A Federated Approach to Enterprise Integration

by

George Fernandez

A Thesis Submitted in Total Fulfillment of the Requirements for the Degree of Doctor of Philosophy

Faculty of Information and Communication Technologies
Swinburne University of Technology

2006
Abstract

In order to remain competitive, the integration of their information systems is an imperative for many large organisations. Applications that originally have been developed independently are now required to interoperate to support new or different functions of the enterprise. Although the mechanisms for application interoperation exist provided by the technology, due to the sheer number and complexity of the running systems, integration solutions — centralised or distributed — appropriate at the local level do not translate successfully to the whole enterprise. Centralised integration approaches often satisfy only some of the integration requirements, they are very expensive, and are fraught with danger since they imply an ‘all or nothing’ approach. Distributed approaches, on the other hand, suffer from complexity and scalability problems as the number of system interfaces to be implemented and the number of execution-time invocations grows with the number of component applications.

This dissertation makes a contribution to the field of Enterprise Application Integration (EAI) within the framework of distributed systems technology. Based on real-life case studies experience, we present here a federated approach that controls the size and complexity of the integration effort by reusing existing systems as much as possible and reducing the number of interacting applications. Only selected local elements are exposed to the organisational milieu, and a consistent supporting infrastructure is provided to make systems interactions possible. Our approach provides a flexible and scalable strategy to enterprise integration, avoiding the shortcomings of traditional approaches. We respect existing organisational structures, and demonstrate how appropriate federation infrastructure and protocols enable the interoperation of existing systems. The three main facets of enterprise knowledge are systematically incorporated into the integration effort: a) by the use of domain ontologies to support data integration; b) by the development of a methodology to include business rules; and c) by the development of FEW, a federated workflow model to implement the business processes of the organisation.
Acknowledgments

I owe a great deal to my Principal Supervisor Professor Doug Grant and to my Second Supervisor Associate Professor Jim McGovern; without their advice, encouragement and generous guidance this project would have not been possible.

I also have a special debt of gratitude with my colleague Inji Wijegunaratne. His industry experience, and his insights into enterprise computing and enterprise architecture have had the most important influence on my approach to enterprise integration.

To all of them, my heartfelt thanks.

George Fernandez
Declaration

I certify that this thesis:

• contains no material which has been accepted for the award of any other degree or diploma;

• to the best of my knowledge contains no material previously published or written by another person except where due reference is made in the text of the thesis; and

• where the work is based on joint research or publications, I disclose the relative contributions of the respective authors in the Appendix.
To

To my family, Maria Ines, Lucia and Ramiro, for their unwavering support.
# Contents

1 An Integrated Enterprise 1

1.1 Introduction .................................................. 1
1.2 Need for Enterprise Integration ................................. 4
  1.2.1 Hypothetical: AllBooks Publishers ......................... 4
1.3 Front-End vs. Back-End Integration ............................. 7
  1.3.1 Enterprise Systems ........................................... 8
  1.3.2 Enterprise Back-End Integration: Problems Faced ............ 10
  1.3.3 The Semantic Integration Problem ........................... 12
1.4 The Highest Level of Software Reuse .............................. 13
1.5 Objectives of This Dissertation: Research Questions ............. 16
  1.5.1 Introduction ............................................... 16
  1.5.2 Research Questions ....................................... 18
1.6 Contributions Of This Research ................................ 19
1.7 Outline of this Dissertation .................................. 20

2 Facets of Enterprise Integration 22

2.1 Enterprise Integration Strategies ............................... 22
2.2 Facets of Organisational Knowledge ............................. 26
  2.2.1 Semantic Heterogeneity ................................... 27
2.3 Data Integration ................................................ 28
  2.3.1 Data Integration vs. Database Integration .................. 28
  2.3.2 Metadata ................................................... 31
2.4 Rules Integration ............................................... 32
2.5 Process Integration ............................................ 34
2.6 Knowledge Integration ......................................... 35

3 Background to Enterprise Integration 37

3.1 Data Integration: Metadata and Data Interpretation ............. 37
| 6.5  | The Integrity of Corporate Entities                                      | 162 |
| 6.6  | The Structure of a Message Adaptor                                      | 164 |
| 7    | Federated Enterprise Integration                                        | 167 |
| 7.1  | A Sale in AllBooks Australia                                           | 167 |
| 7.1.1| Data-Oriented Analysis                                                 | 167 |
| 7.1.2| Process-Oriented Analysis                                               | 173 |
| 7.1.3| Mapping a Workflow to Federation Flows                                  | 178 |
| 7.2  | Evolution of a FEW Workflow                                             | 179 |
| 7.3  | Analysis Methodology:                                                   |     |
|      | Extends Business Rules Methodology of Section 5.4                      | 181 |
| 7.3.1| Data-Oriented Analysis                                                 | 182 |
| 7.3.2| Process-Oriented Analysis                                               | 183 |
| 8    | Conclusions                                                             | 185 |
| 8.1  | AllBooks Hypothetical Revisited                                        | 185 |
| 8.2  | Addressing the Research Questions                                      | 189 |
| 8.3  | Conclusions and Suggestions for Further Research                       | 192 |
List of Figures

1.1 Front-end integration strategies ........................................ 9
1.2 The situation in a major Australian bank-I ............................. 10
1.3 The situation in a major Australian bank-II ............................ 11
1.4 Business Components and Enterprise Systems ....................... 15

3.1 Semantic Domains ................................................................ 39
3.2 Zachman’s Framework of Enterprise Architecture .................... 63
3.3 Classification of Business Rules ............................................ 73

