A NOTION OF A SYSTEM PATTERN (TRI-INTEGRATION) AND ITS INSTANTIATION IN ENTERPRISE APPLICATION DEVELOPMENT

Ph.D.

Hua Zhou

Software Technology Research Laboratory

De Montfort University

2004
BEST COPY AVAILABLE.

VARIABLE PRINT QUALITY
Acknowledgements

I would like to express my gratitude to the many people who helped me in different ways with the development of this thesis.

I wish to express my most profound thanks to my supervisor, Professor Hongji Yang, for his invaluable advice, support and encouragement during my three-year study. He provided me with many useful comments and suggestions for improvement of the thesis.

Meanwhile, I would like to thank colleagues at the Software Technology Research Laboratory at De Montfort University and colleagues at the Information Technology Institute in Yunnan University, for their support and feedback, and for providing such a stimulating and friendly working atmosphere. There are too many to list individually.

I would also like to thank the Research Office in De Montfort University for their outstanding management.

Finally, I must thank my wife, Hongying, my son, Yuxiang and my parents for all their memorable support and encouragement, which are too precious to forget.
Declaration

I declare that the work described in this thesis was originally carried out by me during the period of registration for the degree of Doctor of Philosophy at De Montfort University, U.K., from January 2001 to June 2004. Apart from the degree that this thesis is currently submitted to, no other academic degree or award was applied by me based on this work.
Abstract

How to develop reusable software still needs research. Broadly, there are two approaches to software development: masterplan and piecemeal growth. This thesis investigates how a system pattern can help software development according to a piecemeal growth approach.

Firstly, a new concept for software development, a system pattern, is proposed. It describes a particular recurring development problem that arises in specific application contexts, and presents a generic framework for its solution. The solution framework is specified by describing its constituent components, their responsibilities and relationships, the ways in which they collaborate, the range from architectural level to idiom level and the method for their construction.

Secondly, a unified pattern, Tri-Integration pattern, is defined as a system pattern to build an integrated, secure and expandable network application system. The pattern has three elements: Service, Component and Data. It presents a system-level solution through service integration, component integration and data integration. Service integration represents a mechanism that can integrate various services into one application system; component integration supports the development and deployment of application components using well-designed construction methods; data integration can compose all separate data in a resource repository that may be distributed between different machines.

Thirdly, a software platform based on the Tri-Integration Pattern was developed successfully to accelerate and extend practical software development. Several system tools were integrated with the platform. Through case studies in three typical domains and extensive usage of the software platform, the usefulness and success of the Tri-Integration pattern design and its applications are vindicated.

The proposed system pattern is thus consider to provide valuable assistance in software development, especially in helping support the development of complex, large-scale systems and in supporting effective industrial software production and the integration of application systems.
# Contents

Acknowledgements ................................................................................................................. 1
Declaration ................................................................................................................................. ii
Abstract ................................................................................................................................... iii

Chapter 1  
Introduction .......................................................................................................................... 1

1.1 Proposed Research and Overview of Problem ................................................................. 1
1.2 Scope of the Thesis and Original Contributions .............................................................. 3
1.3 Criteria for Success ............................................................................................................. 5
1.4 Thesis Structure .................................................................................................................. 6

Chapter 2  
Software Development from Architectural Point of View .................................................... 7

2.1 Software Lifecycle .............................................................................................................. 7

2.1.1 The Explosive Rate of Software Development ............................................................. 7
2.1.2 Overview of Existing Models ....................................................................................... 8
2.1.3 Brief Reviews of Methods and Tools ........................................................................... 11

2.1.3.1 Object-Oriented Approach ................................................................................. 12
2.1.3.2 Unified Modelling Language (UML) .................................................................. 12
2.1.3.3 XML Introduction ............................................................................................... 13

2.1.3.4 CASE Tools ......................................................................................................... 13

2.2 Masterplan Approach: Software Architecture ................................................................ 14

2.2.1 Development of Software Architecture ....................................................................... 15
2.2.2 Basic Ingredients of Software Architecture ................................................................. 17
2.2.3 Enabling Techniques for Software Architecture ........................................................... 19

2.2.4 Non-functional Properties of Software Architecture ................................................... 25
2.2.5 Different Research Approaches to Software Architecture ........................................... 30
2.2.6 Architectural Styles ..................................................................................................... 31

2.3 Piecemeal Growth Approach: Software Patterns ............................................................... 32

2.3.1 Pattern Definition ........................................................................................................ 32
2.3.2 Pattern Description ...................................................................................................... 34
2.3.3 Pattern Categories ...................................................................................................... 37

2.3.3.1 Architectural Patterns ....................................................................................... 37
2.3.3.2 Design Patterns ................................................................................................... 38
2.3.3.3 Idioms ................................................................................................................ 38

2.3.4 Relationship between Patterns ................................................................................... 39

2.4 Patterns and Software Architecture .................................................................................. 42

2.4.1 Patterns as Mental Building-Blocks ............................................................................. 43
2.4.2 Patterns versus Methods ............................................................................................. 43
2.4.3 Instantiating Patterns .................................................................................................. 44
2.4.4 Methodologies ............................................................................................................ 44

2.4.5 Software Processes ..................................................................................................... 46

2.5 Summary .......................................................................................................................... 47
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>Integration Approach</td>
<td>50</td>
</tr>
<tr>
<td>3.1</td>
<td>A Taxonomy of Integration Concepts</td>
<td>50</td>
</tr>
<tr>
<td>3.1.1</td>
<td>Integration Concepts</td>
<td>50</td>
</tr>
<tr>
<td>3.1.2</td>
<td>Granularity of Integration</td>
<td>51</td>
</tr>
<tr>
<td>3.2</td>
<td>Conceptual Models of Integration</td>
<td>53</td>
</tr>
<tr>
<td>3.2.1</td>
<td>Integration as a set of Attributes</td>
<td>53</td>
</tr>
<tr>
<td>3.2.2</td>
<td>Integration as a Relationship</td>
<td>55</td>
</tr>
<tr>
<td>3.2.3</td>
<td>Repository based Integration</td>
<td>58</td>
</tr>
<tr>
<td>3.2.4</td>
<td>Multilevel Integration</td>
<td>59</td>
</tr>
<tr>
<td>3.3</td>
<td>Integration Strategies</td>
<td>61</td>
</tr>
<tr>
<td>3.3.1</td>
<td>Strategies for Legacy Systems</td>
<td>61</td>
</tr>
<tr>
<td>3.3.2</td>
<td>Strategies for New Procurements</td>
<td>62</td>
</tr>
<tr>
<td>3.4</td>
<td>Summary</td>
<td>63</td>
</tr>
<tr>
<td>4</td>
<td>Proposed Approach: System Pattern</td>
<td>65</td>
</tr>
<tr>
<td>4.1</td>
<td>Pattern for a System</td>
<td>65</td>
</tr>
<tr>
<td>4.1.1</td>
<td>Three Levels: Architecture, Design and Idiom</td>
<td>65</td>
</tr>
<tr>
<td>4.1.2</td>
<td>System Pattern Definition</td>
<td>67</td>
</tr>
<tr>
<td>4.2</td>
<td>Motivation for One System Pattern</td>
<td>67</td>
</tr>
<tr>
<td>4.2.1</td>
<td>A Motivating Scenario</td>
<td>67</td>
</tr>
<tr>
<td>4.2.2</td>
<td>Recuring Development Problems</td>
<td>69</td>
</tr>
<tr>
<td>4.2.3</td>
<td>Fundamental Solution Principle</td>
<td>72</td>
</tr>
<tr>
<td>4.3</td>
<td>The System Pattern: Tri-Integration Pattern</td>
<td>73</td>
</tr>
<tr>
<td>4.3.1</td>
<td>Definition</td>
<td>73</td>
</tr>
<tr>
<td>4.3.2</td>
<td>Context</td>
<td>74</td>
</tr>
<tr>
<td>4.3.2.1</td>
<td>Service Integration Mechanisms</td>
<td>74</td>
</tr>
<tr>
<td>4.3.2.2</td>
<td>Component Integration Mechanisms</td>
<td>76</td>
</tr>
<tr>
<td>4.3.2.3</td>
<td>Data Integration Mechanisms</td>
<td>81</td>
</tr>
<tr>
<td>4.3.3</td>
<td>Structures</td>
<td>82</td>
</tr>
<tr>
<td>4.3.3.1</td>
<td>Description of Participating Structures</td>
<td>83</td>
</tr>
<tr>
<td>4.3.3.2</td>
<td>The Static Structure of the Pattern</td>
<td>85</td>
</tr>
<tr>
<td>4.3.4</td>
<td>Levels</td>
<td>86</td>
</tr>
<tr>
<td>4.3.5</td>
<td>Dynamic Behaviours</td>
<td>87</td>
</tr>
<tr>
<td>4.4</td>
<td>Summary</td>
<td>92</td>
</tr>
<tr>
<td>5</td>
<td>Tri-Integration Pattern: Architectural Level</td>
<td>93</td>
</tr>
<tr>
<td>5.1</td>
<td>Constructing Architectural Level Elements</td>
<td>93</td>
</tr>
<tr>
<td>5.1.1</td>
<td>Resources Organisation</td>
<td>93</td>
</tr>
<tr>
<td>5.1.2</td>
<td>Proposed Protocol</td>
<td>95</td>
</tr>
<tr>
<td>5.1.3</td>
<td>Core Server Structure</td>
<td>96</td>
</tr>
<tr>
<td>5.1.4</td>
<td>Core Language</td>
<td>100</td>
</tr>
<tr>
<td>5.1.5</td>
<td>Atomic Components Assembly</td>
<td>101</td>
</tr>
</tbody>
</table>
8.4 Case Study 3: An EAI System 
8.4.1 Netcase Solution Framework 
8.5 Summary 

Chapter 9 Conclusions 
9.1 Remarks 
9.1.1 Contrast of Three Software Platforms 
9.1.2 Comparision with Four Network Supported Patterns 
9.2 Evaluation 
9.3 Assessment of Success and Analysis 
9.3.1 The Pattern 
9.3.2 The Platform 
9.4 Conclusions 
9.4.1 Lessons Learnt 
9.4.2 Conclusions 
9.5 Future Directions 

Reference 

Appendix A 
Multiple Resource Transfer Protocol (MRTP) 
A.1 Overview 
A.2 Terminology 
A.3 Overall Operation 
A.4 Protocol Parameters 
A.4.1 DotPath—Resource or Resource Paths 
A.4.2 MsgID—Message Identifier 
A.4.3 FileName—File Name 
A.4.4 Alias—Database Interface Alias 
A.5 Protocol Message 
A.5.1 Request 
A.5.2 Response 
A.5.3 Rights 
A.6 Request Method Command References 

Appendix B 
Key Objects for Tri-Integration Idioms 
B.1 Core Server Base Object: TAnywhereBaseClass 
B.2 Interface Object: TRConnection 
B.3 Syntax of Web Script Language 

Appendix C 
List of Publications by Thesis Author 

List of Figures and Tables

Figure 2.1 Three Static Connectors ............................................................... 20
Figure 2.2 Structure of Proxy ................................................................... 43
Figure 3.1 A Hierarchy of Integration ............................................................ 52
Figure 3.2 Integration and Supporting Notions .............................................. 53
Figure 3.3 Three Integration Dimensions .................................................... 56
Figure 3.4 Integration Relationships between Environment Components ........ 57
Figure 3.5 ER Diagram of a Single Component, Four Relationships and Their Properties ................................. 58
Figure 3.6 A Multilevel Integration Model .................................................. 61
Figure 4.1 The System Pattern Range ........................................................... 68
Figure 4.2 Service Integration through Message Passing .............................. 76
Figure 4.3 Point-to-point Integration ............................................................. 79
Figure 4.4 Integration through Standards ..................................................... 80
Figure 4.5 Static Structure of the Tri-Integration Pattern ............................... 88
Figure 4.6 Resources Operation Scenario .................................................... 90
Figure 4.7 Workflow Scenario .................................................................... 92
Figure 4.8 Application Request Scenario .................................................... 93
Figure 6.1 Application Framework of Tri-Integration Pattern ......................... 110
Figure 6.2 Authorisation Strategies .............................................................. 115
Figure 6.3 Logical Structure of Core Server ............................................... 119
Figure 6.4 Workflow Implementation Logical View ..................................... 121
Figure 7.1 Resource Manager .................................................................... 125
Figure 7.2 Resource Manager Logical View ............................................... 125
Figure 7.3 Main Classes Logical View of Server-side Virtual Machine .......... 131
Figure 7.4 Client Virtual Machine Logical View .......................................... 136
Figure 7.5 Alias Manager ......................................................................... 138
Figure 7.6 Alias Manager Logical View ...................................................... 139
Figure 7.7 Web Page Designer ................................................................... 140
Figure 7.8 Web Pages Designer Logical View .............................................. 141
Figure 7.9 Form Designer ......................................................................... 143
Figure 7.10 Form Designer Logical View .................................................... 144
Figure 7.11 Flow Designer ........................................................................ 145
Figure 7.12 Flow Designer Logical View .................................................... 146
Figure 7.13 Sheet Designer ........................................................................ 147
Figure 7.14 Sheet Designer Logical View .................................................... 148
Figure 7.15 Client Explorer ....................................................................... 151
Figure 7.16 Client Explorer Logical View ................................................... 152
Figure 8.1 Netcase Workflow Model ............................................................ 157
Figure 8.2 Workflow Model Level ............................................................... 159
Figure 8.3 Workflow Process System Implementation View ....................... 160
Figure 8.4 E-Government/E-Commerce Model .......................................... 163
Figure 8.5 Netcase E-Government Architecture ........................................ 163
Figure 8.6 Netcase Solutions Framework ................................................... 164
Figure 8.7 Netcase Framework for Integration Adapter Pattern .................................................. 167
Figure 8.8 Netcase Framework for Integration Messenger Pattern ............................................. 167
Figure 8.9 Netcase Framework for Integration Facade Pattern .................................................... 168
Figure 8.10 Netcase Framework for the Integration Mediator Pattern ........................................... 168
Figure 8.11 Netcase Framework for the Process Automator ......................................................... 169
Table 2.1 Comparison of Two Different Research Approaches to Software Architecture ............... 32
Table 2.2 Pattern Description Template ...................................................................................... 37
Table 4.1 Identified Low Level Integration Standards .................................................................. 81
Table 4.2 The Tri-Integration Pattern Structures ......................................................................... 84
Table 4.3 The Tri-Integration Pattern Levels .............................................................................. 88
Table 6.1 System Information of Resource Description ............................................................... 111
Table 6.2 Description Information of Functionality Resource .................................................... 111
Table 6.3 Description of Authority ............................................................................................. 112
Table 6.4 Basic Information of EMI File ...................................................................................... 112
Table 6.5 Constituents of the EMD File ...................................................................................... 113
Table 6.6 Authorities .................................................................................................................. 114
Table 6.7 Command Classification of the MRTP Protocol .......................................................... 116
Table 6.8 Key Items of Workflow Control File ........................................................................... 120
Table 6.9 Additional Information of Item Description ................................................................. 120
Table 7.1 Kernel Objects of Server-side Virtual Machine ............................................................ 130
Table 7.2 Component Classification Lists of Client-side Virtual Machine ................................... 133
Table 7.3 The Main Classes of the Client-Side Virtual Machine ................................................... 136
Table 9.1 Contrast of Three Software Platforms in the Tri-Integration Pattern ............................. 172
Chapter 1   Introduction

1.1   Proposed Research and Overview of Problem

The focus of architecture-based software development has shifted from lines-of-code to coarser-grained building blocks and their overall interconnection structure in recent years. Software architecture provides software engineers with new methods of understanding software to build larger and more complex software systems [110, 55, 9, 89]. It is the ultimate aim of software architecture research to provide solutions to developing reusable, high quality and maintainable software.

The design of software architecture is more than a simple activity within a limited scope. It comprises technical, methodological and process aspects of software engineering. It explicitly addresses the needs of productive software development and maintenance, and has a great impact on the quality of a software system.

Software patterns help you build on the collective experience of skilled software engineers. They capture existing, well-proven experience in software development and help to promote good design practice. They are not invented or created artificially. Rather they ‘distill and provide a means to reuse the design knowledge gained by experienced practitioners’ [53].

Every pattern deals with a specific, recurring problem in the design or implementation of a software system. Patterns are an important vehicle for constructing high-quality software architectures and can be used to construct software architectures with specific properties. The following analysis and discussions describe what pattern is:

➢ Context  The context extends the plain problem-solution dichotomy by describing situations in which the problem occurs. Specifying the correct context for a pattern is difficult. It is found to be practically impossible to determine all situations, both general and specific, in which a pattern may be applied. A more
pragmatic approach is to list all known situations where a problem that is addressed by a particular pattern can occur. This does not guarantee that every situation in which a pattern may be relevant is covered, but it at least gives valuable guidance.

➢ **Problem** This part describes the problem that arises repeatedly in the given context. It begins with a general problem specification, capturing its very essence—what is the concrete design issue that must be addressed? This general problem statement is completed by a set of *forces*. Borrowed originally from [2], the term *force* is used to denote any aspect of the problem that should be considered when solving it, such as:

❖ Requirements the solution must fulfil.

❖ Constraints the designer must consider.

❖ Desirable properties the solution should have

➢ **Solution** The solution part of a pattern shows how to solve the recurring problem, or more realistically, how to balance the *forces* associated with it. It addresses components and their relationship structures and run-time behaviour.

On the other hand, Enterprise information systems (EIS), which are typically a collection of relational and legacy database systems, enterprise resource planning (ERP) systems, and mainframe transaction processing systems, provide a critical information infrastructure for an enterprise's business processes. These systems hold the information that an enterprise needs to carry out its daily operations. It is essential that new applications developed for an enterprise is able to integrate with an existing EISs. This has given rise to enterprise application integration, or EAI which enables an enterprise to integrate its existing applications and systems, and enables the addition of new technologies and applications. Enterprises must leverage their existing systems and resources even as they adopt new technologies.

In today's environment, a typical enterprise has a multitude of existing applications running
on diverse platforms and operating systems. Considering the cost already invested in these existing applications, no business can afford to discard them. Plus, since these applications often contain valuable data needed by the enterprise, the enterprise is not likely to disrupt them. Yet, at the same time, enterprises continually grow and require new applications. To keep their businesses growing and remain cost effective, enterprises must integrate their existing systems with new applications, and not replace existing systems with new applications written from scratch.

This thesis will investigate what important aspects are among software architecture, software pattern and software integration, what solutions can provide for resolving development and integration problems, and what frameworks can use for constructing software applications rapidly and effectively.

1.2 Scope of the Thesis and Original Contributions

A new concept of system pattern has been proposed in this thesis. A system pattern, the Tri-Integration pattern, has been defined for software development and integration. The pattern can be used as a template for constructing a concrete application system and a solution for integrating various application systems depending on service integration, component integration and data integration. The scope of the research includes investigation into:

- Understanding of relationships between software architecture, software pattern and software integration.
- Capturing of EAI problem as a software design pattern.
- Derivations of solutions for a system pattern that can help solve development and integration problems.
- Development of a generic software platform and a set of tools based on the system pattern for constructing software application systems rapidly and effectively. Several examples are used to explain the validity and applicability of the platform.
➢ Validity and applicability of the software platform at the system pattern through Case studies.

The original contribution of the thesis lies in four aspects:

➢ A new concept, System pattern is proposed. A system pattern separates a software development problem that arises in specific application contexts into three levels: architecture, design and idiom. This is the most important original contribution from the study.

➢ A specific system pattern, the Tri-Integration Pattern is derived. The pattern presents a complete solution for development and integration problems that arise repeatedly in specific application contexts, and elaborates on dynamic characteristics, implementation methods and consequences. It is the core of the thesis which covers three aspects of integration:

✧ The services integration presents a mechanism that can integrate various services into one application system. It is based on the Multiple Resource Transfer Protocol (MRTP). The workflow control supported is also based on MRTP.

✧ The components integration supports application components developed and deployed by well-designed construction methods. It depends on the virtual machines of server and client. To an application system, the server-side virtual machine is used to implement the business logic; the client-side virtual machine is used to implement the presentation logic. An application system extends its functions with the application components whose development is based on the virtual machines.

✧ The data integration can compose all separate data into a resources repository that may be distributed between different machines. It presents a method of working with unified resources. Such a method, which combines resources authority management and its effective usage, gives this kind of data
integration wide applicability.

➢ A Tri-Integration Pattern-based software development and integration platform: Netcase Anywhere is constructed. The platform contains various modules including management and authorisation for integrating data, the virtual machine for integrating components and MRTP (Multiple Resource Transfer Protocol) for integrating services.

➢ Tools that enhance the practicability of the software platform. These tools include Form Designer which supports multiple language programming, Web Page Designer which supports development of business logic and dynamic web pages, Flow Designer which supports the visual design of flow charts, Resources Manager which supports resources' authorisation and properties' accessing and Alias Manager which supports database linking and configuration.

1.3 Criteria for Success

An important criterion for the success of pattern is how well it meets the objectives of software engineering [68]. Patterns must support the development of complex, large-scale systems. They must also support effective industrial software production, otherwise they remain just an interesting intellectual concept, but useless for constructing software.

The following criteria are given to judge the success of the research described in this thesis:

➢ Is the problem addressed by the pattern generic? Is its resolution efficient?

➢ Does the pattern support the development of complex, large-scale systems?

➢ Is it possible to build a generic software platform based on the pattern?

➢ Does the pattern support effective industrial software production?

➢ Does the pattern help to manage software complexity?
1.4 Thesis Structure

The thesis is organised as follows:

➢ Chapter 1 gives the research problem, motivation, scope and original contribution of the thesis.

➢ Chapter 2 provides an overview of software lifecycle, software architecture, software pattern, introduces two approaches to software development: masterplan and piecemeal growth, and discusses relationships between software pattern and software architecture.

➢ Chapter 3 contains a review and a taxonomy of integration concepts as applied to software systems. It identifies pertinent technical integration issues, and reviews and classifies integration models and strategies.

➢ Chapter 4 introduces the concept of system pattern and provides the definition, context, structures, levels and dynamic behaviour of the Tri-Integration Pattern.

➢ Chapter 5 explains the solutions and structural aspects of the Tri-Integration pattern, and discusses architectural level elements of the pattern.

➢ Chapter 6 provides an application framework of the Tri-Integration pattern, and discusses design-level elements of the pattern.

➢ Chapter 7 is concerned with the idiom-level of the Tri-Integration pattern. The development and integration of the relevant tools is covered.

➢ Chapter 8 details the implementation of a software platform: Netcase Anywhere, which is based on the Tri-Integration Pattern. Three case studies: Process Automation Systems, E-Government Systems and Enterprise Application Integration Systems, are given to show how the software platform works.

➢ Chapter 9 evaluates the Tri-Integration Pattern and the software platform using a set of criteria. Conclusions are drawn and future work is discussed.
Chapter 2 Software Development from Architectural Point of View

2.1 Software Lifecycle

2.1.1 The Explosive Rate of Software Development

Fierce competition among enterprises causes internal and external changes within businesses, such as Business Process Re-engineering (BPR), Enterprise Business Integration (EBI) as well as the updating of information infrastructure. These changes help enterprises gain and retain a competitive edge, especially when management and business structures are reorganised. “Change becomes a first-class design goal and requires business and technology architectures whose components can be added, modified, replaced and reconfigured” [46].

On another hand, over the past decade, the information industry has been dominated by the introduction of new and constantly evolving computer technologies. In particular, the Web browser technology, the so-called first-wave of the Internet, allowed for the explosive use of the Internet in terms of shared information resources. The convergence of the Internet and distributed-object technologies extends this “information-based” Internet to a worldwide “service-based” Web. This evolution is referred to as the Internet’s second-wave [40], where software services and content are distributed using open protocols over the Internet, Intranets, and Extranets.

The explosive impact of computers and information technology on everyday life has generated a need to design and develop new computer software systems and to incorporate new technologies in a rapidly growing range of applications. The tasks performed computer software engineers evolve quickly, reflecting new areas of specialisation or changes in technology, as well as the preferences and practices of employers. Computer software
engineers apply the principles and techniques of computer science, engineering, and mathematical analysis to the design, development, testing, and evaluation of the software and systems that enable computers to perform their many applications. In order to economically obtain software that is reliable and works efficiently, whilst avoiding the side effects of the so-called software crisis, sound engineering principles need to be established and used during the lifecycle of the software.

2.1.2 Overview of Existing Models

The objective of software engineering is to produce software products. The software process is the set of activities and associated results that produce a software product whilst the software lifecycle is the period of time that begins when a software product is conceived and ends when the software is no longer available for use. These activities are mostly carried out by software engineers.

As stated in [62, 69, 67], the software life cycle typically includes a concept phase, requirements phase, design phase, implementation phase, test phase, installation and check out phase, operation and maintenance phase and, sometimes, retirement.

In summary, there are four fundamental process activities, which are common to all software processes [130]. These activities are:

1. **Software specification.** The functionality of the software and constraints on its operation must be defined.

2. **Software development.** The software to meet the specification must be produced.

3. **Software validation.** The software must be validated to ensure that it does what the customer wants.

4. **Software evolution.** The software must evolve to meet changing customer needs.

There is no such thing as a 'right' or a 'wrong' software process. Different models of
software lifecycle decompose these activities in different ways. The timing of the activities varies, as do the results of each activity. Different organisations use different processes to produce the same type of product. Different types of project may be produced by an organisation using different processes. However, some processes are more suitable than others for some types of application. If the wrong process is used, this will probably reduce the quality or the usefulness of the software product to be developed. Furthermore, it is not possible to optimise all process attributes simultaneously.

The lifecycle models serve as a high-level definition of the phases that occur during software development and focus on the high-level activities and their relationships. Detailed software process models are still the subject of research but it is now clear that there are a number of different general models [128, 142, 61, 44, 60, 132] or paradigms of software development.

- Waterfall model

"Waterfall model" is the first explicit model of the software development process proposed by Royce in 1970 was derived from other engineering processes. This was enthusiastically accepted by software project management.

In this model, each phase ends with a gate review that reviews the key work products of that phase and determines whether to progress to the next phase or not. It is document driven. The main work products (Software Requirements Specification, Preliminary Design Specification, Detailed Design Specification, etc) are carried from phase to phase. Each of the subsequent work products builds and traces back to the previous work products.

This model works well when the requirements and other factors (like technology) are well understood and will remain relatively stable. Deliverables are frozen at the end of each phase and server as the baseline for the following phases. However, if the requirement or other factors are not well understood, the model does not work well. Changes are difficult to make and can be expensive. The users cannot see the software until the end of the project ('big bang' software development).
• V-Model

V-Model is a variation of the waterfall model [111] to which it is similar except that it emphasises the importance of considering the testing activities up front instead of later in the life cycle. Each test phase is considered in its matching development phase:

- Requirement = System/Functional Testing
- High-level Design = Integration Testing
- Detailed Design = Unit testing

This model emphasises testing up front to validate the software work products and ensure that testing is planned for and not forgotten. Due to the similarity with the waterfall model, it has all the waterfall lifecycle advantages and disadvantages, with some increase in complexity.

• Prototyping Model (Evolutionary Development)

Evolutionary development is based on the idea of developing an initial implementation, exposing this to user comment and refining this through many versions until an adequate system has been developed. Instead of having separate specification, development and validation activities, these activities are carried out concurrently with rapid feedback across them [118].

This model is able to deal well with rapidly changing and volatile requirements. It is useful when developers and customers do not fully understand the requirements at start, which will become apparent through discovery. The other advantage of this model is to introduce good visibility into the progress. However, this model cannot tell how long it will take to complete the project.

• Spiral Model

In the spiral model [14], the software development process is represented as a spiral rather than as a sequence of activities with backtracking. Thus, the innermost loop might be
concerned with system feasibility, the next loop with system requirements definition, the next loop with system design and so on.

The most important distinction between the spiral model and other software process models is the explicit consideration of risk. Risks are dealt with early on in the software lifecycle and can be dealt with through a number of different means. Once the risks have been reduced to acceptable levels, the developer can spin off the development effort. The spiral model encompasses other process models. Prototyping may be used in one spiral to resolve requirements risk. This may be followed by a conventional waterfall development.

Despite of the above advantages, the model is complex and requires effective management. It can be difficult to determine when the level of risk has become acceptable.

- Incremental Model

The incremental model combines elements of the linear sequential model (applied repetitively) with the iterative philosophy of prototyping [111]. It allows a project to construct the software in incremental stages where each stage adds additional functionality. It provides demonstrable progress and feedback to the stakeholders through incremental releases of functionality.

The model makes it easier to accommodate tactical changes in requirements, features or schedules because of the incremental development, which minimises the amount of loss or impact. It also increases understanding of the problem through successive refinements. However, the model requires extra planning at both the management and technical levels.

2.1.3 Brief Reviews of Methods and Tools

In the last decades, a large number of different approaches to software development have been introduced, of which only a few have survived to be used today. Software methods and tools support software development processes and reflect the types of application being developed. Over the past 20 years, applications have evolved significantly, from those based on a single processor to client-server to n-tier Web-based. Future applications will
take advantage of emerging technologies in high-bandwidth networking. Such new applications will follow existing standards, including Internet and middleware standards, as a way of building upon frameworks, components, and libraries. Templates, patterns, and application generation tools extend the ability to create "standard" architectures for common classes of applications. In this section some dominating software methods and tools will be reviewed briefly.

2.1.3.1 Object-Oriented Approach

An object-oriented approach to the development of software was first proposed in the late 1960s. However, it took almost 20 years for object technologies to become widely used. Object-oriented software engineering became the paradigm of choice for many software project builders and a growing number of information systems and engineering professionals. As time passes, object-oriented technologies are replacing classical software development approaches [118, 140, 120, 119, 35, 98]. Object-oriented software is easier to maintain because its structure is inherently decoupled. This leads to fewer side effects when changes have to be made and less frustration for the software engineer and the customer.

2.1.3.2 Unified Modelling Language (UML)

As the strategic value of software increases for many companies, the industry looks for techniques to automate the production of software, to improve quality and reduce cost and time-to-market. The Unified Modelling Language (UML) was designed to respond to these needs. The UML is a standard language for specifying, visualising, constructing, and documenting the artifacts of software systems, as well as for business modelling and other non-software systems. The UML represents a collection of best engineering practices that have proved successful in the modelling of large and complex systems. The UML is a very important part of developing object-oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects [20]. Using the UML helps project teams communicate, explore potential designs,
and validate the architectural design of the software.

2.1.3.3 XML Introduction

The Extensible Markup Language (XML™) is a broadly adopted format for structured documents and data on the Web. XML is a simple and flexible text format derived from standard generalised mark-up language (SGML) (ISO 8879) and developed by the World Wide Web Consortium® (W3C).

XML is expanding from its origin in document processing and becoming a solution for data integration. XML excels in inter-application data exchange because of its flexible and extensible method for describing data and its capability to communicate over the Internet using the standard HTTP protocol.

This flexibility makes XML a powerful mechanism for business-to-business (B2B) application integration. In addition, a growing number of commercial enterprise application solutions are embracing XML.

2.1.3.4 CASE Tools

Computer-aided Software Engineering (CASE) tools give software engineers the ability to automate many of their tasks when developing software [118, 104, 135]. Tools of this kind can be standalone (such as a compiler) or integrated (allowing information to be passed from one tool to another). The main aims of introducing CASE into software development are to [47, 82]:

- Reduce the amount of time and resources spent on projects.
- Improve the quality of the finished product by developing systems that are easier to test and maintain together with good quality documentation.

CASE tools can be classified by function, by their role as instruments for managers or technical people, by their use in the various steps of the software engineering process, by the environment architecture (hardware and software) that supports them, or even by their
origin or cost [24]. There are twenty-four categories of CASE tools classified in [118] using function as a primary criterion. They are listed as follows, Business process engineering tools, Process modelling and management tools, Project planning tools, Risk analysis tools, Project management tools, Requirement tracing tools, Metrics and management tools, Documentation tools, System software tools, Quality assurance tools, database management tools, Software configuration management tools, Analysis and design tools, Prototyping and Simulation (PRO/SIM) tools, Interface design and development tools, Prototyping tools, Programming tools, Web development tools, Integration and testing tools, Static analysis tools, Dynamic analysis tools, Test management tools, Client/Server testing tools and Re-engineering tools. These tools are limited to specific programming languages (although most major languages are addressed) and require some degree of interaction with the software engineer.

2.2 Masterplan Approach: Software Architecture

Software architecture has become an area of intense research in the software engineering community. It provides an efficient way for greater granularity of software reuse. A number of architecture modelling notations and support tools, as well as new architectural patterns, have emerged. The focus of architecture-based software development is shifted from lines-of-code to coarser-grained building blocks and their overall interconnection structure. Explicit focus on architecture has shown tremendous potential to improve the current state-of-the-art in software development and alleviate many of its problems.

As the size and complexity of software systems increase, the design and specification of overall system structure become more significant issues than the choice of algorithms and data structures. Structural issues include the organisation of a system as a collection of components; global control structures; the protocols for communication, synchronisation and data access; the assignment of functionality to design elements; the composition of design elements; physical distribution; scaling and performance; dimensions of evolution; selection among design alternatives and non-functional properties. This is the software
architecture level of design [126].

As a result, good software architecture that is understood by the stakeholders and by the development team members is needed in order to produce a successful product. The consistent, reusable environment provided by certain software architecture gives the following advantages:

- Supports reuse
- Realises software non-functional properties
- Controls system complexity
- Controls consistency and uniformity
- Enforces communication skills
- Reduces system cost
- Reduces system risk

### 2.2.1 Development of Software Architecture

Edgar Dijkstra stressed as early as in 1968 that how software is partitioned and structured is important. In the book Mythical Man Month [17], Fred Brooks Jr. mentioned the concept of software architecture in 1975. The current view of software architecture arose in the 1990s when software architecture became recognised as an independent area of research.

While it has long been recognised that finding an appropriate architectural design for a system is a key element of its long-term success, current practice for describing architectures is typically informal and idiosyncratic. Architectural structures are often described in terms of idiomatic patterns that have emerged informally over time. For example, typical descriptions of software architectures include statements such as these:

In 1994, writing about a method to evaluate architectures with respect to the quality attributes they instil in a system, Bass and his colleagues claim that: "...the architectural
design of a system can be described from (at least) three perspectives -- functional partitioning of its domain of interest, its structure, and the allocation of domain function to that structure."

Definition 1. Perry & Wolf Model [110]:

\[
\text{SA} = \{\text{elements, form, rationale}\}.
\]

Perry and Wolf distinguish three different classes of architectural elements: processing elements, data elements and connection elements. The architectural form consists of weighted properties and relationships. Properties are used to constrain the choice of architectural elements. Relationships are used to constrain how the different elements may interact and how they are organised with respect to each other in the architecture. A part of an architecture is the rationale for the various choices of architectural style, elements, and form.

Definition 2. Garlan & Shaw Model [55]:

\[
\text{SA} = \{\text{components, connectors, constraints}\}.
\]

Components can be a group of code, for example, a procedure module, or an independent program such as SQL server for a database. Connectors represent interactions between components, for example, procedure calls, pipes and RPC. An overall architecture also includes some constraints.

Definition 3. Bass & Clements & Kazman Model [9]:

\[
\text{SA} = \{\text{elements, externally visible properties, relationships}\}.
\]

The software architecture of a program or computing system is the structure or structures of the system, which comprise software elements, the externally visible properties of those elements, and the relationships among them. "Externally visible" properties refer to those assumptions other elements can make about an element, such as its provided services, performance characteristics, fault handling, shared resource usage, and so on.

Definition 4. Recommended Practice for Architectural Description of Software-Intensive
Systems [89, 66]:

\[ \text{SA} = \{\text{components, connectors, environment, principle}\}. \]

Software architecture is defined as the fundamental organisation of a system, embodied in its components, their relationships to each other and the environment, and the principles governing its design and evolution.

Software architecture can therefore be summarised as follows:

1. Software architecture deals with the design and implementation of the high-level structure of the overall software system. The software architecture of a system is an artifact. It is the result of the software design activity.

2. A software architecture is a description of subsystems and components of a software system and the relationships between them. Components and connectors are recognised as the fundamental ingredients of software architecture.

### 2.2.2 Basic Ingredients of Software Architecture

A well-accepted taxonomy of such architectural paradigms, let alone a fully developed theory of software architecture, is still some way off. But a number of basic ingredients of an architectural description can be identified [25, 18, 90, 61, 7, 16, 63, 74, 78].

**Components**

Components are typically specified to show the relevant functional and non-functional properties of a software system. A component is an encapsulated part of a software system. A component has an interface. Components serve as the building blocks for the structure of a system. At a programming-language level, components may be represented as modules, classes, objects or a set of related functions.

Perry and Wolf [110], for example, distinguish three different kinds of components, called elements:
Processing elements

Data elements

Connecting elements.

Processing elements supply transformations of the data elements that contain the
information that is transformed. Connecting elements—which at any time may be either
processing elements, data elements or both—constitute the 'glue' that holds the different
pieces together.

Another categorisation of components developed for the object-oriented programming
paradigm is as follows:

Controller components

Coordinator components

Interface components

Service provider components

Information holder components

Structuring components

Connectors

A connector [126] denotes a relationship between components. A connector may be static
or dynamic. Static connectors show directly in source code. They deal with the placement
of components within an architecture. Dynamic connectors deal with temporal connections
and dynamic interaction between components. They may not be easily visible from the
static structure of source code.

Aggregation and inheritance are examples of static connectors. Object creation,
communication between objects, and data transfer are usually dynamic connectors. An
example of a temporal connector is when an object is inserted into a container at some
point in time and later deleted.
The following Figure 2.1 shows three static connectors in OMT notation [117].

![Figure 2.1 Three Static Connectors](image)

Connectors between components have a great impact on the overall quality of a software architecture. For example, changeability is much better supported by software architectures in which the connectors support the variation of the components, in contrast to architectures in which any change to a component affects the implementation of its clients and collaborators. This explicit exposition of the importance of connectors can be observed in many of the recent definitions and discussions of software architecture [126, 110].

### 2.2.3 Enabling Techniques for Software Architecture

The construction of software is based on several fundamental principles. These principles are called enabling techniques, since the principles involved have become blurred over time. Techniques have been developed to realise these widely-accepted principles, to a degree that it becomes increasingly difficult to differentiate between principles and techniques. A simple approach therefore is taken to using both terms as synonyms.

All enabling techniques are independent of a specific software development method, and most of them have been known for years. They were developed and proposed mainly in the 1970's in connection with publications on structured programming. Classical references are the papers by Parnas and colleagues, see for example [108] and [107]. Although the importance of enabling techniques has been recognised for a long time, their significance for successful software development has increased only over the last few years, strongly
linked to the emerging discipline of software architecture. Patterns for software architecture are explicitly built on these principles, many of them with a special focus on a particular principle. The following sections summarise some of the most important enabling techniques for software architecture:

- Abstraction
- Encapsulation
- Information Hiding
- Modularisation
- Separation of Concerns
- Coupling and Cohesion
- Sufficiency, Completeness and Primitiveness
- Separation of Policy and Implementation
- Separation of Interface and Implementation
- Single Point of Reference
- Divide-and-Conquer

**Abstraction.** Abstraction is one of the fundamental principles humans use to cope with complexity. Grady Booch defines abstraction as 'the essential characteristics of an object that distinguish it from all other kinds of objects and thus provide crisply defined conceptual boundaries relative to the perspective of the viewer' [15]. The word 'object' may be replaced with 'component' to achieve a more general definition of abstraction. Several forms of abstraction exist, such as entity abstraction, action abstraction, virtual machine abstraction and coincidental abstraction [125].

