads to the allocation or reallocation of the cached object. The other benefit of this technique is that the cache can build itself up through time according to the user needs. The first request for a certain database query will put data into the cache and later requests will get it from the cache.

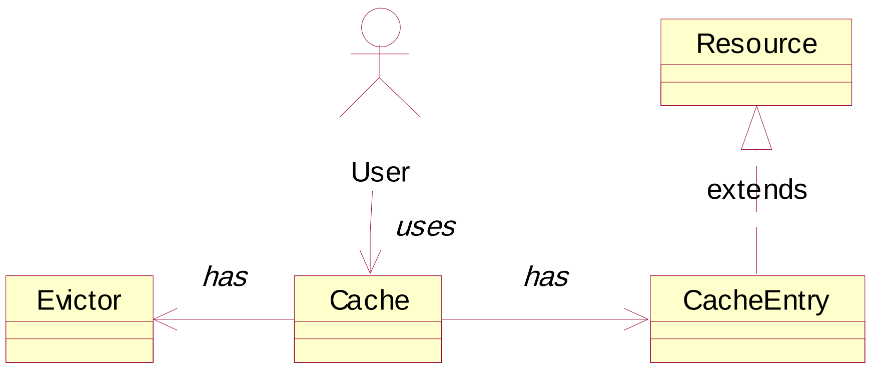


Figure 19 Cache Pattern

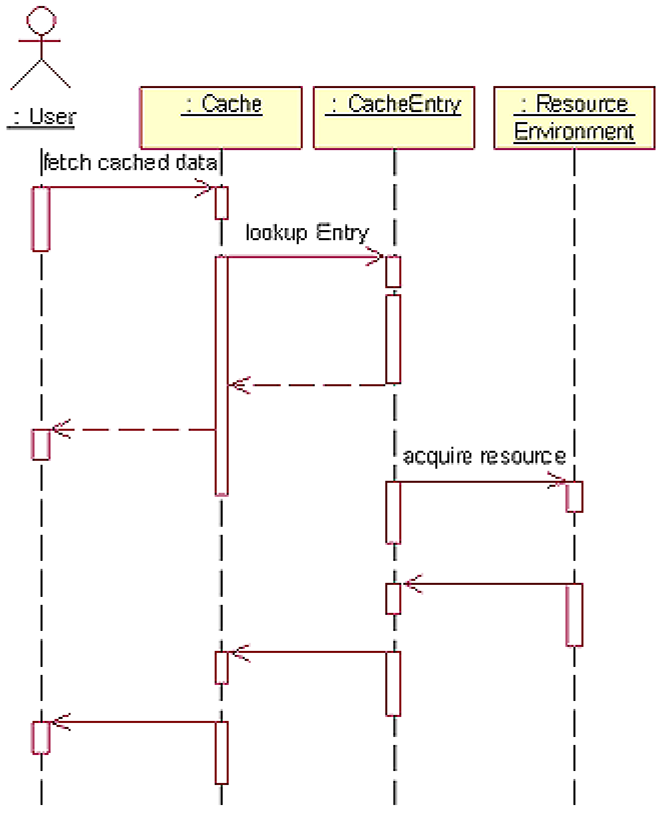


Figure 20 Caching Pattern Sequence Diagram

Planning for caching can highlight the performance optimization path for the developer. Planning can start from the design phase where caching points can be depicted over the object interaction diagrams, Figure 21. Planning for caching can be a quite simple if there is a database access layer designed in the system. It would be a good idea to have a separate artifact that has a roadmap of all the caching points. The *Cache Roadmap* will help in reviewing the performance optimization through caching by the end of the development phase. Moreover, this document can be combined with other documents that tackle other performance issues and a general performance optimization session review can be conducted on the system before the end of the development phase. Table 4 shows a recommended document structure where all caching points are listed against their class methods and their validity period.

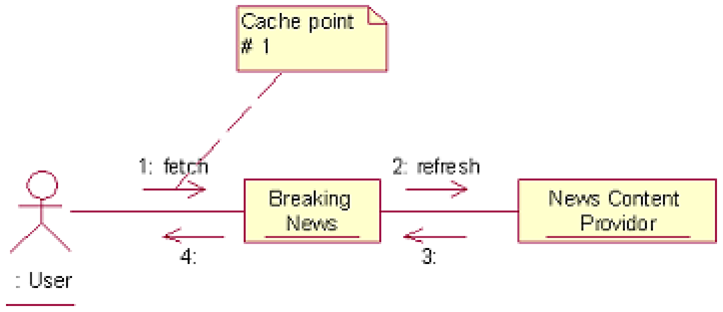


Figure 21 Caching Points shown over the collaboration diagram

Cache Point	Class Method	Validity	Description
1	BreakingNews.fetch()	1 hour	Any additional information
2	SportsNews.fetch()	½ hour	Any additional information

.
.
.

Table 4 Planning for Caching Document

The validity period assigned for every cache point in the *Cache Roadmap* can be accurately specified if type of the information is well known and the users behavior is known or correctly anticipated at least. For example, if the information is changing every one hour and the user demand for this information is very high then it the validity period might be one hour or less. However, if the information is changing quickly but it is not highly demanded by the users then the validity period may be set to one and half an hour.

7.2.5 Consequences

- *Cost versus Performance*: The decision has to be made on how to cache database objects. For example, all the query results could be cached if they will not consume a lot of memory and they are highly demanded. Moreover, the queries that consumes a lot of time to return are the best candidates for caching especially if they are highly demanded. Other queries may be neglected either because they return large results that memory cannot accommodate or because there is no high demand for them. In all cases the cost of memory usage due to caching must be controlled so that the application does not suffer memory over-flow. If memory size is very high, caching policy is known, and caching size is controlled, then performance will be enhanced noticeably.
- *Consistency*: The application has to have a policy by which caching can be invalidated to guarantee that it will contain the latest database object information. The delta between changing information in the database and updating it in the cache cannot be avoided. However, tuning caching periods, derived from the Caching Planning document, will always help to minimize this period or at least it would not be noticeable for the users.

7.2.6 Anti-patterns

Embedding caching inside the database access objects may be tempting, as SQL query caching will be automatically done for the developer. The problem is that query results are always cached regardless of the result size. Systems that have a large size of query results must be given control on whether to cache the results or not. So, caching blindly along the way is not recommended. However, binding the cache pattern with the database connection pool is good if there is a safety exit by which the developer controls the caching criteria.

7.2.7 Related Patterns

- *Read-ahead Caches*: A system can increase its efficiency if some objects are cached upon startup. This gives a good experience to the users since they will not feel any sort of delay. The Read-ahead caches can guarantee an existing instance ready for read before the read operation exists actually [KMJP02].

- *Synchronized Cache*: Where cached data has to be always synchronized with its original source, e.g. database [KMJP02].
- *Lazy Load*: where loading data from database to memory is not done until it is really needed instead of explicitly load all data in memory which hurts the performance [FM02].

7.2.8 References

Caching solutions are very widespread. Almost all applications do caching in one way or another. The following are a few examples that utilize the caching concept: -

- *Web Browsers:* Almost all popular web browsers such as Netscape and Internet Explorer does client side caching. Special Meta tags can be set in the web pages to let the browsers refresh the content from the web server.
- *Hardware Caching*: This is well shown in almost any CPU where caches are very fast areas to avoid expensive program access. As the CPU cache size increases the programs response times increase.
- *Website Accelerators:* They are special hardware devices that save the content closer to the end users. They usually have a large memory capacity.

7.3 Pattern 3: Static Enabler

7.3.1 Context

Dynamic web applications that do not rapidly change their content, and need to optimize their performance can use this pattern.

7.3.2 Problem

The Internet was a major breakthrough in the field of information technology. It was an easy method to share information and as part of this evolution, people started to post their information on websites. A website is a designed location on the Internet where information is published. It normally contains a mixture of different media objects, like images, flashes, and text. Static websites emerged as the first of this technology. The only backend application needed to serve a static website is a web server which serves pages directly to the users.

Static web applications are sufficient for small businesses where information is not expected to change quickly. However, if the business size gets bigger, the need for dynamic content becomes mandatory. The main issues that a dynamic web application is needed for are: -

1. When Content management for a static website is difficult.
2. When A static website can't integrate with other legacy systems. A dynamic web application is mainly needed to achieve an interactive communication with the website users and the existing backend.
3. When Legacy systems have tons of information that can take a very long time and much disk space to be manually posted on a static web application.
4. When periodic and automatic changes of content cannot be supplied using a static website
5. When a content management pool, e.g. a database, better represents logical and consistent relations of components in a web application.

On the other hand, a static web application has the following benefits over a dynamic web application: -

1. It responds much more quickly.
2. It is more robust and reliable.
3. It has fewer problems.
4. Implementation is done more quickly than for a dynamic website which can have many complexities in the backend.

5. It saves server CPU cycles and memory utilization.

The problem here is that the dynamic web application should benefit from the static web application features and at the same time stays dynamic and maintains content relations.

7.3.3 Forces

- *Performance*: Converting dynamic pages to static should optimize for the performance and not the opposite. The resulting text size of the static page should be less than or equal to that of the dynamic page.
- *Reliability*: The web application reliability should not drop, but it should be enhanced.
- *Complexity*: The pattern must not add more complexity to the usage or implementation of the application.
- *Page Reference*: The existing content relations that are represented by the web application should not be spoiled.
- *Content Validity*: The pattern must guarantee that the static pages contain valid content all the time.

7.3.4 Solution

Some applications provide content management tools that build the required content and publish it as static pages. The limitation here is that large database systems can consume a huge disk space if they are published as static pages. The other choice is to have a dynamic web application and accept its limitations for the sake of other benefits as mentioned above.

However, a dynamic web application can benefit a lot if there are static pages bundled inside it for the following reasons: -

1. Highly accessed dynamic pages can be converted to static ones to introduce a faster response, while still keeping the dynamic page for other references and at the same time saving disk space.
2. The decision can be taken to convert all the dynamic pages to static or just leave the dynamic ones according to the capabilities of the working environment.
3. Some backend systems can be temporarily shutdown if their counterpart dynamic pages are converted to static.

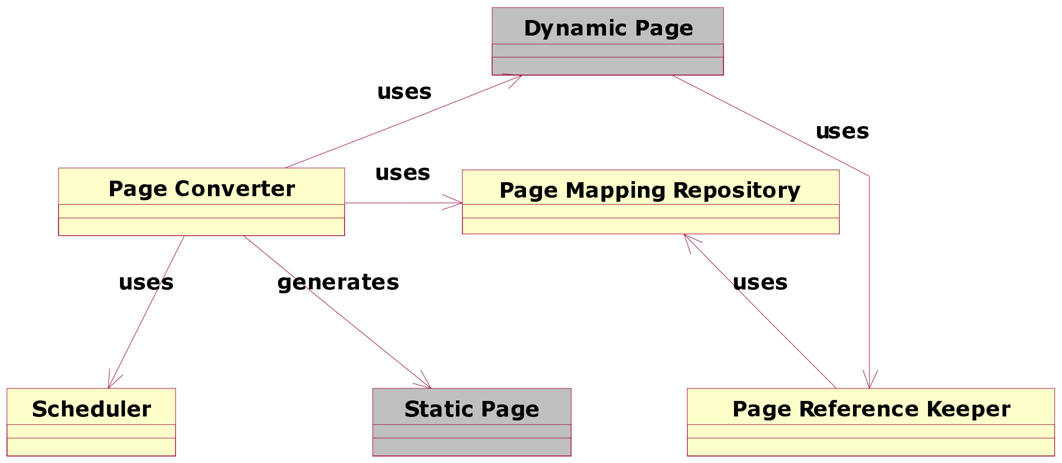


Figure 22 Converting Dynamic Pages to Static ones structure

The pattern consists of four components as shown in Figure 22: -

1. *Page Mapping Repository*: Where the full path of the static page is set for the dynamic page. Only the pages that will be converted are specified here.
2. *Page Converter*: which is responsible for converting all the dynamic pages specified in the repository to their counter part static pages.
3. *Page Reference Keeper*: It makes sure that the embedded links inside the converted page are pointing to the converted static pages, if found for that link. It works directly with the *Page Mapping Repository*. However, its role will be shown when the *Page Mapping Repository* is filled with page mapping entries.
4. *Page Conversion Scheduler*: It schedules the run of the page converter to update the content inside the static pages.

The ideal usage of the pattern is shown in Figure 23.The use of the pattern starts by wrapping all the page links that will be converted to static using the *Page Reference Keeper*. It is recommended to wrap all other pages for future conversion. *Page Reference Keeper* will always get a valid link even if there is no static page for the specified dynamic page. The repository can be filled either before or after wrapping all the links. Actually, the *Page Mapping Repository* is the way to control the conversion. The repository administrator can remove all the entries to make it pure dynamic, fill it with all the page links to make it pure static, or make a mixture of this and that depending on the system capabilities.

One may think of the *Page Converter* component as a client to the system which simply makes scheduled requests by the *Page Converter Scheduler*. It makes use of the web application backend, which is already capable of sending static content out of the dynamic pages, to retrieve the content and save it on disk. This makes the pattern completely independent from the technology used to build dynamic or static pages.

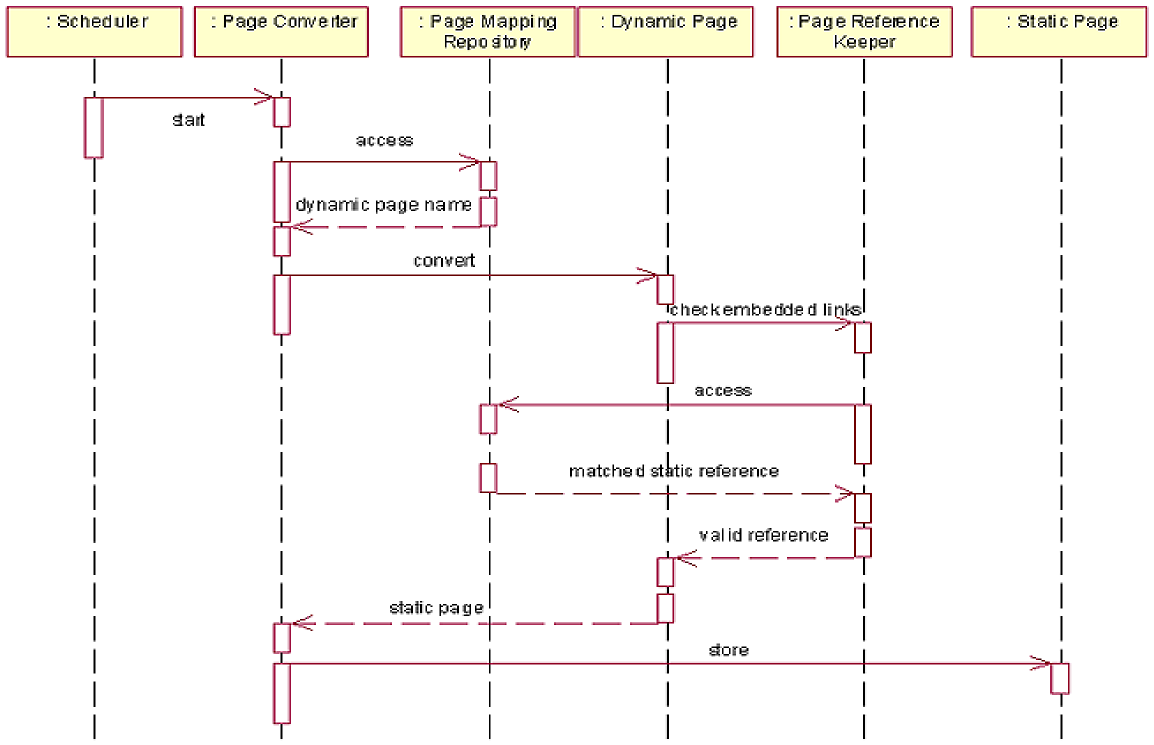


Figure 23 Converting Dynamic to Static Page Pattern (Ideal Scenario)

For example, Assume that there is a web application that contains five pages page1.jsp, page2.jsp, page3.jsp, page4.jsp, and page5.jsp and they are interlinked as shown in Table 5.

Dynamic Page	Embedded Links
Page1.jsp	→ page2.jsp, → page3.jsp, → page4.jsp
Page2.jsp	→ page1.jsp, → page3.jsp
Page3.jsp	→ page4.jsp, → page5.jsp
Page4.jsp	→ page2.jsp, → page4.jsp, → page5.jsp
Page5.jsp	→ page2.jsp, → page4.jsp

Table 5 Example for a web Application Page Tree

Assume also that the *Page Mapping Repository* contains the following mapping

Dynamic Page Full Path	Static Page Full Path
http://<dynamic_path>/page1.jsp	/<static_path>/page1.html
http://<dynamic_path>/page2.jsp	/<static_path>/page2.html
http://<dynamic_path>/page3.jsp	/<static_path>/page3.html
http://<dynamic_path>/page4.jsp	/<static_path>/page4.html

Table 6 Page Mapping Repository

The above table says that all the dynamic pages on the left should be converted from the **<dynamic_path>** server to static pages and saved on the physical machine path **<static_path>**. The assumption is that *Page Converter* has full access to dynamic and static paths. Moreover, a web server should serve the static path. Figure 24 shows that the page converter can work simply as an end user that retrieves the content of the dynamic pages and saves them on different physical machines.