4.1 Three-Layered Enterprise Architecture .................................. 85
4.2 An Information-Oriented Interaction ..................................... 96
4.3 A Service-Oriented Interaction ............................................. 97
4.4 The Federation Highway ..................................................... 101
4.5 A Recursive Federation ..................................................... 104
4.6 Software Architecture Matches Enterprise Structure .............. 105
4.7 Layered Federations .......................................................... 106

5.1 Ontology Segment for a Bill ................................................. 118
5.2 An Ontology Segment for AllBooks Australia ....................... 120
5.3 An Ontology Segment ........................................................ 122
5.4 A FEW workflow ............................................................... 132
5.5 A relay link pair (Link = red, Relay links = green) ................... 137
5.6 Replacing a link (red) with a relay pair (green) ....................... 149
5.7 New workflow including a new checkpoint C ......................... 150

6.1 Levels of Federation Implementation .................................... 158
6.2 A Layered Structure for a Message Adaptor ......................... 165

7.1 Linking Corporate Entities to the Ontology ............................ 170
7.2 Sales data diagram ............................................................. 172
LIST OF FIGURES

7.3 Activity diagram for the Sale Workflow ........................................ 177
7.4 Sale Workflow for AllBooks Australia ........................................... 178
# List of Tables

3.1 Summary of Workflow Characteristics ........................................ 60
3.2 Terms and Facts ................................................................. 70
3.3 Action Assertions ............................................................... 71
3.4 Business rules and data semantics ........................................... 72
4.1 Federated Models Characteristics ............................................ 80
4.2 Service-Oriented (SOI) vs. Information-Oriented Interactions (IOI) .... 98
5.1 Summary of Workflow Characteristics ....................................... 140
6.1 Key Mapping for an AllBooks Customer ..................................... 155
6.2 Placement of Components ..................................................... 164
7.1 Activity Allocation and Scope ................................................. 174
7.2 Task and Applications .......................................................... 174
7.3 Federation Events ............................................................... 174
7.4 Activity Inputs (Primary) ...................................................... 175
7.5 Task Outputs (Primary) ........................................................ 175
7.6 Activity Input and Output (Non-Primary) ................................... 175
7.7 Activity Transactional Behaviour ............................................. 175
7.8 Outgoing Message Types and Parameters .................................. 176
7.9 Incoming Message Types and Parameters .................................. 176
7.10 AB International: Key Mapping for CE Book ............................. 180
Chapter 1

An Integrated Enterprise

1.1 Introduction

The information systems of a modern enterprise are developed and maintained to support the work of people within the organisation. Over the years these systems evolve, as new applications are introduced and old ones are enhanced. After a period, the IT infrastructure of the enterprise consists of a heterogeneous collection of applications supporting the different threads of enterprise activity. Changes in the business milieu continuously call for more and better functionality to be provided and, therefore, enterprise systems change over time creating a very dynamic environment. As a result, the information technology structure of an organisation of a certain (large) size is comprised of a plethora of applications that are not the result of a concerted, strategic approach but that are mainly the solution to individual localised problems. Rather, a closer inspection of enterprise Information Technology (IT) infrastructure reveals that very rarely has there been a concerted effort to develop systems reflecting the way they, as a whole, operate [124]. Instead, enterprise IT systems are usually response to successive tactical decisions made to solve the individual problems encountered by the different groups comprising the organisation. This usually results in systems that focus on, and many times satisfy, local immediate needs but do not offer suitable solutions from a global perspective in terms of scalability, maintainability, flexibility and cost-effective knowledge sharing.

The integration of enterprise systems, however, is an imperative for an organisation that wants to remain competitive in the business of today. In such an environment, applications that originally have been developed independently are now required to interoperate to support new and different functions of the enterprise. Although ideally enterprise software systems should enable the free flow of information through the organisation, and with other businesses, the reality is quite different. Out of more than 20 enterprises studied by Davenport et al [28], none was found to satisfy minimum levels of information sharing, even within the organisations anal-
ysed. Enlarged enterprises as a result of mergers and takeovers always find it very difficult to amalgamate their existing businesses, given the number of applications and the heterogeneity present in their systems [125]. Of late, independent organisations such as government agencies have been required to share information and engage in business-to-business operations, making the solution to the information flow problem even more pressing [44].

A number of standards and technologies, such as CORBA (Common Request Broker Architecture), .NET technology, Message Oriented Middleware (MOM), Web Services and Java-based communication technologies, have been developed to make possible interoperation among modules and applications distributed throughout an enterprise [45]. However, although the mechanisms for the interoperation exist provided by the technology, integration solutions normally successful at the local level do not scale up to enterprise-wide systems. This is mainly due to the sheer number and complexity of the running systems, which include a large number of components, different implementation technologies, disparate programming languages, heterogeneous platforms, non-homogeneous business rules and complex interactions among components. Thus, developing a strategy that enables systems to interact dynamically to effect enterprise application integration — called here back-end integration — is a very difficult task.

Traditionally, back-end enterprise integration strategies have been either centralised, based on all encompassing systems — such as Enterprise Resource Planning (ERP) systems — that replace existing applications, or distributed approaches — such as application suites or point-to-point interfacing — that interconnect existing systems to share data. Centralised integration approaches often satisfy only some of the integration requirements, they are very expensive, and are fraught with danger since they imply an ‘all or nothing’ approach to customisation and roll-out. In addition, centralised strategies usually require extensive business re-engineering and user training. Distributed approaches, on the other hand, suffer from complexity and scalability problems as the number of system interfaces to be implemented and the number of execution-time invocations grows with the number of component applications.