**Encapsulation.** Encapsulation deals with grouping the elements of an abstraction that constitute its structure and behaviour, and with separating different abstractions from each other. Encapsulation provides explicit barriers between abstractions. Encapsulation fosters
non-functional properties like changeability and reusability.

**Information Hiding.** Information hiding involves concealing the details of a component's implementation from its clients, to handle system complexity better and to minimise coupling between components. Any details of a component that clients do not need to know in order to use it properly should be hidden by the component. The principle of encapsulation is often used as a way to achieve information hiding. Information hiding can also be achieved using the principle of separation of interface and implementation.

However, what is to be hidden inside a component sometimes depends on the application. Aspects that clients do not need to know in one application may need to be externally visible in another. For example, in one system direct access to the internal data structures of a component may be necessary for performance tuning. Such access may not be necessary when the component is used in other systems for which its performance is already adequate.

**Modularisation.** Modularisation is concerned with the meaningful decomposition of a software system and with its grouping into subsystems and components. The major task is to decide how to physically package the entities that form the logical structure of an application. The main objective of modularisation is to handle system complexity by introducing well-defined and documented boundaries within a program. Modules serve as physical containers for the functionalities or responsibilities of an application. Modularisation is closely related to the principle of encapsulation.

**Separation of Concerns.** Different or unrelated responsibilities should be separated from each other within a software system, for example by attaching them to different components. Collaborating components that contribute to the solution of a specific task should be separated from components that are involved in the computation of other tasks. If a component plays different roles in different contexts, these roles should be independent and separate from each other within the component.

**Coupling and Cohesion.** Coupling and cohesion are principles originally introduced as
part of the structured design approach. Coupling focuses on inter-module aspects, whereas cohesion emphasises intra-module characteristics.

Coupling is the measure of the strength of association established by a connection from one module to another. Strong coupling complicates a system, since a module is harder to understand, change, or to correct if it is highly interrelated with other modules. Complexity can be reduced by designing systems with weak coupling between modules.

Cohesion measures the degree of connectivity between the functions and elements of a single module. There are several forms of cohesion. The most desirable form is functional cohesion, in which the elements of a module or component 'all work together to provide some well-bounded behaviour' [15]. The worst form is coincidental cohesion, in which entirely unrelated abstractions are thrown into the same module. Other types of cohesion -- logical cohesion, temporal cohesion, procedural cohesion, communicational cohesion, sequential cohesion and informal cohesion -- are described by.

**Sufficiency, Completeness and Primitiveness.** [15] states that 'Every component of a software system should be sufficient, complete, and primitive'. 'Sufficient' means that the component should capture those characteristics of an abstraction that are necessary to permit a meaningful and efficient interaction with the component. 'Completeness' means that a component should capture all relevant characteristics of its abstraction. By 'primitiveness', Booch means that all the operations a component can perform can be implemented easily. It is a major goal of every pattern to be sufficient and complete with respect to the solution of a given problem.

**Separation of Policy and Implementation.** A component of a software system should deal with policy or implementation, but not both:

- A policy component deals with context-sensitive decisions, knowledge about the semantics and interpretation of information, the assembly of many disjoint computations into a result or the selection of parameter values.

- An implementation component deals with the execution of a fully-specified
algorithm in which no context-sensitive decisions have to be made. The context and interpretation are external, and are normally supplied by arguments to the component.

Because of their independence from a certain context, pure implementation components are easier to reuse and maintain, whereas policy components are often application-specific and subject to change.

If it is not possible to separate policy and implementation into different components within a software architecture, there should at least be a clear separation of policy and implementation functionality within a component.

**Separation of Interface and Implementation.** Any component should consist of two parts:

- An interface part that defines the functionality provided by the component and specifies how to use it. This interface is accessible by the clients of the component. An exported interface of this type usually consists of function signatures.

- An implementation part that includes the actual code for the functionality provided by the component. The implementation part may also comprise additional functions and data structures that are only used internally to the component. The implementation part is not accessible by the component's clients.

The main objective of this principle is to protect a component's clients from its implementation details, and only to provide clients with the component's interface specification and guidelines for use. In addition, this principle allows the functionality of a component to be implemented independently of its use by other components. Separation of interface and implementation is, like encapsulation, a technique to achieve information hiding, the principle that states that 'A client should only know what he/she needs to know'.

Separation of interface and implementation also supports changeability -- a component is much easier to change if its interface is separated from its implementation. This separation prevents clients from being directly affected by a change. The principle especially eases the
task of changing a component's behaviour or representation, for example for performance tuning, in cases where the change does not necessitate a change to its interface.

**Single Point of Reference.** Any item within a software system should be declared and defined only once. The main objective of this principle is to avoid problems of inconsistency.

Due to their design principles and implementations, however, many programming languages such as C++ require a single point of definition, but allow or even mandate several points of declaration. In the case of C++ this is mainly due to the limitations of traditional compiler and linker technologies [136]. The consequence for the programmer is an increased workload in manually maintaining consistency.

**Divide-and-Conquer.** This principle is well known, both from the politics of the ancient world as well as from combinatorial algorithms such as Merge-sort. This principle is used heavily in software architecture. Top-down design, for example, divides a task or component into smaller parts that can be designed independently. The Whole-Part pattern [21] approaches this technique at the pattern level. Other patterns also concentrate on such subdivision, although more specifically than the generic Whole-Part. The Microkernel pattern [21], for example, subdivides what once might have been a monolithic block of code. Divide-and-Conquer also often provides a way to realise the principle of separation of concerns.

It is important to note that not all general principles are complementary -- some are contradictory. Examples of this are the principle of separating interface and implementation and that of single point of reference. The first principle -- when enforced by traditional technology--requires at least two points of reference for a particular function, one in the interface part of a component and the other in its implementation part. This is in contradiction to the strict interpretation of the principle of a single point of reference. Generating interfaces from implementations is a solution that is used in more modern approaches.
Other principles are closely related, such as abstraction and encapsulation. A proper abstraction for a particular entity within a software system also requires encapsulation of all the elements that constitute its structure in a single component or module.

### 2.2.4 Non-functional Properties of Software Architecture

A functional property deals with a particular aspect of a system's functionality, and is usually related to a specified functional requirement. A functional property may either be made directly visible to users of an application by means of a particular function, or it may represent aspects of its implementation, such as the algorithm used to compute the function.

In the past, developers concentrated on providing the stated functional properties for software, today non-functional properties are becoming increasingly important:

A non-functional property denotes a feature of a system that is not covered by its functional description. A non-functional property typically addresses aspects related to the reliability, compatibility, cost, ease of use, maintenance or development of a software system.

Non-functional properties of a software system have a great impact on its development and maintenance, its general operability and its use of computer resources. Both functional and non-functional properties have an equal impact on the quality of an application and its architecture. The larger and more complex a software system and the longer its lifetime are, the more important its non-functional properties become.

In this section some of the most important non-functional properties of software architecture are discussed in relation to patterns:

- Changeability
- Interoperability
- Efficiency
- Reliability
- Testability
> Reusability

**Changeability.** Large-scale industrial and commercial software systems usually have a long life-span, sometimes twenty years or more. Many such applications do not remain static after their original development phase; they tend to evolve continuously during their lifetime. Existing requirements change and new ones are added. To reduce maintenance costs and the workload involved in changing an application, it is important to prepare its architecture for modification and evolution.

Parnas lists two reasons why software ages [106]:

> Lack of movement--software ages if it is not frequently updated.

> Ignorant surgery--changes made by people who do not understand the original design gradually destroy the architecture.

In another publication, Parnas adds two further reasons:

> The software is inflexible from the start.

> The documentation is inadequate, allowing understanding of the system to be eroded over time.

The consequent of software aging, as described in [106], is a growing inability to keep up with the market by introducing new features, reduced performance and decreased reliability. These can be prevented by accurate documentation, preserving structure when introducing changes, intense reviewing, and designing for change a priori.

It is considered that changeability has four aspects:

> Maintainability. This deals mainly with problem fixing, 'repairing' a software system after errors occur. A software architecture that is well-prepared for maintainability tends to localise changes and minimise their side effects on other components.

> Extensibility. This focuses on the extension of a software system with new features,
as well as the replacement of components with improved versions and the removal of unwanted or unnecessary features and components. To achieve extensibility a software system requires loosely-coupled components. The aim is a structure that allows you to exchange components without affecting their clients. Support for integrating new components into an existing architecture is also necessary.

- **Restructuring.** This deals with the reorganisation of the components of a software system and the relationships between them, for example when changing the placement of a component by moving it to a different subsystem. Support for the restructuring of a software system needs careful design of the relationships between components. They should ideally allow components to be configured flexibly without affecting major parts of their implementation.

- **Portability.** This deals with adapting a software system to a variety of hardware platforms, user interfaces, operating systems, programming languages or compilers. To be portable, a software system needs to be organised in such a way that dependencies on hardware, other software systems and environments are factored out into special components such as system and user interface libraries.

A software system designed for change also supports the construction of variants for different customers better than a software system that is not so designed.

With the growing use of patterns, classes are no longer simple. Every 'chunk' of code is highly flexible and can adapt to many different contexts. Such flexibility, however, comes at a price. Flexible software often consumes more resources by using more levels of indirection or increasing storage consumption. It also requires more thought and more work in coding. Good designers therefore try to decide in advance which parts of the software should be highly flexible to cope with foreseeable changes, and which parts will probably remain fairly static. If they prove wrong, there are still ways to introduce additional flexibility by carefully restructuring parts of the system, or by using a pattern that supports design for change. This approach is more economical than engineering in total changeability from the start.
**Interoperability.** Software that forms part of a system does not exist independently. It is frequently interacting with other systems or its environment. To support interoperability, a software architecture must be designed to offer well-defined access to externally-visible functionality and data structures. The interaction of a program with software systems written in other programming languages is an aspect of interoperability that also impacts the software architecture of an application.

**Efficiency.** Efficiency deals with the use of the resources available for the execution of software, and how this impacts on response times, throughput and storage consumption. Efficiency is not only a matter of using sophisticated algorithms. The appropriate distributions of responsibilities to components, as well as their coupling, are important architectural activities for achieving efficiency in a given application.

Efficiency also plays a significant role in distributed software systems. The IPC (inter-process communication) mechanisms underlying a distributed application must be fast enough to transfer messages and data with sufficient speed. Many patterns, however, introduce an additional level of indirection to solve a problem, which may decrease rather than increase efficiency.

**Reliability.** Reliability deals with the general ability of a software system to maintain its functionality, both in the face of application or system errors and in situations of unexpected or incorrect usage. Two aspects of reliability can be distinguished:

- **Fault tolerance.** This aims at ensuring correct behaviour in the event of errors, and their internal 'repair', such as losing a connection to a remote component in a distributed software system and subsequently reconnecting to it. After repairing such an error, the software system should resume or repeat the execution of the operation in progress when the error occurred.

- **Robustness.** This deals with protecting an application against incorrect usage and degenerate input, and keeping it in a defined state in the event of unexpected errors. Note that in contrast to fault tolerance, robustness does not necessarily mean that
the software is able to continue computation in the event of errors—it may only guarantee that the software terminates in a defined way.

Software architecture has a major impact on the reliability of a software system. Examples of the way in which software architecture supports reliability include the intentional inclusion of redundancy in an application, or the integration of monitoring components and exception handling.

**Testability.** With the increasing size and complexity of software systems, especially industrial ones, testing is becoming more difficult and expensive. A software system needs support from its architecture to ease the evaluation of its correctness, however, proving correctness is, unfortunately, still out of reach in most cases. Software structures that support testability allow for better fault detection and fixing, and also for temporary integration of debugging code and debugging components.

**Reusability.** Reusability is currently one of the most discussed topics in software engineering. It promises a reduction of both cost and development time for software systems, as well as better software quality [76, 148]. Adele Goldberg once defined reuse as 'the act of achieving what is desired with the help of what already exists' [56]. Reusability has two major aspects; software development with reuse and software development for reuse:

- Software development with reuse means reusing existing components and results from previous projects or commercial libraries, design analyses, design specifications or code components. These reusable artifacts are integrated into the application under development, either as they are or with modifications. Practising software development with reuse requires the construction of software architectures that allow you to 'plug in' prefabricated structures and code components. Software development with reuse aims to support software composition, which means composing an application out of existing components by adapting them to the needs of the development and implementing 'glue' components to connect them.
Software development for reuse focuses on producing components that are potentially reusable in future projects as part of the current software development. This requires software architectures that allow self-contained parts to be taken from the application under development and reused in other systems without significant modification.

Some non-functional properties require similar architectural techniques for their achievement, for example design reusability and changeability. Others serve a similar overall purpose: for example, design portability and interoperability deal with the integration of a software system into its environment, while reliability and efficiency deal with its general usability.

Non-functional properties may contradict as well as complement each other. For example, when replicating the functionality of an application to achieve fault tolerance, the resulting structure is usually less efficient and more expensive than a structure without such redundancy. When specifying non-functional requirements for a software architecture, it is needed explicitly to consider the interdependencies and trade-offs that exist between them. The specification of an ordering priority between different non-functional requirements is also needed, to define a preference of one requirement against another in case of conflict.

### 2.2.5 Different Research Approaches to Software Architecture

Practices show that good software architecture is a key element for the success of software development. However, different viewpoints result in different software architecture definitions, research contents and approaches.

Research approaches to software architecture can be separated into two camps: the Academic approach and the Industrial approach. The Academic approach emphasises particularly formalisation theories of software architecture whilst the Industrial approach regards software architecture design, description and implementation as a whole within traditional software system modelling. Therefore they employ software-modelling techniques to describe software architecture, for example, architectural patterns.
Table 2.1 gives the comparison of two different research approaches to software architecture:

<table>
<thead>
<tr>
<th>Academic Approach</th>
<th>Industrial Approach</th>
</tr>
</thead>
<tbody>
<tr>
<td>Concerns itself with analysis and evaluation on architectural model</td>
<td>Concerns itself with broad development issues</td>
</tr>
<tr>
<td>Single model</td>
<td>Multiple architectural views</td>
</tr>
<tr>
<td>Rigorous modelling token</td>
<td>Emphasis more on practicality than rigorousness</td>
</tr>
<tr>
<td>Powerful analysis technique</td>
<td>Views architecture as a blueprint for development</td>
</tr>
<tr>
<td>Solution for special goals</td>
<td>Solution for general goals</td>
</tr>
<tr>
<td>Ignores implementation</td>
<td>Related closely with implementation</td>
</tr>
</tbody>
</table>

2.2.6 Architectural Styles

In 1992 Dwayne E. Perry and Alexander L. Wolf introduced the notion of architectural style [110]:

An architectural style defines a family of software systems in terms of their structural organisation. An architectural style expresses components and the relationships between them, with the constraints of their application, and the associated composition and design rules for their construction.
Architectural styles have also been proposed in [126] and [131]. Generally speaking, an architectural style expresses a particular kind of fundamental structure for a software system together with an associated method that specifies how to construct it. An architectural style also comprises information about when to use the architecture it describes, its invariants and specialisations, as well as the consequences of its application.

2.3 Piecemeal Growth Approach: Software Patterns

Software development is always through piecemeal growth and rarely through thorough design. Such planned development can lead both to technical problems because the future of a piece of software cannot be known and also to social problems because completely planned development alienates those developers who are not also the planners [26].

Many of the details only become known to developers as they progress in the implementation. Even if they knew all of the relevant facts before they started, experience shows that human beings are unable to comprehend fully the plethora of details that must be taken into account in order to design and build a correct system. Even if they could master all of the detail needed, all but the most trivial projects are subject to change for external reasons [51].

One of the hallmarks of architecture design is the use of idiomatic patterns of system organisation. Many of these patterns have been developed over the years as system designers recognised the value of specific organisational principles and structures for certain classes of software.

2.3.1 Pattern Definition

Patterns help build on the collective experience of skilled software engineers. They capture existing, well-proven experience in software development and help to promote good design practice [10]. Every pattern deals with a specific, recurring problem in the design or implementation of a software system. Patterns can be used to construct software
architecture with specific properties.

When experts work on a particular problem, they often recall a similar problem they have already solved, and reuse the essence of its solution to solve the new problem. It is unusual for experts to tackle a problem by inventing a new solution that is completely distinct from existing ones. This kind of 'expert' behaviour, the thinking in problem-solution pairs, is common to many different domains, such as architecture [1, 3, 4], economics [43] and software engineering [11, 124]. It is a natural way of coping with any kind of problem or social interaction [100].

Abstracting from specific problem-solution pairs and distilling out common factors leads to patterns: 'These problem-solution pairs tend to fall into families of similar problems and solutions with each family exhibiting a pattern in both the problems and the solutions' [75].

A pattern [21] describes a particular recurring design problem that arises in specific design contexts, and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate.

There are several properties of patterns [21]:

1. A pattern addresses a recurring design problem that arises in specific design situations, and presents a solution to it.

2. Patterns document existing, well-proven design experience.

3. Patterns identify and specify abstractions that are above the level of single classes and instances, or of components [53].

4. Patterns provide a common vocabulary and understanding for design principles [53].

5. Patterns are a means of documenting software architectures.

6. Patterns support the construction of software with defined properties.
7. Patterns help to build complex and heterogeneous software architectures.

8. Patterns help to manage software complexity.

### 2.3.2 Pattern Description

Patterns must be presented in an appropriate form if they are to be understood and discussed. A good description helps to grasp the essence of a pattern immediately—what is the problem the pattern addresses, and what is the proposed solution? A good description also provides all the details necessary to implement a pattern and to consider the consequences of its application.

Patterns should also be described uniformly. This helps to compare one pattern with another, especially when alternative solutions to a problem are being sought.

The basic Context-Problem-Solution structure [21] provides a good starting point for a description format that meets the above requirements. It captures the essential characteristics of a pattern, and provides the key ideas. The description template [21] has therefore been based on this structure. A Context-Problem-Solution schema that underlies every pattern can be adopted:

1. **Context**: a situation giving rise to a problem.

The context extends the plain problem-solution dichotomy by describing situations in which the problem occurs. The context of a pattern may be fairly general. On the other hand, the context can tie specific patterns together.

2. **Problem**: the recurring problem arising in that context.

The pattern community uses the term force to denote any aspect of the problem that should be considered when solving it, such as:

- **Requirements** the solution must fulfil – for example, that peer-to-peer inter-process communication must be efficient.

- **Constraints** you must consider – for example, that inter-process communication
must follow a particular protocol.

- Desirable properties the solution should have – for example, that changing software should be easy.

3. Solution: a proven resolution of the problem, or how to balance the forces associated with the recurring problem.

In software architecture such a solution includes two aspects:

Firstly, every pattern specifies a certain structure, a spatial configuration of elements. This structure addresses the static aspects of the solution. Since such a structure can be seen as a micro-architecture [53], it consists, like any software architecture, of both components and their relationships. Within this structure the components serve as building blocks, and each component has a defined responsibility. The relationships between the components determine their placement.

Secondly, every pattern specifies a run-time behaviour. The run-time behaviour addresses the dynamic aspects of the solution. How do the participants of the pattern collaborate? How is work organised between them? How do they communicate with each other?

It is important to note that the solution does not necessarily resolve all forces associated with the problem. It may focus on particular forces and leave others half or completely unresolved, especially if forces are contradictory.

As mentioned above, the Context-Problem-Solution schema as a whole denotes a type of rule that establishes a relationship between a given context, a certain problem arising in that context, and an appropriate solution to the problem. All three parts of this schema are closely coupled.

However, describing a pattern based exclusively on a Context-Problem-Solution schema is not enough. A pattern must be named, preferably with an intuitive name, if it is to be shared and discussed. Such a name should also convey the essence of a pattern. A good pattern name is vital, as it will become part of the design vocabulary [53].
An example can be added to the pattern description to help explain the problem and its associated forces. This example is repeatedly referred to when discussing a solution and the implementation aspects of the general pattern.

Table 2.2 Pattern Description Template

<table>
<thead>
<tr>
<th>No.</th>
<th>Items</th>
<th>Details</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Name</td>
<td>The name and a short summary of the pattern.</td>
</tr>
<tr>
<td>2</td>
<td>Also Known As</td>
<td>Other names for the pattern, if any are known.</td>
</tr>
<tr>
<td>3</td>
<td>Example</td>
<td>A real-world example demonstrating the existence of the problem and the need for the pattern.</td>
</tr>
<tr>
<td>4</td>
<td>Context</td>
<td>The situations in which the pattern may apply.</td>
</tr>
<tr>
<td>5</td>
<td>Problem</td>
<td>The problem the pattern addresses, including a discussion of its associated forces.</td>
</tr>
<tr>
<td>6</td>
<td>Solution</td>
<td>The fundamental solution principle underlying the pattern.</td>
</tr>
<tr>
<td>7</td>
<td>Structure</td>
<td>A detailed specification of the structural aspects of the pattern, including CRC-cards [12] for each participating component and an OMT class diagram [117].</td>
</tr>
<tr>
<td>8</td>
<td>Dynamics</td>
<td>Typical scenarios describing the run-time behaviour of the pattern. The scenarios are further illustrated with Object Message Sequence Charts.</td>
</tr>
<tr>
<td>9</td>
<td>Implementation</td>
<td>Guidelines for implementing the pattern. These are only a suggestion, not an immutable rule. The implementation should be adapted to meet the needs of the project, by adding different, extra, or more detailed steps, or by re-ordering the steps. C++, Smalltalk [57], Java or pSather code fragments may be used to illustrate a possible implementation, often describing details of the example problem.</td>
</tr>
<tr>
<td>10</td>
<td>Example Resolved</td>
<td>Discussion of any important aspects for resolving the example that are not yet covered in the Solution, Structure, Dynamics and Implementation sections.</td>
</tr>
<tr>
<td>11</td>
<td>Variants</td>
<td>A brief description of variants or specialisations of a pattern.</td>
</tr>
<tr>
<td>12</td>
<td>Known Uses</td>
<td>Examples of the use of the pattern, taken from existing systems.</td>
</tr>
<tr>
<td>13</td>
<td>Consequences</td>
<td>The benefits the pattern provides, and any potential liabilities.</td>
</tr>
<tr>
<td>14</td>
<td>See Also</td>
<td>References to patterns that solve similar problems, and to patterns that help to refine the pattern that is being described.</td>
</tr>
</tbody>
</table>

Diagrams and scenarios are further used to illustrate the static and dynamic aspects of the solution. Implementation guidelines are also included with the pattern. These guidelines help to transform a given architecture into one that uses the pattern. Sample code is added, and lists successful applications of the pattern to enhance its credibility.
Variants of a pattern are also described. Variants provide alternative solutions to a problem. However, these variants may not be described with the same level of detail as the original pattern, they need only be described briefly.

A discussion of the benefits and potential liabilities of a pattern highlights the consequences of its application. This provides information to help to decide whether the pattern can be used to provide an adequate solution to a specific problem. Other related patterns are also cross-referenced, either because they refine the current pattern, or because they address a similar problem.

With all this information available and appropriately laid out, a pattern should be capable of being understood, applied and implemented correctly.

The pattern description template [21] shows in Table 2.2. See [30, 33, 34, 2] for the other pattern languages.

### 2.3.3 Pattern Categories

#### 2.3.3.1 Architectural Patterns

Viable software architectures are built according to some overall structuring principles. These principles with architectural patterns are described below.

An architectural pattern expresses a fundamental structural organisation schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships between them.

Architectural patterns are templates for concrete software architectures [54]. They specify the system-wide structural properties of an application, and have an impact on the architecture of its sub systems. The selection of an architectural pattern is therefore a fundamental design decision when developing a software system.

The Model-View-Controller pattern [77] is one of the best-known examples of an architectural pattern. It provides a structure for interactive software systems. Pipes and
Filters [126], Layes [137], Broker [21] and Microkernel [21] are also well-known architectural patterns.

### 2.3.3.2 Design Patterns

The subsystems of a software architecture, as well as the relationships between them, usually consist of several smaller architectural units. These design patterns are described below.

A *design pattern* provides a scheme for refining the subsystems or components of a software system, or the relationships between them.

It describes a commonly-recurring structure of communicating components that solves a general design problem within a particular context [52].

Design patterns are medium-scale patterns. They are smaller in scale than architectural patterns, but tend to be independent of a particular programming language or programming paradigm. The application of a design pattern has no effect on the fundamental structure of a software system, but may have a strong influence on the architecture of a subsystem.

Design Patterns are typically applied in the context of an object-oriented language and are implemented so that the pattern participants correspond to object instances that are created and connected at run-time [42].

Many design patterns provide structures for decomposing more complex services or components, such as Client-Dispatcher-Server [21] pattern, Proxy [53] pattern, Component Configurator [123] pattern and Recoverable Distributor [71] pattern. Others address the effective cooperation between them, such as the Observer [53] pattern.

### 2.3.3.3 Idioms

Idioms deal with the implementation of particular design issues.

An *idiom* is a low-level pattern specific to a programming language. An idiom describes
how to implement particular aspects of components or the relationships between them using the features of the given language.

Idioms represent the lowest-level patterns. They address aspects of both design and implementation, such as Counted Body [32] pattern.

Most idioms are language-specific since they capture existing programming experience. Often the same idiom looks different for different languages, and sometimes an idiom that is useful for one programming language does not make sense in another. For example, the C++ community uses reference-counting idioms to manage dynamically-allocated resources [29]; Smalltalk [81, 115] provides a garbage collection mechanism, so has no need for such idioms.

2.3.4 Relationship between Patterns

A close look at many patterns reveals that, despite initial impressions, their components and relationships are not always as 'atomic' as they first appear to be. A pattern solves a particular problem, but its application may raise new problems. Some of these can be solved by other patterns. Single components or relationships inside a particular pattern may therefore be described by smaller patterns, all of them integrated by the larger pattern in which they are contained.

Example 1

Refinement of the Model-View-Controller pattern.

The Model-View-Controller [77] pattern separates core functionality from human-computer interaction to provide adaptable user interfaces. However, applying this pattern introduces a new problem. Views, and sometimes even controllers, depend on the state of the model. The consistency between them must be maintained: whenever the state of the model changes, all its dependent views and controllers must be updated. However, the ability to change the user interface must not be lost. The Observer Pattern [52] helps to solve this problem; the model embodies the role of the subject while views and controllers
play the roles of observers.

Most patterns for software architecture raise problems that can be solved by smaller patterns. Patterns do not usually exist in isolation. Christopher Alexander puts this in somewhat idealistic terms: 'Each pattern depends on the smaller patterns it contains and on the larger patterns in which it is contained' [1].

A pattern may also be a variant of another. From a general perspective a pattern and its variants describe solutions to very similar problems. These problems usually vary in some of the forces involved, rather than in general character. This is illustrated in the following example.

**Example 2**

The Document-View variant of the Model-View-Controller pattern.

Consider the development of an interactive text editor using the Model-View-Controller pattern. Within such an application it is hard to separate controller functionality from view functionality. Suppose you select text with the mouse and change it from regular to bold face. Text selection is a controller action that does not cause changes to the model. The selected text just serves as input for another controller action, here changing the face of the selected text. However, text selection has a visual appearance since the selected text is highlighted. In a strict Model-View-Controller structure, the controller must either implement this 'view-like' behaviour by itself, or must cooperate with the view in which the selected text appears. Both solutions require some unnecessary implementation overhead.

In such a situation it is better to apply the Document-View variant [21] of the Model-View-Controller pattern, which unifies the view and controller functionality in a single component, the view of the Document-View pattern. The document component directly corresponds to the model of the Model-View-Controller triad. When using the Document-View variant, however, it loses the ability to change input and output functionality independently.

Patterns can also combine in more complex structures at the same level of abstraction. This
happens when your original problem includes more forces than can be balanced by a single pattern. In this case, applying several patterns can solve the problem. Each pattern resolves a particular subset of the forces.

Example 3

Transparent peer-to-peer inter-process communication.

Suppose you have to develop a distributed application with high performance peer-to-peer inter-process communication. The following forces must be balanced:

- The inter-process communication must be efficient. Spending time searching for the location of remote servers is undesirable.

- Independence from a particular inter-process communication mechanism is desirable. The mechanism must be exchangeable without affecting clients or servers.

- Clients should not be aware of or dependent on, the name and location of their servers. Instead, they should communicate with each other as if they were in the same process.

This problem cannot be solved by any single pattern in isolation, but two patterns in combination can achieve this. The Forwarder-Receiver pattern [21] resolves the first and second force. It offers a general interface for sending and receiving messages and data across process boundaries. The pattern hides the details of the concrete inter-process communication mechanism. Replacing this mechanism only affects the forwarders and receivers of the system. In addition, the pattern offers a name-to-address mapping for servers.

The Proxy Pattern resolves the third force. In this pattern, the client communicates with a representative of the server that is located in the same process. This representative, the remote proxy [87], knows details about the server, such as its name, and forwards every request to it.
All three kinds of relationship-refinement, variants and combination -- help in using patterns effectively. Refinement supports the implementation of a pattern, combination helps you compose complex design structures, and variants help when selecting the right pattern in a given design situation.

You can find complementary discussion of relationships between patterns in [147].

2.4 Patterns and Software Architecture

The work on patterns is closely related to much other work in software architecture, object-oriented or procedural analysis, design and programming.

Patterns build on the immense practical experience in software development gathered by designers and programmers over the last three to four decades. None of the patterns described is artificially constructed; they have evolved over time. Software developers recognised that particular solutions solved a problem better than others, and so they reused these solutions again and again. Some of the patterns described have existed for a long time. For example, the Pipes and Filters pattern has been known since the 1960s, and the Model-View-Controller pattern since the late 1970's [77]. Without this practical experience, no patterns would exist.

Patterns are not dedicated solely to object technology but also build explicitly on the many principles that have been developed for structured programming. Many programming principles that were developed in the 1970's form the foundation of the patterns known
today. The relationships of patterns to these principles are discussed in Section 2.2.3, Enabling Techniques for Software Architecture.

Another objective of patterns is to build software systems with predictable non-functional properties. Patterns therefore also build on the principles for developing software for and with reuse, design for change and so on. The relationships of patterns to important non-functional properties are also discussed for software systems in Section 2.2.4, Non-functional Properties of Software Architecture.

2.4.1 Patterns as Mental Building-Blocks

It is has already been established that patterns are useful mental building-blocks for dealing with limited and specific design aspects when developing a software system.

Patterns therefore address an important objective of software architecture – the construction of specific software architectures with defined properties.

General techniques for software architecture, such as guidelines on using object-oriented features, e.g. inheritance and polymorphism, do not address the solution of specific problems. Most of the existing analysis and design methods also fail at this level. They only provide general techniques for building software, for example ‘separating policy from implementation’ [117]. The creation of specific architectures is still based on intuition and experience.

Patterns effectively complement these general problem-independent architectural techniques with specific problem-oriented ones. Note that patterns do not make existing approaches to software architecture obsolete – instead, they fill a gap that is not covered by existing techniques.

2.4.2 Patterns versus Methods

A good pattern description also includes guidelines for its implementation that can be considered as a micro-method for creating the solution to a specific problem. These
micro-methods complement general but problem-independent analysis and design methods, such as Booch [15] and Object Modelling Technique [117], by providing methodological steps for solving concrete recurring problems in software development.

2.4.3 Instantiating Patterns

Another aspect that arises from the integration of patterns with software architecture is a paradigm for instantiating them. Many current software patterns have a distinctly object-oriented flavour [5]. It is tempting to conclude that the only way a pattern can be implemented effectively is in an object-oriented programming language. However, such conclusions are false.

On one hand, it is true that many patterns use object-oriented techniques such as polymorphism and inheritance. Examples of such patterns are the Strategy Pattern [52] and the Proxy Pattern [21].

On the other hand, object-oriented features are not essential for implementing these patterns. Proxy, for example, loses only a small fraction of its elegance by giving up inheritance. Strategy can be implemented in C by using function pointers instead of polymorphism and inheritance.

At the design level, most patterns only require certain abstraction facilities of a programming language, such as modules or data abstraction. You can therefore implement patterns with almost any programming paradigm and in almost any programming language. In addition, every programming language has specific patterns of its own, the idioms of that language. They capture existing programming experience with the language and define a programming style for it.

2.4.4 Methodologies

A common question asked about patterns is how they relate to existing analysis and design methods such as the Booch Method [15], Coad/Yourdon [28], Object Modelling Technique
Before there were patterns these methods were heralded as the solution to 'the design problem'. More recently, people have become increasingly critical of methodologies, or certainly towards the idea of relying on them too heavily. To summarise Michael Jackson writing in [73]:

Failure to focus on problems has harmed many projects. But it has caused even more harm to the evolution of development METHOD. Because problems may not be talked about they may not be analysed or classified. So it is easy to slip into the childish belief that there can be universal development methods, suitable for solving all development problems. Methods are expected to be panaceas: medicines that cure all diseases. This cannot be. It's a good rule of thumb that the value of a method is inversely proportional to its generality. A method for solving all problems can give you very little help with any particular problem.

It is not hard to foresee that people will voice similar complaints about patterns if expectations are not limited. James Coplien wrote [31, 32]:

“One fear for patterns is that designers will look to them first for their design solutions. This happened when the object paradigm was young, too. Many design problems can be solved by well-known paradigms, and good designers should carry those in their toolkits—not always try to use the most recent tools, even if they are the most powerful.”

In the recent past, we've tried to use object tools to solve everything. Patterns have been taken outside pedestrian object-design methods, often into structures that are handled well by no existing paradigm, that's where patterns shine: the dark corners of design. Patterns cover only small holes in the design space: the broader design space lends itself well to the common techniques of well-known paradigms, and those paradigms should be sought to use where they fit.

By curtailing the expectations, both patterns and methodologies can be used to advantage. Methodologies provide many useful steps and guidelines for constructing high-quality software. The implementation sections of the patterns loosely follow these steps, adapted to the needs of the specific problems the patterns address. In addition, these methods define
an overall process for software development that can be adapted and extended to integrate the use of patterns. Patterns complement the existing analysis and design methods with a set of concrete techniques for solving very specific but recurring design problems.

Bear in mind that neither patterns, nor methodologies nor their combination will provide the 'Yellow Brick Road' to a fine architecture. There will be plenty of design problems left that have to be solved by the software architect alone.

2.4.5 Software Processes

The blanket application of methodologies tends to cause even worse problems for software processes. How much harm has been done by enforcing the waterfall process in projects? A defined process has its benefits, but becomes a liability when it causes organisational overheads or enforces a way of working that doesn't fit with a project's goals. How can a process that tries to fit all projects also fit one specific project's special circumstances? How can the crucial insights gained during implementation be used to redesign defined parts of the system if returning to the design stage is not allowed? No methodology or process should be allowed to dictate strictly how design and implementation is to proceed.

How can patterns help here? Patterns can be integrated into an incremental delivery process that gets rid of the strict separation of development phases. Object-oriented analysis and design methodologies tend to blur the boundaries between phases. It can contribute towards making this incremental and sometimes cyclic way of working more predictable. For example, if patterns help to produce better and more stable designs, the number of cycles can be limited through the phases and restrict redesign to well-defined parts of the system.

It is often asked at what point of development should patterns be used: during analysis, high- or low-level design, or even during implementation? There is no single correct answer, but a rule of thumb is that the high-level architectural patterns should be used earlier than medium-level design patterns, which are themselves used before idioms.
2.5 Summary

Software architecture is an important vehicle for constructing high-quality software. There are several techniques, methods, and processes for software architecture construction. Usually, architectures are represented abstractly as box-and-line diagrams, together with accompanying prose that explains the meanings behind the symbols and provides some rationale for the specific choice of components and interactions.

The design of a software architecture is more than a simple activity within a limited scope. It comprises the technical, methodological and process aspects of software engineering. It explicitly addresses the needs of productive software development and maintenance, and has a great impact on the final quality of a software system.

Although non-functional properties are very important in software architecture, their achievement is hard to measure. The detailed criteria a software architecture must satisfy has only been specified for a few such properties, for example reusability and changeability [76]. For this reason, estimating the degree to which a software architecture achieves a given non-functional property is still mainly based on the experience of software engineers.

Patterns fit in well with existing approaches to software development:

> They explicitly build on enabling techniques for constructing well-defined software systems, such as information hiding and the separation of interface and implementation.

> They complement existing problem-independent software development processes and methods with guidelines for solving specific recurring design and implementation problems.

Patterns also make an important contribution to the benefits that can be gained from software architecture:

> They help with the recognition of common paradigms, so that high-level relationships between software systems can be understood and new applications
built as variations on old systems.

➤ They stress the importance of non-functional properties, such as changeability and reliability.

➤ They provide support for finding an appropriate architecture for the software system under development.

➤ They provide support for making principled choices among design alternatives.

➤ They help with the analysis and description of high-level properties of complex software systems.

➤ They provide support for change and evolution of software systems.

Ideally, pattern categories help in the initial selection of potentially useful patterns for a given design problem. They are related to important software development activities. Architectural patterns can be used at the beginning of coarse-grained design, design patterns during the whole design phase as well as idioms during the implementation phase. See [27, 121, 122, 88, 112, 124] for more information on recent patterns.

Architectural styles are very similar to architectural patterns. In fact every architectural style can be described as an architectural pattern. For example, the Multi-phase architectural style corresponds to the Pipes and Filters pattern. On the other hand, architectural styles differ from patterns in several important respects:

➤ Architectural styles only describe the overall structural frameworks for applications. Patterns for software architecture, however, exist in various ranges of scale, beginning with patterns for defining the basic structure of an application (architectural patterns) and ending with patterns that describe how to implement a particular design issue in a given programming language (idioms).

➤ Architectural styles are independent of each other, but a pattern depends on the smaller patterns it contains, on the patterns with which it interacts and on the larger patterns in which it is contained [1].
Patterns are more problem-oriented than architectural styles. Architectural styles express design techniques from a viewpoint that is independent of an actual design situation. A pattern expresses a very specific recurring design problem and presents a solution to it, all from the viewpoint of the context in which the problem arises.
Chapter 3  Integration Approach

A serious concern in the construction of large software systems is integration: the process by which multiple software modules (programs, subprograms, collections of subprograms, etc.) are made to cooperate [8].

This chapter is concerned with software integration and its impact on software projects. It contains a review and a taxonomy of integration concepts as applied to software systems. It identifies pertinent technical integration issues and review and classify integration models. It argues that integration is part of the design activity and illustrates existing best integration practice for project management and engineering of large software systems.

3.1  A Taxonomy of Integration Concepts

3.1.1  Integration Concepts

*Systems Integration* is the practice of combining the functions of a set of subsystems, be it software, hardware or both, to produce a single, unified system that satisfies some need of an organisation [80]. The components of systems integration typically are [99]:

- Software subsystems (applications, system, custom).
- Hardware subsystems (single/multi processor(s)).

*Software integration* is the practice of assembling a set of software components/subsystems to produce a single, unified software system that supports some need of an organisation [141].

Typical software integration components are applications, reusable modules, bespoke modules, Commercial Of-the-Shelf systems (COTS) and Government Of-the-Shelf systems (GOTS).

The term *tool integration* is virtually exclusively used in the literature to denote Computer
Aided Software Engineering (CASE) tool integration. Therefore it is defined as follows [23]:

*Tool Integration* is the practice of combining a set of software development tools to produce a single, unified software development environment.

Typical tool integration components are analysis tools, design and coding tools, requirements traceability tools, configuration management tools and so on.

To summarise then, tool integration is a form of software integration which in turn is a form of systems integration, as shown in Figure 3.1.

![Figure 3.1 A Hierarchy of Integration](image)

### 3.1.2 Granularity of Integration

As well as identifying the true types of integration, it is useful to draw a distinction between the nature of integration depending on the granularity of the components involved.