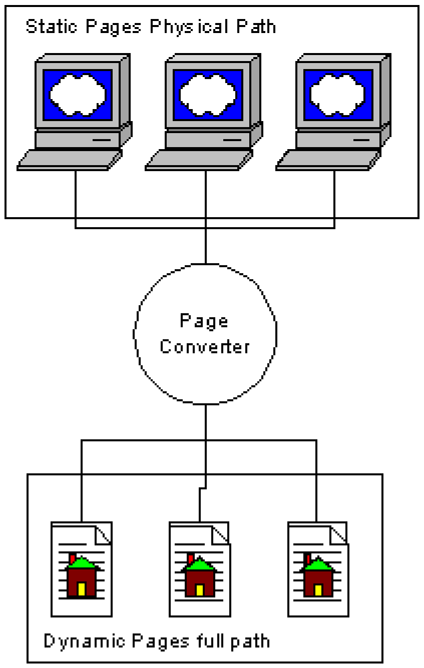


Figure 24 Page Converter

Page Converter starts its work by listing all dynamic pages on the table page1.jsp, page2.jsp, page3.jsp, page4.jsp, converting them, and saving them as static text format pages on the specified static path. *Page Converter* will run periodically according to the scheduler to convert all the pages in the list again. More enhancements can be added to the scheduler to trigger *Page Converter* only if content is changed and convert only the pages that are related to that content. The web application contains now the following pages with their embedded links (Table 7):

Dynamic Page	Static Page	Embedded Links
Page1.jsp	Page1.html	→ page2.jsp, → page3.jsp, → page4.jsp
Page2.jsp	Page2.html	→ page1.jsp, → page3.jsp
Page3.jsp	Page3.html	→ page4.jsp, → page5.jsp
Page4.jsp	Page4.html	→ page2.jsp, → page4.jsp, → page5.jsp
Page5.jsp		→ page2.jsp, → page4.jsp

Table 7 Web Application Page Tree (Dynamic & Static) without using Page Reference Keeper

If we assume that the static html pages are accessible to the end user, then he/she will access them only once and he/she will end at the dynamic pages again. So, before converting the dynamic pages to static, *Page Reference Keeper* must wrap all the links inside all the five pages. It will make sure that there is a link to the static page if found or it keeps the only dynamic link as is. Table 8 shows the web application initially where the dynamic page links are wrapped by the *Page Reference Keeper.* After the Page converter runs, the page links will be set as shown in Table 9.

Dynamic Page	Embedded Links
Page1.jsp	→ Keeper(page2.jsp), → Keeper(page3.jsp), → Keeper(page4.jsp)
Page2.jsp	→ Keeper(page1.jsp), → Keeper(page3.jsp)
Page3.jsp	→ Keeper(page4.jsp), → Keeper(page5.jsp)
Page4.jsp	→ Keeper(page2.jsp), → Keeper(page4.jsp), → Keeper(page5.jsp)
Page5.jsp	→ Keeper(page2.jsp), → Keeper(page4.jsp)

Table 8 Web Application File Tree with Page Reference Keeper

Dynamic Page	Static Page	Embedded Links
Page1.jsp	Page1.html	→ page2.html, → page3.html, → page4.html
Page2.jsp	Page2.html	→ page1.html, → page3.html
Page3.jsp	Page3.html	→ page4.html, → page5.jsp
Page4.jsp	Page4.html	→ page2.html, → page4.html, → page5.jsp
Page5.jsp		→ page2.html, → page4.html

Table 9 Web Application Page Tree (Dynamic & Static) using Page Reference Keeper

Although page5.jsp is not converted to a static one, the *Page Reference Keeper* can still function and get the right reference. This feature allows the web application behavior to stay as designed and any other expected references to dynamic pages will still function correctly. In other words, both dynamic and static pages are kept for use according to the need. Figure 25 shows how the *Page Reference Keeper* as it works for the dynamic pages alone.

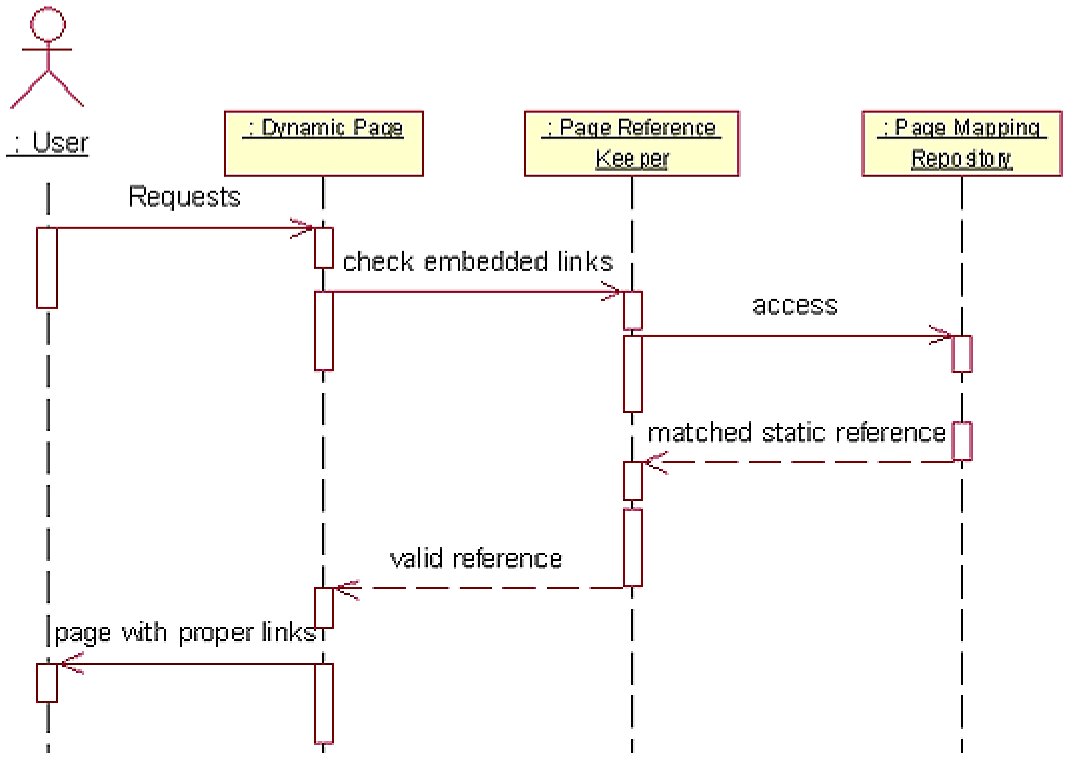


Figure 25 A dynamic page request in cooperation with Page Reference Keeper

7.3.5 Consequences

- *Performance*: Performance is enhanced greatly since the dynamic page access process to build the dynamic page is completely avoided. The server CPU and memory utilization are minimized as well.
- *Reliability*: The web application will be more robust and reliable. This increases the availability of the web application.
- *Content Validity*: Content validity is controlled by the time the *Page Converter* runs after the content is changed. If the system owner accepts this period then there is no problem. However, if the content has to show up on time, then the *Page Converter* must be triggered to the needed pages only.

7.3.6 References

- *Content Management Applications*: there are some content management applications that have publisher modules which buildup static pages out of the content and publishes them on the web.

7.4 Pattern 4: SQL Statement Template

7.4.1 Context

Applications that depend on DBMS for building their business logic and need to break down code complexity, provide common SQL statements, or externalize SQL statements can use this pattern.

7.4.2 Problem

An inherited nature that is found in most database-driven web applications is that SQL statements are considered as an intangible part of the code. This may be reasonable for the application designer as the SQL statements are the building blocks of the business logic; hence, they must not be externalized outside the code. Separation has been presented but in the form of a code layer to access the database. This separation is mainly important to map relational databases to object-oriented software. However, bundling the SQL statements inside the application code has the following main defects: -

1. Tuning any database SQL statement requires re-compilation of the code which may be time consuming. For example, it may be necessary that very simple keywords should be added to the old database queries. This often happens when new constructs are introduced into the SQL or a standard modification across all the queries is needed.
2. Software developers play the roles of both developers and database experts, which deviate their focus from application development.
3. Although SQL statements serve the business logic, they may be constructed at the design phase to relief the developers in the implementation phase. Given that the development cycle is iterative and the changes in the implementation phase reflects back to the design phase, it would be very difficult to map modified SQL statements back to the design phase if they are bundled inside the code.
4. DB Experts cannot easily maintain SQL statements that are bundled inside the code either because of their ignorance of the programming language or because the code is very complex to the degree that it is not easily understandable or even readable.
5. Readability and understandability decreases, as the code that generates the SQL statements gets more complex.

7.4.3 Forces

There are some issues that have to be considered when working with this pattern: -

- *Cost versus flexibility:* The quickest way to access a database from an application is by writing the SQL statements directly inside the code. For the sake of flexibility extra code, storage, or memory will be required. However, a very simple change in the SQL statement may require compiling the whole application which could be very time consuming.
- *Standardization*: Separating SQL statements from code must maintain the standard SQL scripting to facilitate reading, debugging, and modifying them for the DB experts.
- *Maintainability:* The SQL statement interface must be very clear at the design phase. Although, it may be changed in the development phase. However, they must not be changed after they go in production at runtime. SQL statements are still considered an integral part of the business logic and modifications must be handled very carefully.
- *Complexity:* The pattern must provide a solution to organize the SQL statements and group them in a professional way. It must provide a technique also to handle the **IF** conditions by which different blocks of the SQL statements are usually grouped together inside the code.
- *Reusability:* The SQL statements should be available for other applications that may need to access the same Database though they may have different functionality.

7.4.4 Solution

In database-driven applications, there is usually a database access layer where SQL statements are embedded. The solution actually isolates these embedded SQL statements in an external format, for example plain text files, and may be distributed in more than one file. In order to provide an integration interface for the applications, SQL statements are defined as SQL templates.

SQL templates are newly introduced here as normal SQL statements with some special tags to identify the different parts of the statements that will change their values at runtime. Moreover, if SQL templates will be grouped together, then there must be a way to identify every template with a unique name. The beginning and the end of the template must be known. Although the best way to group SQL templates is to put them in one unit, they must also declare their group names to prevent the ambiguity of collecting unrelated templates in one unit. Table 10 provide more information on the recommended tags

Tag	Tag Name	Description	Example
--@begin NAME	SQL template begin	It identifies the start of the SQL template and gives a name to the template. Template names have to be unique in order to guarantee the right access.	**--@begin INVOICE_ACCESS** --@group INVOICE_GRP -- This template allows a -- proper access to the -- invoice table SELECT * FROM INVOICES WHERE INVOICE_TYPE = $INVOICE_TYPE$ AND PRICE > MIN_PRICE --@end INVOICE_ACCESS
--@end NAME	SQL template end	It marks the end of the SQL statement. The name of the template is mentioned as well.	--@begin INVOICE_ACCESS --@group INVOICE_GRP -- This template allows a -- proper access to the -- invoice table SELECT * FROM INVOICES WHERE INVOICE_TYPE = $INVOICE_TYPE$ AND PRICE > MIN_PRICE **--@end INVOICE_ACCESS**
--@group NAME	Template group name	If the template is not assigned to a group it will be added to the default group **default.**	--@begin INVOICE_ACCESS **--@group INVOICE_GRP** -- This template allows a -- proper access to the -- invoice table SELECT * FROM INVOICES WHERE INVOICE_TYPE = $INVOICE_TYPE$ AND PRICE > MIN_PRICE --@end INVOICE_ACCESS
TAG_NAME	Replaceable tag value	This identifies a replaceable tag at runtime. If the tag value is not replaced and the result will be unknown. The tag must start with $ and ends with the same character with no line breaks in between.	--@begin INVOICE_ACCESS --@group INVOICE_GRP -- This template allows a -- proper access to the -- invoice table SELECT * FROM INVOICES WHERE INVOICE_TYPE = **$INVOICE_TYPE$** AND PRICE > **MIN_PRICE** --@end INVOICE_ACCESS
--TAG_NAME	Replaceable comment tag value	This identifies a replaceable comment which is used to enable a certain construct	--@begin INVOICE_ACCESS --@group INVOICE_GRP -- This template allows a

		at runtime. The tag must start with --$ and ends with $ with no line breaks in between. For example, if --$all$ is removed and --$inv_type$ is removed and $invoice_type$ is replaced by 5 then the effective SQL will be
		
		Select * from invoices where invoice_type = 5
			`-- proper access to the`
			`SELECT`
			`--$all$  *`
			`--$invoice_number$ inv_no`
			`FROM`
			`  INVOICES`
			`--$inv_type$ WHERE INVOICE_TYPE = $INVOICE_TYPE$`
			`--$min_price$ WHERE PRICE > $MIN_PRICE$--@end INVOICE_ACCESS`

Table 10 SQL template definition

There are different complexity levels for constructing the SQL statements inside the application code. The simple one does not require replacing anything at runtime. The more complex level is where there are parameters needed to build the SQL statement, but the statement interface is not changed. The most complex level may require existence or non existence of different SQL statement blocks and the statement interface may even change at runtime. Introducing the technique of replaceable runtime tags inside the SQL templates makes the approach to these types very simple and easy. Table 11 gives examples of the listed types.

Type	Example	Description
No replaceable value	`SELECT * FROM INVOICES WHERE INVOICE_TYPE = 1 AND PRICE > 30`	There is no preprocessing needed for this SQL template.
Replaceable value only	`SELECT * FROM INVOICES WHERE INVOICE_TYPE = $INVOICE_TYPE$ AND PRICE > $MIN_PRICE$`	$INVOICE_TYPE$ and $MIN_PRICE$ will be replaced at run time.

Replaceable SQL constructs	`SELECT`
	`--$all$  *`
	`--$invoice_number$ inv_no`
	`FROM`
	`  INVOICES`
	`--$inv_type$ WHERE INVOICE_TYPE = $INVOICE_TYPE$`
	`--$min_price$ WHERE PRICE > $MIN_PRICE$`
		The field names are not always fixed nor the where conditions.

Table 11 SQL Construction complexity

The application will access the SQL templates by their unique name given at runtime. The pattern provides a loader and an indexer. The loader will be responsible for parsing the files that contain the templates and load them in memory as long as there are no errors. The indexer will save the SQL template in an accessible place by which the application can directly request the template by its name. It is clear from the indexer description that it is a cache. So, the cache pattern can be used in this context. Every template has a way to expose the replaceable tags either the normal ones or the comment tags. The class diagram (Figure 26) below depicts the general structure that can be used for this pattern. Figure 27 shows how database access beans can use the SQL statement templates to do their work in cooperation with a database connection.