With the intention of addressing these complexity and scalability problems, federated systems have received significant research attention [44, 125, 130]. Federations are based on an approach that recognises and respects the existence of local systems and loosely integrates them by providing a structure within which meaningful sharing can occur. Thus, local systems maintain their autonomy to process the local tasks for which they have been implemented, while still cooperate for the processing of enterprise-wide activities. For this to occur, the integration strategy must consider the provision of structures and mechanisms for the completion of busi-
ness tasks by cooperating systems that now require the exchange and interpretation of data across their boundaries. Individual systems are still charged with processing enterprise tasks, but in the integrated enterprise they contribute to the overall business goals.

Hence, even if local systems maintain their autonomy, many elements of coordination and integration still have to be present. The various units comprising the organisation need to cooperate in processing the common tasks of the enterprise, and in doing so they are required to share the results of their computations and exchange information. In a centralised environment coherence is guaranteed by the central system. Loosely integrated, heterogeneous systems, however, need to coordinate for tasks that require their multiple intervention, overcoming all facets of their heterogeneity.

This research focuses on a description of organisational business processes and mechanisms to guarantee their successful completion, and the provision of support for the semantic interpretation of enterprise data and rules within a loosely integrated organisation. Enabling information flow, and ensuring the correctness and timeliness of enterprise-wide businesses processes are fundamental goals of this dissertation.

This research makes a contribution to the field of Enterprise Application Integration (EAI) within the framework of distributed systems technology. Supported by the implementation experience of industry case studies, we establish a federated strategy that respects existing organisational structures, and provides a flexible and scalable approach to enterprise integration. Such an enterprise federation enables the interoperation of existing systems, avoiding many of the shortcomings of traditional approaches and providing a more affordable, progressive solution to the integration of the large enterprises that are the focus of this research. In a departure from more traditional approaches, our integration strategy involves three major facets of organisational knowledge and determines how they are incorporated into the framework:

**Enterprise data integration:** making possible the flow and correct interpretation of enterprise data throughout the organisation.

**Enterprise business workflows:** providing mechanisms to ensure that business processes are enacted and completed correctly.

**Enterprise rules:** complying with the rules pertaining to the operational and control knowledge of the organisation.

Our solution controls the number of interacting systems interfaces by grouping applications
into loosely-coupled clusters that are the centres of all processing. Point-to-point interfaces are ruled out, and instead cluster interoperation is made possible via a federated infrastructure that supports the sharing of enterprise information. To this end, we introduce a suitable architecture and information systems structure, and define the required set of services and protocols, to effect enterprise systems integration.

The hypothetical AllBooks situation presented in the next section illustrates the type of problems that large organisations confront for lack of integration.

1.2 Need for Enterprise Integration

1.2.1 Hypothetical: AllBooks Publishers

AllBooks Publishers started as a medium size company in Melbourne selling technical books, but it expanded very quickly by merging with an editorial company in New Zealand specialised in medical books to become AllBooks International. The merger was to take advantage of the synergies between the different products by offering their customers discounts and other advantages if they were loyal to the company by conducting both type of business with AllBooks. Based on the success of their sales force, which were promptly rewarded with generous commissions, the organisation expanded with branches in other capital cities in Australia and New Zealand. It was considered crucial that salespeople maintained at least the same level of satisfaction with their jobs, so new incentive schemes were introduced, such as rewards for the highest selling branch and salesperson. Initially, the attempts to provide the necessary information for the new way of doing business were paper and file based. With the intention of reusing existing systems as much as possible, lists of customers, sales, books, and other data were exchanged on paper or via computer files by Head Office and the different branches and groups within their branches, such as Sales and Warehouse. Very soon a stream of problems started to emerge:

1. The processing of an agent’s commission is housed at Human Resources (HR), which computer also runs another mission-critical legacy application. Each time a new order is issued the sale information is entered into the local client application and a transaction with the central system is fired to update the commission of the employee. However, the HR legacy system is slow, and many times the transaction is left hanging with the order application waiting for the system to respond. Often, the transaction times out, and it is necessary to kill the process and start over, frustrating the branches who get bogged down with the agent’s commission (which is not even their problem). Adding to their frustration, staff at HR are
not very responsive when transaction logs are required back at branch level — it may take up to a week to get them — and branch staff are always under pressure to finish their reports on time. The solution of replacing the central computer system is perceived as too expensive and risky by management, but they are reluctantly considering it because they would like to furnish their sales force with mobile computers to do the data entry only once, and issue invoices directly when they are in the field. They think that the legacy system will not be able to support the new functionality.

2. In the original AllBooks environment, a central database stored most of the business information and was used to drive the data access needs of the company. Despite the initial assessment, after the merger it was apparent that the same scheme housed in Australia could not be used to support the New Zealand business. Firstly, New Zealand had their own systems, using their own data with their own formats and their own access and consistency rules, and they could not contemplate changing them since they were still needed to support their local operations. Secondly, technical staff argued that for performance reasons it would be impossible to call up the Australian database in real time every time a data operation was required in New Zealand. In any case, a central database in Australia would require major changes to support the NZ operation, and that was judged to be too complex and expensive.

3. In the long term, AllBooks International plans to expand into South East Asia, and there are serious concerns about the capacity of some of the planned branches to support the new operations synchronously. Managers are very apprehensive about slow and unreliable communications and the impact that this may have in their operations.

4. Identifying spending patterns to reward good customers proved to be more difficult than expected. In the first instance, some of the applications processing orders and were already interconnected by the use of files, database tables of direct writing to the destination application space, so the same strategy was used to link the remaining ones. Nightly processes perform the updates. However, this not only has significantly increased the number of required intermediate structures, but since these represent point-to-point connections, their format is dependent on the requirements of the two intervening applications. This situation quickly degenerated into a maintenance nightmare, and something had to be done before integrating any new system.