*Fine grain integration*, or *integration in the small*, typically involves software components without independent functionality. These components can only function as part of the integrated system. Examples are program functions, procedures and packages. The granularity of the components is important because it identifies the "glue" used to put them
together. In the case of fine grained integration, the glue is provided by either programming language constructs or the compiler.

On the other hand, coarse grained integration, or integration in the large, normally would involve software or hardware components with well defined, independent functionality. Such components can function independently of the integrated system and might include applications software, reusable modules, bespoke modules, COTS, GOTS and so on. The glue in the case of coarse grained integration is much more varied and open ended than is the case with fine grain integration. This is where integration glue becomes much more of an interesting challenge.

In summary, systems integration aims to support some organisational processes through the supply of some services enabled by the existence of certain integration mechanisms. This view of integration is shown in Figure 3.2. In the rest of this chapter, it is concentrated on coarse grained software integration.

Figure 3.2 Integration and Supporting Notions
3.2 Conceptual Models of Integration

Most of the existing work on modelling coarse grained software integration concepts seems to have originated from the CASE community [19]. CASE researchers have sought a conceptual framework through which integration issues can be more easily understood and effectively communicated.

In this section, the CASE approach is generalised to integration concepts relevant to coarse grained software integration.

Four detailed conceptual models have been identified:

- Integration as a set of attributes.
- Integration as goals for the relationships between components.
- Integration based on a central repository through which components share data.
- Multilevel integration.

To a certain extent, these approaches are based on different notions of what integration actually is. This section will describe each of the four approaches and conclude with a conceptual model best suited to the notion of integration as defined in the previous section.

3.2.1 Integration as a set of Attributes

Wasserman [143] views integration as a set of attributes. These attributes can be used as a means of evaluating, comparing and planning the process of integration. Essentially, this approach allows various aspects of integration to be examined separately through a set of orthogonal dimensions.

Wasserman proposed the following five attributes:

- Platform integration is the set of system services that provide network and operating systems transparency to the components.
Presentation integration refers to the consistency and sophistication of the user interface.

Data integration characterises data sharing between components, be it through a shared repository (database) or files and interprocess communication.

Control integration relates to inter-component communication.

Process integration characterises the extent to which the integrated system supports a user process in a well defined way.

The essential idea here is that any product of system integration can be evaluated in each of these dimensions. For example, in the data integration dimension, one system may use a filestore while another uses a database. These systems would have different data integration attributes. So Wasserman's approach allows some measure of comparison of different systems in terms of their integration approach. These ideas can be easily extended to apply to proposed system designs as well as finalised systems and may be quite useful in that context. Figure 3.3 illustrates three of the integration dimensions proposed by Wasserman.

As a mechanism of evaluating an integrated system, Wasserman's model is somewhat crude since the axes indicate sophistication of integration mechanisms rather than integration quality; more does not necessarily mean better. The five attributes themselves may be debated since the separation is not as clear as Wasserman suggests; for example there is a close and complex relationship between data and control integration. The true value of this model may well lie in identifying these different aspects which can thus be considered and acted upon during the design of an integrated system.
3.2.2 Integration as a Relationship

The approach of Thomas and Nejmeh [139] treats integration not as a property of a component, but as a property of the relationship between components. In particular, they are concerned with defining a set of goals between the components of an integrated system. By doing this, attention is focused on the goals of integration and on the information required to decide whether these goals have been reached. This conceptual model is an improvement on Wasserman's approach since now not only is known which integration aspects to watch out for, but what information it is needed in order to decide whether the integration goals have been met.

Thomas and Nejmeh identify several types of intercomponent relationships [19]:

Figure 3.3 Three Integration Dimensions
- Component to component relationships which expand on Wasserman's dimensions of integration by discussing the ways in which individual components interrelate.

- Component to framework relationship which defines the manner in which each component utilises the framework (e.g. a database or an operating system) services.

- Component to process relationship which captures how well each component supports the objectives of the integrated system. The relationship may be expressed in terms of a component's support for an individual step within the software life cycle (e.g. requirements definition), and its support for multiple steps (e.g. the requirements, design and coding steps).

![Diagram](image)

**Figure 3.4 Integration Relationships between Environment Components**

This aspect of the model is illustrated in Figure 3.4. Thomas and Nejmeh have enriched Wasserman's model in their efforts to characterise component to component integration. They are particularly concerned with the following relationships:
Figure 3.5 shows an entity relationship diagram depicting a component, the four relationships and the properties of each relationship.

Figure 3.5 ER Diagram of a Single Component, Four Relationships and Their Properties

In addition to the elaboration of integration relationships and their properties, Thomas and Nejmeh draw a distinction between two different viewpoints in the discussion of integration; the system user's and the system builder's. This is a useful distinction since it highlights the difference between the user's perspective (is this system well integrated?) and the builder's perspective (how easy is this system to integrate?). For example, there may be a consistent and intuitive user interface for interaction with all system components.
On the other hand, a well integrated system is not necessarily easily integrable with respect to how easy it is for the system builders to assemble the system, tune it for particular needs, and replace one component with another. These two views of integration are independent; there is no guarantee that a system that is easy to assemble is enjoyable and productive to use and vice versa.

The Thomas/Nejmeh integration model is useful to users, component evaluators, component developers and framework technology builders. However it has two important shortcomings. First, although they identify the goals of integration, they do not propose a way for achieving them, what dependencies exist between them and what trade offs must be made. Second, a component is not equivalent to an end user service. This divorces the discussion of component to component relationships from the reality of system use which is the final indicator of success or failure.

### 3.2.3 Repository based Integration

Repository based integration views the issue primarily from the data sharing point of view. In fact there is a pervasive belief amongst many people that a repository of some sort should be the primary means of component integration [19].

A repository has four characteristics:

- **A data storage mechanism.** These range from simple file systems to complex databases possibly distributed over a number of sites. The data storage mechanism is responsible for the safekeeping of persistent instances of data. Frequently used examples of data storage mechanisms are UNIX file systems and relational databases. More advanced instances of the mechanism, such as object-oriented databases and entity-relational object storage systems (such as PCTE [109]) are rarely used.

- **An interface to persistent data.** There may be multiple interfaces to the same data or one interface may provide access to a number of different data storage
mechanisms. The prevalent interfaces are SQL and UNIX (Posix) [70] standards. In addition a number of bodies such as the IEEE, ANSI, ISO and the ECMA all have efforts to define new interfaces to facilitate CASE tool portability across different CASE environments.

➢ *A set of schemata or information models.* A schema is a description of stored data and typically records information about the different types of data in the repository and their relationships. There have been a number of attempts to define generic information models and there still is a great deal of research in the area mainly along the lines of "enterprise modelling and integration". Extensive efforts in this area include the IBM AD/Cycle Information Model and ISO Information Resource Dictionary System (IRDS) [72].

➢ *A set of operations on the data.* This determines how the data in the repository are accessed, by whom and at what frequency. This information is critical to tuning the repository for performance.

It is emphasised that a repository is not in itself a guarantor of integration; it is a mechanism which facilitates it. In order for the mechanism to be used successfully to integrate a system, the components must first come to an agreement on the storage mechanism, the interface, the schema and the operations.

### 3.2.4 Multilevel Integration

Brown et al. [19] have proposed a multilevel model of integration which combines most of the features of the previous models in a meaningful assembly. They distinguish three levels as shown in Figure 3.6.
The central level consists of the services available to system end users. The bottom level consists of the mechanisms that implement the end user services. The top level of interest is that of the process which encodes the set of goals and constraints of the context in which the end user services will be used to meet some objective of an organisation.

- The processes level characterises the use of end user services in the context of a particular operating scenario, in other words, the activities this system supports. Integration at this level can be regarded as a kind of process specification of how the system will be used in the field. This specification may be in terms of:

  - Goals such as deliverables, improvement or measurement, and
  - Constraints due to the life cycle used, the schedule involved and even the politics of operational system use.

- The end user services level corresponds to an abstract description of the functionality that the system offers to the end user. So integration at this level delineates what the component assembly does. These services may include amongst others:

  - Mission related services such as detection and tracking of targets.
  - System management services such as planning, estimating and tracking mission activities.
The mechanisms level includes the architectural concerns and the technology components that will implement the integrated system. So this level describes how the various system components are connected. Concerns here might be:

Architecture aspects such as internal component structure (e.g. client/server) and data management structure (e.g. data dictionary, database).

Technology related aspects such as operating system interfaces and specific integration mechanisms and standards used to connect the various components (e.g. RDBMS, X, PCTE).

The multilevel model, although originally devised for application in CASE tool integration, is clearly capable of generalisation and is applicable to coarse grained software integration. The value of the model comes from systematising many previously ad hoc aspects of integration. It provides a powerful conceptual framework for relating the functionality required from an integrated system to the integration technology that realises it.

This is the preferred model of integration because it encourages and supports integration as a design activity rather than an add-on activity after system development.

3.3 Integration Strategies

An integration strategy determines the way in which an integrated system is to be assembled. Choosing a suitable strategy depends on the type of system under consideration, in particular whether it is a legacy system or a new procurement.

3.3.1 Strategies for Legacy Systems

Hanratty [59] distinguishes three main integration strategies for legacy systems:

- Grafting which involves grafting a function on to an existing system in much the same way that a limb is grafted on to a tree.

- Substitution which implies a more open systems approach in which one
component is removed and another replaces it. The assumption is that some standardised interface permits this substitution.

➢ A Hybrid approach involving both grafting and substitution.

3.3.2 Strategies for New Procurements

For brand new developments, an integration architecture such as a coalition or a federation can be used to hopefully achieve an open system with all the resulting advantages. Therefore, the integration architecture models of either coalitions or federations can be used. Of course having a goal architecture for a new system does not remove the need for an integration strategy. Assuming that the system components are available in a suitable form, there are a number of ways of integrating them together:

➢ Top down integration is used by the Cleanroom Software engineering approach [93]. The strategy is that one starts with the topmost element which is integrated with stubs representing all other components. As integration proceeds, the stubs are gradually replaced by the real components until the entire system has been built.

➢ Bottom up integration starts by assembling together the conceptual building blocks of the system and proceeds by conservatively extending this initial core system.

➢ Sandwich integration is recommended by Myers [97]. This is a compromise between the bottom up and top down strategies. Top down and bottom up integration begin at the same time, integrating the system from both the top and the bottom, eventually meeting somewhere in the middle.

➢ Big bang integration involves putting together all the components in one step and then testing them for conformance to the system specification.

➢ A combination of the top down, bottom up and big bang strategies is proposed by Beizer [13]. He advocates that small assemblies be integrated by bottom up techniques, control structures be put together by shallow top down and the core
system ("backbone") be big bangaed together.

> **Incremental integration** is described by Deutsch [38] and involves introducing components one at a time with the system tested after each component is integrated.

> **Phased integration**, also described by Deutsch [38], requires that components at the next level are integrated all at once and the system be tested before integrating all components at the following level.

It is important to note that all the above approaches are focused on testing of the integrated system without any discernible concern with the whole integration lifecycle as discussed in the introductory section of this chapter. It seems that there is little guidance on broad integration strategies in the literature, the notable exception being the efforts of the SEI group [19]. But even the above integration/testing strategies do not meet with universal approval by the software engineering community. An empirical study by Solheim and Rowland [129] contains some interesting comparisons of these approaches. Their results indicate that top down integration strategies are most effective in terms of defect correction, but also exercise the greatest number of components. Bottom up strategies are generally least effective at correcting defects but exercise relatively few components. They also indicate that the top down and big bang strategies produce the most reliable systems, and the bottom up strategies produce the least reliable systems. Systems integrated by the sandwich strategy are moderately reliable by comparison. Their results do not support a preference for either phased or incremental strategies. It is also true that the extent of success of an integration strategy also depends on what is being integrated (e.g. software elements, tools etc.). The literature does not seem to offer any guidance on this point.

### 3.4 Summary

Integration is a complex task involving a number of trade-offs. Like other software qualities such as robustness and portability, answers about "good" integration are neither obvious nor simple; integration needs to be planned for and designed for.
For any given integration effort, it is needed to identify the goals of the project in order to
discover the ways in which these goals become articulated within a defined process.
Implicit and explicit organisational and project goals are manifested in the process that is
used for developing software, together with the specific needs for integration. Different
goal and process combinations will have different implications for integration purposes.
These differences require different integration solutions.

The solution selected by an organisation must be tempered with the realities of the
multitude of constraints guiding the successful adoption of that solution. Factors such as
available budget, receptiveness to new technology and migration strategy for current tools
and system will all need to be considered. The three level model of use services, integration
mechanisms and process can be used as a reference point for evaluating alternative
decisions and trade-offs.
Chapter 4  Proposed Approach: System Pattern

An important criterion for the success of patterns is how well they meet the objectives of software engineering [68]. Patterns must support the development of complex, large-scale systems. They must also support effective industrial software production, otherwise they remain just an interesting intellectual concept, but useless for constructing software.

4.1 Pattern for a System

Patterns are used to solve very specific but recurring problems. These problems may be architectural problems, design problems, idioms problems or integration problems. All the problems reduce to development problems. For examples, the Model-View-Controller [77] pattern, the Pipes and Filters [126] pattern, the Layers [137] pattern and the Microkernel [21] pattern solve architectural problems; the Client-Dispatcher-Server [21] pattern and Proxy [53] pattern solve the design problems; the Counted Body [32] pattern solves the idioms problems; the Integration Adapter [88] pattern and Integration Messenger [88] pattern solve the integration problems. However, the following factors indicate why such patterns are not sufficient:

➢ No pattern can solve all these problems.
➢ No pattern can be used in three distinct levels: architecture, design and idiom.
➢ No pattern can provide a framework to construct an application system.

In this section, it is to define a unified system pattern to solve recurring development problems and present a framework for its solution.

4.1.1 Three Levels: Architecture, Design and Idiom

A closer look at existing patterns reveals that they cover various ranges of scale and
abstraction. Some patterns help in structuring a software system into subsystems. Other patterns support the refinement of subsystems and components, or of the relationships between them. Further patterns help in implementing particular design aspects in a specific programming language. Patterns also range from domain-independent ones, such as those for decoupling interacting components, to patterns addressing domain-specific aspects such as transaction policies in business applications, or call routing in telecommunications.

To derive a classification, patterns are grouped into three levels:

- **Architectural level** expresses a fundamental structural organisation schema for software systems. It provides a set of predefined subsystems, specifies their responsibilities, and includes rules and guidelines for organising the relationships between them.

- **Design level** provides a scheme for refining the subsystems or components of a software system, or the relationships between them.

- **Idiom level** describes how to implement particular aspects of components or the relationships between them using the features of the given language.

A system pattern covers from architectural level to idiom level. Figure 4.3 shows the system pattern range.

![Figure 4.1 The System Pattern Range](image-url)
4.1.2 System Pattern Definition

A system pattern describes a particular recurring development problem that arises in specific software system contexts, and presents a well-proven generic framework for its solution. The solution framework is specified by describing its constituent components, their responsibilities and relationships, the ways in which they collaborate, the range from architectural level to idiom level and the method to their construction.

4.2 Motivation for One System Pattern

4.2.1 A Motivating Scenario

It is necessary for a sophisticated network application system to be built for organisations, such as enterprises, governments, universities, etc. Such a system should provide the following capabilities:

- Support an internal e-mail system for the organisation with conditional sending and receiving, which is different from a normal email system.
- In order for the enterprise to operate efficiently, it is required that e-management of workflow by means of both Internet and intranet be implemented.
- Every department needs to set up its resources catalogue respectively in the system for the sake of distributing shared information, and supporting exclusive information for special departments and individuals.
- Leaders or decision-makers need rapid and effective access to their organisation's existing information database to assist their decision-making. Systems should be able to support the comprehensive capability to access these databases [145, 146].
- All organisation applications cooperate and interoperate in an integrated platform with a united portal.
- The extension of system functions is transparent. Namely system functions can be
extended and distributed to users according to their needs at any time.

- Applications of the system must be safe. On one hand, information transfer must be secure. On the other hand, all resources usage ought to be authorised and authenticated.

- Application systems can be integrated with other resources, such as organisation websites and other common resources [83].

Normally, it is possible to build such a system as above for organisations. The first prerequisite is that an existing mail system will be set up for supporting information communication within an organisation. Secondly, the organisation's websites and information query systems will be built based on a mature web system. Thirdly, by introducing a workflow-supporting application system, the organisation's management of workflow will be based on the network. Fourthly, using various development tools, business information integration will be implemented gradually. However, the following complexities could arise:

- Complexity of integration. This includes issues such as:
  1. How to integrate the email and workflow systems, the authorisation and sharing of information, and how to control across those two different systems;
  2. How to access business information and web queries through the email system;
  3. How to distribute the integrated information by means of the Web system.

- Complexity of security—there are particular secure mechanisms for each of the email, workflow, Web and business systems. The security of the entire application system is dependent on all of those mechanisms. As a result, any security exception can influence the whole application system.

- Complexity of usage—users access different systems with various approaches and different identifications, which causes difficulty for user training and marketing penetration.
It is the aim of the research for applications to be integrated on a uniform platform based on a unified system pattern.

4.2.2 Recurring Development Problems

The data on the Internet is multifarious and in various formats, such as the database-oriented formats called structured data, and the file-oriented ones called non-structured data. All this data is regarded as resources, and sometimes a set of some relevant data is called as a kind of resource, for example, user resources, website resources, template resources and workflow resources.

In order to be able to integrate and manage those resources, there are some constraints as follows:

1) Resources organisation mode: involving how to classify and represent the resources.

2) Resources authorisation mode: involving authorisation management and resources access control.

3) Resources storage mode: Involving local resources storage strategy and the distributed resources links or copies.

4) Resources management approach: it is required to provide an efficient tool for resources management, involving database connection, authorisation, resources properties configuration and so on.

Although resources may be integrated in an organisation, how to use the resources efficiently is a problem for users. Then, it is necessary to add more constraints such as those listed below:

5) Information transfer: for messages sending, copying and moving, etc. among resources.

6) Resources sharing: for file uploading, downloading, distribution and common data sharing.
7) Resources searching: for the full-text-search and conditional search of files and databases.

8) Workflow: On one hand, following a pre-established workflow, client users are able to read and subscribe, deliver, and distribute e-documents conditionally. On the other hand, client users can flexibly design custom business workflows by theirselves. In other words, with workflow management mechanism, system supports a full-scale workflow management from design to being on the move.

9) Database access: Transparent database access, including connection, query and maintenance for various databases is possible.

10) Authority control: The configuration of authority should be flexible whereas each operation and access to resources must be authorised in order to prevent illegal operation.

11) Remote calling: Commands and component-calling can be executed on the server, and results are returned to the caller.

12) Efficiency of transmission: Since there is a vast body of integrated information, the efficiency of resource access is very important, especially for database queries and large-scale file transmission. In order to improve the efficiency, proper compression algorithms are needed.

13) Security authentication: Pivotal access must be encrypted to promote security, such as authentication and system registration. Setting up a related Certification Authority System (CA) can also raise the system security level.

The extension of these functionalities should be a focus for action after the above services have been considered. Because of frequently changing user-requirements, a stable system architecture is of value to extend system functionality and application by adding the following constraints.

14) Components organisation: components are classified as atomic components,
application components and interface components. Atomic components are basic components, which are involved in most basic applications, such as interface design, data access, data edit, data control, report design, printing, chart display and file operation etc. Those components, a part of the total architecture, are relatively stable and can be upgraded along with the system’s upgrade. Application components, which are application modules based upon atomic components, can also be adjusted with requirement changes. Application components are usually used to implement specific system functions. Interface components encapsulate a specific application’s access to core services. Such components can be deployed at any position where core services are required.

Furthermore, atomic components and application components are classified as client-side components and server-side components. The former runs on the client to communicate with users whilst the latter runs on the server to implement some specific business logic and functions that are irrelevant to the user interface.

15) Component repository: This is used for component storage. If components are regarded as resources, the component repository can be regarded as a specific resources repository.

16) Component assembly: Application components are assembled from atomic components using a specific programming language. With the same programming language, new application components can be assembled using the existing application components. Therefore, this kind of programming language lies at the core of both component assembly and system extension.

17) Component execution: Components that execute on the server are different from those that execute on the client. Both of them should be installed using a run-time version of the core language, namely Virtual Machine (VM). But the server-side virtual machine does not support interface operation. Moreover, the VM is also the core of the software system and upgrades along with that system’s upgrade.
The kernel part of the solution has been discussed. Then essential tools will be provided for the integration of every part. The constraints related to this aspect are shown below:

18) Resources management tools are used to create, delete, rename, assign authority and configure properties, etc.

19) Workflow design tools support flexible and visual design of various business workflows and conditional control on flow direction and each node.

20) Database connection tools allow connections with various databases and realising data transfer between different databases, as well as defining the names for the connection interface.

21) Application development tool: This is the visual development environment, which is used to design application components. All atomic components can be integrated in the tool. It adopts one or multiple object-oriented programming languages as the core languages for development. The application components can be delivered to the component repository as they are developed.

4.2.3 Fundamental Solution Principle

The solution for solving recurring problems that are presented in previous section has been divided into three steps:

The primary step is to make certain how to organise and store resources and set up the resources repository.

The next step is to define a protocol for resources access and set up core server, which provides multiple services based on the protocol.

The third step is to set up the client-side virtual machine and the server-side virtual machine, which is the key to application extension and component assembly. The implementation of the virtual machine depends on the core programming language that is defined and most suitable for the adoption of multiple prevailing programming languages.
The difficulty is that these languages should be able to support common atomic components. Similarly, the suitable and abundant set of atomic components is the basic reason for the virtual machine being built.

4.3 The System Pattern: Tri-Integration Pattern

4.3.1 Definition

A complex software system can be treated as an integrated system. To improve the conceptual models of integration (see section 3.2), Service, Component and Data are considered as key integration elements.

Services are well-defined, self-contained and universally available business functions that respond to service requests from "consumers." They are available across the enterprise. Monolithic business applications such as Enterprise Resource Planning (ERP) systems dissolve into a set of self-contained services that perform specific business functions. These services can be invoked over standard protocols to ensure their availability across the enterprise and beyond.

Components are typically specified to show the relevant functional and non-functional properties of a software system. A component is an encapsulated part of a software system. A component has an interface. Components serve as the building blocks for the structure of a system. At programming-language level, components may be represented as modules, classes, objects or a set of related functions.

Data is multifarious and in various formats on application systems, such as the database-oriented formats, the file-oriented ones or even legacy data stores. Normally, data is the core of an application system, most of functions or operations are based on it.

The Tri-Integration Pattern presents a solution to service integration, component integration and data integration of a software system. It uses a unified framework and application method to build and deploy an extendable network software system.
4.3.2 Context

The context of Tri-Integration is to build an integrated, secure and extendable network software system.

Service integration of the pattern presents a mechanism that can integrate various services into one application system. These services can be invoked over standard protocols to ensure their availability across the enterprise and beyond.

Components integration involves integrating business logic components, presentation logic components and data access components into an enterprise application system. Components integration supports the development and deployment of application components using well-designed construction methods.

Data integration involves integrating existing data living in different enterprise systems, and it often occurs when an enterprise relies on multiple types of database system. For example, some database systems may be relational, others may be hierarchical or based on objects and still others may be file-based or even legacy data stores. Data integration involves not only integrating different data systems, but it also entails integrating different informational or data models.

4.3.2.1 Service Integration Mechanisms

Service integration mechanisms emphasise a common communication approach between applications and a shared understanding of the events that initiate interaction.

Although data sharing can provide a high level of control and coordination between applications, it also imposes a significant overhead on the system, both because of poor performance of existing database mechanisms when used this way, and because of the necessary agreement required between the applications to define a common data schema. An alternative approach is focused on services rather than shared data. In the service integration approach, a system is viewed as a collection of services provided by different
applications. Actions carried out by one application are announced to others via control signals. The receiving applications can then decide if they need to respond in some way. This form of communication is known as message passing (a visualisation of which is shown in Figure 4.2). In the remainder of this subsection, message passing is briefly described and a number of existing systems is reviewed.

![Service Integration Mechanism](image)

**Figure 4.2 Service Integration through Message Passing**

- **The Message Passing Approach**

In the message passing paradigm, applications communicate by passing messages informing other applications of their actions and requesting services from other applications. This requires a mechanism enabling communication as well as a protocol defining the acceptable communication patterns amongst applications.

The service integration mechanism shown in Figure 4.2 is responsible for message distribution and is termed the message server or message broker. Applications never communicate directly with each other. Instead they send messages to the server who is then responsible for selectively forwarding messages to appropriate applications and for initiating execution of an application to handle the current message if necessary.

- **Examples of the Message Passing Approach**

A number of message passing frameworks [116, 103, 41, 36, 22, 50] have been proposed
and built.

- Friendly Integrated Environment for Learning and Development (FIELD) is the origin of much of the work on the message passing approach with an initial implementation available as early as 1987 [116]. The two basic components of FIELD are a consistent graphical user interface as a front-end to all components and a simple message server Msg, implemented as a separate UNIX process, communicating with other processes via sockets.

- The Common Object Request Broker Architecture (CORBA) is intended to provide a common interface that allows applications to exchange information in an object-oriented environment [101, 102, 103]. CORBA allows transparent access to applications operating on multiple computing platforms by requiring that each platform should support a compliant Object Request Broker (ORB). CORBA is a specification for communications between clients requesting a service and objects providing a service.

- Object Linking and Embedding (OLE) is Microsoft's offering and provides users with the ability to embed items created by one application within data produced by another one [41]. OLE provides a client-server model of computing and focuses on communication and data sharing between applications on a single desktop.

### 4.3.2.2 Component Integration Mechanisms

Building an integrated system requires not only a conceptual model of what integration means, but also some practical guidance on how to go about constructing such a system. The structure of the integrated system is the first design choice that needs to be made. This structure defines the way that the components cooperate in order to achieve the objectives of the integrated system, and therefore plays a central role in the success or failure of the integration effort.

Two approaches to component integration are proposed; *component coalitions* and
component federations. These correspond to the notion of closed and open integration, respectively and essentially determine the approach of an organisation to open systems. In the remainder of this section the two approaches are described and the impact that the choice of structure has on the integration process for an organisation is discussed.

- **Coalitions Approach**

Component coalitions are also often referred to as “point-to-point” integration structures [19, 59] as well as “component-based” integration structures. Marty [92] characterises this approach as closed integration. The essence of it is that independently developed components such as tools and applications are integrated through a careful “hardwiring” of their interfaces as shown in Figure 4.3. Each such assembly of components requires a custom integration solution in order to achieve its objectives. The components maintain their independence as attested by a number of characteristics [96]:

- Each component maintains its own interface.
- Each component maintains independent control of its data and processing.
- Components may provide overlapping, even contradictory functionality or services.

![Figure 4.3 “Point-to-point” Integration](image-url)
Although component coalitions are extremely useful for many purposes, they have a number of key problems such as:

- **Lack of generality** since they can only be used for a fixed, well defined objective.
- **Lack of evolvability** since change in either components or system purpose cannot be accommodated within the hardwired solution.
- **Component egocentricism** as reflected in uneven service provision.

Because of the lack of standardisation and the emphasis on "no change" policies, component coalitions can provide an immediate, but partial solution to systems integration. This is already evident in the CASE integration world, where vendor coalitions abound (for instance the coalition of IDE's Software Through Pictures and CodeCenter with Frame Technology's FrameMaker).

- **Federations Approach**

Component federations refer to "integration through standards" [59] structures and open integration [92]. The key issue in this approach is the provision of a "plug and play" platform which can support a variety of components conforming to a set of agreed standards. In contrast to component coalitions, one does not set out to build a custom interface harness to enable integration. Instead, the emphasis is on interface and other standards. Components conforming to such standards can be assembled in a plug and play architecture in a straightforward way, as shown in Figure 4.4. An actual example of a component federation is the Eureka Software Factory reference architecture [45].
Although components can maintain their own data storage and access mechanisms, the platform provides specialised services to allow components to communicate actions and intentions as well as to share data. Typically, the procuring organisation maintains a degree of flexibility in choosing (or substituting) components to be integrated in the system.

Thus, in contrast to component coalitions, component federations can be:

- *General* since components can be used in more than one configuration.
- *Amenable to change* since components are not hardwired together.

The key issue here is that component federations and the associated standards support the notion of open systems which are defined as extensible collections of plug-compatible components and their standards. The open systems approach is advocated by the US DoD through their Technical Architecture for Information Systems Management (TAFIM) programme [59].

**Coalitions or Federations**

The issues involved in integration efforts include component integration, the development
of a supportive framework as well as identification of standards to allow component replacement. The primary technical differences between the component coalition and component federation approaches are determined by how these issues are addressed.

Table 4.1 Identified Low Level Integration Standards

<table>
<thead>
<tr>
<th>Function</th>
<th>Element</th>
</tr>
</thead>
<tbody>
<tr>
<td>Frameworks</td>
<td>Portable common tools environment (PCTE)</td>
</tr>
<tr>
<td></td>
<td>A tool integration standard (ATIS)</td>
</tr>
<tr>
<td>Operating systems</td>
<td>(Extended) POSIX</td>
</tr>
<tr>
<td>User interface</td>
<td>X windows</td>
</tr>
<tr>
<td></td>
<td>Motif</td>
</tr>
<tr>
<td></td>
<td>Open look</td>
</tr>
<tr>
<td>Database management</td>
<td>SQL</td>
</tr>
<tr>
<td></td>
<td>IRDS</td>
</tr>
<tr>
<td>Data interchange</td>
<td>Postscript</td>
</tr>
<tr>
<td></td>
<td>CDIF</td>
</tr>
<tr>
<td></td>
<td>ASCII</td>
</tr>
<tr>
<td></td>
<td>CGM</td>
</tr>
<tr>
<td></td>
<td>IGES &amp; PDES/STEP (ISO 10303)</td>
</tr>
<tr>
<td></td>
<td>SGML</td>
</tr>
<tr>
<td></td>
<td>ODA, ODIF</td>
</tr>
<tr>
<td>Network services</td>
<td>OSI</td>
</tr>
<tr>
<td></td>
<td>TFA (NFS)</td>
</tr>
<tr>
<td>Programming services</td>
<td>C</td>
</tr>
<tr>
<td></td>
<td>FORTRAN</td>
</tr>
<tr>
<td></td>
<td>COBOL</td>
</tr>
<tr>
<td></td>
<td>Ada</td>
</tr>
<tr>
<td></td>
<td>Pascal</td>
</tr>
</tbody>
</table>

Component coalitions provide a specific instance of tool-to-tool integration where sharing of services is tailored to the components in the coalition. Plug compatibility is not addressed or encouraged. Component federations on the other hand are supported by a growing number of standards [80, 96] (as shown in Table 4.1) and can offer greater flexibility as well as support for plug compatibility. Unfortunately, federations are just beginning to develop and in the short term, component coalitions will most likely provide a pragmatic link between the immediate demand for tight integration and the desire for more generalised and adaptable component federation solutions. In the long term, it is expected
that federations will be the preferred option and various governmental organisations are actively preparing for it. For instance, the US National Institute of Standards and Technology (NIST) has developed the Applications Portability Profile (APP), a family of related standards that can improve application portability and thus ease the integration of software developed by different organisations and/or based on a variety of hardware platforms [91, 80]. The US DoD has adopted a similar strategy which is already practised in the Defence Information Infrastructure (DII) Common Operating Environment (COE) effort [39, 133].

4.3.2.3 Data Integration Mechanisms

Data integration mechanisms emphasise storage and persistence, common data structures and shared understanding of these structures. They define the way that data is shared between applications. When applications share data, two issues must be addressed. First, an agreement needs to be established between applications concerning what data are stored, where and how they are accessed, that is data persistence. Second, the applications must share an understanding of the meaning of data, that is data semantics. These two concepts largely provide the backdrop for two classes of data integration mechanisms, each emphasising data persistence or data semantics. Below these two major data integration strategies are discussed and some actual examples of integrating mechanisms currently available are presented.

- Data persistence

There are two strategies supporting storage and sharing of persistent data:

- *Import/Export* where applications maintain separate databases for their unique data and sharing is accomplished by translating the internal data representation to an external form suitable for other components. Typically, data sharing occurs only in the form of data extraction (i.e. export only). Although this strategy is viable when a small number of applications are involved, maintaining consistency between related data becomes unmanageable with larger volumes of data and number of
applications.

- **Common data store** which includes proprietary data stores in a tightly integrated system from a single vendor, traditional databases used to accumulate data from multiple applications and common object bases developed explicitly to support the needs of system integration. This is the most general solution and a number of Object Management Systems (OMSs) have been developed recently, although they do not seem to be the preferred solution of integrated systems users [113], presumably because of issues such as performance, costs and lack of standards.

- **Data semantics**

  The meaning of shared data can be defined in one of two ways:

  - **A common schema** which identifies the types of item represented by the data, characteristics of these items and the types of relationship between items. Shared schemata are often referred to as information models since they focus on the meaning of data to be shared.

  - **Common data formats** which represent either an agreement to store all data in a specified format at all times or to extract a subset of the data and make it available at specific times. The focus here is conveying semantic information based on the structure of the data rather than the meaning of the data to a human observer.

  A problem both these approaches share is control of the evolution of the common schema or data format since vendor compromise and cooperation is required.

### 4.3.3 Structures

In contrast to integrating conceptual models [143, 139, 19] and some related patterns [21, 53], the Tri-Integration pattern defines eight participating structures (as shown in Table 4.2) which are derived from author's development practice and investigation into software architecture, software pattern and software integration.
Table 4.2 The Tri-Integration Pattern Structures

<table>
<thead>
<tr>
<th>Number</th>
<th>Structures</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Atomic components</td>
</tr>
<tr>
<td>2</td>
<td>Interface components</td>
</tr>
<tr>
<td>3</td>
<td>Application components</td>
</tr>
<tr>
<td>4</td>
<td>Resources</td>
</tr>
<tr>
<td>5</td>
<td>Resources repository</td>
</tr>
<tr>
<td>6</td>
<td>Core language</td>
</tr>
<tr>
<td>7</td>
<td>Virtual Machine</td>
</tr>
<tr>
<td>8</td>
<td>Core server</td>
</tr>
</tbody>
</table>

4.3.3.1 Description of Participating Structures

A CRC-card [12] is used for describing each participating structure.

Atomic components are the set of basic components, which are concerned with the basis of an application development.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Atomic components</td>
<td>None</td>
</tr>
<tr>
<td><strong>Responsibility</strong></td>
<td></td>
</tr>
<tr>
<td>• Provides interface design, data access, data edit, data control, report design, print, chart display and file operation etc</td>
<td></td>
</tr>
</tbody>
</table>

Interface Components encapsulate specific application access to core services. Such components can be deployed at any position where core services are required.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interface components</td>
<td>None</td>
</tr>
<tr>
<td><strong>Responsibility</strong></td>
<td></td>
</tr>
<tr>
<td>• Implement access to core server</td>
<td></td>
</tr>
<tr>
<td>• Implement access interface to core services</td>
<td></td>
</tr>
<tr>
<td>• Provide specific implementation differently according to the method of deployment.</td>
<td></td>
</tr>
</tbody>
</table>

Application Components are application modules based upon atomic components, which are used to implement specific system functions.
### Class Application components

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Implement specific system functions</td>
</tr>
<tr>
<td>● Being deployed according to requirements</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core Language</td>
</tr>
<tr>
<td>Interface components</td>
</tr>
<tr>
<td>Resources</td>
</tr>
</tbody>
</table>

**Resources** that are composed of structural data, non-structural data and functionality entities are the core of system management.

<table>
<thead>
<tr>
<th>Class Resources</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Structural data organisation and connection, which are managed by various relational databases within systems.</td>
</tr>
<tr>
<td>● Non-structural data organisation and storage, which collects and manages documents, web pages and multimedia files, etc.</td>
</tr>
<tr>
<td>● Functionality entities, with such dynamic characteristic as system users, variant sheets, schedules, website links, mail configuration, functional code, database queries and application components, etc</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Application components</td>
</tr>
<tr>
<td>Resources</td>
</tr>
</tbody>
</table>

The **Resources Repository** manages all resources within a system. The repository allows the control of resource usage through a central management mechanism.

| Class Resources repository |

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>● Resource definition, classification and storage</td>
</tr>
<tr>
<td>● Resources authorisation and access control</td>
</tr>
<tr>
<td>● Resources properties configuration</td>
</tr>
<tr>
<td>● Resources states track</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources</td>
</tr>
</tbody>
</table>
The Core Language is composed of one or more object-oriented programming languages for developing various applications closely related to the system.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core language</td>
<td>Atomic components</td>
</tr>
<tr>
<td></td>
<td>Application components</td>
</tr>
<tr>
<td></td>
<td>Interface components</td>
</tr>
<tr>
<td></td>
<td>Resources</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Supports object-oriented programming</td>
</tr>
<tr>
<td>• Integrates all atomic components</td>
</tr>
<tr>
<td>• Provides a mechanism to extend atomic components</td>
</tr>
<tr>
<td>• Develops application components</td>
</tr>
<tr>
<td>• Accesses resources by interface components.</td>
</tr>
</tbody>
</table>

The Virtual Machine is a run-time version of the core language and can be deployed according to requirements.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Virtual machine</td>
<td>Core Language</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Interprets and runs core language</td>
</tr>
<tr>
<td>• Provides specific implementation according to different deploying (client-side or server-side)</td>
</tr>
</tbody>
</table>

The Core Server is the core of resources management and system services, it is the implementation of MRTP and the key to the success of services integration.

<table>
<thead>
<tr>
<th>Class</th>
<th>Collaborator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core server</td>
<td>Resources repository</td>
</tr>
<tr>
<td></td>
<td>Interface components</td>
</tr>
<tr>
<td></td>
<td>Virtual machine</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Responsibility</th>
</tr>
</thead>
<tbody>
<tr>
<td>• Implements MRTP</td>
</tr>
<tr>
<td>• Implements resources management</td>
</tr>
<tr>
<td>• Provides system services</td>
</tr>
</tbody>
</table>

4.3.3.2 The Static Structure of the Pattern

An UML class diagram [20] is used for the static structure of the Tri-Integration pattern, as shown in Figure 4.5.
Figure 4.5 Static Structure of the Tri-Integration Pattern

4.3.4 Levels

Table 4.3 The Tri-Integration Pattern Levels

<table>
<thead>
<tr>
<th>Levels</th>
<th>Elements</th>
</tr>
</thead>
<tbody>
<tr>
<td>Architectural Level</td>
<td>Resource Organisation</td>
</tr>
<tr>
<td></td>
<td>Proposed Protocol</td>
</tr>
<tr>
<td></td>
<td>Core Server Structure</td>
</tr>
<tr>
<td></td>
<td>Core Language</td>
</tr>
<tr>
<td></td>
<td>Atomic Components Assembly</td>
</tr>
<tr>
<td></td>
<td>Virtual Machine &amp; Integrated Development Environment Structure</td>
</tr>
<tr>
<td>Design Level</td>
<td>Resource Design and Storage</td>
</tr>
<tr>
<td></td>
<td>Authorisation Strategies</td>
</tr>
<tr>
<td></td>
<td>Multipe Resources Transfer Protocol</td>
</tr>
<tr>
<td></td>
<td>Core Server Design</td>
</tr>
<tr>
<td></td>
<td>Workflow Control</td>
</tr>
<tr>
<td>Idiom Level</td>
<td>Resource Managing Idiom</td>
</tr>
<tr>
<td></td>
<td>Virtual Machine Environment Idioms</td>
</tr>
<tr>
<td></td>
<td>Alias Managing Idiom</td>
</tr>
<tr>
<td></td>
<td>Web Page Designing Idiom</td>
</tr>
<tr>
<td></td>
<td>Form Designing Idiom</td>
</tr>
<tr>
<td></td>
<td>Flow Designing Idiom</td>
</tr>
<tr>
<td></td>
<td>Sheet Designing Idiom</td>
</tr>
<tr>
<td></td>
<td>Client Exploring Idiom</td>
</tr>
</tbody>
</table>
According to the definition of system pattern and the structures of the Tri-Integration Pattern, the elements of the pattern can be divided further into three levels: Architectural level, Design level and Idioms. Table 4.3 shows the levels.