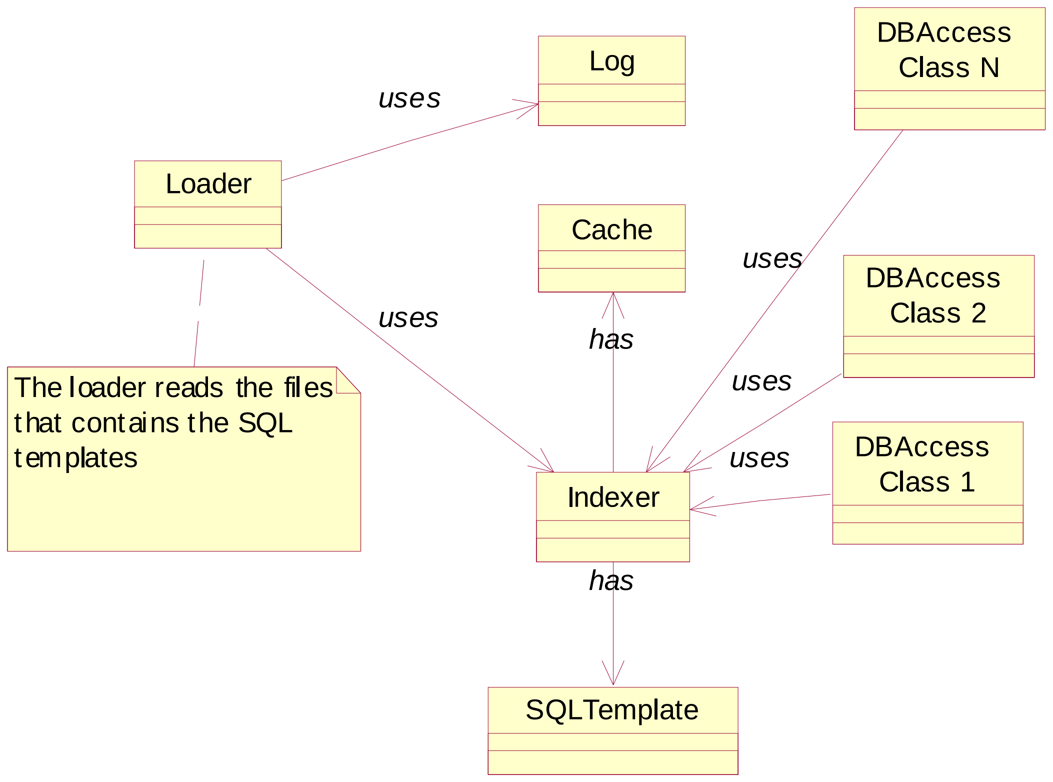


Figure 26 Decoupling SQL Statements from Application Class Diagram

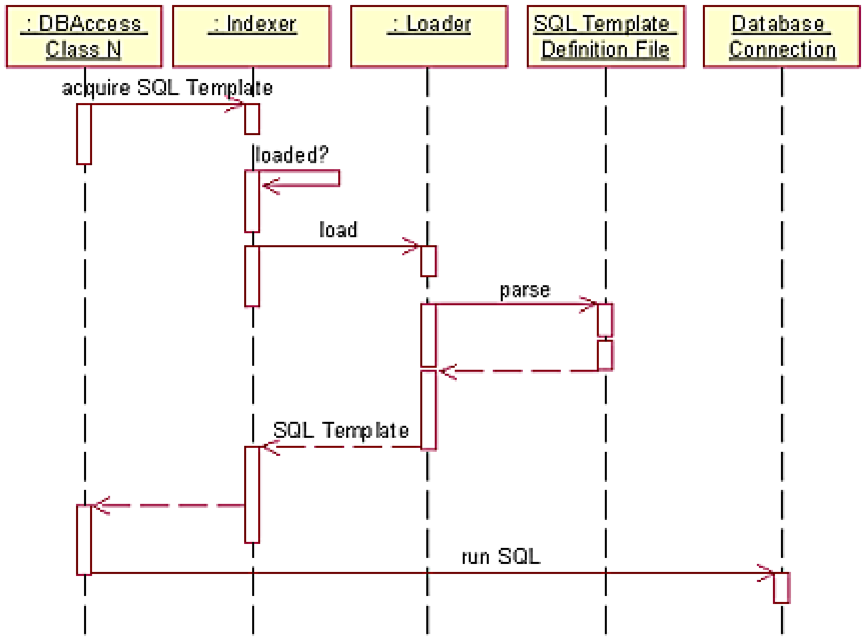


Figure 27 SQL Statement Template Sequence Diagram

The Java code below generates a query to get the top 10 calls for a certain customer according to his/her preferences. The top 10 calls can either be grouped by call, cost or duration. They can also be against National, International, or Roaming. It shows how it could be difficult to get the final SQL statement.

```
if (rpt.equals("call")) {
  query = "SELECT U.O_P_NUMBER AS NUMBER1, TO_CHAR(COUNT(*)) AS VALUE ";
  orderBy = " GROUP BY U.O_P_NUMBER ORDER BY COUNT(*) DESC";
} else if (rpt.equals("cost")) {
  query = "SELECT U.O_P_NUMBER AS NUMBER1, TO_CHAR(SUM(U.RATED_AMOUNT)) AS VALUE ";
  orderBy = " GROUP BY U.O_P_NUMBER ORDER BY SUM(U.RATED_AMOUNT) DESC";
} else if (rpt.equals("dur")) {
  query="SELECT U.O_P_NUMBER AS NUMBER1, TO_CHAR(SUM(U.ROUNDED_VOLUME)) AS VALUE ";
  orderBy = " GROUP BY U.O_P_NUMBER ORDER BY SUM(U.ROUNDED_VOLUME) DESC";
} else {
      stmt.close();
      conn.close();
      return null;
}
query = query + " FROM MY010_DWH_CALL_USAGE U";

if (grp > 0)
    query = query + ", MY010_GROUPS G, MY010_CONTRACT_GROUPS CG ";
String where = " WHERE ";
if (contract_key > 0) {
    where = where + "U.CONTRACT_KEY = " + contract_key;
} else {
    where = where + "U.CUSTOMER_ID = " + customer_id + " ";
}
// end of enhancement
if (type > 0) {
    query = query + ", CALLVIEW CV ";
}
query = query + where;
if (type == 1) {
  query = query + " AND U.CALL_KEY = CV.CALL_KEY AND CV.CALL_TYPE = 'National'";
} else if (type == 2) {
  query=query + " AND U.CALL_KEY = CV.CALL_KEY AND CV.CALL_TYPE = 'International'";
} else if (type == 3) {
  query = query + " AND U.CALL_KEY = CV.CALL_KEY AND CV.CALL_TYPE = 'Roaming' ";
}
if (grp > 0) {
  query = query + " AND CG.CUSTOMER_KEY = G.CUSTOMER_KEY AND CG.GROUPS_KEY =
          G.GROUPS_KEY AND U.CONTRACT_KEY = CG.CONTRACT_KEY AND G.CUSTOMER_KEY = "
          + customer_key + " AND G.GROUPS_KEY = " + grp + " ";
  }

CMy010Cust cust = new CMy010Cust();
cust.setCustomerKey(customer_key);
String bill_cycle = cust.getBillCycleCode();
query = query + " AND U.BILL_CYCLE = '" + bill_cycle + "' ";
query = query + orderBy;
// run the query and fetch the results
```

One may notice from the above code that there are different SQL statements generated with different field names, tables, where conditions, etc. The simplest SQL statement would be like this

```
SELECT
  U.O_P_NUMBER AS NUMBER1,
  TO_CHAR(COUNT(*)) AS VALUE
FROM
  MY010_DWH_CALL_USAGE U
WHERE
  U.CUSTOMER_ID = $customer_id$
GROUP BY U.O_P_NUMBER ORDER BY COUNT(*) DESC
```

The most complex SQL statement will be like this

```
SELECT
  U.O_P_NUMBER AS NUMBER1,
  TO_CHAR(SUM(U.ROUNDED_VOLUME)) AS VALUE
FROM
  MY010_DWH_CALL_USAGE U,
  MY010_GROUPS G,
  MY010_CONTRACT_GROUPS CG ,
  CALLVIEW CV
```

```
WHERE
  U.CUSTOMER_ID = $customr_id$  AND
  U.CALL_KEY = CV.CALL_KEY AND
  CV.CALL_TYPE = '$call_type$'  AND
  CG.CUSTOMER_KEY = G.CUSTOMER_KEY AND
  CG.GROUPS_KEY = G.GROUPS_KEY AND
  U.CONTRACT_KEY = CG.CONTRACT_KEY AND
  G.CUSTOMER_KEY = $customer_key$ AND
  G.GROUPS_KEY = $group_key$
GROUP BY U.O_P_NUMBER ORDER BY SUM(U.ROUNDED_VOLUME) DESC
```

The above Java code is complex to the degree that the developer has to write the code carefully in order not to make any mistake. Moreover, it is not easily readable or understandable though the resulting SQL statements are very clear. The following is a sample code that uses decoupling to solve this problem. It is shown only against the simple SQL statement

```
--@begin CUSTOMER_CALL_USAGE_ACCESS
--@group CALL_USAGE
SELECT
--$call$ U.O_P_NUMBER AS NUMBER1, TO_CHAR(COUNT(*)) AS VALUE
--$cost$ U.O_P_NUMBER AS NUMBER1, TO_CHAR(SUM(U.RATED_AMOUNT)) AS VALUE
--$dur$ U.O_P_NUMBER AS NUMBER1, TO_CHAR(SUM(U.ROUNDED_VOLUME)) AS VALUE
FROM
  MY010_DWH_CALL_USAGE U
WHERE
  U.CUSTOMER_ID = $customer_id$
--$call$ GROUP BY U.O_P_NUMBER ORDER BY COUNT(*) DESC
--$cost$ GROUP BY U.O_P_NUMBER ORDER BY SUM(U.RATED_AMOUNT) DESC
--$dur$ GROUP BY U.O_P_NUMBER ORDER BY SUM(U.ROUNDED_VOLUME) DESC
--@end CALL_USAGE
```

The code that will use this query ideally will be similar to this

```
SQLBlock sqlBlock = SQLIndexer().getSQL("CUSTOMER_CALL_USAGE_ACCESS ");
 // enable where keyword
 sqlBlock.setTagValue("customer_id", customer_id);
 // report_type is call, cost, or dur
 sqlBlock.clearCommentTag(report_type);
 Vector result = select(sqlBlock.getRuntimeSql());
```

Although, it is not recommended to write the SQL template where field names are not known until runtime, the code complexity has been greatly reduced and the SQL structure is now isolated outside the code and it is much better understood.

The SQL statements may also be externalized in XML format. For example, the DAO (Database Access Object) design pattern, which decouples the business logic from the data access logic, externalizes SQL statements to reduce redundancy. This is basically done in order to reduce writing similar APIs that share the same SQL statements [SUN02].

SQL statements that drive the business functionality can be introduced at the design phase as a separate deliverable. This deliverable can be integrated into the code with the minimum amount of effort. The developers can then easily change SQL statements in the development phase, if required. This leads the development team to focus more on achieving the functional requirements and not get too involved in database technical issues.

Actually it happens most of the time that the SQL statements are considered as an individual activity that is done as part of the implementation, even if there are implementation guidelines for the whole team. The implementation guidelines would not be effective in covering all the concerns regarding the database SQL statements. Since every developer is responsible for completing his/her job in the way he/she sees, SQL statements may not have a standard or a common approach. Moreover, there will be a delay in the development if the developer needs consultancy for building complex SQL statements,

For database driven applications, the database design must be introduced in the design phase. The database design task is mandatory because the application database access layer will be built over it. Any design activity in general takes quite some time of discussion in order to arrive at the best shape of the final product. The team that spent its time building a database design would definitely have acquired excellent understanding of the database architecture because they are the ones that took the design decisions. Hence, they are the best candidates to continue building the SQL templates.

The activity of building SQL templates must proceed directly after the database design is finished as the team has fresh knowledge of the database architecture. If they combine their knowledge of the database architecture with the application's functional and non-functional requirements, they would reach most of, if not all, the required SQL statements that would be needed in the implementation phase. Postponing the activity of finding the SQL templates to the implementation phase would lead to a delay in the development because the development team has to understand the database architecture before they start.

7.4.5 Consequences

- *Cost:* The SQL statement templates are loaded in memory at runtime which may be a burden on the system if there are a lot of SQL statements and the average size of every template is large. However, introducing a good caching mechanism can solve this problem easily to guarantee the optimized usage of memory.
- *Maintainability*: This solution may be tempting for some unprofessional

developers to change the SQL statements at runtime which may cause some trouble as changes are not tested. However, all protection methods can be taken in order to guarantee proper access to the application files when they are deployed to the production environment. For example the system administrator may provide read access only on the application files for the non-administrators.

- *Standardization*: Putting the SQL statements in plain text files is a standard way. Usually, both developers and DB experts read these files using an SQL editor. Such an editor makes their work very smooth.
- *Complexity*: The pattern increases the level of understandability and readability of the SQL statements for both the developer and the DB expert. It reduces also the code complexity that would have been used to build SQL statements. The developer no longer needs to trace the code to search for a simple SQL error and the DB expert does not have to understand the programming language. Both of them can simply read the SQL statement stored outside the application and figure out the problem with minimum effort.

7.4.6 Related Patterns

- *DAO (Database Access Objects)*: The pattern introduces the concept of separating the business logic from the data access logic. It also hides the data source from the application in order to make future changes in the data source irrelevant from the application business logic. It suggests also externalizing the data source access interface, like SQL statements, in XML format [SUN02].
- *Query Object*: it is an interpreter object that contains other objects which can be formed to generate SQL queries. The query object hides the knowledge of the SQL inside objects in order to get the change in the SQL statements done in one place only [FM02].

7.5 Pattern 5: Logger

7.5.1 Context

Systems that need to monitor their activities and problems can use this pattern specially when there is a need to trace performance and highlight its bottlenecks.

7.5.2 Problem

Logging is a solution to write, most of the time, certain informative statements in an external media, like plain text files, to record system problems or track system behavior. The log files are useful to the developer because they show the system behavior. The importance of logging is most significant when the application is in production. This practice is usually delayed until the development phase and it is usually done on individual basis and they generally do not conform to a certain logging standard. This leads at the end to messy log files that contain a lot of messages from many sources. This unprofessional development practice happens mostly because: -

1. Product time-to-Market is critical and logging is not one of the main functional activities.
2. The over-trust in the final product, especially if it is well tested.
3. Unprofessional developers do not anticipate future runtime problems. This is why they do not plan for them.

Moreover, if logging is practiced during development, it targets mainly the functional behavior of the system. Performance problems are the least logged issues especially if the system passes the load testing scenarios. However, logging must still be enabled when the system is in the production phase because test scenarios cannot detect all problems. For example, an e-business web application might be tested against a certain number of users and passes successfully. However, the system fails when it is available for public access because the number of users exceeds the tester’s expectation. The system could be retested, but it would be much more efficient if testing is combined with some knowledge from the log files. For example, a general performance problem can be detected with an e-business web application. If there are no logs, then the tester will not be able to tell from where exactly the problem arises. He/She has to test everything again. However, logs can show that a certain database SQL, for example, needs more tuning, if the performance behavior is logged.

The purpose of this design pattern is mainly to introduce a logging design pattern technique whose primary target is proper logging in general and then database query in order to build more robust designs. Once this pattern is applied on the design, it becomes a standard in the development that no one can ignore.

7.5.3 Forces

- *Planning for Logging:* It should be known ahead where to log and what to log.
- *Complexity*: The solution should not introduce unneeded delay or complexity to the system functionality.
- *Standardization:* The pattern must have a standard logging format.
- *System Behavior Handling:* The system behavior can be tracked only by logging events and problems. It shows the existing problems and highlights anticipated ones. A standard and uniform technique must be used across the working application at least.
- *Performance Tracking*: The pattern must give a good vision about the system performance.

7.5.4 Solution

The purpose of this design pattern is to introduce a technique which traces a system's performance in the first place and hence to build more robust applications. If this pattern is introduced to the design phase, it becomes a standard in the development phase that no one can ignore.

Enabling an application with the logging pattern requires two things: a good logging solution and a clear roadmap for logging. If either one of these activities is dropped, then logging will end up into chaos. The logging solution guarantees proper and standard representation of the application activities and problems, does not complicate the system, and introduces configuration capabilities. The *logging roadmap* is invented here as a way to let the developer know what and where to make logging. The roadmap is best started and represented in the design phase. One of the roadmap benefits is that it emphasizes and clarifies the duties of the developers regarding logging. Moreover, it prevents individual developers from inventing their logging methodology which may contradict with others' methodologies by providing a standard logging solution.

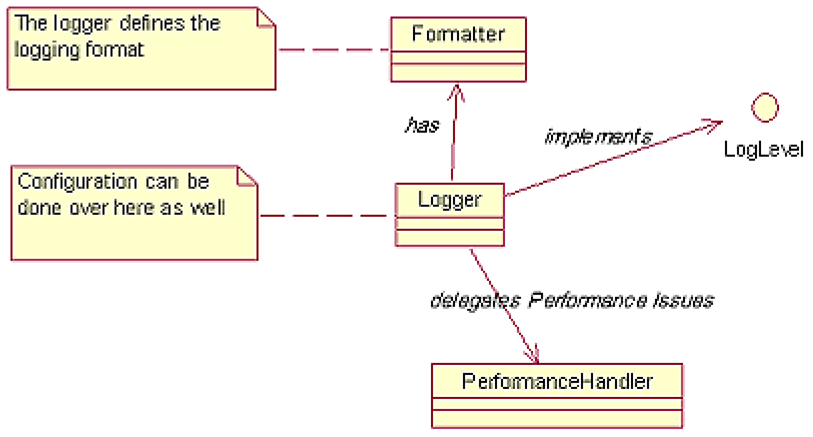

The logger defines the logging format
Formatter
has
implements
LogLevel
Configuration can be done over here as well
Logger
delegates Performance Issues
PerformanceHandler

Figure 28 Logging Pattern Main Structure

Figure 28 shows the main pattern components that the logging solution is providing which are the logging formatter, logging Levels, and performance problems handler. The logging formatter is used to render the message in a standard format, usually in an external media, e.g. a plain text file. The format has basic attributes by which the log line is understood: -

1. *Timestamp*: the time the message is printed.
2. *Log Level*: The type of the message, e.g. (Critical, Error, Warning, Information, Debug, etc).
3. *Logging Point*: The place where the message is logged. This should have been shown in the logging roadmap.
4. *Message*: The message is already defined from the logging roadmap.
5. *Performance Measurement (Optional)*: The criteria by which the system can tell that there is a possibility of a performance problem.

These attributes have to exist even if the solution will be modified or extended. The first four attributes are common for many logging solutions. The performance measurement attribute is a suggested addition in this research. For example, the measurement criteria can be the maximum amount of time that the database SQL statement cannot exceed, e.g. 15 seconds. If an SQL statement exceeds 15 seconds then the logger will log this problem indicating that it is an expected performance bottleneck. In addition, this performance message can be delegated to another handler which can add more explanation, take action to solve the problem, or notify the system administrator.

Logging Levels is the method that makes the system administrator control the amount of information logged out of the application. It is the responsibility of the system administrator to determine the kind of information that should be logged, from the log levels, according to the system stability and the business need. However, the log level tuning must not put a burden over the system itself and cause a performance delay. For example, if the system has a CRITICAL log level at one end and DEBUG log level at the other end, then the administrator may choose to start with the DEBUG level in the soft launch of an e-business application. A DEBUG level means that the application will log all the messages for this level and for the other levels. When the customer visits increase, the level can be set to CRITICAL to log only the CRITICAL messages.

If the logger encounters a possible performance problem, it delegates it to the performance handler which captures the message, analyzes it, and takes an action. For example, the performance handler may check the memory, disk space, database connectivity, etc and try to recover them if possible. However, the message has to be detailed enough to indicate the right source of the problem. Actually, the performance handler can be a system in its own right which makes a full self-recovery and that requires a good Knowledge base. Figure 29 summarizes the previous discussion about the logger.

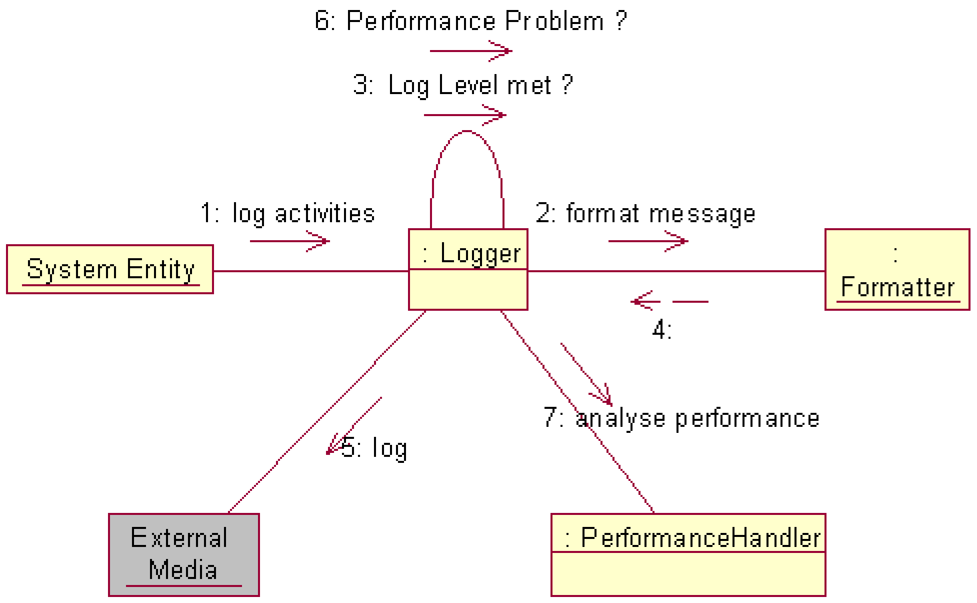


Figure 29 Logger Collaboration Diagram

On the other hand, a *logging roadmap* should be developed on its own to indicate the locations that will hold logging points. This activity can be started from the design phase. A separate iteration should be conducted over the design after it is completed. The logging roadmap iteration process involves the following activities-

1. Visit all the collaboration diagrams and depict where the message should be printed.
2. The message or at least the type of information that should be logged (e.g. error, message, or performance) should be decided at every location.
3. Performance measurement criteria may be decided for some components. Sequence diagrams can be used to highlight the locations that will most affect performance.
4. A separate artifact can be delivered out of this iteration to indicate the locations, log level, message, and whether there is a performance logging or not.