5. The company has instituted a program to reward the better performing units. Each branch is required to provide sales information to Head Office, where it is aggregated, analysed and made public. This has created tension because Australian units report sales totals as they are entered into the Orders database, regardless of whether they have been invoiced and paid
for, whereas New Zealand units use a system that forces them to hold off closing a sale until the payment has been made. The inclusion of 10% Goods and services Tax in the Australian totals makes the problem worse, since the sales in NZ are tallied before tax. Since NZ totals appear lower because of this, these units have protested that this is unfair for them.

6. As illustrated by the two points immediately above, there is an urgent need to manage and control business processes that encompass the newly incorporated units, making them coherent and consistent with the overall operation of the company. There is a general lack of data consistency and integrity throughout the AllBooks organisation. Overlapping data exists in different data stores — customers, products, staff, all have intersecting data segments in different applications — and with the current arrangements is not possible to maintain these segments synchronised. Staff details, for example, which are represented in email, Human Resources, Phone Directory and other Web-based applications, are very hard to keep up to date.

7. The IT staff is unhappy about the high complexity of the systems. The impact of a change is always extensive, and this affects their capacity to respond rapidly to user requests. Furthermore, because of real-time interactions between functions of different modules, a system cannot function without all its components, so when something goes down the IT department is under pressure to fix the problem immediately. A problem with invoicing, for example, often means that invoicing needs to be off-line for half a working day, and this is a major problem since the absence of a working invoicing module also affects order data entry. This will be even more significant to AllBooks’ operations when they want to start issuing invoices in the field since this type of problems occur quite often and they feel they will be powerless to take remedial action.

8. There are also important autonomy and privacy considerations troubling AllBooks. Because of the disclosure required by the rewards program, the different groups are in direct competition with each other for the bonuses. Thus, they would like to have complete control of the information they are making available to the rest of the organisation, and group managers have complained about this to top management. In addition, staff members affected would like to keep the bonuses and the corresponding information as private between management and themselves so they feel uncomfortable with making this public.

9. Due to the high number of requests, central services often have a very poor response time. Branch staff would prefer not to involve central systems unless it is absolutely necessary. They would like to keep a significant degree of autonomy, with their own operations supported by local computer-based resources, which they have demonstrated they are able to manage.
1.3: Front-End vs. Back-End Integration

10. Top management are concerned that if they adopt an ‘all or nothing’ approach and implement an ERP (Enterprise Resource Planning) solution, the required changes could be very expensive and disruptive, and that would have a negative impact on the performance of the units. In addition, despite assurances from the vendors, they are not convinced that this type of solution would be extensible to satisfy not only current but also future needs. They are afraid that once a particular ERP product is chosen it would be very hard to change later on. It is also not clear how such a solution would integrate with the new CRM (Customer Relationship Management) package that they are ready to install.

As illustrated by the AllBooks case, the last decade has produced a shift in the view of an organisation, from a centralised to a distributed structure. The different units or departments comprising a (distributed) enterprise structure find themselves managing their own applications, working with a high degree of autonomy albeit needing to cooperate to carry out the business of the enterprise. In addition, there is a growing need for knowledge intensive applications — Data Warehousing and Data Mining, Decision Support Systems, Information Management Systems, Web-enabled customer management — that require collecting and interpreting data from different heterogeneous sources. A central goal of enterprise integration is to make possible the coordination and interoperation of these systems to produce coherent, efficient enterprise processes. To this end, it is necessary to clearly establish the meaning of enterprise knowledge, to make possible the controlled exchange of data and the proper coordination of enterprise processes.

1.3 Front-End vs. Back-End Integration

The previous section presents the problems faced by AllBooks when trying to consolidate their Australian and New Zealand systems, to enable people and systems to share procedures and information. In the course of their continuous evolution, and as a result of changes such as mergers and takeovers, as enterprises try to automate internal processes and offer better interaction possibilities to external agents, they face new challenges both in terms of their business and technology structures. New developments in technology, such as Web Services, and their possible use as integration tools show promise, but still many problems in terms of connectivity, performance and security are to be resolved before they may be widely adopted [118, 119]. Much of the research and development has been invested in the development of technology, but the crucial question of how to integrate still remains [8, 68].
1.3.1 Enterprise Systems

Enterprise systems gather information as a result of the interaction of the enterprise with users and user processes — customers both inside and outside the organisation, suppliers, other enterprises engaged in business-to-business communication — and by exchanging data between the different units, such as departments, branches and cost centres within the organisation, or with other organisations [84]. An enterprise would in the first place like to exploit new ways of dealing with customers and suppliers (such as telephone, Internet-based, customer representatives in the field) in a secure and flexible way, and then leverage the information so captured throughout the organisation. This highlights the two different types of application services found in a typical enterprise:

- Application services that directly support human-system interaction, both with people internal and external to the organisation. These services support one or more business functions, such as taking an order, creating a new account or issuing an insurance policy. They have been described as Task Oriented Services (TOS), or Transaction Based Services (TBS) [124]. As a result of their interaction with users and their processes, these services typically initiate software events significant for the organisation, such as a New Books Order or an Update to Customer Information. For the modern enterprise, the challenge is to leverage the different types of access technologies (browser, proprietary GUI, phone, WAP, etc.) to maximum effect by providing appropriate, secure, controlled access to the required business functions, and properly support user tasks once access has been granted.

- Application services that store and manage information that may be consumed by other systems within the same organisation, or by external systems when engaged in business-to-business communication. These services are designed to interact mainly with other components, usually without direct human intervention. As a consequence of an event occurring as described above, a system component captures information of global significance and propagates the event related information to other interested systems. Here, the objectives are to make possible the integration of knowledge, keeping down the cost and effort invested, moving to a more responsive, flexible and manageable environment. Ideally, to avoid out-of-synchronicity problems, business events would have to propagate quickly through the enterprise, reducing latency as much as possible.