4.3.5 Dynamic Behaviours

The dynamic behaviour of the Tri-Integration Pattern depends on the requested application mode. This section focuses on the most relevant scenarios in the operation of an application system based on Tri-Integration pattern.

Scenario 1. Resources operation

a) A certain resource requests an operation.

b) After receiving the request, the interface components send it to the core server.

c) The core server authenticates the request to access related resources. If authority is insufficient, the core server returns the denied result to the requesting resource.

d) The core server executes the request through accessing the resources repository.

e) Interface components return the result to the requesting resource.
Figure 4.6 Resources Operation Scenario

Scenario 2. Workflow

a) A certain resource asks for the execution of a task according to the specific workflow.
b) Interface components send the specific workflow to core server.

c) The core server decides whether the workflow can be used according to the authenticating authority of the resource. If authority is not enough, core server returns the denied result to the requesting resource.

d) Interface components return such a decision to the requesting resource.

e) Resources execute related operations according to the request of the workflow node.

f) The results of the operations are sent to the core server according to specific flow direction or the appointed flow direction by interface components.

g) The core server establishes the flow repository in the resources repository or writes down the information of flow in the flow repository.

h) Core server executes the request.

i) Repeats step d).
Figure 4.7 Workflow Scenario

**Scenario 3. Application request**

a) A certain resource requests a service from application components.

b) After receiving the request, interface components send it to the core server.

c) The core server authenticates the authority of the resource. If authority is not enough, core server returns the denied result to the requesting resource.

d) The core server searches components in order to implement the request by
accessing the resources repository.

e) The core server calls the requested application components.

f) Virtual machine executes core language, and accesses related atomic components using the core language.

g) The operation result is returned to the core server.

h) Interface components return the result to the requesting resource.

Figure 4.8 Application Request Scenario
4.4 Summary

Patterns exist in various ranges of scale and abstraction, and cover many different and important areas of software development. Patterns are interwoven with each other and can be used to refine other, larger patterns, which can then be combined to solve more complex problems.

Adopting a System pattern is a big step forward in supporting systematic construction of high-quality software systems with defined properties at three levels. It addresses important aspects of software architecture and complement existing techniques and methods. It can be integrated with programming paradigms and implement them with almost any programming language. In a sense, a system pattern provides a mental method that helps in the construction of software to meet both functional and non-functional requirements of an application.

The Tri-Integration Pattern is a system pattern. It provides a pragmatic method, in a process-independent way to address the development problems that networked software systems must face. It documents existing design knowledge and help developers find appropriate solutions to development problems.

To show that the Tri-Integration Pattern helps build an application system rapidly and effectively, a software platform, Netcase Anywhere, based on the pattern has been constructed and applied successfully in a large number of enterprises. Further discussion on the Tri-Integration Pattern in terms of its three levels and the supporting software platform will be given in the following chapters.
Chapter 5  

Tri-Integration Pattern: 

Architectural Level

At the architectural level, a system pattern expresses fundamental structural organisation schemas for software systems. The Tri-Integration pattern as a system pattern provides a set of predefined subsystems, specify their responsibilities and include rules and guidelines for organising the relationships between them. Every development activity that follows is governed by this structure – for example, the detailed design of subsystems, the communication and collaboration between different parts of the system and its later extension.

5.1 Constructing Architectural Level Elements

The Tri-Integration pattern defines six elements at architectural level (as shown in Table 4.3):

➢ Resources Organisation
➢ Proposed Protocol
➢ Core Server Structure
➢ Core Language
➢ Atomic Components Assembly
➢ Virtual Machine & Integrated Development Environment Structures

5.1.1 Resources Organisation

A resource is composed of a resource description and a resource item. The resource description includes the name, type, signature, configuration and authority of the resource.
The resource item consists of *item description* and *item entity* wherein the former includes the information about the current item's mark, subject, time, flow attributes and authority, etc., and the latter includes the current item's text, attachment and application components, etc. It is permitted that a resource contains no resource item.

In order to distinguish different approaches to resource usage, resources are provided with an essential type. There are two types of resource:

- **Document resource:** this is composed of resource items, which are used to manage e-documents, such as email, archives, technical solutions, regulations and multimedia files, etc.

- **Functional resource:** this is used to implement specific functions, such as schedules, websites access, data queries, data maintenance and statistics, etc. Functions are defined within a resource description. There may be not any resource items for a functional resource.

Each resource may have as many sub-resources as are needed, which is very similar to the directory structure of an operating system. A resource is represented as follows:

<resource name>.<sub-resource name>.<...>

For example, the system administrator, who is also treated as resource, can be represented as: usr.admin, where usr is the resource name for users group.

There are multiple approaches to resources storage but two of them are used most frequently. One is implemented by using the file directory management function of an operating system, namely the file directory mode. The advantages of this are easy to implement and independent of the other system software. However, the lack of security and query efficiency are its disadvantages. Another approach is based on a database, the so-called database mode. This mode is good for security and query efficiency but its implementatuiin is complicated and lack of flexibility. It is also possible that both modes are used together to manage resources and integrate data efficiently.
5.1.2 Proposed Protocol

It is essential to define a protocol to access resources. The Multiple Resources Transfer Protocol (MRTP) is defined for the requirement. MRTP is a generic, stateless, object-oriented and open communication protocol, which is an extended protocol based on other protocols. MRTP refers to SMTP, POP3, FTP, HTTP and TCP/IP [134] that are quite popular and accepted by industry, and extends the following functions based on those protocols:

- Resource creation, authorisation, usage, maintenance, and classification are added, and resources integration can be performed.
- The support for flow is enhanced, such as flow storage management and flow delivery control.
- Accesses to and operations on various databases are added, which support multiple-tier distributed applications based on database technology.
- Facilities for interpreting and executing a specific command language are added, which enable application extension through connecting with server-side components.

MRTP can be used for many tasks, such as distributed object management systems, through extension of its request methods (commands). A feature of MRTP is the typing of data representation, allowing systems to be built independent of the data being transferred. MRTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia applications.

Essentially, MRTP is a protocol based on authorisation. Each command execution with MRTP must satisfy its authority, and anonymous operation is not allowed. The authority of a kind of resources is composed of the following eight rights:

- RIGHT_NONE: No authority authentication
RIGHT_CONFIRM: Resource (user) must be confirmed
RIGHT_READ: Resource read permitted
RIGHT_SEND: Resource send permitted
RIGHT_CREATE: Resource create permitted
RIGHT_MODIFY: Resource modify permitted
RIGHT_DELETE: Resource delete permitted
RIGHT_CONTROL: Resource control permitted

Appendix A gives the whole contents of the Multiple Resource Transfer Protocol (MRTP).

5.1.3 Core Server Structure

A Core Server based on resources is set up to provide multiple services with MRTP. It is the centre of resource management, the core of all system services, and the key to services integration construction.

Core server structure is involved with the following eight aspects:

Resource storage: (but you have not said what it is yet) This is fundamentally different from using the file directory or the database mode for core server implementation. Based on the features of resources storage, e.g. the directory structure of resource storage, it is possible to develop basic classes with common characteristics. No matter what resource storage mode is adopted, its implementation will inherit from these basic classes.

Authority application: Authority application is an important means to ensure information security. When the access authority of a certain resource is judged, the following principles are abided by:

The range of authorisation is from wide to narrow: all resources → grouped resources → roles → current resource. The judgment of whether to authorise all
resources is the first step. After that, whether to authorise grouped resources is considered. The third step is to judge whether to authorise roles, and the last step is to judge whether to authorise the current resource. If it is necessary, the authority of each resource item will be judged.

✧ The level of authorisation is from low to high: if the current resource is not authorised, then the authority of its parent resource can be regarded as the current resource authority. The rest may be deduced by analogy.

➢ Information transmission: As the most basic manipulation of a core service, the information transmission implementation involves the following two aspects:

✧ Promoting efficiency: To acquire higher efficiency of information transmission on the network, especially for large file packages, the data should be compressed with suitable algorithms before transmission so that it can be transmitted in a smaller package. The selected algorithm should meet the requirement of both high compression ratio and acceptable time consumption. It must be mentioned here that the core servers implementing information transmission may use several different, possibly incompatible compression algorithms in a single core server.

✧ Enhancing security: It is also of importance to secure the information when transmitted. A direct solution is encrypting information to avoid intercepting and capturing data directly. Another popular solution is Certification Authority (CA), in which the core server is used as or integrated with the CA server. Then the system can be provided with the basic characteristics of CA, such as Authentication, Integrity, Nonrepudiation and Confidentiality in information transmission [95].

➢ Workflow control: Workflow control is carried out by the workflow engine. The engine interprets and runs part or all of the information defined by the workflow, communicates with external application programs to create, execute and manage
the workflow case, produces work items or tasks, and informs users of carrying out further operations. As the core of the workflow engine, flow execution services must be able to cope with the following three categories of data [64].

♦ Workflow Control Data: Internal data that is managed by the workflow management system and/or workflow engine.

♦ Workflow Relevant Data: Data that is used by a workflow management system to determine the state transition of a workflow process instance.

♦ Workflow Application Data: Data that is application specific and not accessible by the workflow management system.

➢ **Data access:** MRTP supports direct access to databases by interface name. As data interface definition becomes the key to database access, two manners are proposed for data access:

♦ Direct Access uses the interfaces provided by the native database system. Different solutions are required to be fit for corresponding database systems.

♦ Standard Access uses the common interfaces accepted by industry such as JDBC, ODBC or ADO etc. It is easy to implement without worrying about the upgrade. However, using ODBC or ADO, core services could only be run on the Windows platform. To be implemented on multi-platform, for example, core services could be run on Windows, UNIX or Linux, so that JDBC and other multi-platform interfaces are better choices.

➢ **Services integration:** Services integration is the fundamental aim for the design of the core server. With managing resources, transferring information, and accessing data, in practice, the Mail server, FTP server and database service based on authorisation and role are provided as a comprehensive integration solution. In addition, there are two aspects related to integration that should be taken into account.
Web Server and Web Script: It is necessary that a core service is able to support accesses and operations on HTTP, and to be extended on that basis. A flexible and direct approach is to provide an embedded Web Script language, which is integrated with the HTML language and able to implement more complicated tasks to answer requests, such as resources operation, data access and components calling, etc. At present, many prevalent web scripting languages like ASP, PHP, JSP and so on exist.

Web Services: supports SOAP [149] on web servers. On one side, SOAP server, which supports web service register ration, is set up on the core server to provide relevant services. On the other side, it is possible to access and call other web services through SOAP. By providing support to SOAP, each service provided by web services can be integrated into the core server.

Interface providing: the core server is system-level software. It is necessary to develop relevant interfaces, namely interface components, to call core services. Such interfaces ought to be simple and easy to use. There are two modes for that purpose.

Class library mode: This approach is to implement relevant interface classes developed by Java or C++ and so on. The advantage of that approach is implementing integration with application at code-level, so as to call interface flexibly.

Component mode provided with a COM+ or DLL library. This mode allows that application development is not limited within a certain language, which helps to deploy applications.

Tools supported: for the sake of convenience for core server configuration, maintenance and management, the following tools should be provided.

Resources management tools: To create, delete, rename, authorise and configure properties, etc.
Database connection tool: To transact the connection and data transmission among various databases, and define the name of the connection interface.

Web Services configuration tool: To install, configure and manage web services.

Certification management tool: To make or grant CA certification.

5.1.4 Core Language

To integrate components, it is necessary to design or choose one language as the core language, which should have object-oriented programming characteristics and be easily accepted by developers. Therefore, there are two solutions generally:

- **Choosing an existing language:** To choose an eligible language as the core language from currently prevalent development languages, such as Java, C++ and so on. The key to the choice lies in the ability to easily install and deploy that language, in addition it is convenient for the seamless integration between the language interpretation and execution environment and the system.

- **Developing a new language:** To develop a new language as the core language. The advantage is that the new language is easy to integrate with the system but the developers have to learn a new language.

The solution is both of them. Firstly, the core language is composed of several Web Script languages such as Java Script, VB Script, Perl Script and so on. Since those descriptive languages are well known to developers along with their mature run environments. Especially, Java Script and VB Script have been installed in each Windows PC since Internet Explorer became a part of the Windows operating system. Secondly, a C-like or Delphi-like Script language was developed. The purpose is that programmers can develop applications using their familiar language style instead of depending on the run-time environment of that programming language.
5.1.5 Atomic Components Assembly

In the core language environment, the main problem is to assemble atomic components, especially those common Script language components. The atomic components mean the composition of those common components, which are classified into nine types.

➢ User interface design: menu, button, layout, color, text, link, tool bar, status bar, image, etc.

➢ Data access: database connection, data source, table operation, query, storage, procedure calling, etc.

➢ Data edit: text box, check box, list box, dialog box, sheet editor, etc.

➢ Data control: form edit, data guide, field operation (for text, number, date, image, BLOB etc), data transformation and storage, etc.

➢ Report: report design, preview, printout as normal and in particular, predefined formats, etc.

➢ Multimedia play: audio frequency, cartoon, video, music and so on.

➢ Chart display: histogram, pie chart, line chart, display as 2D or 3D etc.

➢ File operation: create, delete, rename, move, maintenance, etc.

➢ System access: core services access, operating system access, workflow control, etc.

The combination of core language and atomic components enables application extensions. It is the basis of application components design and components integration.

5.1.6 Virtual Machine & Integrated Development Environment Structures

Virtual Machine (VM) is not only a run-time version of the core language, but also a
run-time platform for application components. As a vital part of application extension, VM can seamlessly be integrated with core services and run at the client side to implement complex application components. As a result, there are two aspects that should be taken into account when VM is implemented:

- Server-side VM: this type of VM particularly emphasises functionality, such as data access, file operation, system access, except interactive part. Moreover, server-side VM should be able to support multi-user accessing and multi-thread calling.

- Client-side VM: the capability of interaction with users and transparently accessing databases are the main tasks of this kind of VM. In other words, it is not necessary to install the database client interface on the client. The client-side VM needs to be deployed on each client thus multi-user accessing is not compulsory.

According to the different mode of VM implementation, the Integrated Development Environment (IDE) based on the core language and atomic components can be designed from both server and client sides.

- Server-side IDE: is composed of program editor, compiler, debug and running simulator. Among them, running simulator is used to simulate program running on the server and check execution results.

- Client-side IDE: consists of component tool bar, virtual form designer, property editor, program editor, debug and program running machine. Virtual form designer is the core of interactive design. The function and the usage should be as good as current prevalent IDE.

### 5.2 Important Aspects

As web applications are getting prevalent, users are used to operating browsers. For this reason the client-side virtual machine should be seamlessly integrated with the browser, the most commonly web browser is Microsoft Internet Explorer (IE). There are two modes for
such client-side virtual machine implementation based on the integration requirements of IE:

- COM+ components, which can be called by VB Script and Java Script supported by IE, and by other programming languages supporting COM+. However, they have to be installed on client before using, which does not favour automatic upgrade and deployment of components.

- ActiveX components, which are integrated with webpages and regarded as objects called by VB Script and Java Script. Accordingly, component upgrade and deployment are convenient due to their automatic installation on clients.

For each system based on the Tri-Integration Pattern, there are some resource utilisation problems. According to authority, a system is able to access one or more resources that other systems have. Similarly, one or more resources of a system can open to other systems.

The workflow application, across systems, is a requirement in particular environments. While implementing a system, a mechanism should be provided to ensure workflow control across systems.

Design and implementation of transparent database access is very challenging. On one hand, it is necessary to ensure access to various kinds of database without installing a database interface on the client. On the other hand, it is required to reduce the burden of network transmission as much as possible. The access mechanism based on three-tier architecture is set up to implement data transmission by combining MRTP with an efficient data structure.

In order to design different business workflow flexibly and conditionally and control each node and direction, relevant tools are required. The key to implementing workflow control is developing a visual workflow designer, through which the flow node design can be easily implemented that including condition judging, direction control, function configuration and system calling. This visual workflow design tool can provide an efficient...
assistant for designing complex flows.

5.3 Summary

The Tri-Integration Pattern can be used to build an integrated, secure and extendable network software system. It defines eight participating structures at its architectural level. A detailed specification of the structural aspects has been claimed for each structure element. An application system based on the pattern can build on several or all of these structure elements. Therefore, the Tri-Integration Pattern has the following important advantages:

➢ **Integrity.** The application systems based on the Tri-Integration Pattern have high integrity because they adopt a uniform protocol, cooperate with a uniform interface and construct with a uniform method.

➢ **Adaptability.** Since the core services are constructed on various services, such as Internet protocols, database access, information exchanges and workflow applications, etc., they can be applied to information management, business flow control, E-business, E-government, decision support, etc, which makes them comprehensively adaptable.

➢ **Extensibility.** Various application components can be developed using the core language with atomic components, and these components can be deployed by resource management according to certain authority. Additionally, application components can be embedded in the flow, so providing more complex flow control. It means that the extensibility of constructing applications on the system based on the pattern is comprehensive. If a core server is incapable of satisfying system requirements or system efficiency, one or more core servers can be deployed to further the support system till it meets all the demands of the system. Integrating several core servers, and sharing resources and exchanging data among them, so as to form a distributed core server application architecture is more broadly valuable.

➢ **Reliability.** Two issues are important in achieving reliability: availability and fault
tolerance [137]. A distributed Tri-Integration architecture supports availability, because it allows the same core server to be run on more than one machine, increasing availability. If a core server or a machine fails, therefore, the failure does not necessarily have an impact on an application. Fault tolerance may be easily supported because distributed systems allow failures to be hidden from a user.

➤ Security. Initially, MRTP adopted in the core service is based on authority control. Secondly, information transmission of the core service is encrypted. Thirdly, the core service can be integrated with CA server, which gives the system the basic characters of CA, i.e. Authentication, Integrity, Nonrepudiation and Confidentiality.

➤ Transparency. In a Tri-Integration Pattern system, organisation and authorisation of resources, integration of core services, connection to databases, control of workflow and the detail of designing application components are hidden from clients. In other words, accessing and applying the system is transparent to clients.

➤ Interoperability. A good system also can realise interactive operation with the systems based on the host machine or LAN. MRTP ensures that a Tri-Integration Pattern system can completely satisfy these demands. It can be interconnected and coexist with other Mail systems or Web systems and share information or exchange data with file systems.

The Tri-Integration Pattern architectural framework also has liabilities.

➤ Performance. The performance of the Tri-Integration Pattern depends on the cost of the core services. If a software system for a specific application is compared with a Tri-Integration system supporting different applications, the performance of the former will be better in most cases. Therefore there is a price to pay for flexibility and extendibility. If the core service implementation of the Tri-Integration system is optimised for performance, however, this price can be overlooked [137].

➤ Complexity of design and implementation. Developing a Tri-Integration-based
system is a non-trivial task. For example, it can sometimes be very difficult to determine the basic mechanism of MRTP, the core language, VM, core services and system security. In addition, the separation between mechanisms and policies requires in-depth domain knowledge and considerable effort during requirements analysis and design.
Chapter 6  Tri-Integration Pattern: Design Level

At the design level, a system pattern provides a scheme for refining the subsystems or components of a software system, or the relationships between them. The Tri-Integration Pattern as a system pattern helps solve development problems, it provides complete solutions for these problems. So, an application framework of Tri-Integration Pattern has been presented to illustrate what the pattern’s solutions are at this level.

6.1 Application Framework of Tri-Integration Pattern

In order to solve the development problems in the Tri-Integration Pattern, the following four aspects are considered:

➢ Supporting resources shared, which enables various Internet resources such as files, e-mail, web sites and databases etc, to be shared in a uniform format.

➢ Supporting resources management, which provides an effective method to utilise and manage resources according to the user’s authority.

➢ Supporting workflow control, which implements the electronic operation and automatation of the enterprise’s business processing according to the enterprise’s management pattern and combining with its resources.

➢ Supporting application development, which can be used to improve and extend the functions of the system according to the user’s requirements.

So, an application framework of Tri-Integration Pattern supports applications integrated and expandable. It can capture, compose, share and utilise resources, and also helps to manage and provide services with necessary and supports workflow control and complicated development [85, 84, 144].
Figure 6.1 shows the application framework of Tri-Integration Pattern. It is an improvement on the relative framework [144].

![Diagram of Tri-Integration Pattern](image)

Figure 6.1 Application Framework of Tri-Integration Pattern

### 6.2 Constructing Design Level Elements

The Tri-Integration pattern defines five design level elements (as show in Table 4.3):

- Resource Design
- Authorisation Strategies
- Multi-resource Transfer Protocol
- Core Server Design
- Workflow Control

#### 6.2.1 Resource Design and Storage

In the current design of the Tri-Integration pattern, the resource is organised in a tree structure, and is stored in a file directory mode, i.e. one resource is related to one file
directory. For example, the resource USR.ADMIN is related to file directory:

<RESOURCE ROOT>\USR\ADMIN\.

Resource Description is saved in the file Authority.dat, which is a text file and each of its lines represents one information description for the resource. Although the file can be edited in any text editor, in general, the resource manager is a good choice for editing the file Authority.dat.

The file Authority.dat includes two parts, i.e. the description part and the authority part. The description part uses the character P. as its prefix, and its format is:

\(<P\text{.name}> = \langle content\rangle.\)

Table 6.1 shows the description of the authority part:

<table>
<thead>
<tr>
<th>Name</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.FullName</td>
<td>The name of the resource.</td>
</tr>
<tr>
<td>P.UserID</td>
<td>The login name of the resource.</td>
</tr>
<tr>
<td>P.Priority</td>
<td>The priority of the resource.</td>
</tr>
<tr>
<td>P.IsGlobalAdmin</td>
<td>Whether the resource is the administrator.</td>
</tr>
</tbody>
</table>

The functionality resource is also provided in the description part, and its relative descriptions are shown in Table 6.2:

The authority part uses the character A. as its prefix, and its format is:

\(<A\text{.name}> = \langle authority number \rangle.\)

The description of authority is shown in Table 6.3:
Table 6.2 Description Information of Functionality Resource

<table>
<thead>
<tr>
<th>Name</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>P.FormEntry</td>
<td>The form file related to the resource</td>
</tr>
<tr>
<td>P.Scheduler</td>
<td>Time resource</td>
</tr>
<tr>
<td>P.Internet</td>
<td>The resource connects to appointed web address</td>
</tr>
<tr>
<td>P.AppEntry</td>
<td>The resource runs the appointed program</td>
</tr>
<tr>
<td>P.Mail.Server</td>
<td>Specify whether the core Netcase server receives e-mails automatically.</td>
</tr>
<tr>
<td>P.POP3.Server</td>
<td>POP3 server address</td>
</tr>
<tr>
<td>P.POP3.Port</td>
<td>POP3 server port</td>
</tr>
<tr>
<td>P.POP3.Account</td>
<td>POP3 server login name</td>
</tr>
<tr>
<td>P.POP3.AID</td>
<td>POP3 server login password</td>
</tr>
<tr>
<td>P.SMTP.Server</td>
<td>SMTP server address</td>
</tr>
<tr>
<td>P.SMTP.Port</td>
<td>SMTP server port</td>
</tr>
<tr>
<td>P.SMTP.Account</td>
<td>SMTP server login name</td>
</tr>
<tr>
<td>P.SMTP.AID</td>
<td>SMTP server login password</td>
</tr>
</tbody>
</table>

Table 6.3 Description of Authority

<table>
<thead>
<tr>
<th>Name</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>A.Everyone</td>
<td>The authority is assigned to every user in the Netcase system</td>
</tr>
<tr>
<td>A.GlobalAdmin</td>
<td>The authority is assigned to the system administrator</td>
</tr>
<tr>
<td>A.LocalAdmin</td>
<td>The authority is assigned to the resource (group) administrator</td>
</tr>
<tr>
<td>A.DocAdmin</td>
<td>The authority is assigned to the document administrator</td>
</tr>
</tbody>
</table>

The authority is also assigned to a concrete resource, using the format:

<Resource Name> = <Authority number>.

For example, USR.ADMIN=10.

The item description is saved into a text file with EMI postfix in the resource, and the file name is a unique identifier for the item, each line of it represents a piece of description information. Table 6.4 lists the basic information about the item descriptions:
Table 6.4 Basic Information of EMI File

<table>
<thead>
<tr>
<th>Name</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageID</td>
<td>The unique message item ID</td>
</tr>
<tr>
<td>FromName</td>
<td>The resource description of sending message (user name)</td>
</tr>
<tr>
<td>FromUser</td>
<td>The resource name of sending message (login name)</td>
</tr>
<tr>
<td>ToName</td>
<td>The resource description of receiving message (user name)</td>
</tr>
<tr>
<td>ToUser</td>
<td>The resource name of receiving message (login name)</td>
</tr>
<tr>
<td>Subject</td>
<td>The title of the message item</td>
</tr>
<tr>
<td>DateTime</td>
<td>Time of sending the message</td>
</tr>
<tr>
<td>AttachFile</td>
<td>The name of attached file</td>
</tr>
<tr>
<td>Isread</td>
<td>The flag of whether the current message is read, replied to and/or forwarded.</td>
</tr>
</tbody>
</table>

The item entity content of the resource is saved as a text file with an EMD postfix, and the file name is a unique identifier for the item. Each EMD file is related to only one EMI file. Both files use the same file name. The EMD file is a compressed file, which holds many entities’ file content. Table 6.5 lists the basic constituents of item entity content. The CONTENT.DAT file must be included while other files are determined by concrete forming of the information item:

Table 6.5 Constituents of the EMD File

<table>
<thead>
<tr>
<th>Name</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONTENT.DAT</td>
<td>Save text information</td>
</tr>
<tr>
<td>ALLNET.DFM</td>
<td>The form file of the current item</td>
</tr>
<tr>
<td>ALLNET.SHT</td>
<td>The sheet file of the current item</td>
</tr>
<tr>
<td>ALLNET.WEB</td>
<td>The web file of the current item</td>
</tr>
<tr>
<td>&lt;Attach Files&gt;</td>
<td>All attached files</td>
</tr>
</tbody>
</table>

6.2.2 Authorisation Strategies

In the Tri-Integration Pattern, the access to a resource is based on authority control, and the anonymous access mode is not permitted when maintaining a resource. The Tri-Integration Pattern defines eight authorities, listed in Table 6.6 below, among them RIGHT_DENY
and **RIGHT_CONTROL** are absolute authorities. The access authority for some resources is not only either one of the eight authorities, but may also be a combination of another six authorities excluding the above two authorities. For example, A.Everyone=10 means that everyone has authority to send and read the resource, i.e. A.Everyone = **RIGHT_READ or RIGHT_SEND**.

<table>
<thead>
<tr>
<th>Name</th>
<th>Value</th>
<th>Signification</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RIGHT_DENY</strong></td>
<td>0</td>
<td>Deny</td>
</tr>
<tr>
<td><strong>RIGHT_CONFIRM</strong></td>
<td>1</td>
<td>User confirm (the first bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_READ</strong></td>
<td>2(0X2)</td>
<td>Read (the second bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_SEND</strong></td>
<td>8(0X8)</td>
<td>Send (the fourth bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_CREATE</strong></td>
<td>512(0X200)</td>
<td>Create (the tenth bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_MODIFY</strong></td>
<td>2048(0X800)</td>
<td>Update (the twelfth bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_DELETE</strong></td>
<td>8192(0X2000)</td>
<td>Delete (the fourteenth bit is 1)</td>
</tr>
<tr>
<td><strong>RIGHT_CONTROL</strong></td>
<td>32768(0X8000)</td>
<td>Full control (the sixteenth bit is 1)</td>
</tr>
</tbody>
</table>

The authorisation strategies are entirely based on the authorisation mode of the Tri-Integration Pattern, i.e. following the principle of from large to small in scope and from low to high in level. Three special roles are considered, which are group administrator, system administrator and **INET_GUEST** user. **INET_GUEST** user is specially designed in order that an external user may view public information in the real system; its authority is not beyond **RIGHT_READ**. The diagram Figure 6.2 shows how a user (or resource) gets the authority of a certain resource.
Figure 6.2 Authorisation Strategies
### 6.2.3 Multiple Resources Transfer Protocol

MRTP is a basic protocol used in the Tri-Integration Pattern. It is introduced in section 5.4.2. Table 6.7 lists the command classification of the MRTP, and Appendix A gives MRTP in detail and the illustrations of each command.

**Table 6.7 Command Classification of the MRTP**

<table>
<thead>
<tr>
<th>The commands for system login</th>
<th>LOGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>CHANGEPWD</td>
</tr>
<tr>
<td></td>
<td>LOGOUT</td>
</tr>
<tr>
<td>The commands for sending messages</td>
<td>VALUE</td>
</tr>
<tr>
<td></td>
<td>RCPT</td>
</tr>
<tr>
<td></td>
<td>CONTENT</td>
</tr>
<tr>
<td></td>
<td>FINISH</td>
</tr>
<tr>
<td></td>
<td>ATTACH</td>
</tr>
<tr>
<td></td>
<td>RSET</td>
</tr>
<tr>
<td>The commands for messages management</td>
<td>MSGLIST</td>
</tr>
<tr>
<td></td>
<td>MSGINFO</td>
</tr>
<tr>
<td></td>
<td>MSGCONTENT</td>
</tr>
<tr>
<td></td>
<td>MSGFILE</td>
</tr>
<tr>
<td></td>
<td>MSGDELETE</td>
</tr>
<tr>
<td></td>
<td>MSGLOADFILE</td>
</tr>
<tr>
<td></td>
<td>MSGSAVEFILE</td>
</tr>
<tr>
<td></td>
<td>MSGCOPYTO</td>
</tr>
<tr>
<td></td>
<td>MSGUPDATEINFO</td>
</tr>
<tr>
<td></td>
<td>MSGUPDATECONTENT</td>
</tr>
<tr>
<td></td>
<td>MSGUPDATEFILE</td>
</tr>
<tr>
<td></td>
<td>SETMSGINFO</td>
</tr>
<tr>
<td></td>
<td>SETMSGRREAD</td>
</tr>
<tr>
<td></td>
<td>DETECTNEW</td>
</tr>
<tr>
<td>The commands for resource management</td>
<td>RESLIST</td>
</tr>
<tr>
<td></td>
<td>MAKERES</td>
</tr>
<tr>
<td></td>
<td>EMPTYRES</td>
</tr>
<tr>
<td></td>
<td>DELETERES</td>
</tr>
<tr>
<td></td>
<td>RESTREE</td>
</tr>
<tr>
<td></td>
<td>RESRENAME</td>
</tr>
<tr>
<td></td>
<td>RESEXISTS</td>
</tr>
<tr>
<td></td>
<td>RESSIZE</td>
</tr>
<tr>
<td>The commands for authority management</td>
<td>AUTHORITY</td>
</tr>
<tr>
<td></td>
<td>SAVEAUTHORITY</td>
</tr>
<tr>
<td>The commands for task management</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td>---------------------------------</td>
<td>-----------</td>
</tr>
<tr>
<td></td>
<td>TASKINS</td>
</tr>
<tr>
<td></td>
<td>TASKDEL</td>
</tr>
<tr>
<td></td>
<td>TASKINFO</td>
</tr>
<tr>
<td>The commands for file management</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>FILELIST,</td>
</tr>
<tr>
<td></td>
<td>FILEAGE,</td>
</tr>
<tr>
<td></td>
<td>FILEVERSION</td>
</tr>
<tr>
<td></td>
<td>DELETEFILE</td>
</tr>
<tr>
<td>The commands for workflow management</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>FLOWLIST</td>
</tr>
<tr>
<td></td>
<td>FLOWDEL</td>
</tr>
<tr>
<td>The commands for searching and statistics</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>SEARCH</td>
</tr>
<tr>
<td></td>
<td>TOTALON</td>
</tr>
<tr>
<td>The database operating commands</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>ALIASES</td>
</tr>
<tr>
<td></td>
<td>QUERY</td>
</tr>
<tr>
<td></td>
<td>APPLYUPDATES</td>
</tr>
<tr>
<td></td>
<td>TABLenames</td>
</tr>
<tr>
<td></td>
<td>FIELDNAMES</td>
</tr>
<tr>
<td></td>
<td>QUERYTOSTRING</td>
</tr>
<tr>
<td></td>
<td>QUERYTOPAGE</td>
</tr>
<tr>
<td></td>
<td>EXECSql</td>
</tr>
<tr>
<td>The commands for HTTP compatibility</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>GET</td>
</tr>
<tr>
<td></td>
<td>POST</td>
</tr>
<tr>
<td></td>
<td>HEAD</td>
</tr>
<tr>
<td></td>
<td>VIRTUAL</td>
</tr>
<tr>
<td>The commands for running remote script programs</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>SCRIPT</td>
</tr>
<tr>
<td></td>
<td>SCRIPTFILE</td>
</tr>
<tr>
<td>The commands for other uses</td>
<td>USERRIGHT</td>
</tr>
<tr>
<td></td>
<td>DATETIME</td>
</tr>
<tr>
<td></td>
<td>MSGID</td>
</tr>
<tr>
<td></td>
<td>LICENCE</td>
</tr>
<tr>
<td></td>
<td>SERVERADDR</td>
</tr>
<tr>
<td></td>
<td>NOOP</td>
</tr>
</tbody>
</table>

6.2.4 Core Server Design

Based on the MRTP protocol, the core server supports many services. It is designed in detail to include the following contents besides the organisation, storage and authorisation strategies of the resource mentioned above.

- Base class definition: The base class is the kernel of the core server. No matter what kind of resources storage mode is adopted, it should be implemented based on
inheritance of the base class. The design of base class is listed in Appendix B.

➢ Messages Delivering:

✧ Login message is delivered in cryptograph;

✧ Attached files are uploaded and downloaded in a compressed stream;

➢ Data Access:

✧ Microsoft Advance Data Object (ADO) and Borland Database Engine (BDE) are adopted as the data interface to connect various databases;

✧ The query or maintenance of the database is operated based on the data interface name being used, and the name is defined in Alias Manager, and this interface name is transparent to the developer;

➢ Web Server:

✧ The dynamic extension based on ISAPI is supported;

✧ A Web Script language can be embedded in HTML files, and dynamic web page design is supported;

➢ E-mail Service:

✧ E-mail can be received by the core server from an e-mail server supporting the POP3 protocol

✧ E-mail can be sent to the email server supporting the SMTP protocol by the core server;

➢ Virtual Machine:

✧ The virtual machine based on the Web Script language is embedded in the core server;

✧ The program written in Web Script language can be accessed remotely through the MRTP protocol;
Component Using:

- The database can be accessed via the data accessing components in the Web Script language;
- The COM+ components can be accessed via the component object interface functions in the Web Script language;

![Diagram of Logical Structure of Core Server]

Figure 6.3 Logical Structure of Core Server

Although resource management may be operated remotely, the definition of the data interface must be handled in the server because all database-accessing interfaces are installed on the server. Figure 6.3 shows the logical structure of the core server.

### 6.2.5 Workflow Control

In the Tri-Integration Pattern, workflow control is tightly related to the resources. The following three aspects illustrate workflow control implementation:

- **Workflow Control Data:** The available workflow control files are saved in the resource `pub.flowchart`, and every workflow to be started up comes from the resource. Workflow control data is saved in the resource `sys.flowbox` that belongs to
system resources, and every workflow in processing corresponds to a file with EMI as its postfix. This records the basic flow information and delivering history. The key items in this file is shown in Table 6.8.

### Table 6.8 Key Items of Workflow Control File

<table>
<thead>
<tr>
<th>Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Subject</td>
<td>The subject of flow</td>
</tr>
<tr>
<td>DateTime</td>
<td>The start time of the flow</td>
</tr>
<tr>
<td>Flow.sys_user</td>
<td>The user who starts the flow</td>
</tr>
<tr>
<td>Flow.sys_used</td>
<td>The description of the flow steps</td>
</tr>
<tr>
<td>Flow.InsideModel</td>
<td>The template file used by the flow</td>
</tr>
<tr>
<td>Flow.InsideSheet</td>
<td>The sheet file used by the flow</td>
</tr>
<tr>
<td>Flow.InsideForm</td>
<td>The form file used by the flow</td>
</tr>
<tr>
<td>&lt;step&gt;.Flow</td>
<td>The concrete description of flow steps, its format is &lt;step&gt;.Flow = Next Step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowCode</td>
<td>The codes executed before flow moves onto the next step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowForm</td>
<td>The form file executed before the flow turns into next step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowSendCode</td>
<td>The codes executed before sending in flow step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowSendForm</td>
<td>The form file executed before sending in flow step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowAfterSendCode</td>
<td>The codes executed after sending in flow step</td>
</tr>
<tr>
<td>&lt;step&gt;.FlowAfterSendForm</td>
<td>The form file executed after sending in flow step</td>
</tr>
<tr>
<td>S.&lt;Date&gt;</td>
<td>The record of flow transferring using the format S.&lt;Date&gt; = StepX &gt; StepY &gt; UserX &gt; UserY &gt; MessageID</td>
</tr>
</tbody>
</table>

> Workflow Relevant Data: the workflow delivery process is similar to the message transferring process, workflow relevant data is sent to the inbox of the corresponding user (resource) accompanied by the workflow turning operation and saved into the item description of the resource item. Besides similar descriptions of workflow shown in Table 6.8, two additional descriptions are added in Table 6.9.

### Table 6.9 Additional Information of Item Description

<table>
<thead>
<tr>
<th>Names</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flow.ID</td>
<td>It indicates an EMI file name used in the sys.flowbox resource</td>
</tr>
<tr>
<td>&lt;step&gt;.sys_step</td>
<td>The name of the current flow step</td>
</tr>
</tbody>
</table>

> Workflow Application Data: The application data related to the flow has sheet,
form, and relevant file etc. The processing manner is similar to flow-relevant data, but it is saved in the entity content of the resource item. Figure 6.4 shows the logical view implemented in the workflow control.

![Workflow Implementation Logical View](image)

**Figure 6.4 Workflow Implementation Logical View**

### 6.3 Summary

The Tri-Integration Pattern emphasises on solving the cooperation problems of different kinds of network components. It provides a management engine, which can be used for various information resources, strengthening data-sharing and maintenance between heterogeneous databases, providing secure information exchange channels and a powerful means of publishing information. An effective mechanism about network information resource management, workflow management and authority management, is followed, and given in the Tri-Integration Pattern. The component separating principle, all kinds of applications implemented through the Tri-Integration Pattern are much easier to maintain and reuse.

The Tri-Integration Pattern supports the cooperating work among various application units. Most business programs are processed on the core server, data resources are put on the database server, also one or more core servers and database servers can coexist. The database server and core server can therefore be distributed across the different machines in
a large-scale application system.

With many methods such as query constructing and form defining etc, structural information from different databases (such as Sybase, Oracle, MS SQLserver, Informix, DB2 etc.) and non-structural information in other forms can be maintained, shared and published in the application framework of the Tri-Integration Pattern through keeping to the system authority controlling strategies.