For example, Figure 30 clarifies precisely the methods that are needed to log events, errors, and performance behavior. Notice that the diagram does not show how logging will occur as it is not the issue discussed in the logging roadmap. Table 12 shows a suggested logging roadmap document that can be delivered to the development team to start working. This document can evolve to contain more roadmap points. Actually, the logging roadmap is not a waste of time. It can actually be added to the documentation and handed to the system administrator as part of his manuals.

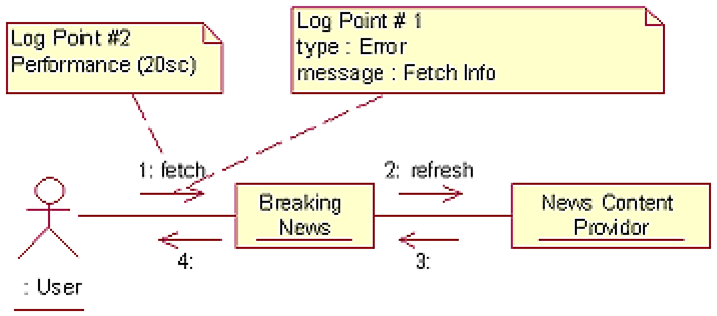


Figure 30 A Collaboration Diagram Showing Logging Roadmap

Log Point	Class Method	Type	Message
1	BreakingNews.fetch()	Error	Fetched info
2	SportsNews.fetch()	Performance (20sc)	Performance problem in database

.

.

.

Table 12 A Logging Roadmap

This standard logging technique opens the door for log analyzers to run over the logs at runtime and generate valuable reports about the system. These reports can be very important for the administrators, developers, and even the system owners. Bug fixing and enhancement can be partially fetched from such reports.

7.5.5 Consequences

- *Cost versus Performance*: Logging is considered an infrastructure practice to any system although it may increase the development time. If a product is overwhelmed with logging points, its performance will suffer greatly. Logging must be classified to levels which would allow the application administrator to change them at runtime.
- *Behavior Monitoring*: The choice is for the developer to either log or not. As mentioned before if he/she logs then the system runtime behavior will be very clear for him/her. If not, then system failures, if any, cannot be explained until the system is retested.
- *Consistency:* By composing a logging roadmap, developers will be obliged to use common messages and by using the same logging format logs are no longer messy.

7.5.6 References

- *Java Logging API*: Java has recently utilized new Java APIs. It has become part of the language.
- *HTTP server logging:* All well-known HTTP Servers like apache, Sun IPlanet, IBM http server, or Microsoft IIS use logging techniques. Moreover, there is a well-known format of the logs which allows other reporting applications to analyze them and generate valuable reports.

- *Windows system logging:* Windows operating systems have a logging tool that shows the results of running applications.
- *UNIX logging*: UNIX operating systems can be configured to log system activities by sending a mail to the administrator. These mails are by default sent to the user mailbox to show the status of the applications in case of failures.

7.6 Performance Pattern Language

The *Performance Pattern Language* is the collection of the selected patterns with guidelines of the best method to use them. Figure 31 shows the three categories of patterns into which the five patterns that resulted from this research study, can be divided: -

1. Performance optimization patterns that have direct impact on enhancing performance. The Database Connection Pool, Cache, and Static Enabler Patterns fall into this category.
2. The Logger pattern monitors application behavior and performance related issues.
3. The SQL Statement Template pattern breaks down the application complexity to help in optimizing database SQL statements.

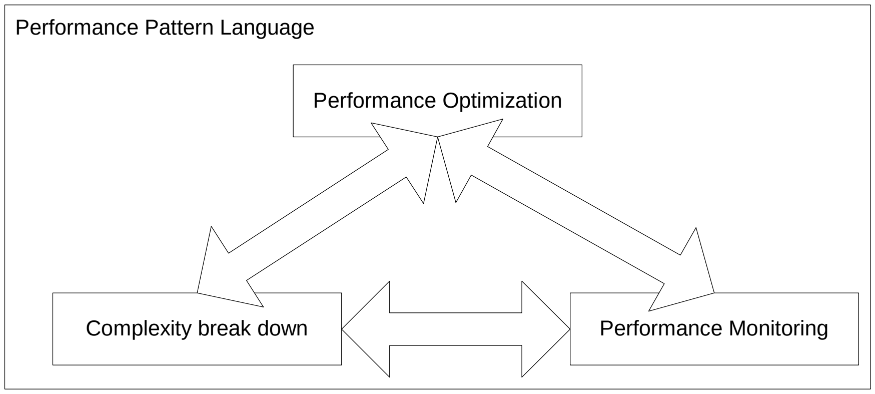


Figure 31 Performance Pattern Language Categories

Every one of the listed patterns can be introduced on its own. However, introducing all of them together makes the developer handle the performance optimization process from many aspects. Figure 32 shows the suggested relationship among all the suggested patterns. The figure shows that the Logger and the Cache are the most recognized patterns. The logger is considered also as an infrastructure for all the other patterns although it is not a mandatory component for all the patterns.

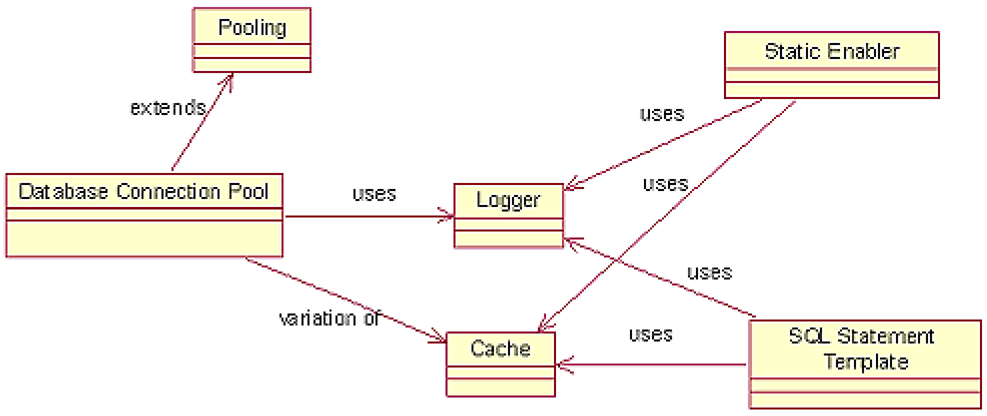


Figure 32 Performance Patterns Relationship

The SQL statement Template provide the developer with an easy and understandable method to tune and fix the database queries which are the access points to the source of information. The Database Connection Pool, the Cache, and the Static Enabler patterns provide direct improvement to the web applications if they are properly applied. However, the logger remains the application eye that tries to see other performance bottlenecks and application problems in general.

For example, if the developer chooses to apply only the performance optimization patterns in a database-driven e-commerce application that have thousands of visits then he/she may face problems. These problems may not be resolved easily because of his/her ignorance of a particular problem and the best way to solve it. Even if the problem is known, which will definitely take more time for investigation, then fixing it may be very expensive if it had resulted from the database SQL statements. The fixing process requires the following: -

1. A thorough investigation of the application code to know the source of the problem. The investigation may require adding ad hoc logging statements, which may be left after the problem is resolved to introduce other side effects to the application.
2. If the source of the problem is due to an SQL statement, after investigation, then the developer has to fix it and maintain at the same time the application code consistency after he/she finishes.
3. The developer has to test the application again not only against the failure point but also against all other related cases in order to ensure that the fixing does not corrupt other components in the application.

4. The new compiled code has to be deployed, which may result into a down time of the web application.

On the other hand, if the application was enabled with the Logger and the SQL Statement Template patterns, then the source of the problem would have been most properly known in details. This logged information provides the technical people with the guidance to investigate, if investigation was required, and may not require investigation at all. Moreover, if the fix of the problem would require tuning an SQL statement, then the SQL statement change is done straightforward in the SQL statements file. Also, the SQL statement fix could be tested against the database directly without getting the application down. The code of the web application in this case was not changed at all.

Figure 33 shows our suggested approach for performance optimization. In the design phase, all the patterns can be developed concurrently, but the roadmap visits for logging and caching should be done at the end of the design phase. The expected artifacts to be delivered are: -

1. *Cache Roadmap Document:* where caching positions are shown clearly against the class methods. [Refer to the caching pattern in section 7.2 for more information.]
2. *Logging Roadmap Document:* where logging points are shown against the class methods. [Refer to the Logger pattern in section 7.5 for more information.]
3. *SQL Statements Templates Document*: This is an initial SQL statement specification, which may be changed in the implementation phase. This document can feed the implementation phase directly without any change in the document format.
4. *Collaboration Diagrams (Revisited for Caching & logging)*: If the roadmap documents show the logging and caching points across the whole application, collaboration diagrams show them across the different scenarios.

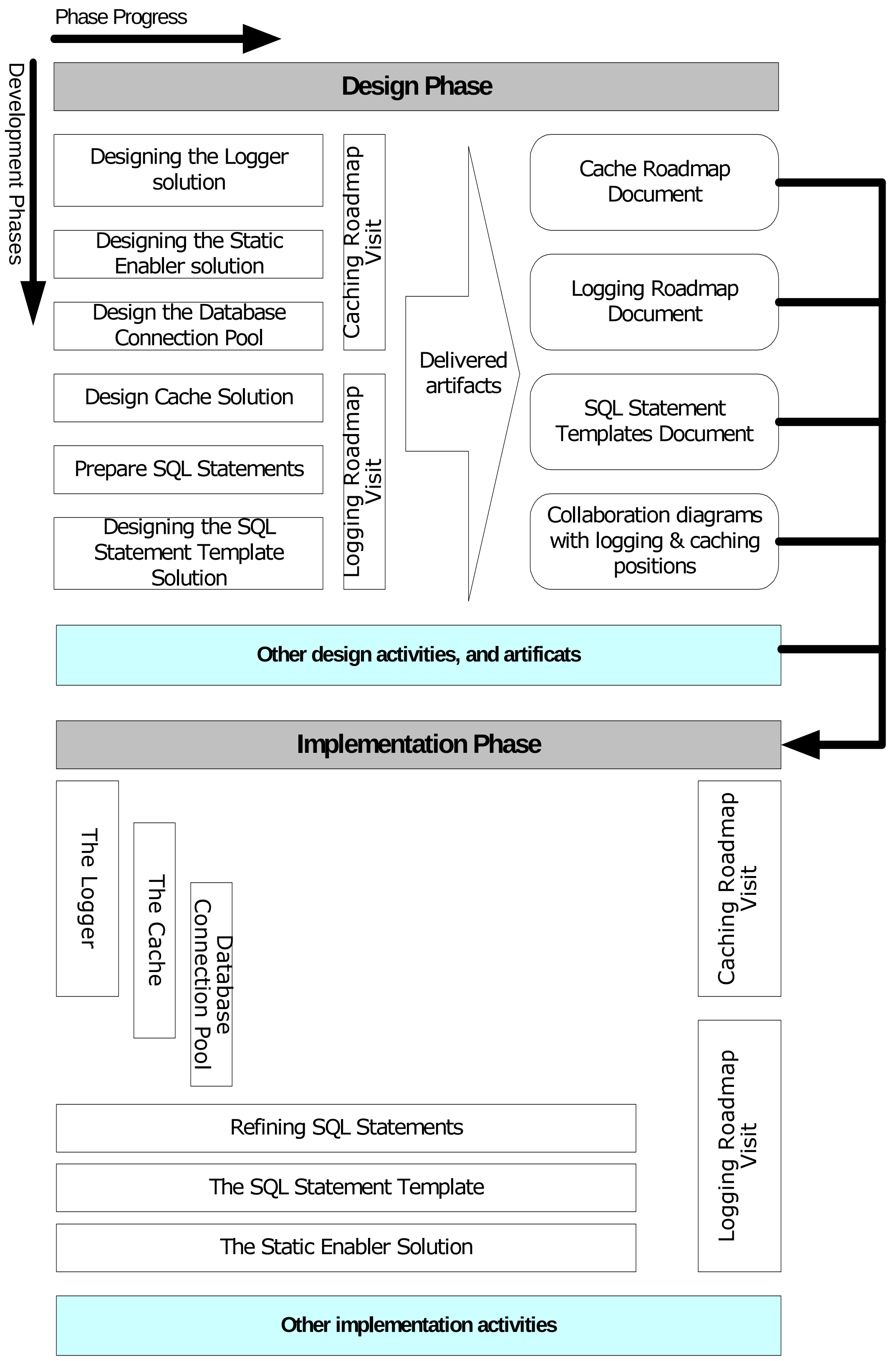

Phase Progress
Development Phases
Design Phase
Designing the Logger solution
Designing the Static Enabler solution
Design the Database Connection Pool
Design Cache Solution
Prepare SQL Statements
Designing the SQL Statement Template Solution
Caching Roadmap Visit
Logging Roadmap Visit
Delivered artifacts
Cache Roadmap Document
Logging Roadmap Document
SQL Statement Templates Document
Collaboration diagrams with logging & caching positions
Other design activities, and artificats
Implementation Phase
The Logger
The Cache
Database Connection Pool
Refining SQL Statements
The SQL Statement Template
The Static Enabler Solution
Caching Roadmap Visit
Logging Roadmap Visit
Other implementation activities

Figure 33 The Pattern Language Approach

The artifacts comprise the starting point for the implementation phase. As shown in Figure 33, there are partial dependencies across logging, Caching, and the Connection Pool. The logging solution should be developed first because it is the infrastructure that will be used in the cache, and the connection pool. Logging is used globally across the whole application. However, it does not prevent the implementation of other scenarios such as refining SQL statements, implementing the SQL Statement template pattern, and implementing the Static Enabler pattern. In order not to disrupt the work of the development team with the non-functional activities, an iteration can be conducted over the whole application to insert the logging and the caching points. This iteration can be conducted on the class, component, package, and application levels with the help of the roadmap documents. Note that the implementation plan may start logging and caching roadmap visits early according to the collaboration diagrams check points.

It is important to understand that the activities that are performed in the design phase and resulting to artifacts are mandatory to develop the suggested design patterns. However, they provide the development team with the best practice to efficiently use the patterns. In other words, the wrong usage of the patterns invalidates their implementation and makes the developed application unreliable.

Chapter 8

ANALYSIS OF THE SUGGESTED PATTERNS

In this chapter, the results of experimentation with the suggested patterns are analyzed and discussed.

8.1 Quality Validation

The quality validations listed below are intended to give an analytical view of the suggested design patterns. They attempt to qualify the suggested pattern language from many aspects in order to prove that it is worth implementing. They do not represent a judgment on the validity and applicability of the pattern language since judgment should be left to the reader. It gives a broad discussion of the suggested pattern language and shows examples from the five patterns, whenever possible, to prove their validity. However, judgment can always differ from one reader to another.

8.1.1 Encapsulation and Abstraction

All the suggested patterns define their problems and give detailed solutions according to their context. As a reminder, all patterns are targeting database-driven web applications. For example, the Static Enabler Pattern specifies its solution for the web applications that want to combine the benefits of both the dynamic and static pages. So, the applications that will live forever as static may not use this pattern as well as the applications that have dynamic pages and will not accept any static pages.

All the patterns suggested in this research were introduced in an abstract format in order to allow the developer to seek the most suitable implementation according to the available resources and the selected programming language. The developers are not forced to use specific method names or even class names. Moreover, almost all the introduced patterns can work elegantly if applied to database-driven desktop applications. If a desktop application has a concurrent access feature and it is database-driven then there will be no problem applying them. In addition, the concept of the SQL Statement Template Pattern could be applied to the client-server applications that depend on XML formatted messages in their communications.

8.1.2 Openness and Variability

Because of the abstract nature of the patterns, they could be extended to give more specific ones according to the application context. For example, the database connection pool pattern can be extended to be specific to certain database systems like Oracle and DB2. Moreover, the Database Connection Pool and Cache patterns can have different variations like lazy (resources are allocated on demand) and eager (resources are allocated at once) acquisition in both of them. The same concept could be applied to the Static Enabler pattern by which the conversion of the dynamic pages to static may be done on demand or according to a scheduler that converts all the dynamic pages.

There is nothing that prevents the implementation of all the patterns using Java, C#, or C++ for example since their designs are generic enough. They could be implemented on different platforms, as there are no constraints regarding this point. Moreover, the patterns are totally independent from any programming language. For example, the SQL Statement Template pattern introduces the *Indexer* component which is responsible for indexing the SQL statement templates in the application memory to speed up their access. Actually, such a concept is natively implemented in a language like *Java* as a class called *java.util.Hashtable.* Although the proof-of-concepts implementation of the patterns was done in *Java* the component itself was introduced completely separate from *Java* to leave up to the readers to implement them according to the chosen programming language.

8.1.3 Equilibrium

All the patterns introduced numerous benefits each according to its context. The optimization patterns are specifically supported by testing results as specified in section 8.2. The side effects shown in every pattern would be expected if the developers apply the patterns in a wrong context or if they mis-implement them. For example, database-driven applications, that have small memory and very few users, may be hurt if connection pooling is used with eager acquisition. Please refer to the Consequences sections for every pattern for more details on their pros and cons.

8.1.4 Minimality

All the patterns have shown their relations with other patterns either in this research or from other researchers. For example, The Connection Pool pattern may recognize other patterns from the research like the Cache and the Logger. The Cache is recognized by itself from the Connection Pool and the SQL Statement patterns. Please refer to section 7.6 for more information about the patterns relations.