Therefore, although both aspects are often called enterprise integration in the literature, we posit that the enterprise integration problem should be essentially divided into two types:

**Front-end integration:** enables controlled access to application services (functions) to different users and user communities, even when these communities use different channels of
access. Figure 1.1 depicts different options for front-end integration strategies, from the simplest (portals) to application launchers, to a common access channel (a single, seamless access point). The figure shows the increase in complexity as the seams between systems are increasingly being hidden by the front end.

Ideally, a common structure would be provided to support user interactions with the various system functions. This structure would consolidate back-end functions into an access hub that takes care of the required access control, user and service mappings, session management, and logging and auditing. Although we have produced some preliminary work on this area, this facet of enterprise integration is out of the scope of this dissertation [41].

**Back-end integration**: application to application or system to system integration. The goal is to provide consistent, reliable support to enable applications and clusters of applications — within the organisation, as well as with other organisations — to share organisational knowledge. This includes the provision of the necessary structures and mechanisms to enable the proper sharing of enterprise knowledge among systems.

Presenting these patterns and bringing them together to form a unified enterprise architecture is a major task. A detailed specification of each, and the corresponding unification process and resultant structure is too large an undertaking to be within the scope of this research. Hence, we would like to centre our attention here exclusively on the second aspect, back-end integration. Although this is frequently called in the literature Enterprise Application Integration, the focus of this research is broader in granularity and scope: we intend to provide an architectural framework and infrastructure to enable enterprise-wide sharing of knowledge between systems, or clusters of systems comprising many applications. In this dissertation we
shall use the generic term enterprise integration (EI) to refer to enterprise back-end integration.

1.3.2 Enterprise Back-End Integration: Problems Faced

As in AllBooks, in the vast majority of cases the computer systems supporting the internal functioning of the enterprise have been developed independently to manage the tasks of individual business units: customer management systems, back office systems, order systems, warehousing, are some examples. Existing systems manage their own data, following their own data models and governed by their own business rules. This leads to the technical problems experienced by AllBooks International, in which applications are not able to communicate with each other with the reliability and efficiency essential to safely implement the complex business processes of a modern enterprise. This is, in a significant part, due to technological difficulties as a result of ad-hoc integration solutions, such as remote calls, direct reading and writing to application space, or bridges and file transfers used for communication. In addition, applications cannot easily exchange information as the formats and representations of enterprise data are different, and therefore numerous complex individual interpretations are required.

Figure 1.2: The situation in a major Australian bank-I

Figure 1.2 and Figure 1.3 depict the actual situation at a major Australian bank in the mid
Figure 1.3: The situation in a major Australian bank-II
90s and 2004 respectively. The boxes represent applications, with arrows or lines showing their point-to-point interconnections. A comparison of the two figures shows that not only the problem has not been solved over the period, but that it may have actually become even worse. The quandary for the bank analysts is how to properly manage the development maintenance and evolution of these systems. Not only it is difficult to determine what is to be shared between applications, but the use of point-to-point interfaces results in a myriad of interface formats and technologies that have to be created and maintained, making it very difficult to accommodate the new applications that inevitably will be required.

1.3.3 The Semantic Integration Problem

These shortcomings, though, only contemplate the syntactic and technological aspects of information exchange. However, it is the size and complexity of the semantic problems concerned that resist the most advanced technical solutions. Despite the significant research effort invested in the provision of frameworks, methods and tools for enterprise integration, the problem does not seem to be close to a satisfactory solution [65]. Unfortunately, the data, rules and procedures relevant to a large organisation are exceedingly complex. Usually they are not a coherent set, nor they are clearly specified in an unambiguous form [128]. They reside within individual organisational units, often embedded into the applications in charge of organisational tasks, having evolved as a consequence of their immediate needs, experience and environment. In this sense, organisational knowledge is essentially a distributed proposition. Consequently, the integration effort has many heterogeneity problems to overcome, including:

- Due to the existing semantic differences and its sheer size, organisational knowledge is usually too vast and complex; hence, knowledge sharing is an overwhelming undertaking for a large organisation.

- Because it has evolved independently within many different groups, the organisational knowledge that needs to be considered for integration often:
  - lacks a common language;
  - possesses different conceptualisations of shared elements;
  - includes imprecise and often contradictory definitions and rules;
  - suffers from poor documentation;
  - is stored and supported by heterogeneous computing systems.
Different applications follow their own business rules for the management of their data and processes. The structure of the data and rules for updates are likely to be different — they are actually likely to be conflicting — for different applications.

Few people in the organization possess intimate knowledge of a given application. This is even more acute in the case of a system outside their immediate jurisdiction, such as an application belonging to a different organisational group.

In a large enterprise, applications show a great degree of heterogeneity in terms of functionality, data models and technology, making any sharing difficult.

The models used for application development, such as Entity-Relationship and Object-Oriented, consider only the structure of the data and, thus, they are not rich enough to represent the complexity of business rules and the relationships among the entities involved. This results in a significant loss of semantic information for stored enterprise data, which results in inaccurate or improper understanding of information exchanges.

The totality of a global business process is not always represented within any one application, but it is the result of tasks coordinated — typically by people — among several of them. This raises issues of task coordination, completion, and error recovery.

There are security and privacy considerations when information originally intended for local, constrained consumption is made available to the wider milieu.

The following section presents such an enterprise integration approach against the background of software reuse.