To depend on the MRTP, the Tri-Integration Pattern provides a security information exchange channel. Through integrating a multi-level security mechanism, the application framework ensures high-security of user data and applications. It provides a good measure for secrecy, integrity, usability, controllability, and non-deniability of the information and information system: ensures that the information is not revealed or available to forbidden-users; ensures that the authority information can be delivered without distortion from the right source to the right destination; ensures that the information and information system are only used by authorised users and avoids the information system refusing to supply services or being used by other users; implements security supervising management for information content and information system and avoids non-authorised modifying; and ensures that any user cannot deny his behaviour and so makes personal behaviour reliable.
Chapter 7  Tri-Integration Pattern: Idiom Level

At the idiom level, a system pattern describes how to implement particular aspects of components or the relationships between them with the features of given languages.

The Tri-Integration pattern defines eight idiom level elements (as show in Table 4.3):

- Resource Managing Idiom
- Virtual Machine Environment Idioms
- Alias Managing Idiom
- Web Page Designing Idiom
- Form Designing Idiom
- Flow Designing Idiom
- Sheet Designing Idiom
- Client Exploring Idiom

These idioms can be implemented by most programming languages, such as Borland Delphi, C++, Java and Visual Basic, etc. An UML class diagram [20] is used for the logic view of each idiom.

7.1  Resource Managing Idiom

The resource manager plays a kernel role in the the Tri-Integration Pattern application framework, which provides a quick, convenient method for the administrator to accomplish many activities such as resource maintenance, resource authorisation, resource properties definition and resource setting. This tool uses a tree directory management solution similar to Windows Explorer to manage the resources and implements remote maintenance.
The resource manager is composed of the following parts:

- **Menu Bar**: Includes such menu items as create resource, delete resource, import resource, export resource, and edit for resource descriptions;

- **Tool Bar**: Includes the following function buttons, refreshing resource, searching resource, creating resource, deleting resource, clear resource content, adding for resource description, deleting for resource description, setting resource password, maintaining resource sign, resource occupied space statistics, and reading system log;

- **Resources Area**: This displays the resource structure in tree form making it easy to rename or move a resource in the resource manager. The resources area contents includes three main parts, USR denotes system user resource; PUB denotes system public resource including form, sheet, template, workflow and user-defined resource; and SYS represents the resources used by the system itself, which are user ranks, system groups, and workflow monitoring depository etc;

- **Property Configuring Area**: This is used to display and edit the description information of each resource. It can be used to maintain resource information items, such as resource description defining, authority setting, priority setting along with type configuring etc;

- **Status Bar**: The status bar is used to show current resource name and corresponding operating information.

Figure 7.1 shows the main interface of the resource information manager.
Figure 7.1 Resource Manager

Figure 7.2 lists the implementation class diagram of the resource manager.

Figure 7.2 Resource Manager Logical View
7.2 Virtual Machine Environment Idioms

7.2.1 Server-side Virtual Machine Environment

The server-side virtual machine supports the design and process enterprise business logic, so powerful functions except interaction with user are emphasised while implementing virtual machine, for example, data accessing, file operating, system accessing and component calling, etc., and the system is designed to support multi-user accessing and multi-thread calling.

7.2.1.1 Web Script Language

The core language adopted by the core server is a Web Script language, which is an object-oriented programming language based on Object Pascal. The syntax of the Web Script is listed in Appendix B.

7.2.1.2 Kernel Functions and Component Objects

The Web Script is designed necessarily for server-side application extension. The following kernel functions and related component objects are implemented:

➢ Embedded script: the script statements written in the Web Script language can be embedded into HTML pages directly, so that the script is executed in the core server rather than in the Web browser. For example:

```html
<HTML>
<BODY>
<%var i: integer;
for i:=3 to 7 do
SendLn('<FONT SIZE='+IntToStr(i)+'>Hello Netcase!<BR>');
%>
</BODY>
</HTML>
```

➢ Accessing core server: It is very convenient for developers to access core server
directly in a web site by using the script language with the embedded TAnywhere component object. For example:

```html
<html>
<head>
</head>
<body>
<%
  var Anywhere: TAnywhere;
  Anywhere := TAnywhere.Create('localhost','admin','aa');
  if Anywhere.Login then
    PrintLn(Anywhere.GetLicence+'<BR>')
  else
    PrintLn('Can not login to Netcase Anwhere.<BR>');</n
Anywhere.Free;
%
</body>
</html>
```

Flexible Database Connecting: With the script in core server and the embedded TQuery database component object, a database can be accessed easily in a web site.

For example:

```html
<html>
<head>
</head>
<body>
<%
  procedure ListDataSet(q: TDataSet);
  var i: integer;
  begin
    SendLn('<tr>');</n
    for i:=0 to q.FieldCount-1 do
      Send('<td>'+q.Fields[i].DisplayLabel+'</td>');
    SendLn('</tr>');
    q.First;
    while not q.Eof do begin
      SendLn('<tr>');</n
      for i:=0 to q.FieldCount-1 do
        Send('<td>'+q.Fields[i].AsString+'</td>');
      SendLn('</tr>');
      q.Next;
    end;
  end;
</body>
</html>
```
end;

var Query: TQuery;
Query := TQuery.Create('Interbase', 'localhost:test.gdb', 'sysdba', 'masterkey');
Query.SQL := 'select * from USERS order by NAME DESC';
Query.Open;

>%
<Table border=1>
<%
ListDataSet(Query);
Query.Free;
%
</Table>
</body>
</html>

State Management: With the inner objects, TRequest and TResponse, the state of
the application based on each user, application and server can be tracked and
managed. The using of inner objects can help to get rid of the limit of the stateless
HTTP. For example:

var Request : TRequest;
var Response: TResponse;

Request := TRequest.Create;
Response := TResponse.Create;
if (Request.Session['username'] = '') then begin
  Request.Session['username'] := 'admin';
  Response.AuthRealm := 'Login Please';
  Response.WriteContent;
end
else begin
  var ss: string;
  ss := Response.Session['username'];
  Response.ContentText := ss+'-'++Request.SessionID;
end;

Response.Free;
Request.free;

Component Accessing: The ability to instance and use programmable components
is most important. Ole components can be developed by using such tools as
Visual C++, Visual Basic, Visual J++, Borland Delphi and Powersoft PowerBuilder etc, and these component objects can be accessed by using the inner CreateOleObject function. For example:

```pascal
var msWord: ComVariant;
var doc, sel, tab: ComVariant;

// Create Word 9.0
msWord:=CreateOleObject('Word.Application');

// Show
msWord.Visible := True;
// New Document
doc := msWord.Documents.Add();
sel := msWord.Selection;
// Add Text
sel.TypeText('Part of system information:');
sel.HomeKey(5 {wdLine}, 1 {wdExtend});
// Set format
sel.Font.Bold := True;
sel.EndKey(5 {wdLine}, 0 {wdMove});
sel.Font.Bold := False;
sel.TypeParagraph();

// Create Table
tab := doc.Tables.Add(sel.Range, 3, 2);
// Add text at the table
sel.TypeText('Operating System');
sel.MoveRight (12 {wdCell});
sel.TypeText(msWord.System.OperatingSystem);
sel.MoveLeft(12 {wdCell});
sel.MoveDown(5 {wdLine});
sel.TypeText('Processor');
sel.MoveRight (12 {wdCell});
sel.TypeText(msWord.System.ProcessorType);
sel.MoveLeft(12 {wdCell});
sel.MoveDown(5 {wdLine});
sel.TypeText('Word Version');
sel.MoveRight (12 {wdCell});
sel.TypeText(MsWord.Version);
sel.MoveLeft(12 {wdCell});
sel.MoveDown(5 {wdLine});
```
7.2.1.3 Implementation of Server-side Virtual Machine

The virtual machine in core server can implemented with any object-oriented languages. The description of main object classes of the virtual machine is shown in Table 7.1 and Figure 7.3 shows the logical view between the main classes of the virtual machine.

Table 7.1 Kernel Objects of Server-side Virtual Machine

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TWebScript</td>
<td>The main component in the Web Script. It manages all information needed for the compilation.</td>
</tr>
<tr>
<td>TFFilter</td>
<td>Transforms any kind of text input with Netcase Web Script instructions between special tags into a valid script the compiler will recognize.</td>
</tr>
<tr>
<td>TProgram</td>
<td>Compiled Web Script program.</td>
</tr>
<tr>
<td>TConfiguration</td>
<td>Compiler configuration data.</td>
</tr>
</tbody>
</table>
7.2.2 Client-side Virtual Machine Environment

What the client-side virtual machine takes into account is the interactive capability between the system and user, and access to the database is transparent for the developer, i.e. it is not necessary to install the database interface in the client. The virtual machine needs to be deployed in every client, and it is unnecessary for multi-user accessing.

7.2.2.1 Core Languages

The core languages used in the client are Delphi Script, VB Script and Java Script. Among them Delphi Script is an object-oriented programming language based on the Object Pascal
language.

VBScript is a subset of the Visual Basic language. It is implemented as a fast, portable, lightweight interpreter for use in World Wide Web browsers and other applications that use ActiveX™ controls, OLE Automation servers [16], and Java applets [6].

JavaScript is the implementation of the ECMA 262 language specification. It is a full implementation, plus some enhancements that take advantage of the capabilities of Internet Explorer.

Delphi Script Interpreter allows working with Pascal scripts. Delphi Script Interpreter supports the subset of the Delphi language that may have the following syntax constructions:

- begin-end
- try-except
- try-finally
- if-then-else
- for, while-do, repeat-until
- exit, break, continue
- with
- goto

Delphi Script Interpreter also supports classes, procedures, functions, variables and constants, all logical operations (and, or, not, xor) and other standard Delphi operators. Logical operations are evaluated in short-circuit style. This Delphi language subset can be used to integrate Pascal scripts into the applications.

7.2.2.2 Kernel Functions and Component Objects

The client-side virtual machine supports large-scale applications. The followings are kernel
functions and related component objects of the virtual machine:

- Multi-language Support: The client-side virtual machine supports such development languages as Delphi Script, VB Script and Java Script. All these languages support common component objects. Under the mechanism, mixed programming is implemented in Netcase, which means different programming languages can be used together in the same application. This provides great convenience for the developers.

- Abundant Developing Components: The client virtual machine has implemented 231 basic components involving interface design, data access, dialogs and report print etc. Moreover, the virtual machine can access the ActiveX components developed in other languages. Table 7.2 lists these components:

Table 7.2 Component Classification Lists of Client-side Virtual Machine

<table>
<thead>
<tr>
<th>Classification</th>
<th>Components</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standard(15)</td>
<td>TPopupMenu, TLabel, TEdit, TMemo, TButton, TBitBtn, TSpeedButton, TCheckBox, TRadioButton, TListBox, TComboBox, TScrollBar, TGroupBox, TRadioGroup, TPanel</td>
</tr>
<tr>
<td>Win32(16)</td>
<td>TTabControl, TPageControl, TImageList, TRichEdit, TTrackBar, TProgressBar, TTreeView, TListView, TAnimate, THeaderControl, TStatusBar, TToolBar, TColorBar, TTimer, TMediaPlayer, TOleContainer</td>
</tr>
<tr>
<td>Additional(22)</td>
<td>TFormActivate, TFormClose, TSourceLib, TScriptEvents, TEnvToolBar, TDirectoryBar, TStringGrid, TDrawGrid, TImage, TShape, TBevel, TScrollBar, TSplitter, THeaderListBox, TCheckListBox, TColorListBox, TDFColorFill, TVirtualStringTree, TVirtualDrawTree, TDosMove, TEval, TExcel, TDateTimeFuncs</td>
</tr>
<tr>
<td>Editor(36)</td>
<td>TMaskEdit, TDateTimePicker, TMonthCalendar, TNumberEdit, TSpinEdit, TNumberSpin, TSheet, THistoryEdit, TButtonEdit, TDFEdit, TDFMaskEdit, TDFMemo, TDFDateEdit, TDFButtonEdit, TDFCheckEdit, TDFImageEdit, TDFSpinEdit, TDFButtonClick, TDFCalcEdit, TDFHyperLinkEdit, TDFTimeEdit, TDFCurrencyEdit, TDFGraphicEdit, TDFBlobEdit, TDFMRUEdit, TDFPopupEdit, TDFLookupEdit, TDFEditStyleController, TDFCheckEditStyleController</td>
</tr>
<tr>
<td>Dialogs(13)</td>
<td>TOpenDialog, TSaveDialog, TOpenPictureDialog, TSavePictureDialog, TFontDialog, TColorDialog, TPrintDialog, TPrinterSetupDialog</td>
</tr>
<tr>
<td>Component Access: The programmable ActiveX component can be used in several development languages. Hence these components can be developed using such development tools as Visual C++, Visual Basic, Visual J++, Borland Delphi and</td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>TRI-INTEGRATION PATTERN: IDIOM LEVEL</td>
<td>TFindDialog, TReplaceDialog, TFilterDialog, TSMEWizardDlg, TSMIWizardDlg</td>
</tr>
</tbody>
</table>
Powersoft PowerBuilder etc, and these components can be accessed via inner function or object, such as the CreateOleObject function in Delphi Script, the CreateObject function in VB Script, and the ActiveXObject object in Java Script.

- Accessing core server: By using inner TRConnection object, the developer can make a seamless join between his application and core server and makes full use of many resources and services it provides.

- Data Access: By using TRConnection object and TRemoteQuery object together the databases can be accessed transparently, i.e. the databases can be accessed without installing any data access interface in client.

### 7.2.2.3 Implementation of Client-side Virtual Machine

Client-side Virtual Machine (CVM) allows integrating of different scripting languages into the applications. Among which there are VBScript, JavaScript, or any other active scripts. Besides that, CVM provides Delphi interpreter that makes it possible to write Delphi scripts. CVM delivers the applications to a scripting host and makes the entire Delphi Visual Component Library (VCL) (or any custom Delphi object, procedure, constant, etc.) available to scripts.

CVM has considerable power being able to use VCL in scripts (for example, VBScript), implement event handlers in scripts and even work with scripts through Microsoft Script Debugger.

Any examination of the internal architecture of CVM should begin with its main element, the TScripter component. This component is one that is placed on the form. When working directly with Tscripter, it can hold and manage all the scripting functionality required.

Depending on the script language that is selected in the Language property, TScripter creates an instance of the TRunner class for every script that is used in the main program. For example, an instance of TDelphiScriptRunner is created when DelphiScript is selected; an instance of TVBScriptRunner is created when VBScript language is selected, etc.
As it can be seen from Figure 7.4, all script runners except TDelphiRunner are inherited from TAxScriptRunner. It is significant that the Delphi Script Interpreter that comes with CVM is implemented in a different way to all other script runners. But no difference should be noticed between working with Delphi scripts and other supported scripts. Table 7.3 gives the description of VCM classes.

![Class Diagram](image)

Figure 7.4 Client Virtual Machine Logical View

Please note that to use any Active Script language the corresponding Active Script Engine needs to be installed.

Table 7.3 Main Classes of the Client-Side Virtual Machine

<table>
<thead>
<tr>
<th>Classes</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TScripter</td>
<td>TScripter is the component that allows adding scripting in the applications. It supports DelphiScript, JScript, VBScript, Python and Perl. TScripter executes scripts that can access VCL classes and procedures and all components and data in the applications from any supported language.</td>
</tr>
<tr>
<td><strong>TScriptRunner</strong></td>
<td>TScriptRunner is a base abstract class for executing scripts. It contains methods that implement the basic behaviour of different scriper classes.</td>
</tr>
<tr>
<td>-------------------</td>
<td>----------------------------------------------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><strong>TAXScriptRunner</strong></td>
<td><strong>TAXScriptRunner</strong> is used for active scripting support in TScripter.</td>
</tr>
<tr>
<td><strong>TVbScriptRunner</strong></td>
<td><strong>TVbScriptRunner</strong> is used for VB Script support in TScripter.</td>
</tr>
<tr>
<td><strong>TJavaScriptRunner</strong></td>
<td><strong>TJavaScriptRunner</strong> is used for Java Script support in TScripter.</td>
</tr>
<tr>
<td><strong>TDelphiRunner</strong></td>
<td><strong>TDelphiRunner</strong> is the script runner that processes scripts written in Delphi.</td>
</tr>
</tbody>
</table>

### 7.3 Alias Managing Idiom

**Alias Manager** is a database alias-defining tool. It can be used to connect databases, migrate data between databases and define interface names that are unique names for database access by objects such as TRemoteQuery supported by CVM.

**Alias Manager** has following main functions:

- Connect data source by using three data interfaces such as ADO, ODBC, JDBC;
- Maintain database tables including creating table, editing table, deleting table, defining index and producing database table structure documents;
- Maintain database table records, including querying, adding, updating and deleting for database table records;
- Migrate data including tables and indexes from one database to another database.
- Delete database tables, one or more database tables can be selected and deleted.
- Generate database table structure documents, the structure information of one or
more selected database tables can be automatically created and saved as a WORD document file which comprises the table name, field name, field type, and field length etc.

Figure 7.5 shows the Main Interface of Alias Manager.

![Netcase Alias Manager Interface](image)

Figure 7.5 Alias Manager

Figure 7.6 lists the Implementation Class Diagram of Alias Manager.
7.4 Web Page Designing Idiom

*Web Pages Designer* is a server-side development tool for application components or WEB pages, based on the Web Script language in the Tri-Integration Pattern. It provides powerful support for server-side application extensions and dynamic WEB page design.

*Web Pages Designer* is composed of the following parts:

- **Menu Bar**: Includes many menu items, such as new, open, save, compile, run and system help etc.

- **Tool Bar**: Includes many function buttons such as find, replace, run, and connect to Netcase etc.

- **Editor**: The editor is used to edit Web Script programs or Web pages, which are sensitive to the syntax element of Web Script;

- **View Area**: This is used to display the running result of Web Script program or
WEB page;

- Message Area: This is used to display the error message resulting from compiling Netcase Web Script programs or WEB pages;
- Status Bar: Displays the current file name and related operating information.

Figure 7.7 shows the Main Interface of Web Pages Designer.

Figure 7.7 Web Page Designer

Figure 7.8 lists the Implementation Class Diagram of Web Pages Designer.
7.5 Form Designing Idiom

Form Designer is an object-oriented, visual, rapid application development tool (RAD), which is designed for further development. Not only can simple client applications be implemented with Form Designer, but also the complex and professional client applications, such as Management Information Systems (MIS), Custom Relationship Management Systems (CRM), and Human Resource Management Systems (HRM) etc, can be developed with it.

Since Form Designer provides a reusable component depository and a method using Client Explorer directly for developers, it simplifies application design and development progress. Therefore it reduces greatly the time period and cost of application development.

Form Designer, a visual integrating development environment, is composed of a series of cooperated working elements such as component bar, code editor, window designer, and object inspector, and supports the setting of environment parameters. The following are
descriptions of each element.

➢ Component Bar: The component bar which is a double layer tool bar contains all installed components belonging to Form Designer itself and ActiveX components. Since these Form Designer components are displayed in a pagination and graph manner, the user can design his application conveniently simply by clicking a component and then putting the selected component object into the application form.

➢ Code Editor: The editor window can be used to set up related events and write related codes. What makes it different from other identifiers is the blackening of reserved word and indicative word.

➢ Window Designer: The form may be used to descript kinds of applications. The user interface of applications is only implemented in the form so by selecting a component from the component bar and putting it into the designed form, it can interact with Form Designer. The location and size of a component placed in the form can be changed with the mouse, and the appearance and behaviour of the component can be controlled by using the object inspector and code editor.

➢ Object Inspector: The function of object inspector is editing the component properties. It contains two elements: one is a combo box located in the top, which contains all components in current form and by high-lighting to show the current editing component; the other, under the combo box, is a pages control, which is used to display the properties of the current component. The property page shows the visual properties of the current component, such as size, colour, and font, etc., and the property page is divided into two columns to display the component properties, the left hand column, called the caption column, lists the property title while the right hand column displays the property value and is accordingly called the value column.

Figure 7.9 shows the Main Interface of Form Designer.
Figure 7.9 Form Designer

Figure 7.10 lists the Implementation Class Diagram of *Form Designer*. 
7.6 Flow Designing Idiom

*Flow Designer* is a visual flow-design tool for the Tri-Integration application framework through which most kinds of resources (e.g. form, sheet, template, and program control, etc.) in the framework can be integrated into the flow, and then various complex workflow applications can be implemented easily.

*Flow Designer* provides the following functions:

- Operations of workflow file, for example, open, save, and save as, etc.
- Creating for workflow object and connection symbol, for example, create object, create connection.
- Editing for workflow object and connection symbol, for example, copy, paste, cut, and delete, etc.
- Union operations for workflow object, for example, new union, clear union, add to
union and remove from union, etc.

- Setting general properties for workflow objects and connection symbols, for example, text, shape, colour, direction, font, layout, align, and frame border, etc.

- Setting accessory properties for a workflow object, for example, users, form, sheet, template, resource limit, display order and node control, etc.

Figure 7.11 shows the main interface of Flow Designer.

![Flow Designer Interface](image)

Figure 7.11 Flow Designer

Figure 7.12 lists the Implementation Class View of Flow Designer.
7.7 Sheet Designing Idiom

*Sheet Designer* provides a powerful design tool for designing tabular input and output interfaces for data information. With the sheet designer, the user can design complex sheets using many elements, such as combo box, image sign, text edit, rich text and URL label, etc., and can put the designed sheet into a form application or a workflow control. Alternatively, the designed sheet can be saved onto disk in order to be used directly while printing a report.

*Sheet Designer* provides the following functions:

- Operation of sheet file, for example, open, save, save as, and print etc;
➤ Editing for sheet content, for example, copy, paste, cut, and delete etc;

➤ Union operations for cell, for example, new union, intercross union, clear union, add to union and remove from union, etc.

➤ Setting general properties for cell, for example, text, colour, font, layout, align, frame, and editing direction etc;

➤ Setting types for cell, for example, character string, digital, date, sign, rich text, image, URL address, and formula, etc.

Figure 7.13 shows the Main Interface of *Sheet Designer*.

![Sheet Designer Interface](image)

Figure 7.13 Sheet Designer

Figure 7.14 lists the Implementation Class View of *Sheet Designer*. 
7.8 Client Exploring Idiom

*Client Explorer* is a client-side application-integration platform for the Tri-Integration application framework, which can integrate resource processing and collaboration as a whole. *Client Explorer* can perform resources organisation, information publication, daily affair management and schedule arrangement, etc. Meanwhile, the user can define reasonable workflow and management patterns to satisfy the different demands. *Client Explorer* advances the applications by resource management, workflow management and
authority management. Its main functions are as follows:

➢ **Resource Management:** Includes personal information management, group information management and classification information management, etc. The user can create a special resource to satisfy different need in *Client Explorer*. Through authorisation, the user can publish and read the information resource. Furthermore, some special resources can be opened to some users; even maintenance authority of a special resource can be authorised to a special user. In addition, *Client Explorer* may be used to define functional resources such as form resource, schedule resource, web site resource, and e-mail resource.

➢ **Authority Configuration:** Seven access-control authorities (deny, read, send, create, edit, delete, and full control) are available for controlling various resources. The system provides a flexible authorisation mechanism that ensures every user has its strict authority and each different authority corresponds to a different process ability. Therefore, strict and flexible authority management is implemented in *Client Explorer*.

➢ **Flow Designing and Business Process Delivering:** *Client Explorer* contains *Flow Designer* thus enabling an authorised user to define a flow himself and save it into the flow resource for any authorised user selecting and delivering it.

➢ **Sheet Design:** *Client Explorer* also contains *Sheet Designer*. The authorised user can define a sheet and save it into the sheet resource for selection by an authorised user.

➢ **Document Delivering and Approval:** *Client Explorer* can deliver a document in the network and correctly send the document to the appointed user. The receiver can reply, read and approve the document. *Client Explorer* also provides a monitor to notify a user about the arrival of his new message. Different process workflows can be defined for document delivering. The document-delivering route can be controled according to the workflow being used, and can sign, update and the
document can be approved according to the user’s authority. As document processing is a daily event in an office, the system can keep a ‘read and approval’ record on the document.

➢ **Information Search:** By taking title, content, scope, and time as searching conditions, you can search information in *Client Explorer*, even full-text-search strategies are available for everyone to find information conveniently.

➢ **Data Access and Share:** *Client Explorer* also provides the *Query Builder* tool, which enables an authorised user to define an SQL query himself and save it into the query resource for other authorised users to select and send it to other users or related resources.

➢ **Customised Configuring and Management:** *Client Explorer* provides the method for every user to set his individual Netcase client-side desktop, for example, to set colour, icon and background images, etc. In addition, the Adjusting Messages tool can be used to classify user messages and delete or backup these messages conditionally.

Figure 7.15 shows the Main Interface of *Client Explorer*
In This Month's Issue:

---------------------

1) Altova Announces Release 4 of 2004 Product Line
- New release available for immediate download

Figure 7.15 Client Explorer

Figure 7.16 lists the Implementation Class View of Client Explorer.
7.9 Summary

It is an important aspect of what that enables several system tools to enhance the practicability of the Tri-Integration Pattern application framework. The eight idioms are also system tools in the Tri-Integration Pattern. These tools include Resources Manager which supports resources' authorisation and properties' accessing, Alias Manager which supports database linking and configuration, Web Page Designer which supports development of business logic and dynamic web pages, Form Designer which supports multiple language programming, Flow Designer which supports the visual design of flow charts, Sheet Designer which supports the visual design of sheet and Client Explorer which supports the client application integration and the client-side virtual machine.

The Tri-Integration Pattern provides an engine that may be applied to the management of various information resources. The engine can be used to organise and share the internal
and external information resources of an enterprise according to different user requirements and authorities. It can be used to implement the seamless joining and integration of various information resources. Furthermore, external data and information can be obtained by e-mail or an embedded browser connected to the Internet. Thus, a fully-individual and content-tight application has become available to users through the Tri-Integration Pattern.

The Tri-Integration Pattern can access different database systems, exchange data, share information and implement remote maintenance through using standard data interfaces such as ADO, BDE, IBX, etc., accepted by the interrelated enterprises.

The Tri-Integration Pattern provides a powerful method for various information deployments. As a development framework, the Tri-Integration Pattern integrates many programming languages (including Delphi Script, VB Script, and Java Script) into its visual development environment. Thus, not only can it be used to design complex applications to meet different needs for different users but also the designed application can be presented in a uniform form and deployed to the interrelated user through the Tri-Integration Pattern. The Tri-Integration Pattern provides a reusable component depository and methods of directly using these components in Client Explorer, which simplifies the design and development process of applications and makes various applications developed easy to maintain and reuse.
Chapter 8  Implementation of Netcase Anywhere and Case Studies

In this chapter, the implementation of Netcase Anywhere software platform is presented firstly, and then three case studies are given to demonstrate how to use Netcase Anywhere solution framework to instantiate specific domains.

8.1  Implementation of Netcase Anywhere

Netcase Anywhere is a software platform whose implementation is based on the Tri-Integration Pattern. The original intention of constructing Netcase Anywhere solves the same development problems as the Tri-Integration Pattern. Netcase Anywhere has realised the entire application framework of the Tri-Integration Pattern and all elements from architecture level to idiom level.

Additionally, Netcase Anywhere makes also necessary extensions in some details, for example, making it easy to install, use and maintain and to realise production configuration and tight integration between various tools. The Netcase Anywhere special features are described as the following:

➢ Operating System: the design of Netcase Anywhere is based on Windows operating system now, but its core server can be designed to support major operating systems, such as UNIX and LINUX, because its implementation is based on MRTP which does not depend on any specific operating system.

➢ Middle-ware Software. The core server of Netcase Anywhere is also an application server, i.e. a middle-ware software. It is designed as a Windows service [37], which may be started during the Windows start-up.

➢ Database Access. Microsoft Advance Data Object (ADO) is adopted as the data interface to connect various databases, and the latest edition of ADO can be
obtained from the Microsoft web site.

- Resources Storage. In the current version of Netcase Anywhere, the resources are organised in a tree structure, and are stored using the file directory mode which is supported by Windows operating system.

At present, Netcase Anywhere as a mature product has been put onto market and accepted by hundreds of enterprises. It is also accepted as an application system building tool and development tool by many developers.

8.2 Case Study 1: A Workflow Process System

8.2.1 Background

Workflow control technique developed rapidly with the tide of Business Process Reengineering (BPR) in the early 1990s. The Workflow Management Coalition (WfMC) [138] was established in August 1993, which indicated that workflow techniques were entering a relatively mature stage. At the present, the WfMC has over 300 member organizations worldwide, representing all facets of workflow, from vendors to users, and from academics to consultants.

Workflow is the computerised facilitation or automation of a business process, in whole or part. It is concerned with the automation of procedures where documents, information or tasks are passed between participants according to a defined set of rules to achieve, or contribute to, an overall business goal.

Process integration uses computer software tools to support integration, e.g. workflow management systems, to realise the ideal of data and resource sharing and collaboration among various applications. The advantage of process integration lies in separating application logic and process logic whilst also separating process modelling from concrete data and function. Consequently, enterprise can modify operation flow and administration management flow very easily without affecting practical applications. This makes it very
convenient for enterprises to rebuild operation flows in order to adapt to the needs of
market competition. Process integration is the trend of current software products.

Workflow techniques pay attention to the optimisation of operation flows, the realisation of
operation flow automation and the integration of current applications. Workflow techniques
can bring following advantages:

➢ Improving efficiency: automation of operation processing can remove unnecessary
steps;

➢ Better process control: realise improvement of operation-flow management using
standard work methods and effective audit trails;

➢ Improving response ability of client services: improve predictability of client
response ability;

➢ Quick flexibility: workflow can be redesigned according to the changes in business
needs;

➢ Improving operation flow: make operation flow simple and streamlined.

8.2.2 Workflow Model

A workflow management system is a system that completely defines, manages and
executes "workflows" through the execution of software whose order of execution is driven
by a computer representation of the workflow logic [64]. It, instead of being a business
system in an enterprise, mainly provides a software environment to support business
systems in the enterprise. It can support the execution of enterprise management processes
only by integrating practical business-application software with operational interfaces.

Netcase Workflow is a centralised workflow management system based on the workflow
reference model of the WfMC and which is designed and developed according to the
practical situation. The model of Netcase Workflow is shown in Figure 8.1:
8.2.3 Workflow Engine

Workflow engine interprets and executes the business information, in whole or in part, as defined in a designed flowchart. By interacting with external applications, the engine implements the creation, execution and management of workflow instances, and then produces work items or tasks, notifying the user to take appropriate action.

The workflow process system can be divided into single engines and multiple engines based on different workflow engines it adopts. Single engines adopt a workflow engine to deal with all workflow, while multiple engines use several workflow engines to deal with workflow cooperatively. When several workflow engines deal with a flow instance cooperatively, the control data of the flow instance must be understood by all workflow engines, and the control data is produced by the main engine or a combination of the data is produced by several engines. Accordingly, workflow definition and flow supervision can be processed by one or more engines.

The workflow process system of a multiple engine has better extensibility and can satisfy
the needs of medium to large sized enterprises, but it is difficult to implement and expensive. The single engine flow management system is easier to realise and especially suitable for medium to small sized enterprises. Also its price is more attractive.

*Netcase Workflow* is a kind of centralised, single engine flow management system. The flow engine of *Netcase Workflow* is used to manage various resources and call diverse databases, and provides interfaces to components of flow monitor, flow designer, client application and resources management.

There are two cases to process application data (see section 6.2.5):

1) Work that needs be accomplished by a participant in the workflow.

2) Where the system needs to process automatically work that allows the flow to continue.

For the former, for each activity of the flow, the *Netcase Workflow* engine decides which work must be accomplished first in the activity and then decide the participants of this work at the same time. In the latter, the system calls relevant external applications to be processed. These applications are components of an “invoked application”.

### 8.2.4 Netcase Solution Framework

*Netcase Workflow* is an aggregation of a series of relevant software and models. Considering the high layer functions, they can be divided into Build-time and Run-time functions. See Figure 8.2:
Figure 8.2 Workflow Model Level

Build-time Functions: The Build-time functions are those which result in a computerised definition of a business process. During this phase, a series of modelling tools have been included such as Flow Designer, Sheet Designer, Form Designer, Resource Manager, etc.

Run-time Process Control Functions: These run-time process control functions act as the linkage between the process as modelled within the process definition and the process as it is seen in the real world, reflected in the runtime interactions of users and application tools. The Netcase Anywhere Server is the basic workflow management control software (or "engine"), responsible for process creation and deletion, control of the activity scheduling within an operational process and interaction with application tools or resources.

Workflow Enactment Service: A software service that may consist of one or more workflow engines in order to create, manage and execute workflow instances. Applications may interface to this service via the workflow application-programming interface (WAPI).

The MRTP is a WAPI in the Netcase Anywhere/Explorer software platform.

According to the above function model in combination with Netcase Anywhere, the view of the system realisation of the Workflow Process System is reflected in Figure 8.3. This demonstrates that Netcase Anywhere can be a solution of a Workflow Process System.
8.3 Case Study 2: An E-Government System

8.3.1 Background

Few technologies have revolutionised business more than the advent of the Internet. Since the mid-nineties, organisations all over the world have come to realise that the Internet’s true value is not in people’s ability to browse the Web or send e-mail, but rather, in the new opportunities it creates for enhancing business processes, reducing costs and delivering better services.

E-government is the application of e-business to government. It is not as simply defined as e-commerce transactions; it is about using technology to redefine organisational models in order to extend relevance and maximise value. It is the realm of technology-enabled transformation.

With the evolution of e-business technology, non e-government models and organisational structures will be faced with increasing pressures. People used to say that government had no competitors, but that was clearly wrong. Government’s competitors exist in the shape of
other governments and jurisdictions, intermediaries, private sector providers, etc.

These competitors will exploit new technologies as much as possible. New business models are emerging, along with enhanced constituent experiences, which offer alternatives and weaken a particular government’s “mind share”. In order to compete in this new era, governments at all levels must be able to react quickly to challenges; constantly innovating their processes to stay relevant in meeting public service expectations and priorities.

To accomplish this, it is becoming imperative for government or organisations to build technical infrastructures flexible enough to absorb new technologies quickly and rapidly to alter the scope and function of applications to support changes in the government business model.

This example investigates the requirements e-government places on Information Technology (IT) infrastructures, and provides guidelines for creating an infrastructure that offers the flexibility and reliability necessary to support the constant evolution of processes in the e-government world.

### 8.3.2 E-Government Model

E-Government is considered as a network-based information service and information process system, that accords with technology standards of the Internet and faces to the government's interior, other government institutions, enterprises and the public.

- **E-government** is a comprehensive information system.

- **Services objects**: E-government includes internal government departments and other departments, groups, enterprises and public.

- **Processes objects**: E-government includes internal information concerning government departments, information communicated in certain areas and information published openly.

- **Process methods**: E-government process workflow simulation, cooperation and
information publishing accepts all kinds of applications, complaints, suggestion and important files including not only information publishing and incepting, but also interactive processes.

Thus it can be seen that constructing an e-government need a solid, reliable, controllable platform following Intranet/Internet standards. The platform should also provide the following services:

➢ Abundant mechanisms for data access, storage, operation and management provide comprehensive data processing ability for e-government.

➢ Application development platform: applications of e-government are different from the traditional data centralised computing. The former focuses on extensive flows, involves a great deal of people and data, needs great amount of interaction and harmony, lacks standard models and has constant flow changes. Using traditional development methods and tools to develop this application can be inefficient and difficult to maintain.

Working together with government, helping government to extend information services to other government departments, enterprises and public:

➢ Establish interaction with external users of government, based on a uniform technical platform.

➢ Integrate with traditional technology and systems.

➢ Rebuild core business flows to provide faster and more effective services.

➢ Make full use of the knowledge of government staff; use Netcase Anywhere/Explorer to realise services for public.
Figure 8.4 E-Government/E-Commerce Model

Figure 8.4 shows the location and environment of e-government applications in entire e-commerce [86]. E-government applications are typical cooperating applications. The further model based on the Netcase Anywhere/Explorer is shown as Figure 8.5:

Figure 8.5 Netcase E-Government Architecture
8.3.3 Netcase Solution Framework

*Netcase Anywhere* successfully integrates the technology of the Internet/Intranet/Extranet applications. The following Netcase Solutions Framework describes nine different types of e-government application solutions based on Netcase Anywhere/Explorer.

![Netcase Solutions Framework Diagram]

Figure 8.6 Netcase Solutions Framework

The following shows how the nine applications are realised using in *Netcase Solutions*:

- **Workgroup Communication**: improves the efficiency of information exchange between workgroup members, such as information transfer, e-mail, and calendar arrangement.

- **Enterprise-Wide Communication**: spans the boundary of departments in an enterprise, provides a complex information transfer system for the whole enterprise. For example, transfer of information between resources based on authority.

- **Cross-Enterprise Communication**: transfers messages, exchanges electronic files and publishes special data between enterprises.
Workgroup Collaboration: utilises on-line discussion, shares resources and common applications to bring together skills within the workgroup to explore options and reach collective decisions.

Enterprise Knowledge Management: spans the boundaries of departments in an enterprise, making good use of the enterprise's various intelligent resources and avoiding various valuable experiences and expertise being covered up by individuals or departments within the enterprise. Integrates these intelligent resources with the working environment of the enterprise to form "knowledge" that can be used to guide staff practical work and training. For example, corresponding resources can be defined or customised according to knowledge characteristics, and can also be integrated as part of the total business information system.

E-community Development: spanning the boundary of the enterprise, entities that have common interests in some aspects form a virtual electronic common space that is used to serve each participant. For example, checking contract rules on-line and distributing application components.

Workgroup Innovation: applies information flows and knowledge flows within the workgroup into workflow to create new working and flow or improve current working and flow. For example, automation of sales.

Enterprise Process Innovation: also spans departmental boundaries in the enterprise, redesigns workflow to decrease breakdown time, avoid redundant and incompatible objectives and promote mechanisation. Many enterprises are rebuilding product-manufacturing flow "from concept to market" to make it more effective and rapid. For example, checking projects automatically.

Value Chain Innovation: extends the enterprise-level flow innovation beyond the enterprises, begins with the social and economic value chain of the enterprise, rebuilds workflow between enterprises and brings benefits to all enterprises. For example, checking contracts and automatic subscriptions.
8.4 Case Study 3: An EAI System

Enterprise Application Integration (EAI) is a rapidly growing segment in enterprise computing [49]. Multiple forces are driving this trend. Many of them center on rapidly changing business needs and opportunities driven by the Internet, merger/acquisitions, competition, regulation and deregulation. Increased attention on Customer Relationship Management (CRM) and supply chain integration in many industries is driving the demand for intra- and inter-enterprise system integration.

8.4.1 Netcase Solution Framework

Five EAI patterns have been advanced [88]. All the patterns share the goal of decoupling applications. The patterns differ, however, in the level at which decoupling occurs. The integration adapter decouples at the interface level and the integration messenger at the communication level. Both the integration façade and integration mediator decouple at the application level. Proper decoupling at the architectural level is essential for robust, flexible EAI solutions.

The intent of each pattern and its Netcase implementation framework are as follows:

- Use the integration adapter to convert an existing application interface to a desired interface. Figure 8.7 shows the Netcase implementation framework for the integration adapter pattern.
Figure 8.7 Netcase Framework for Integration Adapter Pattern

Use the integration messenger to minimise communication dependencies between applications. Figure 8.8 shows the Netcase implementation framework for integration messenger pattern.

Figure 8.8 Netcase Framework for Integration Messenger Pattern

Use the integration façade to provide a simplified interface to back-end applications, minimising dependencies between client and server applications. Figure 8.9 shows the Netcase implementation framework for the integration façade pattern.
Use the integration mediator to encapsulate the application interaction logic, minimising application dependencies. Figure 8.10 shows the Netcase implementation framework for the integration mediator pattern.