8.2 Testing

The test cases were tested on two environments, windows and Solaris operating systems. The load testing was conducted on the Solaris operating system with an average number of virtual users. The results were found satisfactory since it is a load testing experiment to prove the validity of the proposed solutions and not to verify the reliability of the web application used in the testing. Testing was not intended to compare design patterns implementation with other solutions as the research proposes design solutions that could be implemented differently from one developer to another.

The Vodafone Website was used as the testing web application in this research. Two sample pages from the website promotions channel were mainly used in all the test cases. They are of average size content and receive a fair number of database requests as follows:

- *A List of All Vodafone Promotion Types:* The content size is almost 20K and content is retrieved out of 19 database requests.
- *A List of A promotion Type Details:* The content size is almost 24k and content is retrieved out of 26 database requests.

8.2.1 Memory Utilization

The memory utilization is an implicit job for any application where memory objects are allocated and de-allocated. As the number of memory allocation and de-allocation operations increases, the performance of the application decreases. Both allocation and de-allocation operations are application decisions. If a memory object is allocated and not de-allocated, this will lead to a memory leak; hence the application becomes unstable and most probably it will crash.

A good optimization of the memory utilization according to its capacity can make the application live longer. Hence, a memory utilization analysis has to be done for the application to estimate the needed memory. The memory analysis has to take into consideration the expected number of objects, their size, and the expected lifetime of every object. Other systems that have concurrent behavior like the web applications must be analyzed against memory utilization and user load. For example, there could be a web application that does not have memory leaks at all, but it has very high utilization of memory and a high number of users. In the web application rush hours, the high number of users will slow down the application because there will be wasted CPU cycles on the server for memory allocation and de-allocation operations.

A Database driven web application in particular can suffer greatly from the database connections utilization, and database objects. The database connections usually consume a good memory size, take good number of CPU cycles, and consume a fair I/O percentage of the server operations. By allocating a pool of connections, the heavy memory utilization is greatly reduced especially if the pool size stays stable for a long time since all memory operations will be restricted in the read operation. The number of database objects is basically tied with the number of connections. However, the database objects can vary in size dramatically. Their memory utilization behavior may be very close from the database connections memory utilization behavior if not more. Caching usually solves most of the database objects utilization problems. However, cache size usually changes more quickly than the database connections pool size.

To summarize the above discussion, the memory utilization for a database driven web application is directly affected by the following factors: -

1. The number of objects in the memory.
2. The size of every object.
3. The lifetime of the objects in memory.
4. Memory allocation and de-allocation rate.
5. The number of users at a certain time.

CPU is the main victim of all memory utilization operations. It has to strive all the time to allocate and de-allocate memory for the application process. Some operating systems utilize the disk space to allocate virtual memory for the processes. Some other operating systems, like UNIX, swap processes to disk directly if there is not enough physical memory. It can be understood that processes that have part of its memory swapped to disk will perform slower than those that work from physical memory. The application process that requires more memory will suffer greatly if the part of its memory will be swapped to disk because its memory utilization's normal delay will increase until the required memory blocks are loaded to the physical memory.

8.2.2 Performance Optimization Evolution

The first testing scenario shows that a web application performance can be enhanced greatly if it implements the performance optimization techniques suggested in the research. This testing has been run on a IBM WebSphere application server test environment. The test environment is by nature slower than the real products. However, since the environment is not changed, the result can be acceptable to verify the optimization point. The following are the runs that have been tried: -

1. *No Connection Pool & no Caching*: A database connection was allocated with every database hit and released after the database query returns. The query result was always fetched from the database as well. This scenario represents the situation in the absolute absence of the minimum available patterns. It could be thought of the control of the experience.
2. *Connection Pool:* A connection pool solution was introduced but without caching.
3. *Caching*: Caching is enabled in addition to the database connection pool.
4. *Static Page*: Identical copies of the dynamic pages have been tried.

The following settings have been fixed during all the runs

1. The test scenario ran for one minute
2. Number of users = 1
3. Number of test pages = 2

Page	Throughput				Avg. Latency (sec)				Avg. Response Time (sec)			
	R1	*R2*	*R3*	*R4*	*R1*	*R2*	*R3*	*R4*	*R1*	*R2*	*R3*	*R4*
A list of All Vodafone Promotion Types	5	12	14	613	0.2	0.1	0.095	0.04	5.5	3.1	2.35	0.047
A list of A promotion Type Details	5	11	14	613	0.1	0.11	0.097	0.05	8	2.5	2.05	0.05
Over All Avg.	5	12	14	613	0.2	0.1	0.096	0.045	6.7	2.8	2.2	0.048

Table 13 Performance Evolution Testing Results

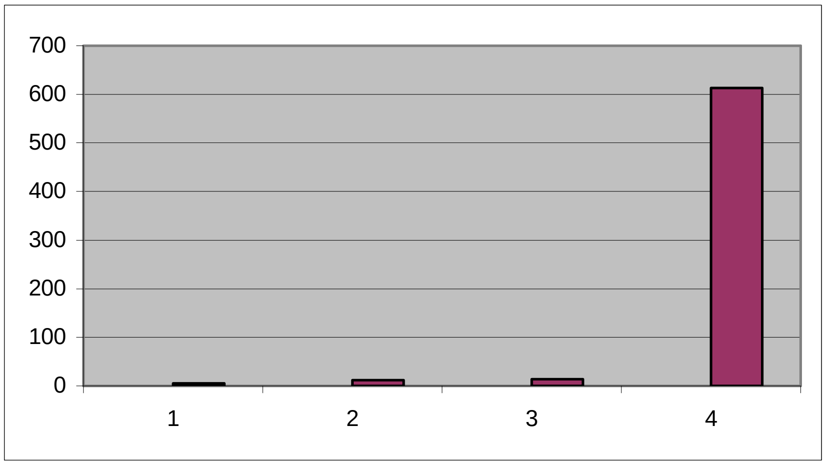


Figure 34 Average throughput per page

We note the following on the above results as following: -

1. In general throughput, latency and response times show improvement.
2. There is a noticeable improvement from run1 (no connection pool and no caching) to run2 (connection pool and no caching). However, the improvement from run2 to run3 (connection pool and caching) is not as large.
3. Throughput, latency, and response time are dramatically much better for the fourth run (static html) than any other run.
4. Latency values for all the runs are very small compared to the Response time values.
5. Latency values for all the runs barely improved from the first run until the fourth run.

The analysis of the above points and data gathered in Table 13 shows that the solutions used to optimize the performance are generally effective. Moreover, implementing a connection pool without implementing caching can be sufficient if the database SQL statements are fast enough. However, implementing a caching solution can show improvement but it does not mean that the database connection pool should not be implemented since the cache will be refreshed and the application will open a database connection at any point in time. The optimization solutions show that the best optimization solution is to have static pages as shown in Figure 34.

To summarize, dynamic web applications that do not plan to have static pages as replacement must have connection pooling. Caching is mandatory if the database

SQL statements are very time consuming, but it can be neglected if the SQL statements show a fast enough response time. Static web applications do not have to worry about performance since they have good throughput and response time by nature.

8.2.3 Load Testing

A test environment was established as described in Appendix D and a load testing utility was used to impose virtual users over the web applications. The utility used is described in more details in Appendix E . Load testing results were gathered from the testing tool and the web application server. These results are listed in detail in Appendix F . Load testing was conducted over several scenarios as following: -

1. *S1 (No Connection Pool & no Caching)*: A database connection was allocated with every database hit and released after the database query returns. The query result was always fetched from the database as well. This scenario represents the situation in the absolute absence of the minimum available patterns. It could be thought of the control of the experience.
2. *S2 (Connection Pool)*: A connection pool solution was introduced but without caching.
3. *S3 (Caching)*: Caching is enabled in addition to the database connection pool.
4. *S4 (Static Page)*: Dynamic pages have been converted to static ones to test their performance and compare it with the previous three scenarios.

The following settings have been fixed during all the runs

1. The test scenario ran for one minute
2. The Number of test pages was fixed to two (Refer to section 8.2 for more information on pages).
3. The numbers of virtual users used are 1, 10, 30, 60, and 100 virtual users.

The following points were noted from the load testing: -

1. There is a noticeable improvement from the first to the fourth scenario.
2. The fourth scenario is generally much better than all other scenarios.
3. The first scenario is the worst scenario for all the measurements.
4. When there were 100 virtual users the throughput declined for the third and the fourth scenarios.

5. The fourth scenario experienced some failures for 60 and 100 users.

The analysis of the above points and the data in Table 14 show that a connection pool solution is a must especially when the number of users increases otherwise the users will always experience bad latency and response time (as in Figure 35 and Figure 36). Moreover, a connection pool solution is enough for an environment that has a low number of users. However, other solutions, like caching, are required to improve the performance when the number of users is expected to increase. Caching may be enough for the environments which normally have a large number of users to achieve a good response time. Hence, converting dynamic pages to static ones shows a fair improvement in throughput than caching (see Figure 37).

	Throughput				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
Over All Avg.	**S1**	**S2**	**S3**	**S4**	**S1**	**S2**	**S3**	**S4**	**S1**	**S2**	**S3**	**S4**	**S1**	**S2**	**S3**	**S4**
1 User	7	53	222	455	0.02	0	0.06	0.03	4.3	0.6	0.1	0.1	0	0	0	0
10 Users	10	369	901	1694	0.38	0.1	0.05	0.04	35.8	0.8	0.3	0.2	0	0	0	0
30 Users	30	377	930	1761	17.1	0.5	0.19	0.09	116	2.4	1	0.5	0	0	0	0
60 Users	60	418	954	1789	89.5	2.7	1.14	0.6	197	4.5	1.9	1	0	0	0	0.0083
100 Users	100	430	947	1601	170	5.6	2.45	1.44	277	7.5	3.3	2	0	0	0	0.0085

Table 14 Over All Average (Load Testing)

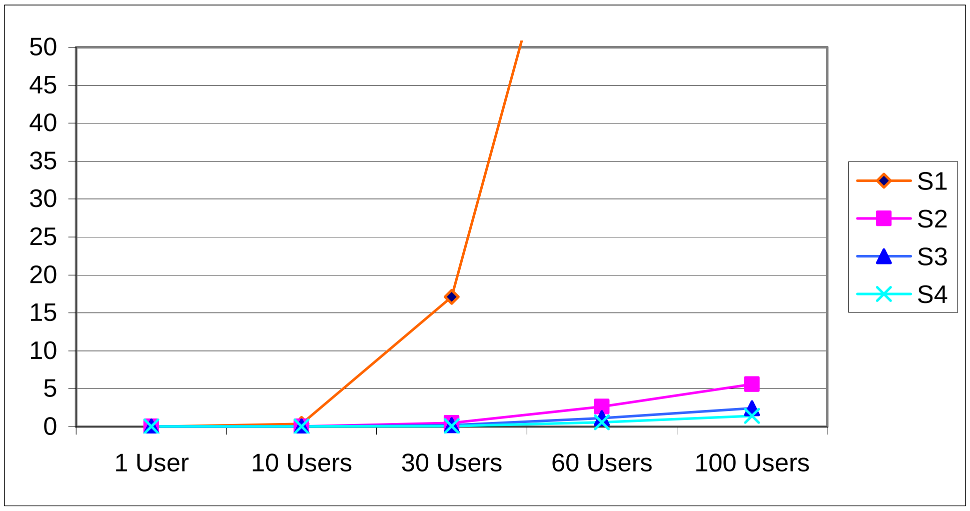


Figure 35 Average Latency (Load Testing)

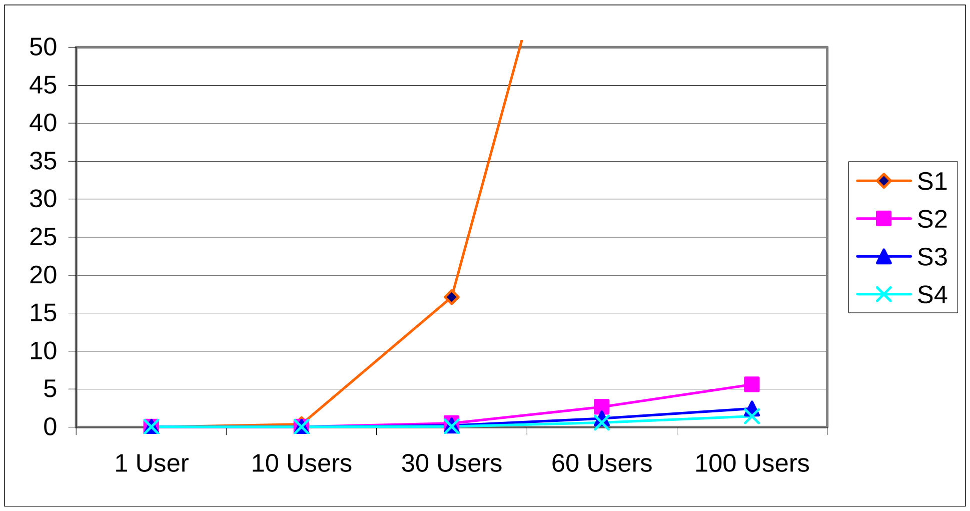


Figure 36 Average Response Time (Load Testing)

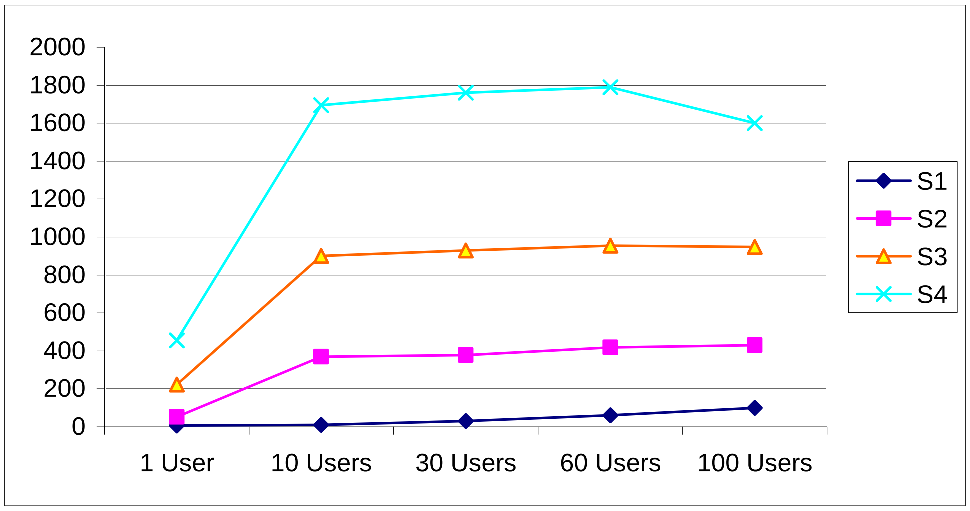


Figure 37 Average Throughput (Load Testing)

On the other hand, the server side analysis did not show server memory utilization work. This is mainly due to the nature of the application server which is based on *Java*. Java garbage collection policy works as a lazy process where memory is allocated when needed. However, the CPU utilization has shown some noticeable change as a result of the load testing. Figure 38 shows that the CPU utilization for the first scenario. It is mainly because every virtual user was very busy retrieving a page that takes a very long time to download which left the CPU idle with almost no work. However, the case was different with the other three scenarios where CPU utilization was in the normal distribution shape. As shown in Figure 38 the fourth scenario (static pages) has the lowest CPU utilization behavior, followed by the third scenario (caching), and then the second scenario (connection pooling).

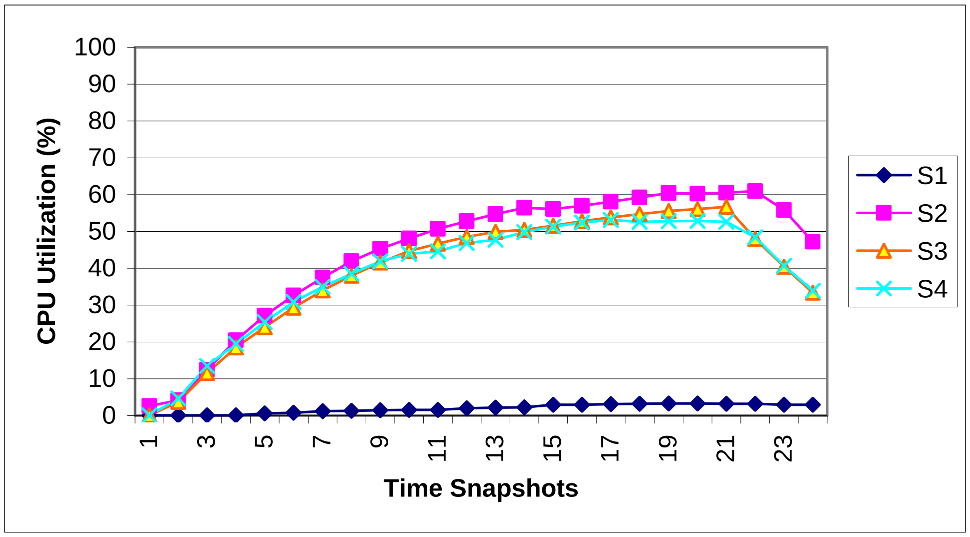


Figure 38 CPU Utilization for load testing (100 users)

In the second scenario, the memory utilization is high, as database queries are run and allocated in the memory every time there is a database access. The third scenario relaxes the memory utilization by using the cache. The fourth scenario does not go to the application server at all, but it still has a high CPU utilization due to the very high throughput.

The following may be observed from the obtained results in (Appendix F ,Table 22) which are mapped in Figure 39 regarding CPU I/O waiting: -

1. CPU I/O waiting for all the scenarios did not exceed 50% of the CPU processing in general.
2. The third scenario (Caching) shows zero I/O waiting.
3. The second scenario (Connection Pool) shows noticeable fluctuating in CPU I/O waiting.
4. The fourth scenario shows almost zero I/O waiting but fluctuate abnormally at one point in time.
5. The first scenario shows very small I/O waiting on scattered intervals.