1.4 The Highest Level of Software Reuse

The issue of enterprise integration is very closely linked to the concept of systems and software reuse. If it were possible to develop all systems from scratch every time new functionality was required, there would be no need for enterprise integration. The reality is, however, that organisations cannot afford the expense and disruption that this approach would engender. On the contrary, the goal of organisations such as AllBooks ought to be to tackle the problems encountered by leveraging off their existing systems, minimising the changes required to their IT structure. The recognition of existing organisational processing groups and their supporting systems as contributing parts of the overall IT effort is a first step towards this goal. This approach is the result of the evolution, both in the time and granularity dimensions, of the
notion of software reuse.

Traditionally, software engineering has been concerned about software reuse. In its simplest form, reuse is based on reusing code segments. In the 1960s and 70s this originated the notion of structured programming, supported by programming languages such as Algol, Modula and Pascal that made possible writing programs that were better structured, easier to understand and that, due to their higher time and address space localisation properties, produced more efficient compilation and execution code.

Structured programming, however, provided only code implementation reusability. That is, although code segments and functions could be reused, programmers still required the inclusion of that code into their programs. During the 80s and 90s object-oriented programming provided the possibility of reusing the constructed version of an object. Once an object is created, it is usable ‘as is’ via its publicised interface, without the need to incorporate the object’s code into the program. Objects may be downloaded when needed from where they reside; however, they are not addressable at a network node, the object may not be referred to as a network-based entity. In contrast, a collection of such objects with a standardised interface protocol (e.g. EJB, CORBA, .NET) housed and addressable at a network node may provide also execution time reusability as a distributed component. These are the building blocks of modern application engineering.

There are, however, higher levels of software reuse afforded by the new developments in distributed technology from the late 90s to date, which are applicable to IT based organisations. Firstly, a business component housed at a network node — such as an application — may provide information and functionality to other similar components at a different node, even if they are based on different platforms and technology. Examples of this may be an Order Management module, an Insurance Policy application, an Invoice Manager, or a Customer Relationship Management application. Secondly, a collection of such components may be grouped to provide a related collection of services such as Insurance Policy Management, Billing, Human Resources and Warehouse Management. These enterprise-level components are typically the ones used to support tightly integrated, enterprise-level functions.

As mentioned, this is the result of an evolution along the time dimension from the 1960s, but it is also an evolution on the size of the reusable components, which has increased with the time. From simple code segments to enterprise-level components, software reuse targets elements of ever increasing size. Moreover, developments in technology have enabled the ready incorpora-
tion of smaller reusable elements into coarser grained components. Thus, objects are grouped into collections to form business components, and business components are themselves grouped into enterprise-level systems (See Figure 1.4).

At the highest end of the reusability spectrum we find the efforts of governments and similar authorities to consolidate information and services from disparate systems servicing government agencies. Due to the size of their investment in IT — approximately U$S50 billion annually [44] — a clear leader in this effort in the area is the Federal Government of the USA. The intention of the Federation of Government Information Processing Councils/Industry Advisory Council (FGIPPC/IAC) is to manage information technology and systems to be able to provide visibility and accessibility to the different federal agencies IT ‘silos’. Due to the size of the problem and the inherent heterogeneity present in their systems, the idea of integration by coalescing systems together into a whole is replaced instead by the idea of interoperation, in which systems are able to communicate with each other meaningfully. Thus, the intention of FGIPPC/IAC is to enhance the understandability and interoperability of the existing systems, rather than presenting a unified system. In this way, agencies are able to keep control of their own data and processing, but agreeing on common syntax, semantics and mechanisms for interoperation.

The evolution of information systems has followed this trend of increasing size of reuse, from code segments to enterprise systems. Accordingly, the problem of a complete centralised integration to be managed by an all encompassing system has become intractable, and the solution instead should rely on the provision of support for system interoperation.
1.5 Objectives of This Dissertation: Research Questions

1.5.1 Introduction

Back-end enterprise integration as it is the focus of this dissertation is related to enterprise-level components in a large, IT-mature organisation. Such an enterprise is likely to be spread over geographically dispersed locations, where autonomous processing of enterprise tasks occurs. Our assumption is that these identifiable enterprise groups are supported by enterprise-level components — e.g. clusters of applications and systems — which are the elements that will become the building blocks of our integration strategy. Our intention is to map units of organisational architecture to enterprise-level components, albeit not necessarily in a one-to-one relationship. Sometimes this mapping may be immediate, because the software component stands out by itself — e.g. a Customer Relationship Manager (CRM) package, Order Management at the Sales department, Inventory Management at Warehouse — some other times the enterprise group comprises several interoperating systems that may be considered a single enterprise-level component. Thus, we take here a flexible view of an enterprise level component as a collection of people, software components, tasks and procedures — Head Office, Warehouse, Sales, the New Zealand branch — that produce well-identified enterprise processes. Although a detailed discussion of this issue is postponed until Chapter 4, this results in clustering together applications and systems that support an organisational unit. Our integration strategy aims to reuse these components whole, by establishing protocols and mechanisms to make possible their interoperation.

The most basic mechanism of back-end enterprise integration involves the sharing of data across different systems. Although this often is taken to mean that a remote system accesses data items stored in a database, the reality of enterprise systems indicates that for safety and security reasons this is very rarely the case. The most common situation is that a database may only be accessed by executing a service — function, stored procedure, method — provided by a standard interface. Additionally, this mechanism also allows information belonging to different systems to be aggregated and returned as a structure to the client as a result of a single request. We consider this derived or calculated data also part of the enterprise data to be integrated.