Use the process automator to minimise dependencies between process automation logic and IT systems. Figure 8.11 shows the Netcase implementation framework for the process automator pattern.
Figure 8.11 Netcase Framework for the Process Automator

To sum up, the Tri-Integration Pattern can be a powerful ally to EAI development organisations in the battle against complexity in application integration within an enterprise. Tri-Integration Pattern helps reduce integration complexity and improve the chances for success in several ways:

- Robust EAI architectures can be created quickly by using Netcase framework based on the Tri-Integration Pattern.
- Tri-Integration Pattern can provide a framework for evaluating the fit of a tool and vendor for a given EAI problem and to identify tool strengths and weaknesses.
- Tri-Integration Pattern can serve as a framework for establishing EAI standards in the enterprise.

8.5 Summary

The Netcase Anywhere solution for a workflow process system demonstrates that the Tri-Integration Pattern support solving the workflow control problem of an application system. Its solution framework shows how the elements of a workflow may cooperate and Netcase Anywhere may be used to build a workflow process system with a single workflow engine.
The solution for an E-Government system demonstrates that the Tri-Integration Pattern helps solve the cooperation problem of an application system. Its solution framework describes how Netcase Anywhere may be used to build nine different types of e-government applications.

The solution for an EAI system demonstrates that the Tri-Integration Pattern supports solving the integration problem of an application system. Its solution framework describes how Netcase Anywhere is used to perform integration. The Tri-Integration Pattern provides a solid technical foundation that significantly reduces complexities and risks inherent in EAI.

Because of its capabilities in data access, application components development and supporting various services, Netcase Anywhere can also provide solutions for Management Information Systems (MIS) and other network application systems.

In conclusion, the Tri-Integration Pattern can solve a range of development problems at system level. Its solution framework is effective and applicable to building large-scale application systems.
Chapter 9  Conclusions

9.1  Remarks

9.1.1  Contrast of Three Software Platforms

In the light of the Tri-Integration Pattern, the following contrasts three software platforms - Exchange Server/Outlook, IBM Lotus Domino/Notes, and Netcase Anywhere/Explorer.

Exchange Server 2000 is the Microsoft messaging and collaboration server designed to support effective communication. Along with the rich client-functionality provided by Microsoft Office Outlook® 2000, Exchange Server 2000 offers [94]:

- Mobile, remote, and desktop e-mail access with state-of-the-art security and privacy.
- Lower cost of ownership through the services provided by Microsoft Windows Server.
- High reliability and outstanding performance.
- E-mail-based collaboration.
- Easy upgrading, deployment, and administration.

IBM LotusDomino/Notes products enable customers to quickly and easily build secure and reliable messaging, calendaring and scheduling along with collaborative application solutions. It has the following features [65]:

- Enterprise-grade messaging server for companies of all sizes.
- Open, secure Web application server.
- Reliable, available and scaleable server infrastructure.
- Simple, flexible administration.
Innovations in collaboration, discovery tools.

Localisation Features.

Netcase Anywhere/Explorer is an integrated expandable platform that can capture, compose, share and utilise resources. This platform also has the ability to manage and provide services at all levels and support workflow and complicated S/W development [144] requirement. Its main features are described as the following:

Information integration tool: this is used to organise and share internal and external information resources based on different user requirements and authority for seamless connection of all kinds of information and accordingly provide users with fully-individualised and context-sensitive application.

Data management platform: This is used for data maintenance, sharing and distribution in databases, such as Sybase, Oracle, Ms SQLserver, Informix and DB2, through query building and data form-designing which abide by the strategies of authority control.

Workflow control is supported and a visual flow designer has been provided.

Development platform: Multiple languages (Delphi Script, VB Script, Java Script, Perl, Python etc.) can be integrated within a visual environment. By this platform, complicated network applications can be designed for users with different requirements and different levels that can also be distributed to users as united forms by the system.
### Table 9.1 Contrast of Three Software Platforms in the Tri-Integration Pattern

<table>
<thead>
<tr>
<th>Data Integration</th>
<th>Exchange Server &amp; Outlook</th>
<th>Lotus Domino &amp; Notes</th>
<th>Netease Anywhere &amp; Explorer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Resources Organisation</td>
<td>File type and functional type</td>
<td>File type and functional type</td>
<td>File type and functional type</td>
</tr>
<tr>
<td>Resources Storage</td>
<td>Microsoft Web Storage</td>
<td>Domino documentation database</td>
<td>OS directory or Relational Database</td>
</tr>
<tr>
<td>Authority Control</td>
<td>Integrating with Windows OS</td>
<td>Build-up authority control system</td>
<td>Authority Control system Based on MRTP</td>
</tr>
<tr>
<td>Information Transmission</td>
<td>Transmission based on email and sharing folder</td>
<td>Transmission based on email and documentation database</td>
<td>Transmission based on MRTP</td>
</tr>
<tr>
<td>Information Search</td>
<td>Full-text search based on Web storage</td>
<td>Full-text search based on documentation database</td>
<td>Full-text search based on OS directory and relational database</td>
</tr>
<tr>
<td>Workflow Engine</td>
<td>CDO for Workflow Objects</td>
<td>Lotus Workflow Engine</td>
<td>MRTTP Workflow Control</td>
</tr>
<tr>
<td>Database Access</td>
<td>ADO</td>
<td>Domino Connector</td>
<td>ADO&amp;BDE&amp; Native Connector</td>
</tr>
<tr>
<td>Web Server</td>
<td>Internet Information Server</td>
<td>Domino Application Server</td>
<td>MRTP(HTTP Support)</td>
</tr>
<tr>
<td>Web Services</td>
<td>Integrate with .Net</td>
<td>Integrate with WebSphere</td>
<td>MRTTP(SOAP Support)</td>
</tr>
<tr>
<td>Service Interface</td>
<td>CDO</td>
<td>Lotus API</td>
<td>COM+ LIB.</td>
</tr>
<tr>
<td>External Integration</td>
<td>NetMeeting MSN Messenger SQL Server</td>
<td>Lotus Enterprise Integrator</td>
<td>COM+ Web Services</td>
</tr>
<tr>
<td>Client-side Core Language</td>
<td>Microsoft Visual Basic Scripting Edition</td>
<td>Lotus Script</td>
<td>Delphi Script &amp; VBScript &amp; JavaScript</td>
</tr>
<tr>
<td>Client-side VM</td>
<td>VBScript VM</td>
<td>Lotus Script Interpreter</td>
<td>Netease VM</td>
</tr>
<tr>
<td>Client-side IDE</td>
<td>Outlook IDE</td>
<td>Lotus Script IDE</td>
<td>Netease Form Designer</td>
</tr>
<tr>
<td>Application Component Distribution</td>
<td>Outlook Forms</td>
<td>Lotus Script Application</td>
<td>Netease Forms</td>
</tr>
<tr>
<td>Server-side Core Language</td>
<td>Visual Studio</td>
<td>Java</td>
<td>Delphi Script</td>
</tr>
<tr>
<td>Server-side VM</td>
<td>ASP&amp;ASP.NET</td>
<td>Java VM</td>
<td>Delphi Script Runtime</td>
</tr>
<tr>
<td>Server-side IDE</td>
<td>Visual Studio IDE</td>
<td>Java IDE</td>
<td>Netease Web Designer</td>
</tr>
<tr>
<td>Workflow Design</td>
<td>Workflow Designer</td>
<td>Lotus Workflow</td>
<td>Netease Workflow Designer</td>
</tr>
</tbody>
</table>

### 9.1.2 Comparison with Four Network Supported Patterns

The Broker Pattern [21] is suitable for distributed software systems that consist of
interacting and decoupled components. In the Broker Pattern clients access the services provided by servers using remote procedure calls or message-passing. In the Tri-Integration Pattern, application components are similar to clients in the Broker Pattern and core services to brokers. The request from application components to core services is processed through the MRTP.

**Client-Dispatcher-Server Pattern [21]** introduces an intermediate layer between clients and servers, the dispatcher component. It provides location transparency by means of a name service, and hides the details of the establishment of the communication connection between clients and servers. In the Tri-Integration Pattern, core services act as the dispatcher, but offer services rather than servers. It establishes communication connections between each client and service provider. For example, to access resources, databases and Web services by a named service.

**Microkernel Pattern [21]** applies to software systems that must be able to adapt to changing system requirements. It separates a minimal functional core from extended functionality and customer-specific parts. In the Tri-Integration Pattern, core services can be regarded as a microkernel system used to implement access between client and server. Its implementation is the key to services integration. The virtual machine environment, including core language, atomic components and virtual machine, is the microkernel system of application extensions. Most application components are designed and developed in this way. Therefore, the virtual machine environment is vital for component integration.

**Layers Pattern [137]** helps to structure applications that can be decomposed into groups of subtasks in which each group of subtasks is at a particular level of abstraction [58, 105]. The relationship between the Layers Pattern and the Tri-Integration Pattern is twofold. Firstly, a Tri-Integration system may also be considered as a variant of a Layers Pattern. Various resources are the lowest layer; the core service is a transaction system built on the resources, representing the layer on top of the resources. Application components constitute the top layer. Secondly, for some application domains both patterns may be applied alternatively. Consider architectures for business applications [48]. A very common
approach is to separate these systems into three layers:

- The lowest layer includes database management system (DBMS).
- The middle layer contains the business logic.
- The highest layer comprises different business applications.

As employing the Tri-Integration pattern to set up business applications based on such an architecture, the following methods can be advised. Firstly, the database is regarded as a resource, and DBMS is connected with a defined interface. Secondly, central business logic is implemented by the core language that is integrated into the core services, or by any programming languages with certain components, such as COM+, that can be deployed on the server and called by the core language. Lastly, client-side application components are developed to implement business applications. If, however, all clients build upon the same view of the underlying business logic, then the Tri-Integration Pattern should not be applied.

9.2 Evaluation

The evaluation of the system pattern, Tri-Integration Pattern and the software platform is listed as following:

- There is no researcher to propose the concept - system pattern. To declare this point, the following journals of most recent years are searched manually:

  - IEEE Transaction on Software Engineering
  - IEEE Software
  - ACM Transaction on Software Engineering and Methodology
  - ACM Software Engineering Notes, BIDS Database and Internet.

The following BIDS databases are also searched:

  - Science Citation Index (SCI), which includes all the international and
important national journals of most recent ten years.

- Index to Scientific and Technical Proceedings, which includes all the proceedings of international conferences of the most recent ten years.

Internet is searched thoroughly through the following agents:

- Goole
- Excite
- Infoseek
- Yahoo

The Tri-Integration Pattern is a system pattern. It is not invented or created artificially. Rather it builds on the design knowledge gained by experienced software engineers in the network application systems development.

An effective and applicable software platform, Nectase Anywhere, has been implemented successfully. The Tri-Integration Pattern is a well-design pattern and helps to support application development and integration.

At present, more than six hundreds organisations have used Nectase Anywhere as their application platform in China. For example:

- Enterprises: Yunnan Hongta Group which is one of bigest enterprise group in China, Yunnan Honghe Tobacco Factory, etc.
- Governments: Yunnan Finance Office, Yunnan Education Office, etc.
- Universities: Shanghai Jiao Tong University, Guangdong Zhong Shan University, Yunnan University, etc.
9.3 Assessment of Success and Analysis

9.3.1 The Pattern

In Chapter 1, a set of criteria is proposed to judge the success of the Tri-Integration Pattern described in this thesis. In this section, detailed analyses of the pattern are presented based on these criteria.

➢ Is the problem concerned by the pattern universal? Is its resolution efficient?

Service integration, component integration and data integration are usually involved in an enterprise application integration system. To resolve ‘isolated islands of information’ which existed widely in enterprises, necessary integration should be carried out. Maintenance and evolution of systems are inevitable for enterprise application systems to keep up with development demands. That is to say, in order to fulfil the business ability of the original system with the most advanced techniques platform and solution, integration must be the only way for an enterprise application system. The thesis presented the Tri-Integration Pattern to resolve the recurring integration problem, discussed the solution extensively and confirmed that the solution is feasible and effective through implementing a custom software platform – Netcase Anywhere – based on the Tri-Integration Pattern.

➢ Does the pattern support the development of complex, large-scale systems?

The Tri-Integration Pattern is a system pattern. Although it is not bound to any particular enterprise size, the pattern is most concerned with large scale integration for enterprise application systems. Techniques employed by the pattern, for example, the MRTP (supporting various services), the virtual machine (supporting the extension of system functionality), the data access for almost relational databases and the policies of resource authorisation and management, are beneficial to large scale system integration.

➢ Is it possible to build a common software platform based on the pattern?

The Tri-Integration Pattern involves some common problems of service integration,
component integration and data integration, which exist in many fields. A common platform to resolve these comprehensive integration problems, *Netcase Anywhere*, has been implemented based on the pattern.

\[ \textbf{Does the pattern support effective industrial software production?} \]

Through simple configuration, *Netcase Anywhere* based on the Tri-Integration Pattern can be employed as an enterprise application platform. Through designing and configuring application components, enterprise business applications can be constructed rapidly. Application components can also be maintained and developed independently in terms of requirement changes. The application development or system extending methods are valid in practical implementation; therefore industrial software production is effectively supported.

\[ \textbf{Does the pattern help to manage software complexity?} \]

The Tri-Integration Pattern provides an effective and practicable approach that can be used to integrate services, components and data. When you encounter a concrete design situation covered by the pattern there is no need to waste time inventing a new solution to your problem. If you implement the pattern correctly or use the software platform based on the pattern, you can rely on the solution it provides. Therefore the pattern can help you to manage software complexity.

### 9.3.2 The Platform

In this part, according to a series of criteria, the software platform—*Netcase Anywhere*—based on the Tri-Integration Pattern is evaluated.

\[ \textbf{Integrity} \quad \textit{Netcase Anywhere} \enspace \text{possess high integrity because it adopts a uniform protocol, cooperates with a uniform interface and constructs with a uniform method.} \]

\[ \textbf{Adaptability} \quad \textit{Netcase Anywhere} \enspace \text{allows adaptability. Because the software platform is constructed using various services, such as Internet protocols, database access, information exchanges and workflow applications, it can be applied to information} \]
management, business process control, E-business, E-government, decision support and so on.

**Extensibility.** Various application components can be developed by *Netcase Form Designer* and *Netcase Web Pages Designer*, and deployed by resource management according to certain authority. Additionally, application components can be embedded in flow, and provide more complex flow control. It means that the extensibility of applications constructed using the system based on the pattern is powerful. If a core server is not enough to satisfy system requirements or system efficiency, more core servers can be deployed to further support the system. The integration of several core servers which can be used to share resources and data, and form distributed application architecture is of more broad value.

**Reliability.** Two issues are important in achieving reliability: availability and fault tolerance [137]. A distributed *Netcase Anywhere* supports availability, because it allows *Netcase Anywhere* to be run on more than one machine. If a *Netcase Anywhere* or a machine fails, therefore, the failure does not necessarily have an impact on an application. Fault tolerance may be easily supported because distributed systems allow you to hide failures from a user.

**Security.** Firstly, the MRTP adopted in *Netcase Anywhere* is based on authority control. Secondly, information transmission within *Netcase Anywhere* is encrypted. Thirdly, *Netcase Anywhere* can be integrated with a Certificate Authority (CA) server, which makes the system possess the basic characteristics of CA, i.e. Authentication, Integrity, Nonrepudiation and Confidentiality.

**Transparency.** In *Netcase Anywhere*, organisation and authorisation of resource, integration of core services, connection of databases, control of workflow and the detail of designing application components are hidden from clients. In other words accessing and applying the system is transparent to clients.

**Interoperability.** A good system can also realise interactive operation with the systems
based on a host machine or LAN. The MRTP ensures that Netcase Anywhere can completely satisfy these demands. It can be interconnected and coexist with other Mail systems or Web systems, and share information or exchange data with file systems.

9.4 Conclusions

9.4.1 Lessons Learnt

Through developing the approach, the following lessons have been learnt:

Resource organisation — The basis of the Tri-Integration Pattern is resources. Data integration is also based on resources. Therefore, the pattern construction is affected by the resources constituent model, storage structure, authorised access and management. Different resource organisation models directly impact the complexity and flexibility of the pattern construction.

MRTP — The MRTP is the core of the Tri-Integration Pattern. Service integration is based on it. Thus, the definition, command constituent, and authority control of the MRTP are extremely important to decide the level of service integration. This directly relates to the feasibility of the pattern implementation.

Virtual machine implementation — Core language and virtual machine implementations are the basis for application extension using the Tri-Integration Pattern. Component integration depends on virtual machine implementation. It plays a core role in supporting application component development, extension and maintenance that are the constituents of the atomic components in the virtual machine. This directly affects the extensibility of the pattern implementation.

Tools integration — Powerful tools are important for a pattern to be useful in supporting complex, large-scale system development. It involves many application and development tools such as resource management, component development, flow definition and form design. Tools integration directly affects the practicability of the pattern implementation.
Software platform based on the pattern — The pattern presents an effective and practicable solution to deal with development problems in specific application contexts. Constructing a common software platform based on the pattern is helpful to support effective industrial software production, shorten the software development cycle, improve software quality and to manage software complexity.

9.4.2 Conclusions

The features of the proposed system pattern, Tri-Integration, including the software platform based on the pattern, are as follows:

- Resources management and storage.
- MRTP definition.
- Core server design and implementation.
- Construct interface components for calling the core server.
- Resource Manager implemented to manage resources via the core server.
- Alias Manager implemented to be used to configure and manage databases which can be linked and accessed by the core server.
- Workflow control implementation.
- Flow Designer implementation to support visual flow design.
- Sheet Designer implementation – for visual sheet design.
- Define the server-side core language – Web Script, and can be embedded into HTML pages to make them execute on the core server rather than the Web browser.
- Atomic components of server core language construction.
- Virtual machine for server implementation.
- Web Pages Designer implementation to support development of server application
components and dynamic Web pages.

- Define client core languages – Delphi Script, VB Script, and Java Script.
- Construct atomic components of client core languages.
- Client virtual machine implementation – to support mixed compilation and execution of three core languages and share the same atomic components.
- *Form Designer* implementation – to support the development of client application components.
- *Client Explorer* implementation – it is not only an integrated application tool for client, but also a carrier of the client virtual machine.

To conclude this thesis: The Tri-Integration Pattern emphasises service integration, component integration and data integration. A software platform based on the pattern was developed to accelerate and extend practical pattern applications. Several tools supporting pattern applications were developed to improve further practicality of the pattern platform. The problem of multiple integrations has been addressed by software engineering researchers for many years, but there exist no dedicated solutions used in service integration, component integration and data integration using a system pattern.

Through the discussion in this thesis, it is concluded that the Tri-Integration Pattern can assist software development; its system framework is powerful for enterprises integrating services, components and data. It is helpful to construct an integrated, safe and extensible network application system, especially to help support the development of complex, large-scale systems and also support effective industrial software production.

9.5 Future Directions

In this section, some possible extensions to the present work are explored.

**System Pattern-Mining** — Although the system pattern concept has been proposed and a
special system pattern, Tri-Integration Pattern, is defined successful, mining new system patterns will remain an important activity for the future in the widely domains.

Mechanism for services integration — Because services integration cannot include all kinds of services or predict future service models, a mechanism that can integrate new services needs to be explored. A tentative plan is to add services configuration to the core service, and extend the MRTP to support additive services calling.

Information exchange standards — Although the pattern provides a data integration solution, it is inevitable that a system still need to exchange data with other information sources. In other words, a common exchange format is needed, for example, the exchange format defined by XML Schema.

Dynamic extension of the virtual machine — Applications development and integration are extended constantly; it is necessary to expand the core part of the virtual machine – the atomic components. It is necessary that this expansion can be dynamically implemented, for example, to support the DLL objects library dynamic load in Windows systems [16]. Establishing the mechanism of atomic components dynamic extension will contribute to improve application component development and expand the scope of component integration.

Wider mechanism for workflow control — In the software platform based on the Tri-Integration Pattern, a workflow management system with a single-engine was implemented. But, with the development of workflow applications, workflow management systems with multiple engines must be implemented. Only in this way, can they be adapted to the needs of medium to large enterprises and improve the application extensibility of the workflow system.
Reference


and Case Studies, Addison-Wesley, 2002.


[38] Deutsch, M.S. Software Verification and Validation: Realistic Project Approaches.


[50] Frankel, R. *Introduction to the tooltalk service*. Sun Microsystems Inc., Mountain


[52] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns - Elements of Reusable Object-Oriented Software*, Addison-Wesley, 1995.


[65] IBM, About Lotus Domino,


UK Ministry of Defence, MOD, Procurement Executive, SM836, Ash1, Abbeywood #93, P.O. Box 702, Bristol BS12 7DU, 1996.


Appendix A

Multiple Resource Transfer Protocol (MRTP)

A.1 Overview

Multiple Resource Transfer Protocol (MRTP) is a user-defined promotional protocol at application-level, which is based on the other protocols. MRTP refers to SMTP, POP3, FTP and so on which are quite popular and accepted by industry, and extends the following functions based on those protocols:

- Resource creation, authorisation, usage, maintenance, and classification are added, and resources integration can be performed.
- The supports to flow are enhanced, such as flow storage management, and flow delivery control.
- Accesses to and operations on various databases are added, which supports multiple-tier distributed applications based on databases development.
- Interpreting and executing specific command language are added, and connect with server-side components for the sake of application extension.

MRTP is a generic, stateless, object-oriented and open communication protocol. It can be utilised for many kinds of purposes through extension of its request methods (commands). MRTP can be used for many tasks, such as distributed object management systems, through extension of its request methods (commands). A feature of MRTP is the typing of data representation, allowing systems to be built independently of the data being transferred. MRTP is an application-level protocol with the lightness and speed necessary for distributed, collaborative, hypermedia applications. At present, MRTP still has been extended and increased continuously.
A.2 Terminology

This section uses some terms as following to refer to the roles played by participants in, and objects of, the MRTP communication.

**Connection:** A transport layer virtual circuit established between two application programs for the purpose of communication.

**Request:** A MRTP request message (as defined in A.5.1).

**Response:** A MRTP response message (as defined in A.5.2).

**Server:** An application program that accepts connections in order to service requests by sending back responses.

**Resource:** In MRTP, DotPath, MSGID and common file ID may uniquely identify data object or service.

There are various types of resource, such as: schedule resource, procedure resource, document resource, mail resource, form resource and web resource, etc.

**DotPath:** is a descriptive way that may locate and identify a certain resource object on the server or other resource object containers (as described in detail in A.4.1).
A.3 Overall Operation

MRTP is a communication protocol based on a request/response paradigm. A client establishes a connection with a server and sends a request to the server in the form of a request method. This request may follow a MIME-like message containing DotPath, client information, and possible body content.

The server responds with a status line, including the message's protocol version and a success or error code, followed by a MIME-like message containing server information, entity meta-information, and possible body content.

Most MRTP communication is initiated by a user agent and consists of a request to be applied to a resource on some origin server. In the simplest case, this may be accomplished via a single connection (v) between the user agent (UA) and the origin server (O).

```
Request chain ------------->
UA ---------------v-------------O
<-----------------Response chain
```

MRTP communication generally takes place over TCP/IP connections. The default port is TCP 24464, but other ports can be used. This does not preclude MRTP from being implemented on top of any other protocol on the Internet, or on other networks. MRTP only presumes a reliable transport; any protocol that provides such guarantees can be used, and the mapping of the MRTP request and response structures onto the transport data units of the protocol in question is outside the scope of this specification.

MRTP communication process is that the connection be established by the client prior to each request and closed by the server after sending the response. Both clients and servers should be aware that either party may close the connection prematurely, due to user action, automated time-out, or program failure, and should handle such closing in a predictable fashion. In any case, the closing of the connection by either or both parties always terminates the current request, regardless of its status.
A.4 Protocol Parameters

A.4.1 DotPath—Resource or Resource Paths

DotPath is a notation that is used to locate and identify resource in MRTP. It is a string used to describe server resource hierarchy or objects, and uses "." as separator between resource names of each layer. The hierarchy of server resource is in accordance with that of OS file system. For an application, the top root resource is the installation direction of the application. For example, on the assumption that an application is installed in “C:\program files\app\server\”, the corresponding path of Dotpath: Pub.forms in file system is “C:\program files\app\server\pub\forms\”.

It needs to be pointed out that DotPath is not only able to describe path of resource access and repository of other resource objects, but also describe an idiographic resource, such as user resource and so on.

A.4.2 MsgID—Message Identifier

MsgID is a unique name identifier string used to identify resource item in MRTP. Currently, the length of MsgID string is 30 bytes.

For example, “20030928095311E3A7763C6524C331”, the first eight-byte is resulted from current date (yyyymmdd), the following six-byte is produced from current time (HHMMSS), and the rest sixteen-byte is formed with random codes.

For the purpose of guaranteeing a unique MsgID, usually MsgID is produced with a random mechanism by the server.

A.4.3 FileName—File Name

FileName is file name representation in OS file system. Its named rules are accordant with that used in OS file system.
A.4.4 Alias — Database Interface Alias

Alias is a name of a database connection interface, through which server can access various current popular relation databases, such as Sybase, Oracle and MS SQL server, etc., with many database interface access methods accepted by industry (ADO, BDE, and ODBC, etc.). Since all database operations on client can be executed by server agent, wherever client runs, it does not need to install client-side interface of related database.

A.5 Protocol Message

MRTP message is constituted by both request message from a client to a server and response message from a server to a client.

A.5.1 Request

A request message from a client to a server includes, within the first line of that message, the method to be applied to the resource (command word), the identifier of the resource (Dotpath), and other parameters that a certain request method requires. A valid request format of MRTP is below:

\[
\text{<Request method> + <SP> + [command parameter ... ]}
\]

For example, LOGIN admin aa.

The elements in message are separated by SP characters. No CR or LF is allowed except in the final CRLF sequence.

A.5.2 Response

After receiving and interpreting a request message, a server responds to the client in the form of an MRTP response message according to the request content of the request method, which is called a response process.

MRTP response message consists of the Status-Line and the response entity.
The first line of a Full-Response message is the Status-Line, consisting of the protocol version followed by a numeric status code and its associated textual phrase, with each element separated by space (SP), ";", or "|" characters.

Status codes in the Status-Line have following formats which represent different meanings.

- **+OK** request method has been accepted, and server deal with successfully.
- **+ERR** request method has been accepted, but server deal with unsuccessfully.
- **-ERR** request method is erroneous, so that server cannot identify that request method.

The response entity is the result content that request method expects. Its organisation form and structure vary with different request method. MRTP uses many structures defined for Internet Mail (RFC822) and MIME (Multipurpose Internet Mail Extensions). So it allows the response entity to be transferred by a kind of open and expansive mechanism. In fact, some characteristics in MRTP are different from the mail discussed in RFC1521. These differences are used to optimise binary transferring performance, enhance security, and provide more freedom in using media types.

As for the idiographic response entity, please refer to the command references of each request method.

### A.5.3 Rights

Considering security, every MRTP request method command relates to current registered user, that is to say it must meets one or more following rights:

- **RIGHT_NONE** : No authority authentication
- **RIGHT_CONFIRM** : Resource (user) must be confirmed
- **RIGHT_READ** : Resource read permitted
• **RIGHT_SEND** : Resource send permitted

• **RIGHT_CREATE** : Resource create permitted

• **RIGHT_MODIFY** : Resource modify permitted

• **RIGHT_DELETE** : Resource delete permitted

### A.6 Request Method Command References

There are 64 request method commands totally in this document. They are described respectively as followings:

#### LOGIN

**Format:**

'LOGIN' [LoginName Password]

**Description:**

Send user login request to the server.

**Parameters:**

LoginName: The legal resource name used as user in MRTP system.

Password: resource password (case sensitive).

**Rights:**

**RIGHT_NONE.**

**Returns:**

If request is successful, it returns resource +'-' Resource description

#### LOGOUT
Format:

'LOGOUT'

Description:

Cancel login and stop current connection.

Parameters:

No parameter

Rights:

RIGHT_NONE.

Returns:

No return result

VALUE

Format:

'VALUE' {ValueID Value}

Description:

Assign value to a certain parameter of server.

Parameters:

ValueID: The name of the parameter assigned value

Value: value assigned to target parameter

Rights:

RIGHT_CONFIRM.

Returns:

No return result
Example:

If you want to realise UserName='test', then corresponding command is:

```
VALUE username test
```

**RCPT**

Format:

```
'RCPT' {Name}
```

Description:

Set user that receives the information.

Parameters:

- Name: user's name

Rights:

**RIGHT_SEND**

Returns:

The successful operation indicates that user has authority to receive the information, or else, that user has no adequate authority.

**CONTENT**

Format:

```
'CONTENT' {FileSize Subject ...}
```

Description:

Text content the client sends to the server.

Parameters:

- FileSize: The text content size
Subject: The text title

Rights:

RIGHTCONFIRM.

Returns:

No return result

FINISH

Format:

'FINISH'

Description:

Notify the server that the sending of the message has finished.

Parameters:

No parameter

Rights:

RIGHTCONFIRM.

Returns:

No return result

ATTACH

Format:

ATTACH {FileSize FileName ...}

Description:

The client sends attachment content to the server.
Parameters:

FileSize: the attachment size

FileName: the attachment name

Rights:

RIGHT_CONFIRM.

Returns:

No return result

RSET

Format:

'RSET'

Description:

Reset all message content.

Parameters:

No parameter

Rights:

RIGHT_CONFIRM.

Returns:

No return result

MSGLIST

Format:

'MSGLIST' {Dotpath}
Description:

Get the messages list from the specified resource path.

Parameters:

DotPath: resource path

Rights:

RIGHT_CONFIRM.

Returns:

MsgID list of the messages, and MsgIDs are separated by Ctrl (\n). Example:

MSGLIST usr.admin

MSGINFO

Format:

'MSGINFO' [Dotpath MsgID]

Description:

Get the item description from a certain message in specified resource path.

Parameters:

DotPath: resource path

MsgID: message identifier

Rights:

RIGHT_READ

Returns:

The item description of the message, the description lines are separated by Ctrl (\n).
Example:

MSGINFO usr.admin 200201171315516A0948F6DB85C8D3

Return information as follows:

MessageID=200201171315516A0948F6DB85C8D3
FromName= Professor Zhou
FromUser=usr.xw
Subject= This is the subject
ToName=Administrator
ToUser=usr.admin
DateTime=2002.01.17 13:15
Importance=no
AttachFile= attachment1. zips; attachment2. zips
IsRead=yes

MSGCONTENT

Format:

'MSGCONTENT' {Dotpath MsgID}

Description:

Get the text of a certain message from specified resource path.

Parameters:

DotPath: resource path
MsgID: message identifier

Rights:

RIGHT_READ

Returns:

The text content of the message (RTF format)

MSGFILE
Format:

'MSGFILE' {Dotpath MsgID FileName}

Description:

Save the specified attached file of a certain message as a new file.

Parameters:

DotPath: resource path

MsgID: message identifier

FileName: attached file name

Rights:

RIGHT_READ

Returns:

No return result

MSGDELETE

Format:

'MSGDELETE' {Dotpath MsgID}

Description:

Delete the specified message.

Parameters:

DotPath: resource path

MsgID: message identifier

Rights:

RIGHT_DELETE
Returns:

No return result

**RESLIST**

Format:

'RESLIST' {DotPath}

Description:

Obtain the resource list from the specified resource path.

Parameters:

DotPath: resource path

Rights:

RIGHT_CONFMIR

Returns:

The resource list from the specified resource path is separated by Ctrl (\r\n).

**AUTHORITY**

Format:

'AUTHORITY' {DotPath}

Description:

Obtain the resource descriptor of the specified resource.

Parameters:

DotPath: resource path

Rights:
RIGHT_CONFIRM

Returns:

A list that consists of property value pairs that left side is property name, right side is value, i.e. property name= value. The list pairs are separated by Ctrl (\r\n).

SAVEAUTHORITY

Format:

SAVEAUTHORITY' {DotPath}

Description:

Set the property information of the specified resource.

Parameters:

DotPath: resource path

Rights:

RIGHT_CREATE or RIGHT_MODIFY

Returns:

No return result

MAKERES

Format:

'MAKERES' {DotPath}

Description:

Create the name-specified resource in the server.

Parameters:

DotPath: the resource name to be created
Rights:

RIGHT_CREATE

Returns:

No return result

EMPTYRES

Format:

'EMPTYRES' (DotPath)

Description:

Empty all resource items in the specified resource in the server.

Parameters:

DotPath: the resource path to be emptied

Rights:

RIGHT_DELETE

Returns:

No return result

DELETERES

Format:

'DELETERES' (DotPath)

Description:

Delete the specified resource.

Parameters:
DotPath: the resource path to be deleted

Rights:

RIGHT_DELETE

Returns:

No return result

SETMSGINFO

Format:

'SETMSGINFO' {DotPath MsgID ...}

Description:

Set the related properties of the specified message.

Parameters:

DotPath: resource path

MsgID: message identifier

Values: the pair of the property name and property value (property name=value)

Rights:

RIGHT_MODIFY

Returns:

No return result

SETMSGREAD

Format:

'SETMSGREAD' {DotPath MsgID Read}
Description:

Set the flag that represents whether the message has been read already or not.

Parameters:

DotPath: resource path

MsgID: message identifier

Read: Whether have been read or not (yes/ no)

Rights:

RIGHT_CONFIRM

Returns:

No return result

MSGLOADFILE

Format:

'MSGLOADFILE' {DotPath FileName}

Description:

Download the specified file to the local computer from the server.

Parameters:

DotPath: the resource path of the file to be downloaded

RemoteFile: the file name to be downloaded

Rights:

RIGHT_READ

Returns:

No return result
MSGSAVEFILE

Format:

'MSGSAVEFILE' {DotPath FileSize FileName}

Description:

Upload the file to the server.

Parameters:

DotPath: the target resource path that the uploaded file will be saved

FileSize: the size of the uploaded file

FileName: the file name in the target resource

Rights:

RIGHT_CREATE or RIGHT_MODIFY

Returns:

No return result

MSGCOPYTO

Format:

'MSGCOPYTO' {DotPath MsgID bDotPath}

Description:

Copy the specified message from DotPath to bDotPath.

Parameters:

DotPath: resource path

MsgID: message identifier
bDotPath: target resource path

Rights:

RIGHT_SEND or RIGHT_CREATE

Returns:

No return result

MSGUPDATEINFO

Format:

'MSGUPDATEINFO' {Dotpath MsgID Value}

Description:

Update the property values of the specified message.

Parameters:

DotPath: resource path

MsgID: message identifier

Value: the property value pairs to be updated

Rights:

RIGHT_MODIFY

Returns:

No return result

MSGUPDATECONTENT

Format:

'MSGUPDATECONTENT' {Dotpath MsgID FileStream}
Description:

Update the text content of the specified message.

Parameters:

DotPath: resource path

MsgID: message identifier

FileStream: the file stream of the text content to be updated

Rights:

RIGHT_MODIFY

Returns:

No return result

**MSGUPDATEFILE**

Format:

'MSGUPDATEFILE' (Dotpath MsgID FileName FileStream)

Description:

Update the attached file of the specified message.

Parameters:

DotPath: resource path

MsgID: message identifier

FileName: the name of the attached file to be updated

FileStream: the attached file stream

Rights:

RIGHT_MODIFY
Returns:

No return result

**USERRIGHT**

**Format:**

'USERRIGHT' {DotPath}

**Description:**

Obtain the right information of the current user (resource) in the specified resource path.

**Parameters:**

DotPath: the target resource path from which right information is detected

**Rights:**

RIGHT_NONE

**Returns:**

The right values owned by the user (resource) in the specified resources path.

The right value is represented by a word type integer, the concrete meanings as follows:

- **RIGHT_DENY=0;** // deny
- **RIGHT_CONFIRM=1;** // user confirms (the first bit is 1)
- **RIGHT_READ=2;** // ($2) read (the second bit is 1)
- **RIGHT_SEND=8;** // ($8) send (the forth bit is 1)
- **RIGHT_CREATE=512;** // ($200) create (the tenth bit is 1)
- **RIGHT_MODIFY=2048;** // ($800) modify (the twelfth bit is 1)
RIGHT_DELETE=8192; // ($2000) delete (the fourteenth bit is 1)

RIGHT_CONTROL=32768; // ($8000) full control (the sixteenth bit is 1)

**DATETIME**

**Format:**

'DATETIME'

**Description:**

Get current date and time of the server.

**Parameters:**

No parameter

**Rights:**

RIGHT_NONE

**Returns:**

Current date and time of the server, it is described by a float number of double type, the integral part means the date, the fraction part means the time.

**MSGID**

**Format:**

'MSGID'

**Description:**

Request the server to produce randomly a unique MSGID.

**Parameters:**

No parameter

**Rights:**
RIGHT_NONE

Returns:

A unique MSGID, for its composing construction please refer to the "protocol parameter" part in the front.

TASKINS

Format:

'TASKINS' [UserName sDateTime sDuration sImageIndex sText]

Description:

Add the new task to the schedule of the specified user (resource).

Parameters:

UserName: user's login name (resource path)

sDateTime: start date and time

sDuration: durative minutes

sImageIndex: image Index

sText: task content

Rights:

RIGHT_MODIFY

Returns:

No return result

TASKDEL

Format:

'TASKDEL' [UserName sDateTime sDuration sImageIndex sText]
Description:

Delete a task from the schedule of the specified user (resource).

Parameters:

UserName: user's login name (resource path)

Sdate: start date and time

sDuration: durative minutes

sImageIndex: image index

Stext: task content

Rights:

RIGHT_DELETE

Returns:

No return result

TASKINFO

Format:

'TASKINFO' {DotPath}

Description:

Get the schedule information of the specified user (resource).

Parameters:

DotPath: user's login name (resource path)

Rights:

RIGHTCONFIRM

Returns:
The schedule arrangement records that use Ctrl (\n) as separator, and each record consists of four fields: (they use TAB as separator)

<start date time><end date time><Duration><task content>

FILELIST

Format:

'FILELIST' {DotPath UseExt SpecPath}

Description:

Get file list in the specified resource path.

Parameters:

DotPath: the resource path used to get file list

UseExt: indicate whether file list returned includes file extension or not

SpecPath: indicate the asterisk wildcard of the file name to be obtained

Rights:

RIGHT_READ

Returns:

File list in the specified resource path.

FILEAGE

Format:

'FILEAGE' {DotPath FileName}

Description:

Get the file age of the specified file.

Parameters:
DotPath: the resource path where the target file exists

FileName: the name of the target file

Rights:

RIGHT_NONE

Returns:

Integer value which represents the creating date of the target file

SEARCH

Format:

'SEARCH' {DotPath FileType|Content}

Description:

Search the files that satisfy with the specified conditions in the specified resource path.

Parameters:

DotPath: the resource path searched

FileExt: file extension, usually is "emx"

Context: the searching conditions and the fields to be obtained,

Its format is:

<Condition expression> [[field1; field2; ...]]

< Condition expression>=<identifier><comparison operators><value>.

<Comparison operators> is one of the following operators:

$LE$ = ' less or equal';

$GE$ = ' great or equal ';
$LT$ = 'less than';

$GT$ = 'greater than';

$NE$ = 'not equal';

$EQ$ = 'equal';

$CT$ = 'contain';

If there are several condition expressions, they may be connected by the symbol &.

Example:

SEARCH usr.admin emx FromName $EQ$ Professor Zhou | MessageID

Rights:

RIGHT_CONFIRM

DETECTNEW

Format:

'DetectNEW' {DotPath}

Description:

Detect whether new files exist in the specified resource path or not.

Parameters:

DotPath: the resource path detected

Rights:

RIGHT_CONFIRM

Returns:

The "true" value means that some messages in the resource path haven't been read; while the "false" value means that all messages have been read.
FLOWLIST

Format:

'FLOWLIST' {DotPath}

Description:

Get workflow list.