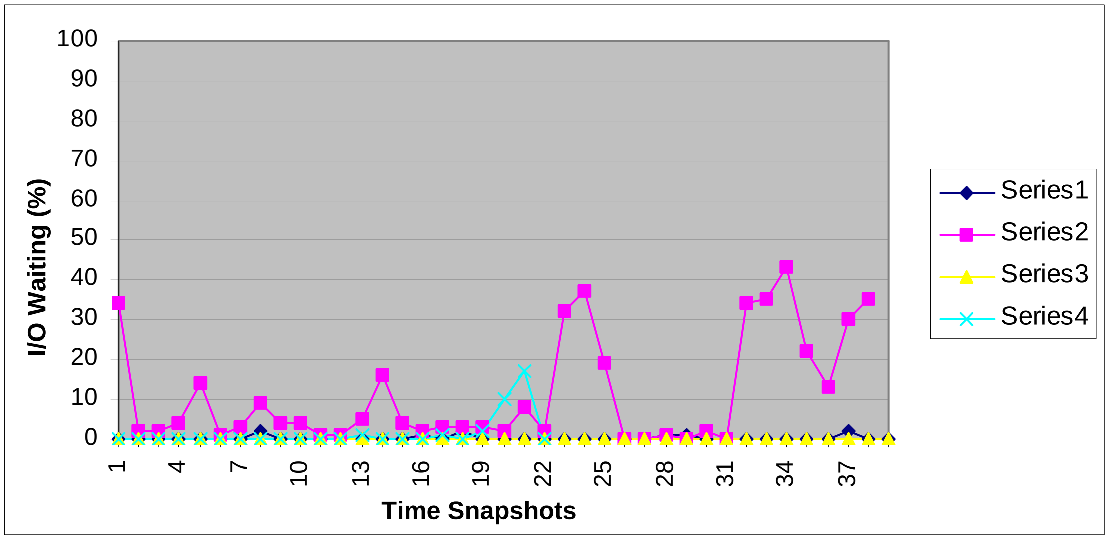


Figure 39 I/O Waiting for load testing (100 users)

A large value for I/O waiting is a bad indicator for the application because it means that the process is idle due to blockage in I/O. The second scenario which made the highest fluctuation in I/O waiting time curve due to the fact that the throughput was high and the hits that were not served directly with a database connection had to wait until a connection was returned or until a new database connection is created. In the third scenario there was no access to the database at all; that is why the throughput in the third scenario is far beyond that of the second scenario. The fourth scenario experienced some I/O waiting mainly because the very high throughput and the limited connections allowed by the web server made the virtual users wait for sometime acquiring a connection. The first scenario did not show any active I/O waiting values because its throughput was very low compared to the other scenarios and I/O was distributed across the whole load testing period.

I/O waiting analysis shows that the Cache and the Static Scenarios achieve d the maximum utilization of the CPU because they are served with the information directly. However, the second scenario is served from the memory directly, but the fourth scenario is served from the files on the disk. The database connection pool wastes CPU cycles because the users had to wait for connections or they had to acquire new connections.

To sum up, if one wants to achieve the best performance for a database-driven web application, then he/she must have a database connection pool, a cache, and may be some static pages. It is left to the user to decide which content should be cashed and which should be converted to static, given that the dynamic pages are still there. The web applications that have a very high number of hits may also build redundant hardware to achieve better performance. However, web applications that have small to medium visits will stay stable and reliable if the optimization solutions are implemented without the need to add more hardware.

Chapter 9

SUMMARY AND CONCLUSION

The aim of this research work was to suggest a number of essential design patterns and a pattern language that handles performance problems for database-driven web applications. The emphasis is to show the importance of design patterns and their use as stand-alone designs and with other technologies like frameworks. The suggested design patterns in this study are meant to help in solving performance problems, breaking down database SQL statements complexities, as well as monitoring resource utilization. They could be applied as well in normal desktop applications. However, from experience, they are very effective on web applications where the customer load is most probably unknown. It is very important to mention here that performance issues must be addressed as early as the design phase, or even earlier, and it is not enough to tune some parameters after the development phase to get the required performance.

9.1 Research Conclusion

Studying web applications in details within the context of a certain organization shows that it is the web application that varies where the web server, application server, and database are all products that have been tested and verified. Problems arise, most of the time, from the application itself and its misuse of the available resources.

There are many benefits for working with the suggested design patterns for web applications in general: -

1. The patterns help in building a solid performance optimization infrastructure for web applications. This can be integrated later on inside a framework.
2. Performance bottlenecks are considered at the early phases of design. For example, if developers properly use connection pooling throughout a project, then establishing a connection with a database will not be blamed for delay.
3. Post implementation performance problems will be easily tracked and analyzed where log analysis engines could be implemented to give accurate reports on system performance.

4. Testing scripts can give more weight for functional and non-functional testing. Performance testing must be conducted as well. However, testing scripts will not take much time to complete.
5. Tuning database queries will be an easy process that is not tightly coupled with the code.
6. Highly accessed dynamic pages can be converted to static versions, if possible, which puts the entire load on the HTTP server and leaves the Application Server, and the database, to be utilized more efficiently in other areas.

It would be practically more useful if a pattern language was bundled within a framework that is tailored for web applications. Actually, the suggested pattern language in this research can then be used to document the framework's performance issues. Or it can become the nucleus of a new framework that focuses on web applications.

9.2 Directions for Future Work

The area of design patterns is still in need for future research in many aspects. The mentioned patterns are just a small number of patterns that may be used in an application. The following points outline some ideas of anticipated research: -

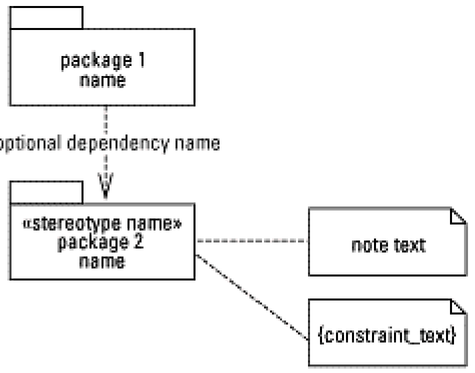
- *Decoupling Database SQL Statements from Applications:* An application can be developed to help the designer build the required SQL templates and produce them in the required format.
- *Logging to Debug Performance Issues:*
    - Generic Log analyzers can be developed to report against system behavior and performance issues.
    - The developed reporting tool can link the logged runtime SQL statements with their templates to help the developer trace application problems.
- *Web Application Framework*: The suggested patterns in this research can be collected in a framework that focuses on web applications, or they can be added to existing frameworks such as Expresso [Refer to section 3.8.2 for more information on Expresso framework.]

Appendix A UML Notations

GENERAL-PURPOSE CONCEPTS

Can be used on various diagram types

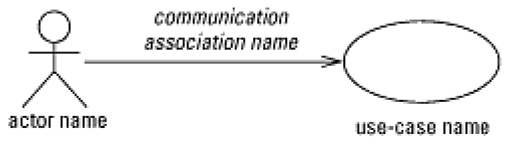
Package, dependency, note



USE-CASE DIAGRAM

Shows the system's use cases and which actors interact with them

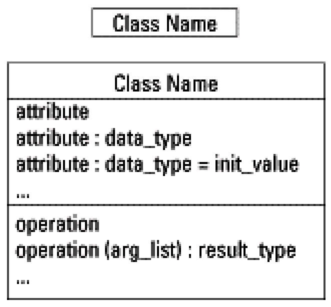
Actor, use case, and association



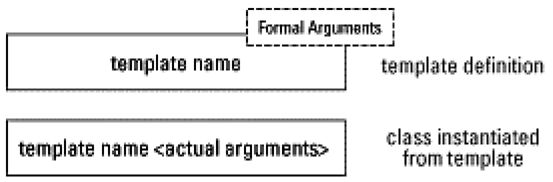
CLASS DIAGRAM

Shows the existence of classes and their relationships in the logical view of a system

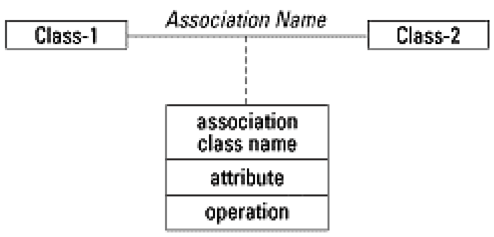
Class


Class Name
Class Name
attribute
attribute : data_type
attribute : data_type = init_value
...
operation
operation (arg_list) : result_type
...

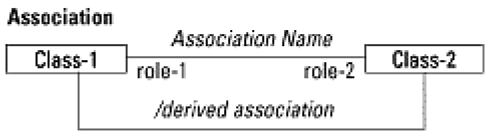
Parameterized class



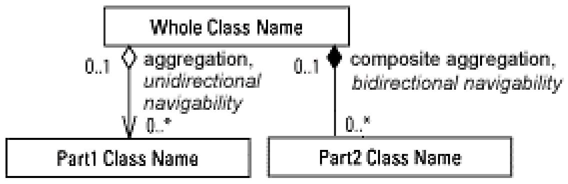
Association classes



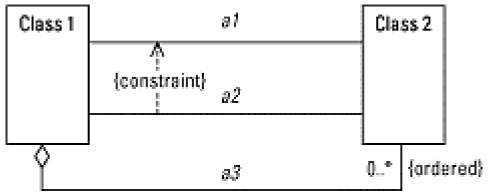
Role names and derived associations



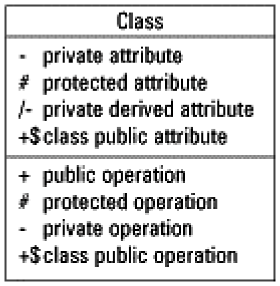
Aggregation, navigability, and multiplicity


Whole Class Name
0..1
aggregation, unidirectional navigability
0..1
composite aggregation, bidirectional navigability
0..*
0..*
Part1 Class Name
Part2 Class Name

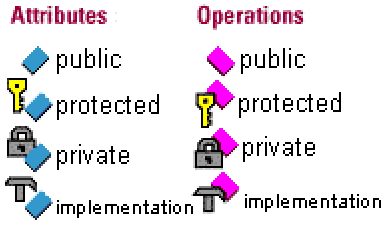
Constraints



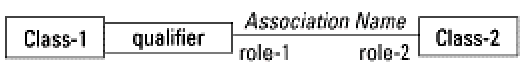
Visibility and properties



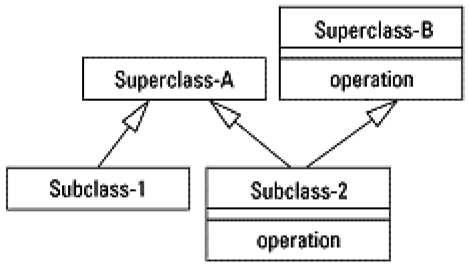
Optional visibility icons



Qualified association



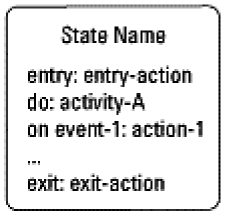
Generalization/specialization



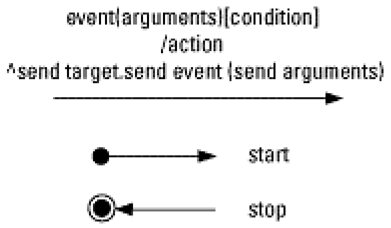
STATE-TRANSITION DIAGRAM

Shows the state space of a given context, the events that cause a transition from one state to another, and the actions that result

State icon

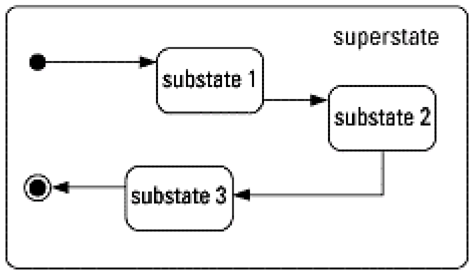


State transitions


event(arguments)[condition]
/action
^send target.send event (send arguments)
start
stop

History Ⓗ

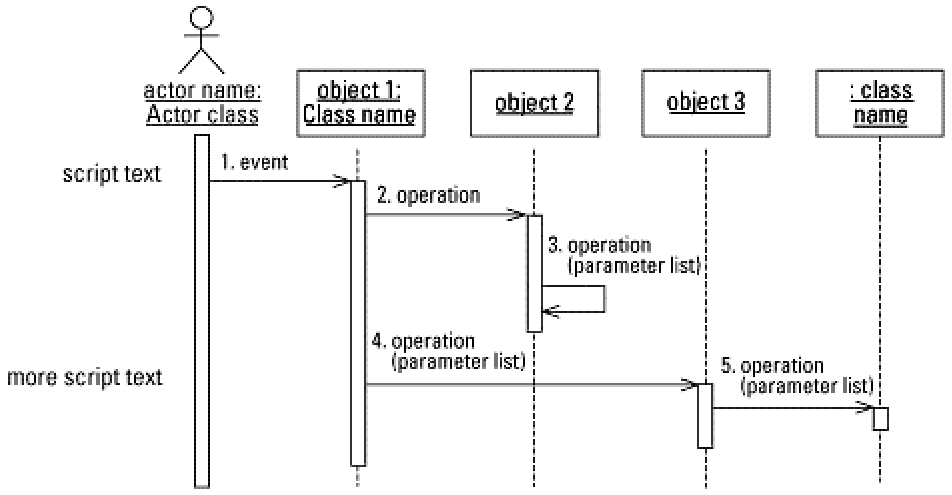
Nesting



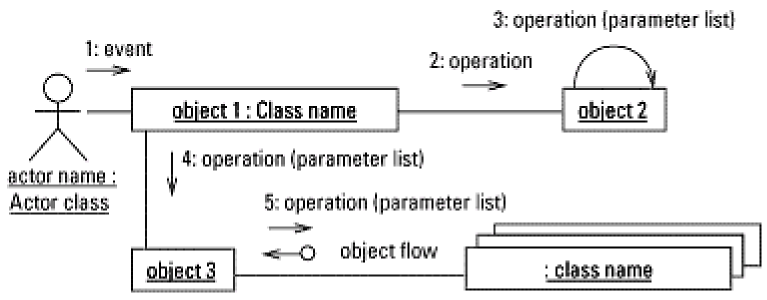
INTERACTION DIAGRAMS

Show objects in the system and how they interact

Sequence diagram

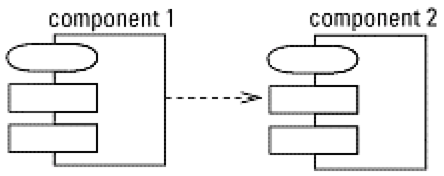


Collaboration diagram



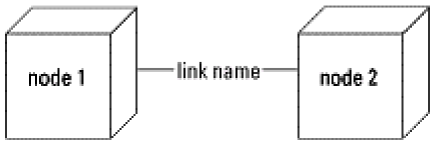
COMPONENT DIAGRAM

Shows the dependencies between software components



DEPLOYMENT DIAGRAM

Shows the configuration of runtime processing elements



Appendix B Tuning the Apache Web Server

The following parameters are the most common ones that make difference when tuned in the apache web server. There are other parameters that are related to the hardware and the operating system which can add more to the over all performance. However, they are not considered in this section. The following are configuration parameters that are found in **httpd.conf** of the Apache Web Server [GD].

- **KeepAlive** - The default for this setting is "on." Make sure "on" is selected if it is not already. There are two good reasons for using this setting:
  - The other "number of servers" settings do not work when **KeepAlive** is set to "off" -- they are no longer optimizations and will probably slow your execution.
  - Building the connection with each mouse click is time-consuming. This option alleviates that pressure from your server, and the server assumes you'll ask for more data.
- **TypeOfServer** - If you are performance-tuning your server, make sure it is set to "Standalone." This is the current default, but double-check just to be sure. When running in inetd mode, Apache uses **inetd** (or **xinetd**) to make its connections, which is much slower.
- **KeepAliveTimeOut** - The values used here are in seconds. Set the value so it does not keep connections lingering too long after the user has disconnected, but long enough that connections in use don't have to be rebuilt frequently. If all your customers are on 1200-baud dial-up, the default of 15 seconds is probably too low, but if you're serving on an intranet, it might be too high. As long as this connection is "live," one of your connections is being used and is not available. When visitors click away from your site, you'll be holding on to a dead connection for as long as this value is set, so don't turn it up arbitrarily.
- **MaxKeepAliveRequests** - This setting controls how many requests a single connection will stay alive for. In a situation where your server is overtaxed, reducing this number will cause more users to have slower access. The default is high -- 100 or 300, depending upon the operating system (or distribution) you're

running. It should be high. Use the **KeepAliveTimeOut** value to clean up most connections, and give it a high number so that any long-running resource issues are cleaned up once in a while. If you don't want your connections to time out, set this value to "0." However, be aware that your server will be kept alive to listen to the connections.

- **StartServers** - This is the number of child processes to start up when Apache is run. This number can be a problem only in heavily loaded servers. Since there are other settings to control the maximum number of servers when the system is stressed, the **StartServers** setting only applies at start-up. If you restart your Web server a lot, it is a good idea to set this number reasonably high, so that your users can get back in quickly. If you do not reboot often, the default (normally 5 for a server installation) should be enough.

- **MaxSpareServers** - This is the number of servers that are hanging around, waiting for a connection. If you are well tuned, you'll have a few of these, but not many. The servers are there to handle spikes in service, but they're not wasting your resources when you don't need them. Leave this setting at the default unless your site is "busy" (for example, you get suddenly swamped at certain times of the day). Don't set it too low, because starting servers up takes quite a chunk of CPU time.

- **MinSpareServers** - This value indicates when Apache should run more copies of child servers to hang about waiting for connections. The default is 5 on most server installations. This means when you only have 4 idle servers, Apache should start a new one. Playing with this value can have wild results, depending upon the CPU speed, load, and memory of your server. If you use settings you're comfortable with, and your performance degrades, use "top" or an equivalent program to look at CPU and memory usage. You may need to upgrade your hardware.