However, from our discussion of the previous sections we conclude that two other aspects are also relevant to enterprise integration. Firstly, the tasks and procedures that make possible the completion of the organisational business processes. Secondly, the business rules that govern these organisational processes. Although the latter two aspects may have received less research attention within the context of enterprise integration than data integration, we argue that the
three aspects should be considered in unison when attempting enterprise integration, since all three aspects are manifestations of enterprise knowledge [130]. Considering data integration by itself results in ad-hoc interpretation of processes and rules, rather than a comprehensive integration strategy. The integration must contemplate how a process involving several units is able to correctly complete all its tasks, and provide adherence to the rules imposed by the organisation as a whole, as well as the individual units and the applications within them.

However, for an application or an organisational unit to make sense of remote knowledge it is necessary that the meaning of the knowledge, including all facets relevant to the integration, is clearly established. From this point of view, we regard organisational semantics as the information that makes the meaning of knowledge aspects as clear and unambiguous as possible, thereby providing the understanding necessary to properly make use of the knowledge. As discussed in Chapter 4, a federation offers an infrastructure that facilitates the exchange of information between component domains, but it does not determine what information is to be exchanged, or how it is to be interpreted by data sources and receivers. This is a primary concern of Enterprise Integration as it is presented here.

We intend to address the problems of Figures 1.2 and 1.3 by aiming to:

- Develop a consistent and efficient system interfacing paradigm, instead of point-to-point interfaces.
- Establish what forms should these interfaces take, in terms of technology and structure.
- Provide an architecture and support structures for system interoperation.
- Establish protocols to keep the integrity of shared data.
- Determine how, and what which systems, an application may share information and processes with the rest of the enterprise.
- Show how it is possible to contain the complexity and cost of the integration project.
- Present a flexible, progressive integration strategy.
- Show how the integration of new applications can evolve following the evolution of the IT systems structure.

The operation of an enterprise is based on the storage and management of its corporate knowledge, based on three different, but complementary, aspects:
• Data: The concepts of the organisation, their characteristics and the values that represent them. This aspect includes the terms used by the enterprise, their meaning (semantics), and their formats (syntax).

• Processes: The procedures that carry out the enterprise tasks, producing outcomes by managing, and reasoning about, the data.

• Rules: The business rules that govern processes and the management of data. These rules are often set locally by the organisational units — mainly through the business rules of their applications — to ensure the consistency of their tasks and the integrity of local data. Local rules must properly interrelate with the rules that govern the organisation as a whole.

It is, therefore, along all these three dimensions that our integration effort is focused.

1.5.2 Research Questions

This main objective of this dissertation is:

\[\text{to provide an architecture and mechanisms for enterprise back-end integration, to ensure the proper interpretation of data semantics for the correct completion of business processes, according to the organisation’s business rules.}\]

This main objective has been refined into the following sub-objectives:

• Classify the problems presented by the integration of enterprise knowledge.

• Identify the shortcomings of the current enterprise back-end integration strategies.

• Establish conditions for a distributed enterprise integration strategy to be successful.

• Present and justify a federated approach to enterprise integration.

• Develop a set of federation services to support the integration strategy.

• Establish appropriate sharing protocols to ensure data integrity.

• Develop a methodology to include business rules and processes in the integration strategy.

• Demonstrate an ontology-based approach for data integration.

• Develop a low coupling workflow model with transactional characteristics to effect process integration.

• Show how a federated integration strategy may follow the path of enterprise evolution.
1.6 Contributions Of This Research

This dissertation contributes to the field of Enterprise Application Integration as follows:

• It identifies the shortcomings of current EAI strategies.

• It presents a low-coupling federated architecture to properly support the integration by breaking down the integration problem into smaller manageable segments, and by the progressive incorporation of processes, clusters of processing and layers of services.

• It includes in the integration three crucial facets of enterprise knowledge.

• It presents a semantic, ontology-based approach to enterprise data integration.

• It develops a set of federation protocols to ensure the correct management of shared enterprise data.

• It develops a methodology to include business rules in the analysis of the integration.

• It introduces FEW, a low coupling, publish-and-subscribe transactional workflow model to carry out enterprise business processes within a federated framework.

• It presents a methodology for the analysis and implementation of enterprise workflows based on the FEW model.

• It develops a scalable, flexible, progressive and extensible strategy for integration.

To answer these research questions, and to address the research objectives of the previous section, this dissertation aims to reuse as much as possible the experience of the experts in each field, by utilising existing standard structures and tools — standard ontologies, business rules and workflow tools — rather than developing new ones. For example, in Section 5.1, we argue that to store semantic descriptions standard ontologies should be preferred to in-house ones, just linking the latter to the former when necessary. In addition, there has also been considerable research into business rules, and how to properly represent and express them; we follow here a particular form, but others may also equally be used. Similarly, workflow research and corresponding tools have received significant attention by researchers and practitioners, and many of these developments may be used within the context provided by this research.

Instead, we intend to provide an architecture, and appropriate structures and behavioural rules so these standard constructs may be appropriately used in the integration. An enterprise may have many different reasons to choose one ontology over another, or one workflow description or tool over another. Thus, the focus of this dissertation is to provide a framework to use these
regardless of the choice, and to present the necessary models and protocols for a successful integration of the three aspects of enterprise knowledge considered.

Some aspects of this research, however, are predicated on certain assumptions — e.g. the existence of delineated enterprise groups, or certain capabilities of the middleware infrastructure — but these will be part of the requirements of the framework and clearly specified in the discussion.

1.7 Outline of this Dissertation

The remainder of this dissertation proceeds as follows:

Chapter 2: Discusses current integration strategies, their advantages and shortcomings. In addition, it presents the different facets of enterprise knowledge, and discusses the requirements and constraints associated with the integration of these facets.

Chapter 3: Investigates research work closely aligned with this research, along the three facets of enterprise knowledge.