Parameters:

DotPath: The workflow resource path

Rights:

RIGHT_CONFIRM

Returns:

Workflow list which items is separated by Ctrl (\n
FLOWDEL

Format:

'FLOWDEL' {DotPath, MsgID}

Description:

Delete the specified workflow message file.

Parameters:

DotPath: the workflow resource path

MsgID: the identifier of workflow message file

Rights:

No return result
LICENCE

Format:

'LICENCE'

Description:

Get the authorisation information of the server.

Parameters:

No parameter

Rights:

RIGHTNONE

Returns:

The authorisation information string of the server

RESTREE

Format:

'RESTREE' {DotPath}

Description:

Get the resource tree records in the specified resource path.

Parameters:

DotPath: resource path

Rights:

RIGHTCONFIRM

Returns:
Resource list in the specified resource path, each record uses Ctrl (\n) as last symbol.

There are five fields in a record, the meanings of each field is described as follows:

Field 1: Resource directory layer

Field 2: Resource name

Field 3: Resource full name

Field 4: The right information of the resources

Field 5: Resource type

Example:

If the command “GetResTree pub” is executed, the return results may as follows:

0; pub.Catalog; Catalog; 1; Unknown

1; pub.Catalog.technical service; technique service; 3; Unknown

RESRENAMEN

Format:

'RESERNAME' {OldDotPath NewDotPath}

Description:

Rename the specified resource.

Parameters:

OldDotPath: old resource path

NewDotPath: new resource path

Rights:

RIGHT_MODIFY

Returns:
No return result

**RESEXISTS**

**Format:**

'RESEXISTS' {Dotpath}

**Description:**

Determine whether the resource exists or not.

**Parameters:**

DotPath: resource path

**Rights:**

RIGHT_CONFIRM

**Returns:**

If the resource exists, the return result is true, otherwise false

**RESSIZE**

**Format:**

'RESSIZE' {DotPath}

**Description:**

Get the size of the resource occupied space.

**Parameters:**

DotPath: the resource path

**Rights:**

RIGHT_CONFIRM
Returns:

The size of the resource occupied space

**DELETEFILE**

Format:

'DELETEFILE' {DotPath FileName}

Description:

Delete the specified file from the server.

Parameters:

DotPath: the resource path where the target file is placed

FileName: the name of the target file

Rights:

RIGHT_DELETE

Returns:

No return result

**FILEVERSION**

Format:

'FILEVERSION' {DotPath FileName}

Description:

Get the version information of the specified file.

Parameters:

DotPath: the resource path where the target file is placed
FileName: the name of the target file

Rights:

RIGHT_CONFIRM

Returns:

The version information string of the specified file

TOTALON

Format:

'TOTALON' {DotPath Recursive Condition}

Description:

Get total numbers of the files in the specified resource path with the specified conditions.

Parameters:

DotPath: the specified resource path proceeded

Recursive: indicate whether make recursive totalling or not

Condition: statistical conditions, (described in detail in SEARCH command)

Rights:

RIGHT_CONFIRM

Returns:

Return the total of the files that satisfy with the specified conditions

CHANGEPWD

Format:

'CHANGEPWD' {DotPath OldPassword NewPassword}
Description:

Change the password.

Parameters:

DotPath: user's name or resource (for example: Usr.admin)

OldPassword: old password

NewPassword: new password

Rights:

RIGHT_MODIFY

Returns:

If operation is successful, the return result is true, otherwise false.

ALIASES

Format:

'ALIASES'

Description:

Get the aliases of the remote database.

Parameters:

No parameter

Rights:

RIGHT_NONE

Returns:

The aliases of the remote database
QUERY

Format:

'QUERY' {Alias; UserName; Password; SqlStr}

Description:

Execute SQL query in the server.

Parameters:

Alias: the alias of the database interface

UserName: the login name of the remote database user

Password: the login password of the remote database user

SqlStr: SQL string to be executed

Rights:

RIGHT_NONE

Returns:

The result produced by SQL's execution

APPLYUPDATES

Format:

'APPLYUPDATES' {Alias; UserName; Password; SqlStr; FileStream}

Description:

Execute the database updating from the server.

Parameters:

Alias: the alias of the database interface
UserName: the login name of the remote database user

Password: the login password of the remote database user

SqlStr: SQL string to be executed

FileStream: the related data stream

Rights:

RIGHT_NONE

Returns:

If operation is successful, it returns empty string, or else it returns the error information of the database operation

TABLENAMES

Format:

TABLENAMES' {Alias; UserName; Password}

Description:

Through the database interface, get the table names from the remote database server.

Parameters:

Alias: the alias of the database interface

UserName: the login name of the remote database user

Password: the login password of the remote database user

Rights:

RIGHT_NONE

Returns:

Table names list of the database
FIELDNAMES

Format:

'FIELDNAMES' {Alias; UserName; Password; TableName}

Description:

Through the database interface, get the fields information in the specified table from
the remote database server.

Parameters:

 Alias: the alias of the database interface

 UserName: the login name of the remote database user

 Password: the login password of the remote database user

 TableName: the table from which the field information is obtained

Rights:

RIGHT_NONE

Returns:

Fields information list

QUERYTOSTRING

Format:

'QUERYTOSTRING' {Alias; UserName; Password; q|Str}

Description:

Execute SQL query command and return query result in the form of string.

Parameters:

 Alias: the alias of the database interface
UserName: the login name of the remote database user

Password: the login password of the remote database user

SqlStr: SQL string to be executed

Rights:

RIGHT_NONE

Returns:

The result strings produced by SQL’s execution

**QUERYTOPAGE**

Format:

'QUERYTOPAGE'\{Alias; UserName; Password; Page; PageSize; SqlStr\}

Description:

Execute SQL query on the database specified alias, query results are return by the specified page.

Parameters:

Alias: the alias of the database interface

UserName: the login name of the remote database user

Password: the login password of the remote database user

Page: the number of the page brought back

PageSize: record number in each page

SqlStr: SQL string to be executed

PageCount: the total of the pages brought back

Rights:
RIGHT_NONE

Returns:

The specified page in the total record set of the query results

EXECSQL

Format:

'EXECSQL' {Alias; UserName; Password; SqlStr}

Description:

Execute SQL string in the server, which will not bring back result set.

Parameters:

Alias: the alias of the database interface

UserName: the login name of the remote database user

Password: the login password of the remote database user

SqlStr: SQL string to be executed

Rights:

RIGHT_NONE

Returns:

If operation is successful, it returns an empty string, or else it returns the error information of the database operation

VIRTUAL

Format:

'VIRTUAL' {Dir}

Description:
Corresponding to the virtual directory in the WWW server, return its real directory in the file system.

Parameters:

Dir: virtual directory name

Rights:

**RIGHT.NONE**

Returns:

The real directory path of the virtual directory in the file system

**SCRIPT**

Format:

'SCRIPT' {Source}

Description:

Summit Script codes to the server that will execute the string.

Parameters:

Source: script codes to be executed in the server.

Rights:

**RIGHT.CONFIRM**

Returns:

Return result is set by script codes.

**SCRIPTFILE**

Format:

'SCRIPTFILE' {DotPath,FileName}
Description:

Request the server to execute a certain script file in the specified resource path.

Parameters:

DotPath: the resource path that the script file exists

FileName: the name of the script file

Rights:

\texttt{RIGHT\_CONFIRM}

Returns:

Return result is set by script file.

\textbf{SERVERADDR}

Format:

\texttt{'SERVERADDR'}

Description:

Get IP address of the server.

Parameters:

No parameter

Rights:

\texttt{RIGHT\_NONE}

Returns:

The IP address information string of the server.

\textbf{NOOP}
Format:

'NOOP'

Description:

Execute empty operation.

Parameters:

No parameter

Rights:

RIGHT_NONE

Returns:

No return result

GET

Format:

'GET' {HTTP Cmd}

Description:

Execute GET command of the HTTP protocol, following the standards of HTTP/1.1s, please refer to the RFC2616.

Parameters:

HTTP: Following the standards of HTTP/1.1s, please refer to the RFC2616

Cmd: command string

Rights:

RIGHT_NONE

Returns:
Follows the standards of HTTP/1.1s, please refer to the RFC2616

**POST**

Format:

'POST' {HTTP Cmd}

Description:

Execute POST command of the HTTP protocol.

Parameters:

Follows the standards of HTTP/1.1s, please refer to the RFC 2616

Rights:

RIGHT_NONE

Returns:

Follows the standards of HTTP/1.1s, please refer to the RFC 2616

**HEAD**

Format:

'HEAD' {HTTP Cmd}

Description:

Execute HEAD command of the HTTP protocol.

Parameters:

Follows the standards of HTTP/1.1s, please refer to the RFC 2616

Rights:

RIGHT_NONE
Returns:

Follows the standards of HTTP/1.1s, please refer to the RFC 2616
Appendix B

Key Objects for Tri-Integration Idioms

B.1 Core Server Base Object: TAnywhereBaseClass

<table>
<thead>
<tr>
<th>MRTP base object</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>TAnywhereBaseClass = class(TObject)</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>function anGetAuthority(DotPath: string): string; virtual;</td>
</tr>
<tr>
<td>function anGetValue(DotPath, aID: string): string; virtual;</td>
</tr>
<tr>
<td>function anCheckRight(const DotPath, UserName, FlowID: string, Right: word): Boolean; virtual;</td>
</tr>
<tr>
<td>function anCheckMsgIDRight(const DotPath, UserName, MsgID: string, Right: word): Boolean; virtual;</td>
</tr>
<tr>
<td>function anCheckUser(const DotPath, UserName, MsgID: string, IsConfirm: Boolean=False): word; virtual;</td>
</tr>
<tr>
<td>function anCheckPath(DotPath: string): Boolean; virtual;</td>
</tr>
<tr>
<td>function anUserLogin(var UserName: string; Password: string): Boolean; virtual;</td>
</tr>
<tr>
<td>function anGetPassword(UserName: string): string; virtual;</td>
</tr>
<tr>
<td>function anGetInfo(DotPath, MsgID: string): string; virtual;</td>
</tr>
<tr>
<td>function anGetResource(DotPath: string): string; virtual;</td>
</tr>
<tr>
<td>function anGetResTree(DotPath, UserName: string; Level: integer): string; virtual;</td>
</tr>
</tbody>
</table>

//Following methods must be implemented in the inheritance classes. |
| function anSaveAuthority(DotPath: string; AuStrings: TStrings): Boolean; virtual; |
| function anMakeResource(DotPath: string): Boolean; virtual; |
| function anEmptyResource(DotPath: string): Boolean; virtual; |
| function anDeleteResource(DotPath: string): Boolean; virtual; |
| function anRenameRes(OldDot, NewDot: string): Boolean; virtual; |
| function anResourceSize(DotPath: string): Longint; virtual; |

| procedure |
| anLoadAllResource; virtual; |
| function anGetList(DotPath: string; Cond: string = ": string; virtual; |
| function anDelete(DotPath, MsgID: string): Boolean; virtual; |
| function anSetInfo(DotPath, MsgID: string; AuStrings: TStrings): Boolean; virtual; |
| function anSetRead(DotPath, MsgID, ReadFlag: string): Boolean; virtual; |
| function anMsgCopyTo(DotPath, MsgID, bPath: boolean): boolean; virtual; |
| function anGetFileContent(DotPath, MsgID, LoadFileName: string; Stream: TStream): Boolean; virtual; |
| function anUpdateFileContent( DotPath, MsgID, LoadFileName: string; Stream: TStream ): Boolean; virtual; |
| function anTaskIns(UserName, sDateTime, sDuration, sImageIndex, Text: String): Boolean; virtual; |
| function anTaskDel(UserName, sDateTime, sDuration, sImageIndex, Text: String): Boolean; virtual; |
| function anTaskInfo(DotPath, UserName: String): String; virtual; |
| function anGetFileList(DotPath, sPath: string; Ext: Boolean): String; virtual; |
| function anSearchFor( DotPath, UserName, FileExt, Context: string; Recursive, CaseSensitive: Boolean): string; virtual; |
| function anPullSearch(DotPath, Context: string; Recursive, CaseSensitive: Boolean): string; virtual; |
| function anDetectNew(DotPath: string; var Msg: string): Boolean; virtual; |
| function anGetFlowList(DotPath, UserName: string): string; virtual; |
| function anFlowDelete(DotPath, MsgID, UserName: string): Boolean; virtual; |
| function anDeleteFile(DotPath, aFileName: string): Boolean; virtual; |
| function anGetFileString(DotPath, aFileName: string): string; virtual; |
| function anSaveEMI(DotPath, aUid: string; emiStrings: TStrings): Boolean; virtual; |
| function anSaveEMD(DotPath, aUid, emdFilePath: string): Boolean; virtual; |
| function anGetFileVersion(DotPath, FileName, wwwroot: string): string; virtual; |
| function anGetFileAge(DotPath, aFileName: string): integer; virtual; |

end;
implementation

function TAnywhereBaseClass.anGetAuthority(DotPath: string): string;
var i: integer;
begin
  // Extract DotPath Authority Information
  Result := ";
  Lock;
  i := FAuthoritiesList.IndexOf(DotPath);
  if i >= 0 then
    Result := TStringList(FAuthoritiesList.Objects[i]).Text;
  UnLock;
end;

function TAnywhereBaseClass.anGetValue(DotPath, aID: string): string;
var i: integer;
begin
  // Extract DotPath item aID's value in Authority Information
  Result := ";
  Lock;
  i := FAuthoritiesList.IndexOf(DotPath);
  if i >= 0 then
    Result := TStringList(FAuthoritiesList.Objects[i]).Values[aID];
  UnLock;
end;

function TAnywhereBaseClass.anCheckPath( DotPath: string): Boolean;
begin
  // Detect if DotPath exists
  Lock;
  Result := (FAuthoritiesList.IndexOf(DotPath) >= 0) or
            SameText(DotPath, 'usr.INET_GUEST') or
            SameText(DotPath, 'usr.sa');
  UnLock;
end;

function TAnywhereBaseClass.anCheckMsgIDRight( const DotPath, UserName, MsgID: string; Right: word): boolean;
var
tmpList: TStringList; ss: string;
rt: word;
procedure AssignRight( var r: word; Value: string );
begin
  if Value = " then Exit;
  if Value = '0' then r := 0
  else r := r or StrToIntDef(Value,1);
end;

begin
  // Detect the right of UserName in DotPath with MsgID
  Result := False; // None
  rt := 0;
  if DotPath = " then Exit;
  if MsgID <> " then begin
    ss := anGetInfo(DotPath, MsgID);
    tmpList := TStringList.Create;
    tmpList.Text := ss;
    AssignRight(rt, tmpList.Values[UserName]); // Assign right to User
    tmpList.Free;
    Result := (rt and Right)>0;
  end;
end;

function TAnywhereBaseClass.anCheckUser( const DotPath, UserName, MsgID: string; IsConfirm: Boolean=False): word;
var
EntryStrings, UserStrings, tmpList: TStringList;
i: integer;
ss: string;

function GroupValue: string;
var i: integer;
ss, sGroup: string;
begin
  Result := "";
  for i := 0 to UserStrings.Count-1 do begin
    ss := UserStrings[i];
    if UpperCase(Copy(ss,1,2)) = 'D.' then begin
      ss := UserStrings.Names[i];
      sGroup := Copy(ss,3,Length(ss));
      ss := UserStrings.Values[ss];
      if ss = 'I' then begin
        ss := EntryStrings.Values['G'+sGroup];
        if ss > Result then
          Result := ss;
      end;
    end;
  end;
end;

function GroupValue2: string;
var i: integer;
ss, sGroup: string;
begin
  Result := "";
  for i := 0 to UserStrings.Count-1 do begin
    ss := UserStrings[i];
    if UpperCase(Copy(ss,1,2)) = 'D.' then begin
      ss := UserStrings.Names[i];
      sGroup := Copy(ss,3,Length(ss));
      ss := UserStrings.Values[ss];
      if ss = 'I' then begin
        ss := tmpList.Values['G'+sGroup];
        if ss > Result then
          Result := ss;
      end;
    end;
  end;
end;

function IsOwnerGroup: boolean;
var i: integer;
ss, sGroup: string;
begin
  Result := False;
sGroup := "";
  for i := 0 to UserStrings.Count-1 do begin
    ss := UserStrings[i];
    if UpperCase(Copy(ss,1,2)) = 'D.' then begin
      ss := UserStrings.Names[i];
      sGroup := Copy(ss,3,Length(ss));
      Break;
    end;
  end;
  if sGroup <> "" then begin
    if pos(UpperCase('pub.group.'+sGroup),UpperCase(DotPath)) > 0 then begin
      Result := True;
      Exit;
    end;
  end;
  else begin
    //if EntryStrings.Values['D.'+sGroup] = 'I' then begin
    i := StrToIntDef(EntryStrings.Values['G'+sGroup], 0);
    if ((i and RIGHT_READ) > 0) or ((i and RIGHT_CONTROL) > 0) then
      Result := True;
  end;
end;
procedure AssignRight(var r: word; Value: string);
begin
  if Value = " then Exit;
  if Value = '0' then r := 0
  else r := r or StrToIntDef(Value, 1);
end;

begin
  // Get User's right in DotPath, if MsgID is not empty, then Get User's right in
  // DotPath with MsgID
  // if you can not get the User's right in DotPath, then get the User's right from its parent resource.
  Result := 0; // RIGHT_DENY
  if DotPath = " then Exit;

  if anCheckPath(UserName) then begin
    Result := Result or 1; // RIGHT_CONFIRM
    if IsConfirm then Exit;
  end;

  EntryStrings := TStringList.Create;
  if Pos(';', DotPath) = 0 then
    EntryStrings.Text := anGetAuthorization(DotPath)
  else
    EntryStrings.Text := StringReplace(DotPath, ';', '#13#10[rfReplaceAll]);
  UserStrings := TStringList.Create;
  UserStrings.Text := anGetAuthority(UserName);

  with EntryStrings do begin
    AssignRight(Result, Values['A.EveryOne']); // Detect everyone
    AssignRight(Result, GroupValue); // Detect group
    AssignRight(Result, Values['L.4+UserStrings.Values[\r\n P.Position\r\n]']); // Detect user's range
    if (UserStrings.Values['P.IsLocalSecretary'] = '1') and IsOwnerGroup then
      AssignRight(Result, Values['A.LocalSecretary']); // Detect Local Secretary
    if (UserStrings.Values['P.IsLocalAdmin'] = '1') and IsOwnerGroup then
      AssignRight(Result, Values['A.LocalAdmin']); // Detect Local Administrator
    if UserStrings.Values['P.IsSecretary'] = '1' then
      AssignRight(Result, Values['A.Secretary']); // Detect Secretary
    if UserStrings.Values['P.IsDocAdmin'] = '1' then
      AssignRight(Result, Values['A.DocAdmin']); // Detect Document Manager
    if UserStrings.Values['P.IsGlobalAdmin'] = '1' then
      AssignRight(Result, Values['A.GlobalAdmin']); // Detect global Administrator
    AssignRight(Result, Values[UserName]); // Detect the user
  end;

  if MsgID <> " then begin
    tmplList := TStringList.Create;
    tmplList.Text := anGetInfo(DotPath, MsgID);
    AssignRight(Result, tmplList.Values['A.EveryOne']); // Detect everyone
    AssignRight(Result, GroupValue2); // Detect group
    AssignRight(Result, tmplList.Values['L.4+UserStrings.Values[\r\n P.Position\r\n]']); // Detect user's range
    if UserStrings.Values['P.IsSecretary'] = '1' then
      AssignRight(Result, tmplList.Values['A.Secretary']); // Detect Secretary
    if UserStrings.Values['P.IsDocAdmin'] = '1' then
      AssignRight(Result, tmplList.Values['A.DocAdmin']); // Detect Document Manager
    if UserStrings.Values['P.IsGlobalAdmin'] = '1' then
      AssignRight(Result, tmplList.Values['A.GlobalAdmin']); // Detect global Administrator
    AssignRight(Result, tmplList.Values[UserName]); // Detect the user
    tmplList.Free;
  end;

  if SameText('usr.'+Values['P.UserID'], UserName) or
    SameText(DotPath, UserName) or
    (Pos(LowerCase(UserName)+'.'), LowerCase(DotPath)) > 0) then
    Result := Result or $8000; // Resource is the user or user's subresource
  if (Pos('INET_GUEST', UserName) > 0) then begin
    if ((Result and RIGHT_READ) > 0) or ((Result and RIGHT_CONTROL) > 0) then
      Result := 2 // Read only
else
    Result := 1;
end;
end;

UserStrings.Free;
EntryStrings.Free;

if (Result = 1) and (not IsConfirm) and (Pos(INET_GUEST, UserName) = 0) then begin  //Detect parent resource
    if (pos('.', DotPath) > 0) and (pos(';', DotPath) = 0) then begin
        EntryStrings := TStringList.Create;
        EntryStrings.Text := StringReplace(DotPath, '.', #13#10,[rfReplaceAll]);
        ss := "";
        for i := 0 to EntryStrings.Count-2 do
            if ss = "" then ss := EntryStrings[i]
        else ss := ss + "." + EntryStrings[i];
        EntryStrings.Free;
        Result := Result or anCheckUser(ss, UserName, ",", IsConfirm);
        if Result > RIGHT_CONTROL then Result := RIGHT_CONTROL;
    end;
end;

function TAnywhereBaseClass.anCheckRight(const DotPath, UserName, FlowID: string; Right: word): boolean;
var
    UserRight: word;
    FileName, ss: string;
    FilesList: TStringList;
begin
    //Detect if user has specific right in DotPath,
    //if user has no any right and FlowID is not empty, then detect if user has right in
    //flowbox with FlowID
    Result := False;
    if not anCheckPath(DotPath) and ((Right and RIGHT_CREATE) = 0) then Exit;
    UserRight := anCheckUser(DotPath, UserName, FlowID, Right = RIGHT_CONFIRM);
    Result := ((UserRight and Right) > 0) or ((UserRight and RIGHT_CONTROL) > 0);
    if (not Result) and (FlowID <> '') then begin
        FilesList := TStringList.Create;
        FilesList.Text := anSearchFor('sys.flowbox', UserName, 'emi', FlowID, False, False);
        if FilesList.Count > 0 then begin
            FileName := FilesList.Names[0];
            FilesList.Text := anGetInfo(FileName, FilesList.Values[FileName]);
            ss := FilesList.Values[UserName];
            if ss <> "" then begin
                UserRight := StrToIntDef(ss, 0);
                Result := ((UserRight and Right) > 0) or ((UserRight and RIGHT_CONTROL) > 0);
            end;
        end;
    end;
    FilesList.Free;
end;

function TAnywhereBaseClass.anUserLogin(var UserName: string; Password: string): Boolean;
var
    DotPath: string;
begin
    function CheckUserPwd(DotPath: string): Boolean;
    begin
        Result := Decrypt(anGetValue(DotPath, 'P.PrivateID')) = Password;
    end;

    begin
        if User login successful, then return the FullName in the variant UserName
        DotPath := UserName;
        if Pos(';', UserName) = 0 then
            DotPath := $usr+DotPath;
        Result := CheckUserPwd(DotPath);
        if not Result and (Pos(';', UserName) = 0) then begin
            DotPath := $pub.group+$UserName;
Result := CheckUserPwd(DotPath);
end;
if Result then UserName := DotPath;
end;

function TAnywhereBaseClass.anGetPassword(UserName: string): string;
var ss: string;
begin
  // Get User's password
  Result := "
  if Pos('USR', UpperCase(UserName)) = 0 then
    UserName := 'usr.' + UserName;
    ss := anGetValue(UserName, 'P.PrivateID');
  Result := Decrypt(ss);
end;

function TAnywhereBaseClass.anSaveAuthority(DotPath: string; AuStrings: TStrings): Boolean;
begin
  // Save authority information with AuStrings. If saveing is ok, then return TRUE.
  Result := False;
end;

function TAnywhereBaseClass.anGetResList(DotPath: string): string;
var FilesList: TstringList;
  EntryList: TstringList;
  i, ll: integer;
  ss, SubPath: string;
begin
  // Get DotPath resources list
  Result := "
  FilesList := TStringList.Create;
  GetFoldList(FilesList, DotPath, True);
  EntryList := TStringList.Create;
  EntryList.Sorted := True;
  for i:=0 to FilesList.Count-1 do begin
    ss := anGetValue(FilesList[i], 'P.Priority');
    ss := GetUserOrder(ss);
    SubPath := FilesList[i];
    SubPath := Copy(SubPath, Length(DotPath)+2, Length(SubPath));
    if Pos('.', SubPath) > 0 then
      SubPath := Copy(SubPath, 1, Pos('.', SubPath)-1);
    EntryList.Add(ss + '=' + SubPath);
  end;
  FilesList.Free;
  EntryList.Sorted := False;
  for i:=0 to EntryList.Count-1 do begin
    ll := Pos('=', EntryList[i]);
    EntryList[i] := Copy(EntryList[i], ll+1, Length(EntryList[i]));
  end;
  Result := EntryList.Text;
  EntryList.Free;
end;

function TAnywhereBaseClass.anGetResTree(DotPath, UserName: string; Level: integer): string;
var ss, tt: string;
  FilesList, tmpList: TStringList;
  i: integer;
begin
  // Get DotPath resources tree
  Result := "
  FilesList := TStringList.Create;
  FilesList.Text := anGetResList(DotPath);
  for i:=0 to FilesList.Count-1 do begin
    tmpList := TStringList.Create;
    tmpList.Text := anGetAuthority(DotPath+'.'+FilesList[i]);
    ss := tmpList.Values['P.FullName'];
    if ss = " then ss := 'Unknown';
    tt := 'Unknown';
  end;
end;
if SameText(tmpList.Values['P.Scheduler'], 'YES') then
tt := 'SCHEDULED';
else if tmpList.Values['P.FormEntry'] <> " then
tt := 'FORM';
else if tmpList.Values['P.AppEntry'] <> " then
tt := 'APPLICATION';
else if SameText(tmpList.Values['P.Mail.Server'], 'TRUE') then
tt := 'MAIL';
else if tmpList.Values['P.Internet'] <> " then
tt := 'INTERNET';
tmpList.Free;
//Result := Result+IntToStr(Level)+"+IntToStr(i)+"+ss+";
Result := Result+IntToStr(Level)+"+DotPath+"+FilesList[i]+"+ss+";
Result := Result+IntToStr(anCheckUser(DotPath+"+FilesList[i], UserName, "));
Result := Result+";tt;
Result := Result+#13#10+anGetResTree(DotPath+"+FilesList[i], UserName, Level+1);
end;
FilesList.Free;
end;

---------------------------------------------------------------

procedure TAnywhereBaseClass.anLoadAllResource;
begin
  //Load all resources authority information to FAuthoritiesList
end;

function TAnywhereBaseClass.anMakeResource(DotPath: string): Boolean;
begin
  //Create resource DotPath, if successful, then return TRUE
  Result := False;
end;

function TAnywhereBaseClass.anEmptyResource(DotPath: string): Boolean;
begin
  //Empty resource DotPath, if successful, then return TRUE
  Result := False;
end;

function TAnywhereBaseClass.anDeleteResource(DotPath: string): Boolean;
begin
  //Delete resource DotPath, if successful, then return TRUE
  Result := False;
end;

function TAnywhereBaseClass.anRenameRes(OldDot, NewDot: string): Boolean;
begin
  //Rename resource OldDot as NewDot, if successful, then return TRUE
  Result := False;
end;

function TAnywhereBaseClass.anResourceSize(DotPath: string): Longint;
begin
  //Calculate occupied spaces of resource DotPath
  Result := 0;
end;

function TAnywhereBaseClass.anGetInfo(DotPath, MsgID: string): string;
begin
  //get the information of DotPath with MsgID
  Result := ";
end;

function TAnywhereBaseClass.anGetList(DotPath: string; Cond: string = ": string;
begin
  //get messages list in DotPath
  Result := ";
end;
function TAnywhereBaseClass.anMsgCopyTo(DotPath, MsgID, bPath: string): boolean;
begin
  //Copy message with MsgID from DotPath to bPath
  Result := False;
end;

function TAnywhereBaseClass.anSetInfo(DotPath, MsgID: string; AuStrings: TStrings): Boolean;
begin
  //Set the information of resource DotPath with MsgID using values AuStrings
  Result := False;
end;

function TAnywhereBaseClass.anSetRead(DotPath, MsgID, ReadFlag: string): Boolean;
begin
  //Set message read flag with ReadFlag
  Result := False;
end;

function TAnywhereBaseClass.anDelete(DotPath, MsgID: string): Boolean;
begin
  //Delete message with MsgID from DotPath
  Result := False;
end;

function TAnywhereBaseClass.anGetFileContent(DotPath, MsgID, LoadFileName: string; Stream: TStream): Boolean;
begin
  //Convert inside file content to specific stream in DotPath with MsgID
  Result := False;
end;

function TAnywhereBaseClass.anUpdateFileContent(DotPath, MsgID, LoadFileName: string; Stream: TStream): Boolean;
begin
  //Update inside file content form specific stream in DotPath with MsgID
  Result := False;
end;

function TAnywhereBaseClass.anTaskIns(UserName, sDateTime, sDuration, sImageIndex, Text: string): Boolean;
begin
  //Append task to user
  Result := False;
end;

function TAnywhereBaseClass.anTaskDel(UserName, sDateTime, sDuration, sImageIndex, Text: string): Boolean;
begin
  //Delete user's specific task
  Result := False;
end;

function TAnywhereBaseClass.anTaskInfo(DotPath, UserName: string): string;
begin
  //Get user's task information in the DotPath
  Result := "";
end;

function TAnywhereBaseClass.anGetFileList(DotPath, sPath: string; Ext: Boolean): string;
begin
  //Get files list from resource DotPath
  Result := "";
end;

function TAnywhereBaseClass.anSearchFor(DotPath, UserName, FileExt, Context: string; Recursive, CaseSensitive: Boolean): string;
begin
function TAnywhereBaseClass.anFullSearch( DotPath, Context: string; Recursive, CaseSensitive: Boolean): string;
begin
  //Full text search from the resource DotPath
  Result := ";
end;

function T何处BaseClass.anDetectNew(DotPath: string; var Msg: string): Boolean;
begin
  //Detect if there is any new message in the DotPath
  Msg := ";
  Result := False;
end;

function T何处BaseClass.anGetFlowList(DotPath, UserName: string): string;
begin
  //get flow list from resource DotPath
  Result := ";
end;

function T何处BaseClass.anFlowDelete(DotPath, MsgID, UserName: string): Boolean;
begin
  //Delete flow item from resource DotPath with MsgID
  Result := False;
end;

function T何处BaseClass.anDeleteFile( DotPath, aFileName: string): Boolean;
begin
  //Delete a file from resource DotPath
  Result := False;
end;

function T何处BaseClass.anGetString(DotPath, aFileName: string): string;
begin
  //Get a file content in string format from resource DotPath
  Result := ";
end;

function T何处BaseClass.anSaveEMI(DotPath, aUid: string; emiStrings: TStrings): Boolean;
begin
  //Save EMI format file to resource DotPath with aUid
  Result := False;
end;

function T何处BaseClass.anSaveEMD(DotPath, aUid, emdFilePath: string): Boolean;
begin
  //Save EMD format file to resource DotPath with aUid
  Result := False;
end;

function T何处BaseClass.anGetFileVersion(DotPath, FileName, wwwroot: string): string;
begin
  //get a file version from resource DotPath
  Result := ";
end;

function T何处BaseClass.anGetFileAge( DotPath, aFileName: string): integer;
begin
  //get a file age from resource DotPath
  Result := -1;
end;
end.
B.2 Interface Object: TRConnection

<table>
<thead>
<tr>
<th>Interface object</th>
</tr>
</thead>
<tbody>
<tr>
<td>type</td>
</tr>
<tr>
<td>TRConnection = class</td>
</tr>
<tr>
<td>private</td>
</tr>
<tr>
<td>procedure DoConnect;</td>
</tr>
<tr>
<td>public</td>
</tr>
<tr>
<td>function Login: Boolean;</td>
</tr>
<tr>
<td>procedure Logout;</td>
</tr>
<tr>
<td>function Hello: Boolean;</td>
</tr>
<tr>
<td>function SendToUser(SubEntry, Subject, ContentFile, Values, Files: string): Boolean;</td>
</tr>
<tr>
<td>function GetMessageList(DotPath: string): string;</td>
</tr>
<tr>
<td>function GetMessageInfo(DotPath, MsgID: string): string;</td>
</tr>
<tr>
<td>function GetMessageContent(DotPath, MSGID: string): string;</td>
</tr>
<tr>
<td>function GetMessageFile(DotPath, MSGID, FileName: string): string;</td>
</tr>
<tr>
<td>function DeleteMessage(DotPath, MSGID: string): Boolean;</td>
</tr>
<tr>
<td>procedure SaveMessageFileTo(DotPath, MSGID, FileName, DiskFile: string);</td>
</tr>
<tr>
<td>function GetResList(DotPath: string): string;</td>
</tr>
<tr>
<td>function GetAuthority(DotPath: string): string;</td>
</tr>
<tr>
<td>function SaveAuthority(DotPath, Values: string): Boolean;</td>
</tr>
<tr>
<td>function MakeResource(DotPath: string): Boolean;</td>
</tr>
<tr>
<td>function EmptyResource(DotPath: string): Boolean;</td>
</tr>
<tr>
<td>function DeleteResource(DotPath: string): Boolean;</td>
</tr>
<tr>
<td>function ResourceExists(DotPath: string): Boolean;</td>
</tr>
<tr>
<td>function GetAdbPath(AdbName: string): string;</td>
</tr>
<tr>
<td>function GetADBDList: string;</td>
</tr>
<tr>
<td>function GetMessageInfo(DotPath, MSGID, Values: string): Boolean;</td>
</tr>
<tr>
<td>function LocalToRemote(LocalFile, DotPath, RemoteFile: string): Boolean;</td>
</tr>
<tr>
<td>function RemoteToLocal(DotPath, RemoteFile, LocalFile: string): Boolean;</td>
</tr>
<tr>
<td>function UserRight(DotPath: string): integer;</td>
</tr>
<tr>
<td>function CopyMessageToADotPath, MSGID, bDotPath: string): Boolean;</td>
</tr>
<tr>
<td>function TaskInsert(UserName, sDateTime, sDuration, sImageIndex, sText: string): Boolean;</td>
</tr>
<tr>
<td>function TaskDelete(UserName, sDateTime, sDuration, sImageIndex, sText: string): Boolean;</td>
</tr>
<tr>
<td>function GetFileList(DotPath, SpecPath: string; Ext: WordBool: string);</td>
</tr>
<tr>
<td>function GetFileAge(DotPath, FileName: string): integer;</td>
</tr>
<tr>
<td>function SearchFor(DotPath, FileExt, Context: string): string;</td>
</tr>
<tr>
<td>function DetectNew(DotPath: string): Boolean;</td>
</tr>
<tr>
<td>function GetFlowList(DotPath: string): string;</td>
</tr>
<tr>
<td>function DeleteFlow(DotPath, MSGID: string): Boolean;</td>
</tr>
<tr>
<td>function GetLicence: string;</td>
</tr>
<tr>
<td>function GetResTree(DotPath: string): string;</td>
</tr>
<tr>
<td>function GetMessageRead(DotPath, MSGID, ReadFlag: string): Boolean;</td>
</tr>
<tr>
<td>function RemoteQuery(qType, qAlias, qUserName, qPassword, qSqlStr, qFileName: string): string;</td>
</tr>
<tr>
<td>function RemoteUpdate(qAlias, qUserName, qPassword, qSqlStr, qFileName: string): string;</td>
</tr>
<tr>
<td>function RemoteExecSQL(qAlias, qUserName, qPassword, qSqlStr: string): string;</td>
</tr>
<tr>
<td>function RemoteTableNames(qAlias, qUserName, qPassword: string): string;</td>
</tr>
<tr>
<td>function RemoteFieldNameNames(qAlias, qUserName, qPassword, qTable: string): string;</td>
</tr>
<tr>
<td>function RemoteObjectNames: string;</td>
</tr>
<tr>
<td>function RenameResrcResource(OldDotPath, NewDotPath: string): Boolean;</td>
</tr>
<tr>
<td>function RemoteDeleteFile(DotPath, FileName: string): Boolean;</td>
</tr>
<tr>
<td>function RemoteFileVersion(DotPath, FileName: string): string;</td>
</tr>
<tr>
<td>function SendAll(ToUsers, Subject, ContentFile, Values, Files: string): string;</td>
</tr>
<tr>
<td>function TotalOn(DotPath, Condition: string; Recursive: boolean): integer;</td>
</tr>
<tr>
<td>function RemoteQueryToString(qAlias, qUserName, qPassword, qSqlStr: string): string;</td>
</tr>
</tbody>
</table>
function RemoteQueryToPage(qAlias, qUserName, qPassword: string; qPage, qPageSize: Integer; qSqlStr: string; out qPageCount: Integer): string;
function ChangePassword(DotPath, OldPassword, NewPassword: string): boolean;
function GetMessageHtml(DotPath, MsgID: string): string;
function Tasklnfo(DotPath: string): string;
function UpdateMessageInfo(DotPath, MsgID, Values: string): Boolean;
function UpdateMessageContent(DotPath, MsgID, ContentFile: string): Boolean;
function UpdateMessageFile(DotPath, MsgID, AttachFile, FileName: string): Boolean;

function RunScript(Script, Params: string): string;
function RunScriptFromFile(DotPath, FileName, Params: string): string;

function anGetValue(EntryName, MsgID: string): string;
end;

implementation

procedure TRConnection.DoConnect;
begin
  if not Connected then
    try
      Connect;
    except end;
  end;
function TRConnection.Login: Boolean;
var ss: string; ll: integer;
begin
  Result := False;
  try
    DoConnect;
    if Connected then
      try
        SendCmd ('LOGIN ' + Encrypt(FUserName) + ' ' + Encrypt(FPassword), 1);
        ss := ReadLn;
        ll := Pos('~', ss);
        if ll > 0 then begin
          FUserName := Copy(ss, 1, ll-1);
          FFullName := Copy(ss, ll+1, Length(ss));
        end else
          FFulName := ss;
        Result := True;
      except end;
    except
      Disconnect;
      except end;
      //ShowMessage('Can’t connect to remote server.');
      Result := False;
    end;
  end;
procedure TRConnection.Logout;
begin
  if Connected then Disconnect;
end;
function TRConnection.Send(ToUser, SubEntry, Subject, ContentFile,
Values, Files: string): Boolean;
var tmpStrings: TString;
i: integer;
begi
  Result := False;
  tmpStrings := nil;
  if not Login then Exit;
  if connected then
    try
      SendCmd ('SEND ' + Encrypt(ToUser) + ' ' + Encrypt(Subject) + ' ' + Encrypt(ContentFile), 1);
      //if exists, then do nothing
      Result := True;
    except end;
end;

tmpStrings := TStringList.Create;
SendCmd ( 'RCPT +ToUser+' +SubEntry, 1);
SendCntent( Subject, ContentFile );
tmpStrings.Text := StringReplace(Values,,\#13#10,[rfReplaceAll]);
for i:=0 to tmpStrings.Count-1 do begin
  SendCmd( 'VALUE ' +StringReplace(tmpStrings[i],',',',',1));
end;
tmpStrings.Text := StringReplace(Files,,\#13#10,[rfReplaceAll]);
for i:=0 to tmpStrings.Count-1 do begin
  SendAccachFile(tmpStrings[i]);
end;
SendCmd ( 'FINISH', 1);
tmpStrings.Free;
Result := True;
except
  if Assigned(tmpStrings) then tmpStrings.Free;
  SendCmd( 'RSET', 1);
end
else begin
  ShowMessage('Can't connect to remote server. ');
end;
Disconnect;
end;