- **MaxClients** - So you changed all the previous settings, and nothing happened? Well, this setting is the "overlord" of child processes. It controls the maximum number of children that can be running at any one time. Do not drop this number too low because the server will reject connections when this number of connections is reached. The default is 150. Consider raising it if you have problems with connection rejections. According to the httpd.conf documentation, this setting's purpose is to keep the Web server from dragging the entire system down if the server goes wild. So make sure this value is not so high that you cannot log in if it is ever reached. Keep the default unless you know that "connection denied" is something your customers are seeing occasionally.

- **MaxRequestsPerChild** - This setting determines the number of separate requests the server handles before it becomes obsolete. It should be set to a reasonably high number. Like **MaxKeepAliveRequests**, this setting is useful because it keeps your system from running out of resources if there is a problem with the server or a shared object the server uses. Starting with the default of 150 is a good idea, but the higher this number, the fewer times Apache will kill a child process than create a new one to replace it.

- **HostnameLookups –** assigning this parameter to on adds latency to the web server where every request has to be looked up in the DNS Server. Note that it's possible to scope the directives, such as within a <Location /server-status> section. In this case the DNS lookups are only performed on requests matching the criteria. Here's an example which disables lookups except for .html and .cgi files:

```
HostnameLookups off
<Files ~ "\.(html|cgi)$">
    HostnameLookups on
</Files>
```

- **DirectoryIndex –** If possible avoid content negotiation where wildcard file name is used such as:

```
DirectoryIndex index
```

Try to use a complete list of options:

```
DirectoryIndex index.cgi index.pl index.shtml index.html
```

Appendix C Java Naming Convention

Naming conventions make the code more readable and understandable. Programmers can know from the first look if a certain name is a package name or a constant. Code-naming convention is important for many reasons [JAVACONV]: -

1. It enhances code readability and understandability.
2. 80% of the software lifetime cost goes to maintenance.
3. The original software developer is rarely the one that maintenances the software.
4. If the software source code is shipped, then it must be guaranteed that the source code is as good as any other product.

The following table is an excerpt from the Java documentation that describes briefly the recommended naming conventions. For more information about Java Code Convections visit http://java.sun.com/docs/codeconv/index.html.

Table 15 Java Naming Convention Summaries

Identifier Type	**Rules for Naming**	**Examples**
Packages	The prefix of a unique package name is always written in all-lowercase ASCII letters and should be one of the top-level domain names, currently com, edu, gov, mil, net, org, or one of the English two-letter codes identifying countries as specified in ISO Standard 3166, 1981. Subsequent components of the package name vary according to an organization's own internal naming conventions. Such conventions might specify that certain directory name components be division, department, project, machine, or login names.	com.sun.eng com.apple.quicktime.v2 edu.cmu.cs.bovik.cheese

Classes	Class names should be nouns, in mixed case with the first letter of each internal word capitalized. Try to keep your class names simple and descriptive. Use whole words-avoid acronyms and abbreviations (unless the abbreviation is much more widely used than the long form, such as URL or HTML).	class Raster;
		class ImageSprite;

Interfaces	Interface names should be capitalized like class names.	interface RasterDelegate;
		interface Storing;

Methods	Methods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized.	run();
		runFast();
		getBackground();

Variables	Except for variables, all instance, class, and class constants are in mixed case with a lowercase first letter. Internal words start with capital letters. Variable names should not start with underscore _ or dollar sign $ characters, even though both are allowed.
	Variable names should be short yet meaningful. The choice of a variable name should be mnemonic- that is, designed to indicate to the casual observer the intent of its use. One-character variable names should be avoided except for temporary "throwaway" variables. Common names for temporary variables are i, j, k, m, and n for integers; c, d, and e for characters.
		int i;
		char c;
		float myWidth;

Constants	The names of variables declared class constants and of ANSI constants should be all uppercase with words separated by underscores ("_"). (ANSI constants should be avoided, for ease of debugging.)	static final int MIN_WIDTH = 4;
		static final int MAX_WIDTH = 999;
		static final int GET_THE_CPU = 1;

Appendix D The Test Environment

The power of the pattern language cannot be shown unless all its patterns are integrated and collectively tested. The testing environment simulates the real one. Moreover, it explores the tradeoffs of the pattern language on similar environments. The pros and cons of the pattern language will appear clearly after checking the cooperation of the different design patterns. This can be seen as an integration testing although there may be no clear division of the integrated modules.

A commercial environment is suggested here as the test bench. Vodafone Egypt website www.vodafone.com.eg is selected to be the test bench. Vodafone Egypt website satisfies the required tools and it has a very good design on the level of the application or the database. Selecting an existing application as the test bench makes the testing results related mostly to the design patterns without focusing on the testing environment problems. The purpose is not to test the Vodafone website, but to show the benefits of the suggested pattern language for a commercial web application. So, testing will ignore any business logic and will mainly concentrate on the system before and after applying the pattern language.

The Following are the technical specification for the environment that the research development and testing will be conducted on:

- **Development Environment**
  2. **Visual Age for Java 3.5.3**: the development kit that is used. It has a robust testing environment for web applications. It uses the rapid application development (RAD) to build Java applications, servlets, applets, and java beans [TOHF01].
  3. **WebSphere Studio 3.5.2**: It is used to develop HTML and JSP pages.
  4. **UML**: It is used to depict the design pattern notations. The Unified Modeling Language (UML) is the standard language nowadays for designing, visualizing, documenting software. This notation is flexible enough to design even a database which helps in bridging the gap between the developers and the users [UMLCENTER]. Almost all the design patterns, if not all of them, will have blue prints of UML to visualize them. Appendix A has more information on the UML notations.
  5. A Java-based load testing utility is used. A Commercial load testing software was not possible.

6. **JDK (Java Development Kit) 1.2.2**: It is used as the implementation language. Java is currently very widespread in the internet community for its network flexibility and its innovative solutions like applets [JAVA1.2].
7. **JSP 1.1**: It is used to develop the dynamic pages for the web application. Java Server Pages (JSP) are like HTML pages, but they provide dynamic content inside the HTML. It is very useful to have this separation since the developers will no longer need to know how to build HTML and leave it to the graphics designers. All that they need to know is where to put their dynamic content [WUFM].

- **Deployment Environment**
  1. Oracle Database Engine/Management
  2. **WebSphere Application Server Advanced Edition 3.5.6**: This is the application server that hosts the web application. It is a java based application that supports servlets, JSP, XML, and EJBs technology.
  3. **IBM HTTP Server**: It is a web server that is based on Apache. Apache is the most widespread web server in the world. It is bundled with WAS (WebSphere Application Server).
  4. Windows 2000 is used as the platform for the client connections.
  5. IE6 is used as the testing browser.
  6. A Unix machine with dual processors and 2GB memory running on Solaris 2.6 is used to host WebSphere Application Server 3.5.6.

Appendix E Load Testing Utility Source Code

This testing utility is built fully in Java. It consists mainly of two classes and one property file where configurations are saved. Tester.java is responsible for reading the properties file and loading its configurations, creating virtual users, and generating the report. URLRun.java is a virtual user that keeps calling the pages specified in the properties file and accumulate statistics about its usage. Tester.properties is used to specify the load testing duration in minutes, the number of users to run, and the pages to be called. Please refer to Appendix C for more information on Java naming conventions. The command used to run the load testing utility is

```
java -cp .;<jar file path> Tester
```

Sample Report

```
Users ran concurrently for at least 1 minutes
Number of users = 1
Start Time = Sat Nov 01 15:24:15 EET 2003
End Time = Sat Nov 01 15:25:15 EET 2003
Number of test pages = 2
------------------------------------------
Details on every page
---------------------
page (http://10.230.91.29/JSP/personal/promotions/promotionTypeDetails_14.html)
--------------------------------
Total number of requests = 455
Failed requests = 0
Total successful response/download time = 13.99 seconds.
Total successful latency time = 7.931 seconds.
Average response Time (total success response time / (No. of requests - No. of request
errors)) = 0.03074725274725275 seconds
Average latency Time (total success latency time / (No. of requests - No. of request
errors)) = 0.017430769230769232 seconds

page (http://10.230.91.29/JSP/personal/promotions/promotions.jsp)
--------------------------------
Total number of requests = 454
Failed requests = 0
Total successful response/download time = 45.939 seconds.
Total successful latency time = 22.909 seconds.
Average response Time (total success response time / (No. of requests - No. of request
errors)) = 0.10118722466960352 seconds
Average latency Time (total success latency time / (No. of requests - No. of request
errors)) = 0.05046035242290749 seconds

------------------------------------------------------------------------------
total requests = 909
total failed requests = 0
Failed ration % = 0.0
total request time = 59.929 sc
Average latency time per page (total response time/(total requests - total failed
requests)) = 0.03392739273927393 sc
Average response/download time per page (total request time/ (total requests - total
failed requests)) = 0.06592849284928493 sc
```

Tester.properties

```
# number of users to go on with the test
```

```
usersCount=1
# the pages that will go in the scenario
page1=http://<path>/JSP/personal/promotions/promotions.jsp
page2=http://<path>/JSP/personal/promotions/promotionTypeDetails.jsp?Pro
mTypeID=14
# duration of the test in minutes
duration=1
```

Tester.java

```
import java.net.*;
import java.util.*;
import java.io.*;
/**
 * The test manager.  It is responsible for creating the virtual users.  It loads also the
pages
 * that will be called by the user.
 * @author: Osama Mabrouk
 */
public class Tester extends Thread {
    public static Boolean finished = new Boolean(true);
    ResourceBundle rs = null;
    /** The number of users that will run the test */
    private int usersCount = -1;
    /** sequences of pages to test */
    private String[] pages = null;
    /** duration of testing in seconds */
    private int duration = -1;
    java.util.Date startTime = null;
    java.util.Date endTime = null;

    public Tester() {
        super();
        try {
            String time = new Date().toString();
            time = time.replace(':', '_');
            File f = new File("report_" + time + ".txt");
            f.createNewFile();
            PrintStream out = new PrintStream(new FileOutputStream(f));
            System.setOut(out);
        } catch (FileNotFoundException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }

        rs = ResourceBundle.getBundle("Tester");
        /** The number of users that will run the test */
        usersCount = Integer.parseInt(rs.getString("usersCount"));
        /** sequences of pages to test */
        Enumeration keys = rs.getKeys();
        String key = null;
        Vector v = new Vector();
        while (keys.hasMoreElements()) {
            key = (String) keys.nextElement();
            if (key.indexOf("page") != -1)
                v.add(rs.getString(key));
        }
        pages = new String[v.size()];
        for (int i = 0; i < v.size(); i++) {
            pages[i] = (String) v.get(i);
        }
        /** duration of testing in seconds */
        duration = Integer.parseInt(rs.getString("duration"));
    }

    public int getDuration() {
        return duration * 1000 * 60;
    }

    public java.lang.String[] getPages() {
        return pages;
    }

    public int getUsersCount() {
        return usersCount;
    }

    public static void main(String[] args) {

        Tester tester = new Tester();
        tester.run();
    }

    private void report(URLRun[] user) {
        double[] singlePageAvgTime = new double[pages.length];
        double totalRequestTime = 0;
        double[] singlePageResponseTime = new double[pages.length];
        double totalResponseTime = 0;
        int[] singlePageNumOfLoadTrials = new int[pages.length];
        int totalRequests = 0;
        int[] singlePageNumOfErrors = new int[pages.length];
        int totalErrors = 0;
        double pageAvgTime = 0;
        for (int i = 0; i < user.length; i++) {
            for (int j = 0; j < singlePageAvgTime.length; j++) {
                singlePageAvgTime[j] += user[i].getPageAvgLoadTime()[j];
                singlePageNumOfLoadTrials[j] += user[i].getNumberOfLoads()[j];
                singlePageNumOfErrors[j] += user[i].getPageError()[j];
                singlePageResponseTime[j] += user[i].getPageResponseTime()[j];
            }

        }
        System.out.println("Users ran concurrently for at least " + getDuration() / (1000
* 60) + " minutes");
        System.out.println("Number of users = " + getUsersCount());
        System.out.println("Start Time = " + startTime);
        System.out.println("End Time = " + endTime);
        System.out.println("Number of test pages = " + pages.length);
        System.out.println("----------------------------------------");
        System.out.println("Details on every page");
        System.out.println("--------------------");

        for (int i = 0; i < pages.length; i++) {
            System.out.println("page (" + pages[i] + ")");
            System.out.println("--------------------------------");
            System.out.println("Total number of requests = " +
singlePageNumOfLoadTrials[i]);
            System.out.println("Failed requests = " + singlePageNumOfErrors[i]);
            System.out.println("Total successful response/download time = " +
singlePageAvgTime[i] / 1000 + " seconds.");
            System.out.println("Total successful latency time = " +
singlePageResponseTime[i] / 1000 + " seconds.");
            double avgTime = (singlePageAvgTime[i] / 1000) / (singlePageNumOfLoadTrials[i]
- singlePageNumOfErrors[i]);
            double avgLatency = (singlePageResponseTime[i] / 1000) /
(singlePageNumOfLoadTrials[i] - singlePageNumOfErrors[i]);
            System.out.println(
                "Average response Time (total success response time / (No. of requests -
No. of request errors)) = "
                    + avgTime
                    + " seconds");
            System.out.println(
                "Average latency Time (total success latency time / (No. of requests - No.
of request errors)) = "
                    + avgLatency
                    + " seconds");
            System.out.println();
        }

        for (int i = 0; i < pages.length; i++) {
            totalRequestTime += (singlePageAvgTime[i] / 1000);
            totalResponseTime += (singlePageResponseTime[i] / 1000);
            totalRequests += singlePageNumOfLoadTrials[i];
            totalErrors += singlePageNumOfErrors[i];

        }
        System.out.println("-------------------------------------------------------------
----------------");
        System.out.println("total requests = " + totalRequests);
        System.out.println("total failed requests = " + totalErrors);
        System.out.println("Failed ration % = " + ((float) totalErrors / (float)
totalRequests) * 100);

        System.out.println("total request time = " + totalRequestTime + " sc");
        System.out.println(
            "Average latency time per page (total response time/(total requests - total
failed requests)) = "
                + totalResponseTime / (totalRequests - totalErrors)
                + " sc");
        System.out.println(
            "Average response/download time per page (total request time/ (total requests
- total failed requests)) = "
                + totalRequestTime / (totalRequests - totalErrors)
                + " sc");
    }
    public void run() {
        URLRun[] user = new URLRun[getUsersCount()];

        try {
            for (int i = 0; i < getUsersCount(); i++) {
                user[i] = new URLRun(getPages());
                user[i].setDaemon(true);
            }
            startTime = new java.util.Date();
            System.err.println(startTime);
            for (int i = 0; i < getUsersCount(); i++) {
                user[i].start();
            }
            System.err.println("Total users " + new java.util.Date() + " " +
URLRun.getCurrentRunsCount());
            Thread.currentThread().sleep(getDuration());
            URLRun.setRunning(false);
            // sleep for one minute until all threads are killed
            while (URLRun.getCurrentRunsCount() > 0) {
                try {
                    Thread.currentThread().sleep(500);
                    System.err.println("Total users =" + URLRun.getCurrentRunsCount());
                } catch (InterruptedException e) {
                }
            }

            endTime = new java.util.Date();

        } catch (InterruptedException e) {
        }
        report(user);
    }

    public void setDuration(int newDuration) {
        duration = newDuration;
    }

    public void setPages(java.lang.String[] newPages) {
        pages = newPages;
    }

    public void setUsersCount(int newUsersCount) {
        usersCount = newUsersCount;
    }
}
```

URLRun.java

```
import java.io.*;
import java.net.*;
import java.util.*;
import com.clickgsm.internet.web.common.*;
/**
 * A virtual User
 * @author: Osama Mabrouk
 */
public class URLRun extends Thread {
    private String[] pages = null;
    private double[] pageAvgLoadTime = null;
    private double[] pageResponseTime = null;
    private int[] pageError = null;
    public static Boolean running = new Boolean(true);
    private int[] numberOfLoads = null;
    static public int currentRunsCount = 0;

    public URLRun() {
        super();
    }

    public URLRun(String[] pages) {
        super();
        this.pages = pages;
        pageAvgLoadTime = new double[pages.length];
        pageError = new int[pages.length];
        numberOfLoads = new int[pages.length];
        pageResponseTime = new double[pages.length];
        setCurrentRunsCount(getCurrentRunsCount() + 1);

    }

    public URLRun(Runnable target) {
        super(target);
    }

    public URLRun(Runnable target, String name) {
        super(target, name);
    }

    public URLRun(String name) {
        super(name);
    }

    public URLRun(ThreadGroup group, Runnable target) {
        super(group, target);
    }

    public URLRun(ThreadGroup group, Runnable target, String name) {
        super(group, target, name);
    }

    public URLRun(ThreadGroup group, String name) {
        super(group, name);
    }

    synchronized public static int getCurrentRunsCount() {
        return currentRunsCount;
    }

    public int[] getNumberOfLoads() {
        return numberOfLoads;
    }

    public double[] getPageAvgLoadTime() {
        return pageAvgLoadTime;
    }

    public int[] getPageError() {
        return pageError;
    }

    public double[] getPageResponseTime() {
        return pageResponseTime;
    }

    public static boolean isRunning() {
        return running.booleanValue();
    }
    public void run() {
        URL url = null;
        long startUrlTime = 0;
        long endResponseTime = 0;
        long endUrlTime = 0;
        InputStream input = null;
        byte[] bytes = null;
        // continue loading as long as running is true
        while (isRunning()) {

            // go over the pages one by one
            for (int i = 0; i < pages.length && isRunning(); i++) {
                try {
                    startUrlTime = System.currentTimeMillis();
                    url = new java.net.URL(pages[i]);
                    input = url.openConnection().getInputStream();
                    endResponseTime = System.currentTimeMillis();
                    bytes = new byte[1000];

                    while (input.read(bytes) != -1) {

                    }
                    input.close();
                    endUrlTime = System.currentTimeMillis();
                    pageResponseTime[i] = pageResponseTime[i] + (endResponseTime -
startUrlTime);
                    pageAvgLoadTime[i] = pageAvgLoadTime[i] + (endUrlTime - startUrlTime);
                } catch (java.net.MalformedURLException e) {
                    pageError[i]++;
                    System.err.println(url.toString() + " " + e);
                } catch (IOException e) {
                    pageError[i]++;
                    System.err.println(url.toString() + " " + e);

                } catch (Exception e) {
                    pageError[i]++;
                    System.err.println(url.toString() + " " + e);
                } catch (Throwable e) {
                    pageError[i]++;
                    System.err.println(url.toString() + " " + e);
                } finally {
                    numberOfLoads[i]++;
                    yield();
                }
            }
        }
        setCurrentRunsCount(getCurrentRunsCount() - 1);
    }

    synchronized public static void setCurrentRunsCount(int newCurrentRunsCount) {
        currentRunsCount = newCurrentRunsCount;
    }

    public void setNumberOfLoads(int[] newNumberOfLoads) {
        numberOfLoads = newNumberOfLoads;
    }
    public void setPageAvgLoadTime(double[] newPageAvgLoadTime) {
        pageAvgLoadTime = newPageAvgLoadTime;
    }

    public void setPageError(int[] newPageError) {
        pageError = newPageError;
    }

    public void setPageResponseTime(double[] newPageResponseTime) {
        pageResponseTime = newPageResponseTime;
    }

    public static void setRunning(boolean newRunning) {
        running = new Boolean(newRunning);
    }
}
```

Appendix F Detailed Load Testing Results

The following are the settings that have been used in the load testing scenarios

- Load Test Duration = 1 minute
- Number of Pages = 2
- IBM HTTP Server Maximum Clients = 150
- IBM WebSphere Application Server Max Connections=25
- Database Connection Pool startup Size = 5 and maximum size=30
- Cache is considered infinite.

The following are the different scenarios that have been tried

- S1 = No connection pool and no Caching
- S2 = Connection Pool and No Caching
- S3 = Connection Pool and Caching
- S4 = Static Pages

The number of the virtual users which have been tried are 1, 10, 30, 60, 100 virtual users. The following tables list all the results for all the runs across the previous four scenarios

Page (1 User)	**Throughput**				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
	S1	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*
List All Vodafone Promotion Types	7	53	221	454	0.02	0.07	0.05	0.017	3.6	0.5	0.13	0.03	0	0	0	0
List A promotion Type Details	7	53	222	455	0.02	0.01	0.066	0.05	5	0.64	0.14	0.1	0	0	0	0
Over All Avg.	7	53	222	455	0.02	0.04	0.058	0.034	4.3	0.57	0.14	0.07	0	0	0	0

Table 16 1 User (Load Testing)

Page (10 User)	**Throughput**				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
	S1	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*
List All Vodafone Promotion Types	9	367	898	1692	0.013	0.05	0.05	0.03	45	0.8	0.3	0.3	0	0	0	0
List A promotion Type Details	11	371	903	1695	0.74	0.04	0.04	0.04	26.6	0.8	0.3	0.06	0	0	0	0
Over All Avg.	10	369	901	1694	0.377	0.05	0.045	0.035	35.8	0.8	0.3	0.18	0	0	0	0

Table 17 10 Users (Load Testing)

Page (30 User)	**Throughput**				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
	S1	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*
List All Vodafone Promotion Types		370	920	1759		0.5	0.18	0.084		2.4	0.94	0.9	0	0	0	0
List A promotion Type Details	30	383	939	1763	17.1	0.5	0.2	0.098	116.2	2.4	1	0.12	0	0	0	0

Over All Avg.	30	377	930	1761	17.1	0.5	0.19	0.091	116.2	2.4	0.97	0.51	0	0	0	0

Table 18 30 Users (Load Testing)

Page (60 User)	**Throughput**				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
	S1	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*
List All Vodafone Promotion Types		402	935	1778		2.68	1.157	0.6		4.5	1.92	1.42	0	0	0	0.007
List A promotion Type Details	60	433	972	1799	89.5	2.61	1.13	0.6	196.6	4.39	1.9	0.6	0	0	0	0.0095
Over All Avg.	60	418	954	1789	89.5	2.65	1.144	0.6	196.6	4.45	1.91	1.01	0	0	0	0.00825

Table 19 60 Users (Load Testing)

Page (100 User)	**Throughput**				**Avg. Latency (sec)**				**Avg. Response Time (sec)**				**Reliability**			
	S1	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*	*S1*	*S2*	*S3*	*S4*
List All Vodafone Promotion Types		400	922	1583		5.8	2.5	1.47		7.8	3.3	2.4	0	0	0	0.011
List A promotion Type Details	100	460	972	1619	170	5.4	2.4	1.4	276.7	7.24	3.2	1.5	0	0	0	0.006
Over All Avg.	100	430	947	1601	170	5.6	2.45	1.435	276.7	7.52	3.25	1.95	0	0	0	0.0085

Table 20 100 Users (Load Testing)

Time	**S1**	**S2**	**S3**	**S4**
0	0.1	2.6	0.1	0.1
2	0.1	4.2	3.7	4.7
4	0.1	12	11.6	13.5
6	0.1	20	18.5	19.5
8	0.6	27	24	25.4
10	0.8	33	29.3	30.9
12	1.2	38	34	35
14	1.3	42	38	38.5
16	1.5	45	41.6	41.9
18	1.6	48	44.7	43.9
20	1.6	51	46.7	44.6
22	2	53	48.5	46.9
24	2.2	55	49.9	47.7
26	2.3	56	50.4	49.8
28	3	56	51.6	51.3
30	3	57	52.8	52.4
32	3.1	58	53.8	53.1
34	3.2	59	54.7	52.6
36	3.3	60	55.6	52.8
38	3.3	60	56.1	52.9
40	3.2	61	56.7	52.6
42	3.2	61	48	48.4
44	3	56	40.4	40.7
46	3	47	33.4	33.9

Table 21 CPU Utilization for load testing (100 users)

Time	S1	S2	S3	S4
0	0	34	0	0
2	0	2	0	0
4	0	2	0	0
6	0	4	0	0
8	0	14	0	0
10	0	1	0	0
12	0	3	0	0
14	2	9	0	0
16	0	4	0	0
18	0	4	0	0
20	0	1	0	0
22	0	1	0	0
24	0	5	0	1
26	0	16	0	0
28	0	4	0	0
30	1	2	0	0
32	0	3	0	1
34	2	3	0	0
36	0	3	0	2
38	0	2	0	10
40	0	8	0	17
42	0	2	0	0
44	0	32	0	N/A
46	0	37	0	N/A
48	0	19	0	N/A
50	0	0	0	N/A
52	0	0	0	N/A
54	1	1	0	N/A
56	1	0	0	N/A
58	0	2	0	N/A
60	0	0	0	N/A
62	0	34	0	N/A
64	0	35	0	N/A
66	0	43	0	N/A
68	0	22	0	N/A
70	0	13	0	N/A
72	2	30	0	N/A
74	0	35	0	N/A
76	0	N/A	0	N/A

Table 22 I/O Waiting for load testing (100 users)

BIBLIOGRAPHY

1. [**AB00**] Brad Appleton. *Patterns and Software: Essential Concepts and Terminology* (2000). http://www.enteract.com/~bradapp/docs/patterns-intro.html.
2. [**BB02**] Bradley D. Brown (2002). *HIGH PERFORMANCE WEB SITES WITH WEB CACHE*. Oracle Development Tools. www.odtug.com.
3. [**BDCR02**] Diego Bonura, Rosario Culmone, and Emanuela Merelli (2002). *Patterns for Web applications.* ACM Press.
4. [**BG94**] Grady Booch (1994). *Object-Oriented Analysis and Design with Applications, Second Edition.*
5. [**BH99**] Hans Bergsten (1999). *Improved Performance with a Connection Pool*. http://www.webdevelopersjournal.com/columns/connection_pool.html
6. [**BJAR02**] James M. Bieman, Roger Alexander, P. Willard Munger III, Rin Meunier (2002). *Software Design Quality: Style and Substance. ACM*
7. [**BKCW87**] Kent Beck, Ward Cunningham. *Using Pattern Languages for Object-Oriented Programs* (1987). OOPSLA-87. http://c2.com/doc/oopsla87.html
8. [**BWMR98**] William Brown, Raphael Malveau, Hays McCormick, Thomas Mowbray, and Scott W. Thomas (1998). Anti Patterns Refactoring Software Architectures, and Projects in Crisis. http://www.antipatterns.com/briefing/index.htm
9. [**CA79**] Alexander, C. (1979). *The Timeless Way of Building.* New York: Oxford University Press.
10. [**CAUJ02**] A.Cranmore, J.Ure, R.G.Dewar, A.D.Lloyd, R.J.Pooley (2002). *Capacity Planning for e-Business.* EuroPLoP 2002.
11. [**CC02**] Charles Connell (2002). *Most Software Stinks!*. http://www.chc-3.com/pub/beautifulsoftware.htm
12. [**CJCB00**] JAMES E. CAREY, BRENT A. CARLSON (2000). *Deferring Design Decisions in an Application Framework.* ACM Computing Surveys.
13. [**EESW00**] Ezra Ebner, Weiguang Shao, Wei-Tek Tsai (2000). *The Five-Module Framework for Internet Application Development.* ACM Computing Surveys.
14. [**FM02**] Martin Fowler and others (2002). *Patterns of Enterprise Application Architecture*. http://www.martinfowler.com/eaaCatalog/.
15. [**FP99**] PIERO FRATERNALI. *Tools and Approaches for Developing Data-Intensive Web-Applications: A survey.* ACM Computing Surveys, Vol. 31, No. 3, September 1999.
16. [**GA]** Antonio Garicia. *The Strategy Design Pattern.* http://www.exciton.cs.rice.edu/JavaResources/DesignPatterns/StrategyPattern.htm.
17. [**GD**] Dean Gaudet. *Apache Performance Notes*. http://httpd.apache.org/docs/misc/perf-tuning.html
18. [**GEHR95**] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides (1995). *Design Patterns, Elements of Reusable Object-Oriented Software.*
19. [**GRSA01**] Richard L. Gimarc and Amy C. Spellmann (2001). *Performance Modeling for Web Application Optimization Part I – Modeling the Customer Experience.* HyPerformix Inc.
20. [**HENDER96**] Brian Henderson-Sellers (1996). *Object-Oriented Metrics, Measures of Complexity.* Prentice Hall PTR.

21. [**HPATT01**] *History of Patterns* (2001). http://c2.com/cgi-bin/wiki?HistoryOfPatterns
22. [**HREW00**] R. Hariharan, W.K. Ehrlich, D. Cura, and P.K. Reeser (2000). *End to End Performance Modeling of Web Server Architectures*
23. [**JAVA1.2**] *Java 2 Platform, Standard Edition (J2SE)*. http://java.sun.com/products/jdk/1.2/
24. [**JAVA1.3**] *Java 2 SDK Standard Edition Documentation Version 1.3.* http://java.sun.com/products/jdk/1.3/index.html
25. [**JAVACONV**] *Code Conventions for the JavaTM Programming Language*. http://java.sun.com/docs/codeconv/index.html
26. [**JCORP**] *http://www.jcorporate.com/*
27. [**JP01**] P. Jain (2001). *Evictor Pattern*. http://www.cs.wustl.edu/~pjain/papers/Evictor.pdf.
28. [**JPKM02**] Prashant Jain and Michael Kircher (2002). *Partial Acquisition Pattern*. PLoP 2002. http://www.cs.wustl.edu/%7Emk1/PartialAcquisition.pdf.
29. [**JR97**] Ralph E. Johnson (1997). *Frameworks = (Components + Patterns).* Communications of the ACM.
30. [**KC92**] CHARLES W. KRUEGER (1992). *Software Reuse.* ACM Press.
31. [**KM**] Mika Kujala. *Java in Design Patterns*. Tik-76.270, Research Seminar: Java-based Software Technologies.
32. [**KM01**] Michael Kircher (2001). *Lazy Acquisition Pattern*. EuroPLoP 2001. http://www.cs.wustl.edu/%7Emk1/LazyAcquisition.pdf.
33. [**KM02**] Michael Kircher (2002). *Eager Acquisition Pattern*. EuroPLoP 2002. http://www.cs.wustl.edu/%7Emk1/EagerAcquisition.pdf
34. [**KMJP02**] Michael Kircher, Prashant Jain (2002). *Pooling*. Corporate Technology, Siemens AG. Munich, Germany. http://www.cs.wustl.edu/%7Emk1/Pooling.pdf.
35. [**KMJP03**] Michael Kircher, Prashant Jain (2003). *Caching*. EuroPLoP 2003 http://www.cs.wustl.edu/%7Emk1/Caching.pdf.
36. [**LBCV03**] Barry M. Leiner, Vinton G. Cerf, David D. Clark, Robert E. Kahn, Leonard Kleinrock, Daniel C. Lynch, Jon Postel, Larry G. Roberts, Stephen Wolff (2003). *A brief history of the Internet*. http://www.isoc.org/internet/history/brief.shtml
37. [**LCGR00**] Chris Loosley, Keynote Systems Inc. Richard L. Gimarc and Amy C. Spellmann, HyPerformix Inc (2000). *E-COMMERCE RESPONSE TIME: A REFERENCE MODEL*
38. [**LD93**] Doug Lea (1993). *Christopher Alexander: An Introduction for Object-Oriented Designers*. http://gee.cs.oswego.edu/dl/ca/ca/ca.html
39. [**LKZS00**] K. Liu, S. Zhou and H. Yang (2000). *Quality Metrics of Object Oriented Design for Software Development and Re-development.* IEEE.
40. [**LQNJ99**] Qiong Luo, Jeffrey F. Naughton, Rajasekar Krishnamurthy, Pei Cao, Yunrui Li (1999). *Active Query Caching for Database Web Servers.*
41. [**LT01**] Timothy C. Lethbridge and Robert Laganière (2001)**.** *Object-Oriented Software Engineering: Practical Software Development using UML and Java.* McGraw Hill.
42. [**MBMT01**] Berna Massingil, Timothy Mattson, Beverly Sanders (2001). *A Pattern Language for Parallel Application Programming*. http://www.cise.ufl.edu/research/ParallelPatterns/
43. [**MC94**] Dr. Carma McClure (1994). *Reuse Engineering: Extending Information Engineering to Enable Software Reuse*. Extended Intelligence, Inc. http://www.reusability.com/papers4.html

44. [**MC95**] Dr. Carma McClure (1995). *Getting Started with Software Reuse: Secrets to Reuse Success Revealed*. Extended Intelligence, Inc. http://www.reusability.com/papers5.html
45. [**MREW00**] R.D. van der Mei, W.K. Ehrlich, P.K. Reeser, J.P. Francisco (2000). *A Decision Support System for Tuning Web Servers in Distributed Object Oriented Network Architectures*. ACM SIGMETRICS Performance Evaluation Review, volume 27 Issue 4.
46. [**NDWS01**] Dung ("Zung") Nguyen, Stephen B. Wong (2001). *Design Patterns for Sorting*. ACM SIGCSE Bulleti**,** Volume 33 Issue 1.
47. [PAFJ96] Allison L. Powell, James C. French, John C. Knight (1996). *A Systematic Approach to Creating and Maintaining Software.* ACM Press.
48. [**PFDL96**] F. Prefect, L. Doan, S. Gold, Th. Wicki, W. Wilcke (1996). *Performance Limiting Factors in Http Web) Server Operations*. HAL Computer Systems, Inc.
49. [**RDZH96**] Dirk Riehle and Heinz Zullighoven (1996). "Understanding and Using Patterns in Software Development". http://citeseer.nj.nec.com/rd/0%2C14269%2C1%2C0.25%2CDownload/http%253A%252F%252Fwebfuse.cqu.edu.au/Information/Resources/Readings/papers/tapos-96-survey.pdf
50. [**RUP**] *Rational Unified Process*. http://www.rational.com/
51. [**SD96**] Douglas C. Schmidt (1996). *Using Design Patterns to Guide the Development of Reuseable Object-Oriented Software*. ACM Computing Surveys.
52. [**SDC96**] Douglas C. Schmidt (December 1996). *Using Design Patterns to Guide the Development of Reusable Object-Oriented Software.* ACM Computing Surveys.
53. [**SHASHA96**] DENNIS SHASHA (1996). *Tuning Databases for High Performance***.** ACM Computing Surveys*.*
54. [**SI95**] Ian Sommerville (1995). *Software Engineering, Fifth Edition*. Addison-Wesley Publishing Company.
55. [**SUN02**] Sun Microsystems (2002). *Data Access Object (DAO)*. http://java.sun.com/blueprints/patterns/DAO.html.
56. [**TEST01**] University of South Australia. School of Computer and Information Science. *Software Engineering*. http://louisa.levels.unisa.edu.au/se1/testing-notes/test01_1.htm
57. [**TM02**] Moisés Daniel Díaz Toledano (2002). *Meta Patterns: A new Approach for Design Patterns*. http://www.moisesdaniel.com/wri/metapatterns.doc.
58. [**TOHF01**] Osamu Takagiwa, Frederik Haesbrouck, Veronique Quiblier, and Sarah Poger. *Programming with VisualAge for Java Version 3.5.* http://www.redbooks.ibm.com/redbooks/SG245264.html
59. [**UMLCENTER**] *UML Resource Center*. http://www.rational.com/uml/index.jsp?SMSESSION=NO
60. [**USPROB01**] *Usability problems*. http://www.leafdigital.com/class/lessons/usability/3.html
61. [**VFWE96**] Filippos I. Vokolos, Elaine J. Weyuker (1996). *Performance Testing of Software Systems***.** ACM
62. [**VOZU02**] Oliver Vogel and Uwe Zdun (2002). *Content Conversion and Generation on the Web: A Pattern Language.* EuroPLoP.
63. [**WAS02**] IBM (2002). *WebSphere Application Server*. http://www.ibm.com/websphere/
64. [**WUFM**] Ueli Wahli, Mitch Fielding, Gareth Mackown, Deborah Shaddon, and Gert Hekkenberg. *Servlet and JSP Programming with IBM WebSphere Studio and VisualAge for Java*. http://www.redbooks.ibm.com/

INDEX