function TRConnection.GetMessageList( 
  DotPath: string): string;
begin
  Result := "
  if not Login then Exit;
  if Connected then begin
    SendCmd( 'MSGLIST '+DotPath, 1);
    Result := GetStreamStr;
    end;
  Disconnect;
  end;
end;

function TRConnection.GetMessageInfo(DotPath, 
  MsgID: string): string;
begin
  Result := "
  if not Login then Exit;
  if Connected then begin
    SendCmd ('MSGINFO '+DotPath+' '+MsgID, 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;

function TRConnection.GetMessageFile(DotPath, MsgID, 
  FileName: string): string;
var fStream: TMemoryStream;
  FileSize: integer;
begin
  Result := "
  if not Login then Exit;
  if Connected then begin
    SendCmd ('MSGFILE '+DotPath+' '+MsgID+' '+FileName, 1);
    fStream := nil;
    try
      fStream := TMemoryStream.Create;
      FileSize := ReadInteger;
      ReadStream(fStream, FileSize);
      if GetResponse((1,0)) <> 1 then
        Result := "
    else begin
      Result := CompressionStreamToString(fStream);
    end;
    fStream.Free;
  end;
except
  if Assigned(fStream) then fStream.Free;
end;
end;
Disconnect;
end;

function TRConnection.GetMessageContent(DotPath, MsgID: string): string;
var fStream: TMemoryStream;
  FFileSize: integer;
begin
  Result := "\n  if not Login then Exit;
  if Connected then begin
    SendCmd ( 'MSGCONTENT '+DotPath+' '+MsgID, 1);
    fStream := nil;
    try
      FFileSize := ReadInteger;
      if FFileSize > 0 then begin
        fStream := TMemoryStream.Create;
        ReadStream(fStream, FFileSize);
        if GetResponse([1,0]) <> 1 then
          Result := "\n        else begin
          Result := CompressionStreamToString(fStream);
        end;
        fStream.Free;
      end;
    end;
    fStream.Free;
  end;
end;

function TRConnection.DeleteMessage(DotPath, MsgID: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ( 'MSGDELETE '+DotPath+' '+MsgID, 1);
      Result := True;
    except
      Result := False;
    end;
  end;
  Disconnect;
end;

procedure TRConnection.SaveMessageToFile(DotPath, MsgID, FileName, DiskFile: string);
var
  fStream: TMemoryStream;
  FFileSize: integer;
begin
  if not Login then Exit;
  if Connected then begin
    SendCmd ( 'MSGFILE '+DotPath+' '+MsgID+' '+FileName, 1);
    fStream := nil;
    try
      FFileSize := ReadInteger;
      if FFileSize > 0 then begin
        fStream := TMemoryStream.Create;
        ReadStream(fStream, FFileSize);
        WriteStream(fStream, FileName);
        fStream.Free;
      end;
    except
      Result := False;
    end;
  end;
  Disconnect;
end;
if FileSize > 0 then begin
  FileStream := TMemoryStream.Create;
  ReadStream(fStream, FileSize);
  CompressionStreamToFile(fStream, DiskFile);
  fStream.Free;
end;
except
  if Assigned(fStream) then fStream.Free;
end;
Disconnect;
end;

function TRConnection.GetResList(DotPath: string): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    SendCmd (\'RESLIST\'+DotPath, 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;

function TRConnection.GetAuthority(DotPath: string): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    SendCmd (\'AUTHORITY\'+DotPath, 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;

function TRConnection.SaveAuthority(DotPath,
Values: string): Boolean;
var
  AuStrings: TStringList;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    AuStrings := TStringList.Create;
    AuStrings.Text := StringReplace(Values,'\',\'\#13\#10\',[rfReplaceAll]);
    SendCmd (\'SAVEAUTHORITY\'+DotPath, 1);
    NewWriteStrings(AuStrings);
    AuStrings.Free;
    if GetResponse([1,0]) <> 1 then raise Exception.Create('Save Authority Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.anGetValue(EntryName, MsgID: string): string;
var
  IdList: TStringList;
begin
  IdList := TStringList.Create;
  if Pos(\',', EntryName) = 0 then
    EntryName := \'usr-\'+EntryName;
  try
    IdList.Text := GetAuthority(EntryName);
    except end;
    Result := IdList.Values[MsgID];
    IdList.Free;
  end;

function TRConnection.MakeResource(DotPath: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    Writeln('MAKERES ' + DotPath);
    Result := (GetResponse([1.0]) = 1);
  end;
  Disconnect;
end;

function TRConnection.EmptyResource(
  DotPath: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    Writeln('EMPTYRES ' + DotPath);
    Result := (GetResponse([1.0]) = 1);
  end;
  Disconnect;
end;

function TRConnection.DeleteResource(
  DotPath: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    Writeln('DELETERES ' + DotPath);
    Result := (GetResponse([1.0]) = 1);
  end;
  Disconnect;
end;

function TRConnection.ResourceExists(DotPath: string): Boolean;
begin
  Result := False;
  DoConnect;
  if Connected then begin
    try
      SendCmd ('RESEXISTS ' + DotPath, 1);
      Result := True;
    except end;
  end;
  Disconnect;
end;

function TRConnection.GetAdbPath(AdbName: string): string;
begin
  Result := ";
  DoConnect;
  if Connected then begin
    SendCmd ('ADBPATH ' + AdbName, 1);
    Result := ReadLn;
  end;
  Disconnect;
end;

function TRConnection.GetADBList: string;
begin
  Result := ";
  DoConnect;
  if Connected then begin
    SendCmd ('ADBLIST', 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;
function TRConnection.Hello: Boolean;
begin
  DoConnect;
  Result := Connected;
  Disconnect;
end;

function TRConnection.SendMessageInfo(DotPath, MsgID, Values: string): Boolean;
var
  AuStrings: TStrings;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    AuStrings := TStringList.Create;
    AuStrings.Text := StringReplace(Values, ',', '#13#10,[IfReplaceAll]);
    SendCmd ('SETMSGINFO '+DotPath+' '+MsgID, 1);
    NewWriteStrings(AuStrings);
    AuStrings.Free;
    if GetResponse([1,0]) <> 1 then raise Exception.Create('SendMessageInfo Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.LocalToRemote(LocalFile, DotPath, RemoteFile: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    SendCmd ('MSGSAVEFILE ' +DotPath+ '+IntToStr(SizeOfFile(LocalFile))+
    ' +ExtractFileName(RemoteFile), 1);
    WriteAFile(LocalFile);
    if GetResponse([1,0]) <> 1 then Result := False
else
    Result := True;
end;
Disconnect;
end;

function TRConnection.RemoteToLocal(DotPath, RemoteFile, LocalFile: string): Boolean;
var
  mStream: TMemoryStream;
  fStream: TFileStream;
  FileSize: integer;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    fStream := nil;
    mStream := nil;
    try
      //SendCmd ('MSGLOADFILE '+DotPath+' '+ExtractFileName(RemoteFile), 1);
      WriteLn('MSGLOADFILE '+DotPath+' '+ExtractFileName(RemoteFile));
      if GetResponse([1,0]) = 1 then begin
        mStream := TMemoryStream.Create;
        FileSize := ReadInteger;
        ReadStream(mStream, FileSize);
        mStream.Position := 0;
        fStream := TFileStream.Create(LocalFile, fmCreate);
        DecompressStream(fStream, mStream);
        fStream.Free;
        mStream.Free;
      end;
    finally
      if not mStream.Free then mStream.Free;
      if not fStream.Free then fStream.Free;
    end;
  end;
end;
Result := (GetResponse([1,0]) = 1);
end
else
  Result := False
except
  if Assigned(fStream) then fStream.Free;
  if Assigned(mStream) then mStream.Free;
end;
end:

function TRConnection.UserRight(DotPath: string): integer;
begin
  Result := -1;
  if not Login then Exit;
  SendCmd ('USERRIGHT' + DotPath, 1);
  Result := ReadInteger;
  Disconnect;
end;

function TRConnection.CopyMessageTo(aDotPath, MsgID,
bDotPath: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ( 'MSGCOPYTO' + aDotPath + ' ' + MsgID + ' ' + bDotPath, 1);
      Result := True;
    except
      Result := False;
    end;
    Disconnect;
  end;
end;

function TRConnection.TaskInsert(UserName, sDateTime, sDuration,
sImageIndex, sText: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ( 'TASKINS' + UserName + ' ' + sDateTime + ' ' + sDuration + ' ' + sImageIndex + ' ' + sText, 1);
      Result := True;
    except
      Result := False;
    end;
    Disconnect;
  end;
end;

function TRConnection.TaskDelete(UserName, sDateTime, sDuration,
sImageIndex, sText: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ( 'TASKDEL' + UserName + ' ' + sDateTime + ' ' + sDuration + ' ' + sImageIndex + ' ' + sText, 1);
      Result := True;
    except
      Result := False;
    end;
    Disconnect;
  end;
end;
function TRConnection.GetFileList(DotPath, SpecPath: string;
   Ext: WordBool): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    if Ext then
      SendCmd ( 'FILELIST '+DotPath+' TRUE '+SpecPath, 1)
    else
      SendCmd ( 'FILELIST '+DotPath+' FALSE '+SpecPath, 1);
    Result := GetStreamStr;
    end;
  Disconnect;
end;

function TRConnection.GetFileAge(DotPath,
   FileName: string): integer;
begin
  Result := -1;
  DoConnect;
  if Connected then begin
    try
      SendCmd ( 'FILEAGE '+DotPath+' '+FileName, 1);
      Result := ReadInteger;
    except end;
    end;
  Disconnect;
end;

function TRConnection.SearchFor(DotPath, FileExt,
   Context: string): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ( 'SEARCH '+DotPath+' '+FileExt+' '+Context, 1);
      Result := GetStreamStr;
    except end;
    end;
  Disconnect;
end;

function TRConnection.DetectNew(DotPath: string): Boolean;
begin
  Result := False;
  DoConnect;
  if Connected then begin
    try
      SendCmd ( 'DETECTNEW '+DotPath, 1);
      Result := True;
    except end;
    end;
  Disconnect;
end;

function TRConnection.GetFlowList(DotPath: string): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    SendCmd ( 'FLOWLIST '+DotPath, 1);
    Result := GetStreamStr;
    end;
  Disconnect;
end;
function TRConnection.DeleteFlow(DotPath, MsgID: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ('FLOWDEL '+DotPath+' '+MsgID, 1);
      Result := True;
    except
      Result := False;
    end;
    Disconnect;
  end;
end;

function TRConnection.GetLicence: string;
begin
  Result := ";"
  DoConnect;
  if Connected then begin
    SendCmd ('LICENCE', 1);
    Result := ReadLn;
  end;
  Disconnect;
end;

function TRConnection.GetResTree(DotPath: string): string;
begin
  Result := ";"
  if not Login then Exit;
  if Connected then begin
    SendCmd ('RESTREE '+DotPath, 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;

function TRConnection.SetMessageRead(DotPath, MsgID, ReadFlag: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    SendCmd ('SETMSGREAD '+DotPath+' '+MsgID+' '+ReadFlag, 1);
    if GetResponse((1,0)) <> 1 then raise Exception.Create('SetMessageRead Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.RemoteQuery(qType, qAlias, qUserName, qPassword, qSqlStr, qFileName: string): string;
var
  mStream: TMemoryStream;
  fStream: TFileStream;
  FileSize: integer;
begin
  Result := 'Connect Error:';
  DoConnect;
  if Connected then begin
    SendCmd ('QUERY '+qType+';'+qAlias+';'+qUserName+';'+qPassword+';'+qSqlStr, 1);
    if GetResponse((1,0)) = 1 then begin
      mStream := nil;
      try
        FileSize := ReadInteger;
        mStream := TMemoryStream.Create;
        if FileSize > 0 then begin
ReadStream(mStream, FileSize);
mStream.Position := 0;
FileStream := TFileStream.Create(qFileName, fmCreate);
DecompressStream(fStream, mStream);
fStream.Free;
end;
FileStream.Free;
Result := "";
except on e: exception do begin
  if Assigned(mStream) then mStream.Free;
  Result := e.Message;
end;
end;
else
  Result := LastCmdResult.Text.Text;
end;
Disconnect;
end;

function TRConnection.RemoteUpdate(qAlias, qUserName, qPassword,
  qSqlStr, qFileName: string): string;
var
  fStream: TFileStream;
begin
  Result := "";
  DoConnect;
  if Connected then begin
    SendCmd ('APPLYUPDATES '+qAlias+';'+qUserName+';'+qPassword+';'+qSqlStr, 1);
    try
      fStream := TFileStream.Create(qFileName, fmOpenRead);
      WriteInteger(fStream.Size);
      fStream.Position := 0;
      WriteStream(fStream);
      fStream.Free;
      if GetResponse([1.0]) <> 1 then
        Result := LastCmdResult.Text.Text;
    except on e: exception do begin
      if Assigned(fStream) then fStream.Free;
      Result := e.Message;
    end;
    end;
  Disconnect;
end;

function TRConnection.RemoteExecuteSQL(qAlias, qUserName, qPassword, qSqlStr: string): string;
begin
  Result := 'Connect Error';
  DoConnect;
  if Connected then begin
    try
      SendCmd ('EXECSQL '+qAlias+';'+qUserName+';'+qPassword+';'+qSqlStr, 1);
      Result := ReadLn;
      except
        Result := 'EXECSQL Error.';
      end;
    end;
  Disconnect;
end;

function TRConnection.RemoteTableNames(qAlias, qUserName,
  qPassword: string): string;
begin
  Result := "";
  DoConnect;
  if Connected then begin

SendCmd ( 'TABLENAMES '+qAlias+';'+qUserName+';'+qPassword, 1);
Result := GetStreamStr;
end;
Disconnect;
end;

function TRConnection.RemoteFieldName(qAlias, qUserName, qPassword, qTable: string): string;
begin
Result := ";
DoConnect;
if Connected then begin
SendCmd ( 'FIELDNAMES '+qAlias+';'+qUserName+';'+qPassword+';'+qTable, 1);
Result := GetStreamStr;
end;
Disconnect;
end;

function TRConnection.RemoteAliases: string;
begin
Result := ";
DoConnect;
if Connected then begin
SendCmd ( 'ALIASES', 1);
Result := GetStreamStr;
end;
end;

function TRConnection.RenameResource(OldPath, NewPath: string): Boolean;
begin
Result := False;
if not Login then Exit;
if Connected then begin
SendCmd ( 'RESRENAME '+OldPath+' '+NewPath, 1);
if GetResponse([1,0]) = 1 then
Result := True;
end;
Disconnect;
end;

function TRConnection.RemoteDeleteFile(DotPath, FileName: string): Boolean;
begin
Result := False;
if not Login then Exit;
if Connected then begin
try
SendCmd ( 'DELETEFILE '+DotPath+' '+FileName, 1);
Result := True;
except
Result := False;
end;
end;
Disconnect;
end;

function TRConnection.RemoteFileVersion(DotPath, FileName: string): string;
begin
Result := ";
DoConnect;
if Connected then begin
try
SendCmd ( 'FILEVERSION '+DotPath+' '+FileName, 1);
Result := ReadLn;
except
Result := ";
end;
end;
Disconnect;
end;

function TRConnection.SendAll(ToUsers, Subject, ContentFile, Values, Files: string): string;
var UsersList, tmpStrings: TStringList;
i: integer;
ss, tempFile: string;

procedure ErrorBreak;
begin
  if Assigned(UsersList) then UsersList.Free;
  if Assigned(tmpStrings) then tmpStrings.Free;
  try
    SendCmd ( 'RSET', 1);
    except end;
    Disconnect;
  end;
end;

begin
  Result := ";
  UsersList := nil;
  tmpStrings := nil;
  if not Login then begin
    Result := ToUsers;
    Exit;
  end;
  if connected then begin
    UsersList := TStringList.Create;
    UsersList.Text := StringReplace(ToUsers, ';', '#13#10, [rfReplaceAll]);
    for i:=0 to UsersList.Count-1 do begin
      if UsersList[i] = " then Continue;
      try
        SendCmd ( 'RCPT ' + UsersList[i], 1);
      except
        Result := Result + UsersList[i] + #13#10;
      end;
    end;
    if Result <> " then begin
      ErrorBreak;
      Exit;
    end;
    if FileExists(ContentFile) then begin
      try
        SendContent( Subject, ContentFile )
      except
        Result := Result + ToUsers + #13#10;
      end
    end
    else if ContentFile <> " then begin
      tempFile := GetTempFile;
      CreateRtfFile( tempFile, ContentFile);
      if FileExists(tempFile) then begin
        try
          SendContent( Subject, tempFile );
        except
          Result := Result + ToUsers + #13#10;
        end;
        DeleteFile(tempFile);
      end;
    end;
    if Result <> " then begin
      ErrorBreak;
      Exit;
    end;
    tmpStrings := TStringList.Create;
    tmpStrings.Text := StringReplace(Values, ';', '#13#10,[rfReplaceAll]);
    try
      for i:=0 to tmpStrings.Count-1 do begin
        SendCmd ( 'VALUE ' + StringReplace(tmpStrings[i], '"','=",[f]), 1);
      end;
    except
      
    end;
  end;
end;
Result := Result + ToUsers + #13#10;
ErrorBreak;
Exit;
end;
try
  tmpStrings.Text := StringReplace(Files,"","#13#10,[rfReplaceAll]);
  for i:=0 to tmpStrings.Count-1 do begin
    SendAccachFile( tmpStrings[i] );
  end;
except
  Result := Result + ToUsers + #13#10;
  ErrorBreak;
  Exit;
end;
try
  SendCmd ('FINISH', 1);
  Result := StringReplace(LastCmdResult.Text.Text, '#13#10, ".", [rfReplaceAll]);
except on E:Exception do begin
  ss := StringReplace(E.Message,'-ER","+rfReplaceAll);
  Result := Result + StringReplace(ss, ".", #13#10, [rfReplaceAll]);
  try
    SendCmd ('RSET', 1);
    except end;
  end;
tmpStrings.Free;
UsersList.Free;
Disconnect;
end;
end;

function TRConnection.TotalOn(DotPath, Condition: string; Recursive: boolean): integer;
var ss: string;
begin
  Result := 0;
  if not Login then Exit;
  if Connected then begin
    if Recursive then
      ss := 'TRUE'
    else
      ss := 'FALSE';
    try
      SendCmd ('TOTALON'+'DotPath+'+ss+' '+Condition, 1);
      Result := ReadInteger;
    except end;
  end;
  Disconnect;
end;

function TRConnection.RemoteQueryTostring(qAlias, qUserName, qPassword, qSqlStr: string): string;
begin
  Result := ";
  DoConnect;
  if Connected then begin
    try
      SendCmd ('QUERYTOSTRING '+qAlias+';'+qUserName+';'+qPassword+';'+qSqlStr, 1);
      Result := ReadLn;
    except end;
  end;
  Disconnect;
end;

function TRConnection.RemoteQueryToPage(qAlias, qUserName, qPassword: string; qPage, qPageSize: Integer;
  qSqlStr: string; out qPageCount: Integer): string;
begin
  Result := ";
  qPageCount := 0;
DoConnect;
  if Connected then begin
    try
      SendCmd
        ( 'QUERYTOPAGE
          +qAlias+q.Username+qPassword\+IntToStr(qPage)+\+IntToStr(qPageSize)+\+SqlStr, 1);
        qPageCount := ReadInteger;
        Result := ReadLn;
      except end;
    end;
  Disconnect;
end;

function TRConnection.ChangePassword(DotPath, OldPassword, NewPassword: string): boolean;
var OldPwd, NewPwd: string;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    OldPwd := Encrypt(OldPassword);
    NewPwd := Encrypt(NewPassword);
    SendCmd ('CHANGEPWD '+DotPath+'|'+OldPwd+'|'+NewPwd, 1);
    if GetResponse('[1,0]) = 1 then
      Result := True;
  end;
  Disconnect;
end;

function TRConnection.GetMessageHtml(DotPath, MsgID: string): string;
var ss: string;
begin
  Result := GetMessageContent(DotPath, MsgID);
  if Result <> '' then begin
    ss := UpperCase(Result);
    if Copy(ss, 1, 5) = 'rf\' then
      Result := RtfToHtml(Result)
    else if (Pos('<HTML>', ss) > 0) or
      (Pos('<TABLE', ss) > 0) or
      (Pos('<TR', ss) > 0) or
      (Pos('<TD', ss) > 0) or
      (Pos('<P>', ss) > 0) or
      (Pos('<BR', ss) > 0) then
      Result := Result
    else
      Result := StringReplace(Result, #13#10, '<br>', [rfReplaceAll]);
  end;
end;

function TRConnection.TaskInfo(DotPath: string): string;
begin
  Result := '';
  if not Login then Exit;
  if Connected then begin
    SendCmd ('TASKINFO '+DotPath, 1);
    Result := GetStreamStr;
  end;
  Disconnect;
end;

function TRConnection.UpdateMessageInfo(DotPath, MsgID, Values: string): Boolean;
var
  AuStrings: TStrings;
begin
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    AuStrings := TStringList.Create;
    AuStrings.Text := Values;
    SendCmd ('MSGUPDATEINFO 'DotPath+' 'MsgID, 1);
    NewWriteStrings(AuStrings);
    AuStrings.Free;
    if GetResponse([1,0]) <> 1 then raise Exception.Create('SetMessageInfo Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.UpdateMessageContent(DotPath, MsgID, ContentFile: string): Boolean;
var tempFile: string;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    if FileExists(ContentFile) then begin
      SendCmd ('MSGUPDATECONTENT 'DotPath+' 'MsgID, 1);
      WriteInteger(SizeOfFile(ContentFile));
      WriteAFile(ContentFile);
    end;
    else if ContentFile <> "" then begin
      tempFile := GetTempFile;
      CreateRtfFile( tempFile, ContentFile);
      if FileExists(tempFile) then begin
        SendCmd ('MSGUPDATECONTENT 'DotPath+' 'MsgID, 1);
        WriteInteger(SizeOfFile(tempFile));
        WriteAFile(tempFile);
      end;
      WriteInteger(SizeOfFile(ContentFile));
      DeleteFile(tempFile);
    end;
    if GetResponse([1,0]) <> 1 then raise Exception.Create('UpdateMessageContent Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.UpdateMessageFile(DotPath, MsgID, AttachFile, FileName: string): Boolean;
begin
  Result := False;
  if not Login then Exit;
  if Connected then begin
    SendCmd ('MSGUPDATEFILE 'DotPath+' 'MsgID+' 'AttachFile, 1);
    if FileExists(FileName) then begin
      WriteInteger(SizeOfFile(FileName));
      WriteAFile(FileName);
    end;
    else
      WriteInteger(0);
    if GetResponse([1,0]) <> 1 then raise Exception.Create('UpdateMessageFile Error');
    Result := True;
  end;
  Disconnect;
end;

function TRConnection.RunScript(Script, Params: string): string;
var
  sStrings: TStrings;
begin
  Result := "
  ```
if not Login then Exit;
if Connected then begin
  try
    SendCmd ('SCRIPT' ' Params, 1);
    sStrings := TStringList.Create;
    sStrings.Text := Script;
    NewWriteStrings(sStrings);
    sStrings.Free;
    Result := GetStreamStr;
  except end;
  Disconnect;
end;
end;

function TRConnection.RunScriptFromFile(DotPath, FileName, Params: string): string;
begin
  Result := ";
  if not Login then Exit;
  if Connected then begin
    try
      SendCmd ('SCRIPTFILE ' + DotPath + FileName + Params, 1);
      Result := GetStreamStr;
    except end;
    Disconnect;
  end;
end;

B.3 Syntax of Web Script Language

EBNF

The description of the syntax uses the Enhanced Bachus Nauer Form (EBNF).

- [ ] = Optional
- { } = Zero, one or many times
- Names in capital letters are symbols (non-terminals)
- Names in quotes ("name") are part of the input string (terminals)

Syntax

The start non-terminal is SCRIPT

SCRIPT =
  [ ROOTSTATEMENT { "," ROOTSTATEMENT } [ ",;" ]
ROOTSTATEMENT =
  TYPEDECL |
  PROCDECL |
```plaintext
STATEMENT =
  VARDECL |
  CONSTDECL |
  BLOCK |
VARDECL =
  "var" NAME ":=" TYPEDEF [ ":=" EXPR ]
TYPEDECL =
  "type" NAME ":=" TYPEDEF |
  "Boolean" | "Integer" | "Float" | "String" | "DateTime" | "Variant"
  NAME |
ARRAYDEF |
RECORDDEF |
CLASSDEF |
ARRAYDEF =
  "array" [ "[" EXPR ",.." EXPR [ "," EXPR ",.." EXPR ] ] ] "of" TYPEDEF |
RECORDDEF =
  "record" ARGLIST "end"
ARGLIST =
  ARGDECL [ ";" ARGDECL ]
ARGDECL =
  NAME ":=" TYPEDEF |
CLASSDEF =
  "class" [ "(" NAME ")" ] |
  [ "private" | "public" | "protected" ] |
  [ ARGLIST ",;" ] |
  [ METHODDECL [ ",;" METHODDECL ] ",;" ] |
  [ "property" NAME ":=" TYPEDEF [ "read" NAME ] [ "write" NAME ] ",;" ] |
  "end"
METHODDECL =
  [ "class" ] "procedure" NAME [ "(" ARGLIST ")" ] [ ",;" "override" | "virtual" |
  "reintroduce" ] |
  [ "class" ] "function" NAME [ "(" ARGLIST ")" ] ":=" TYPEDEF [ ",;" "override" |
  "virtual" | "reintroduce" ] |
  "constructor" NAME [ "(" ARGLIST ")" ] [ ",;" "override" | "virtual" | "reintroduce" ] |
  "destructor" NAME [ "(" ARGLIST ")" ] [ ",;" "override" | "virtual" | "reintroduce" ] |
PROCDECL =
  PROCHEAD ",;" PROCBODY |
  PROCHEAD ",;" "forward" |
  METHODDECL ",;" PROCBODY |
PROCHEAD =
  "procedure" NAME [ "(" ARGLIST ")" ] |
  "function" NAME [ "(" ARGLIST ")" ] ":=" TYPEDEF
```

PROCBODY =
    { VARDECL ";" } "begin" SCRIPT "end"

METHODDEF =
    [ "class" ] "procedure" NAME ";" NAME ["(" [ARGLIST] ")] |
    [ "class" ] "function" NAME ";" NAME ["(" [ARGLIST] ")] ";" TYPEDEF |
    "constructor" NAME ";" NAME ["(" [ARGLIST] ")] |
    "destructor" NAME ";" NAME ["(" [ARGLIST] ")] |

CONSTDECL =
    "const" "=" EXPR

BLOCK =
    "begin" [STATEMENT ";" STATEMENT] [";" ] "end" |

INSTR =
    "if" EXPR "then" BLOCK |
    "if" EXPR "then" BLOCK "else" BLOCK |
    CASEINSTR |
    "for" VAR ":=" EXPR "to" EXPR "do" BLOCK |
    "for" VAR ":=" EXPR "downto" EXPR "do" BLOCK |
    "while" EXPR "do" BLOCK |
    "repeat" [BLOCK ";" BLOCK] [";" ] "until" EXPR |
    VAR ":=" EXPR |
    FUNC |
    EXCEPT |
    FINALLY |
    BLOCK

CASESECOND =
    EXPR |
    EXPR ";" EXPR |

CASEINSTR =
    "case" EXPR "of" |
    { CASESECOND ";" CASESECOND } : BLOCK ";" |
    [ "default" ";" BLOCK ";" ]
    "end"

FUNC =
    NAME ["(" [EXPR , EXPR ] ")] |

EXCEPT =
    "try"
    BLOCK [ ";" BLOCK ] [";" ]
    "except"
    { "on" NAME ";" NAME "do" BLOCK; } |
    "end"

FINALLY =
    "try"
    BLOCK [ ";" BLOCK ] [";" ]
"finally"
    { "on" NAME ":" NAME "do" BLOCK; }
"end"

EXPR =
    EXPRADD [ "+" | "-" | "OR" EXPRADD]
EXPRADD =
    EXPRMUL [ "+" | "-" | "OR" EXPRADD]
EXPRMUL =
    TERM [ "+" | "/" | "mod" | "div" EXPRMUL]
TERM =
    "+" TERM |
    "-" TERM |
    "not" TERM |
    CONST |
    VAR |
    FUNC |
    "(" EXPR ")"

CONST =
    INT |
    HEXINT |
    FLOAT |
    STR |
    CHAR |
    "True" |
    "False"

VAR =
    NAME |
    NAME "[" INT "]" |
    NAME "." VAR |
    VAR "." FUNC

NAME =
    LIT { LIT | "0".."9" | "." }
LIT =
    "A".."Z", "a".."z"
STR =
    CHAR |
    STRING { CHAR [ STRING ] }
STRING =
    "+" { STRINGCHAR } "+" { "+" STRING }
STRINGCHAR =
    ASCII(0)..ASCII(255) - "+" - ASCII(13) |
    "+"
CHAR =
    "+#" INT | "+#" HEXINT
HEXINT =
"$" HEXNUM { HEXNUM } 
HEXNUM =
"0"..'9' | "A"..'F' | "a"..'f'
FLOAT =
   INT [ "." INT] [ "E" | "e" ["+" | "-" ] INT ]
INT =
   NUM {NUM}
NUM =
"0"..'9"

Reserved Words
- and
- array
- as
- begin
- case
- class
- const
- constructor
- destructor
- div
- do
- downto
- else
- end
- except
- finally
- for
- forward
- function
- if
- inherited
- is
- label
- mod
- nil
- not
- of
- or
- procedure
- property
- raise
- record
- repeat
- string
Database independence access object

```pascal
type
  TRemoteQuery = class(TADODataSet)
private
  function DetectPrimaryKey: Boolean;
  procedure SetTableName(const Value:string);
  procedure SetPrimaryKey(const Value:string);
  procedure SetEditFields(const Value:string);

  function GetSQL(TStringList;
  procedure SetSQL(value: TStringList);

  procedure WriteRecord(Writer: TWriter);
  function InfoToFile(State: Integer; FileName: string): boolean;
  function GetActive: Boolean;
  procedure SetRConnection(Value: TRConnection);

protected
  procedure SetActive(Value: Boolean); override;
  procedure InternalClose; override;
  procedure InternalDelete; override;
  procedure InternalPost; override;

public
  function AssignParams: string;
  function ExecSql: string;

published
  property Active: Boolean Read GetActive write SetActive;
  property RemoteConnection: TRConnection Read FRCconnection write SetRConnection;
  property Alias: String Read FAlias write FAlias;
  property UserName: String Read FUserName write FUserName;
  property Password: String Read FPassword write FPassword;
  property TableName: String Read FTableName write SetTableName;
  property PrimaryKey: String Read FPrimaryKey write SetPrimaryKey;
  property EditFields: String Read FEditFields write SetEditFields;
  property SQL: TStringList read GetSQL write SetSQL;
  property Params: TParams read FParams write FParams;
end;

implementation

procedure TRemoteQuery.SetTableName(const Value:string);
begin
  CheckInactive;
  FTableName := Value;
  FSQL.Text := 'select * from '+FTableName;
end;

procedure TRemoteQuery.SetPrimaryKey(const Value:string);
begin
  CheckInactive;
  FPrimaryKey := Value;
```
procedure TRemoteQuery.SetEditFields(const Value:string);
begin
  CheckInactive;
  FEditFields := Value;
end;

function TRemoteQuery.GetSQL: TStringList;
begin
  Result := FSQL;
end;

procedure TRemoteQuery.SetSQL(value: TStringList);
begin
  Close;
  FSQL.Assign(value);
end;

function TRemoteQuery.DetectPrimaryKey: Boolean;
var S, S1: string;
  Field: TField;
begin
  Result := True;
  if FPrimaryKey <> "" then begin
    S := FPrimaryKey;
    while S <> "" do begin
      S1 := StrDivide(S, ";");
      Field := FindField(S1);
      if Field=nil then begin
        Result := False;
        Break;
      end;
    end;
  end;
end;

function TRemoteQuery.GetActive: Boolean;
begin
  Result := Inherited Active;
end;

procedure TRemoteQuery.SetActive(Value: Boolean);
var i: integer;
  sqlStr, tmpFile, Err: string;
  tmpRConnection: TRConnection;
begin
  if not Assigned(FRConnection) then
    Inherited SetActive(Value)
  else if FLoading then Inherited SetActive(Value)
  else begin
    if Value then begin
      if FSQL.Text = "" then
        Raise Exception.Create('SQL is empty');
      tmpRConnection := TRConnection.Create(nil);
      tmpRConnection.Host := FRCConnection.Host;
      tmpRConnection.Port := FRCConnection.Port;
      sqlStr := StringReplace(AssignParams, #13#10, ", [rReplaceAll]);
      tmpFile := getTempFile;
      FLoading := True;
      Err := tmpRConnection.RemoteQuery('OPEN', FAlias, FUserName, FPassword, sqlStr, tmpFile);
      if Err = "" then begin
        LoadFromFile(tmpFile);
        Inherited SetActive(Value);
      end;
      FLoading := False;
      DeleteFile(tmpFile);
    end;
end;
tmpRConnection.Free;
if Err <> "" then
  Raise Exception.Create(Err);
end
else
  Inherited SetActive(Value);
if Active and (FPrimaryKey <> ") and (not DetectPrimaryKey) then
  Raise Exception.Create('Primary key invalid');
if FEEditFields="" then begin
  for i:=0 to FieldDefs.Count-1 do begin
    if i>0 then FEEditFields := FEEditFields + ":
    FEEditFields := FEEditFields + FieldDefs[i].Name;
  end;
end;
end;
end;

procedure TRemoteQuery.SetRConnection( Value: TRConnection );
begin
  FRConnection := Value;
end;

procedure WriteVariant(Writer: TWriter; Value: Variant);
var
  VType: Integer;
  procedure WriteValue(V: TValueType);
  begin
    Writer.Write(V, SizeOf(V));
  end;
begin
  VType := VarType(Value);
  with Writer do begin
    //WriteInteger(VType);
    case VType and VarTypeMask of
      varEmpty: WriteValue(varNil);
      varNull: WriteValue(varNull);
      varOleStr: WriteWideString(Value);
      varString: WriteString(Value);
      varByte, varSmallInt, varInteger: WriteInteger(Value);
      varSingle: WriteSingle(Value);
      varDouble: WriteFloat(Value);
      varCurrency: WriteCurrency(Value);
      varDate: WriteDate(Value);
      varBoolean:
        if Value then
          WriteValue(varTrue) else
          WriteValue(varFalse);
      else
        try
          WriteString(Value);
        except
          WriteValue(varNil);
        end;
    end;
  end;
end;

procedure TRemoteQuery.WriteRecord(Writer: TWriter);
var i: integer;
begin
  with Writer do begin
    WriteInteger(FieldCount);
    for i:=0 to FieldCount-1 do begin
      Writer.WriteString(Fields[i].FieldName);
      Writer.FlushBuffer;
      WriteVariant(Writer, Fields[i].AsVariant);
      Writer.FlushBuffer;
    end;
function TRemoteQuery.InfoToFile(State: Integer; FileName: string): boolean;
var i: integer;
  mStream: TFileStream;
  Writer: TWriter;
  FldList: TStrings;
begin
  mStream := nil;
  Writer := nil;
  try
    mStream := TFileStream.Create(FileName, fmCreate);
    Writer := TWriter.Create(mStream, 10240);

    //Write PrimaryKey
    FldList := TStringList.Create;
    FldList.Text := StringReplace(FPrimaryKey,';', '#13#10,[rfReplaceAll]);
    Writer.WriteString(FldList.Text);
    Writer.FlushBuffer;
    for i:=0 to FldList.Count-1 do begin
      Writer.WriteString(FldList[i]);
      Writer.WriteString('#13#10');
      Writer.FlushBuffer;
      if State = 2 then
        WriteVariant(Writer, FieldByName(FldList[i]).Value)
      else
        WriteVariant(Writer, FieldByName(FldList[i]).OldValue);
    end;
    FldList.Free;

    //Write State
    Writer.WriteInteger(State);
    Writer.FlushBuffer;

    //Write Current Record Value
    WriterRecord(Writer);

    Result := FileExists(FileName);
    finally
      if Assigned(Writer) then Writer.Free;
      if Assigned(mStream) then mStream.Free;
    end;
  end;
end;

procedure TRemoteQuery.InternalPost;
var tmpFile, ss: string; Flag: integer;
begin
  if (FPrimaryKey <> '') and (not FLoading) then begin
    tmpFile := getTempFile;
    if State = dInsert then
      Flag := 0
    else
      Flag := 1;
    if InfoToFile(Flag, tmpFile) then begin
      ss := StringReplace(AssignParams, #13#10, ',', [rfReplaceAll]);
      ss := FRemoteConnection.RemoteUpdate(FAlias, FUserName, FPassword, ss, tmpFile);
      if ss <> '' then begin
        DeleteFile(tmpFile);
        raise Exception.Create(ss);
      end;
      end;
      DeleteFile(tmpFile);
      end;
inherited InternalPost;
end;
procedure TRemoteQuery.InternalClose;
begin
  inherited InternalClose;
end;

procedure TRemoteQuery.InternalDelete;
var tmpFile, ss: string;
begin
  if FPrimaryKey <> " then begin
    tmpFile := getTempFile;
    if InfoToFile(2, tmpFile) then begin
      ss := StringReplace(AssignParams, #13#10, ", [rfReplaceAll]);
      ss := FRCConnection.RemoteUpdate(FAlias, FUserName, FPassword, ss, tmpFile);
      if ss <> " then begin
        DeleteFile(tmpFile);
        raise Exception.Create(ss);
      end;
    end;
    DeleteFile(tmpFile);
  end;
  inherited InternalDelete;
end;

function TRemoteQuery.AssignParams: string;
var p: integer;
  NewValue: string;

function GetParamValue(Param: TParam): string;
begin
  Result := ";
  if not Assigned(Param) then Exit;
  case Param.DataKind of
    ftString, ftFixedChar, ftMemo, ftTime, ftDateTime:
      Result := QuotedStr(ParamAsString);
    else
      Result := ParamAsString;
  end;
end;

procedure SetParam(var Value: string; FromPos: integer);
var ss, ParamName: string;
  i: integer;
  Param: TParam;
begin
  ParamName := ";
  ss := Copy(Value, FromPos+1, Length(Value));
  for i := 1 to Length(ss) do begin
    if ss[i] in [', ',', ', ']', #13, #10] then Break;
    if ss[i] = #30 then Continue;
    ParamName := ParamName + ss[i];
  end;
  Param := FParams.FindParam(ParamName);
  Value := Copy(Value, 1, FromPos-1) + GetParamValue(Param) + Copy(Value, FromPos+1+Length(ParamName), Length(Value));
end;

begin
  Result := FSQl.Text;
  if FParams.Count = 0 then Exit;
  if Pos("", Result) = 0 then Exit;
  NewValue := StringReplace(Result, ", #30, [rfReplaceAll]);
  p := Pos(#30, NewValue);
  while p <> 0 do begin
    SetParam(NewValue, p);
    p := Pos(#30, NewValue);
  end;
  Result := NewValue;
end;
function TRemoteQuery.ExecSql: string;
var ss: string;
begin
  ss := StringReplace(AssignParams, #13#10, ",", [rfReplaceAll]);
  Result := FRConnection.RemoteExecSQL(FAlias, FUserName, FPassword, ss);
end.
end.
Appendix C

List of Publications by Thesis Author