Chapter 4: Discusses federated strategies and software systems architecture to introduce the federation. It introduces the notions of transient inconsistencies and information-oriented interactions, which are the basis of the federated architecture. Two very significant case studies of proto-federations implementations are discussed at the end of chapter.

Chapter 5: Introduces a knowledge-based approach to enterprise integration, including an ontological approach to data integration, integration contexts, a business rules methodology and the \textit{FEW} workflow model.

Chapter 6: Presents the federated infrastructure to support the integration, including the different services that the federation should provide, and consequently the different levels of federation involvement in the strategy. This chapter also introduces federation protocols to ensure the integrity of shared data.

Chapter 7: Introduces the methodology to perform the required data-oriented and process-oriented analyses. It also discusses how to map a business workflow as a combination of federation flows, and how a \textit{FEW} workflow may follow the path of business evolution.

Chapter 8: Revisits the hypothetical case of AllBooks International, and shows how the federated implementation has addressed their original problems. The chapter also demonstrates
how the research questions have been addressed by this dissertation, and presents concluding remarks and suggestions for future research.
Chapter 2

Facets of Enterprise Integration

This chapter provides in Section 2.1 first a discussion of the most common current strategies for enterprise back-end integration, and an analysis of the advantages and the shortcomings of each. The following sections present the different facets of enterprise knowledge, and their characteristics with respect to enterprise integration, that are the concern of this dissertation.

2.1 Enterprise Integration Strategies

The problems illustrated in Figure 1.2 and Figure 1.3 stem from the fact that enterprise systems such as the ones shown have been developed, and have subsequently evolved, independently. When there was a need, the systems were furnished with ad-hoc, point-to-point interfaces to make possible the data exchanges. This exemplifies a particular approach to application integration, the most common in enterprises since very rarely there is a concerted approach to developing and deploying systems with evolution in mind.

However, point-to-point is not the only approach to Enterprise Integration. In general terms, EI strategies may be classified as follows:

**Point-to-point strategies**: A very common approach to application integration, that partially replaces people-intensive interactions such as emails and file transfers. This approach has obvious advantages in terms of providing a quick solution and return on investment, and it may be a good strategy if the systems to integrate are only a few. However, this strategy often:

- results in a number of interfaces that grows inordinately with the number of applications, in each case needing to map the different data formats and semantics;
• it is based on non-reusable interface development; hence there is a need to start anew every time;
• lacks a standard, established target system architecture; this implies that each change has to be handled in its own individual way, making it very difficult for analysts to see what impact any alteration might have on the overall system structure;
• it is of limited flexibility, as the interactions are hard-wired, making it difficult to accommodate new developments;
• it is characterised by a high number of complex interconnections; thus, once the integration project is advanced it is very difficult to contain the ripple effects of changes to existing applications;
• it necessarily follows the rules of the individual applications; thus is not possible to ensure coherence with organisation-wide rules; and
• it lacks overarching control over the sharing process to ensure that enterprise-wide procedures are complete and consistent.

**Enterprise Resource Planning (ERP) packages:** This strategy aims to support the work of at least a major part of the organisation. ERP systems provide comprehensive support for the correct management of enterprise data and processes, and that is a main reason for their success. When such systems are deployed with minor modifications, they tend to provide an affordable solution to organisational needs, and since they are supported by vendors and consulting firms, they are often preferred by organisations.

However, an ERP strategy may have shortcomings:

• seldom covers the totality of a large organisation, often requiring messaging services to integrate autonomous systems [65].
• is rarely found to suffice completely, even for restricted integrations. In a recent experience, a large global enterprise implemented an ERP package with the intent of standardising their back office operations. However, it was later found that in the order of 500 integration touch points were required, a significant number against existing back office systems that, contrary to their original thinking, the analysts in the end decided could not discard [123].
• if may requires extensive customisation or otherwise impose significant changes to the organisation;
• its complexity may result in a considerable integration burden, and may result in disappointing functionality, late delivery and cost overruns [10, 59].

• it may result in local systems being replaced or made compliant to the all-embracing scheme;

• may be unsettling to the normal functioning of an enterprise, since such new systems tend to change existing rules and processes and replace them with their own embedded ones, they take a long time to bed down, and their roll-out is complex, involving staff re-training;

• with the introduction of new versions, the maintenance of these systems tends to be a continuous process, with new systems becoming legacy in an ever-changing environment; and

• lacks flexibility, since due to the big investment the enterprise finds itself ‘locked-in’ with the proprietary ERP system.

**Application suites:** These provide an infrastructure to interconnect systems supporting organisational processes, such as Customer Relationship Managers (CRM), ERP systems and Data Warehouses. Often, the supporting software provides an enterprise backbone that connects to the applications via the use of pre-built adaptors (See for example Hasselbring [65]). A rule-based system manages data transformation and routing. These solutions enable data sharing and ensure data integrity, but at a cost:

• implementation is long and complex, and usually requires expensive outside expertise;

• since there is application logic embedded into the backbone infrastructure, the strategy is not flexible to accommodate enterprise evolution and integration to other systems;

• maintenance must be performed by specialists hired for the purpose;

• since the solutions use proprietary protocols, it is not possible to extend the integration to heterogeneous systems in the enterprise or other enterprises, unless they use the same product;

• it is difficult to cross network boundaries, since the protocols used are not standard; and

• scalability is limited;

**Event-driven strategies:** These strategies are based on the detection and propagation of software events within the organisation. When an event occurs the change of data is delivered to interested parties across the organisation. This type of connectivity delivers the
information where it is required, to existing applications. In this way, applications are not
discarded, but they continue their normal processing with a more comprehensive set of data.
Data sources and receivers need not be aware of each other, since the interaction is usually
based on publish-and-subscribe mechanisms. Nonetheless, the following shortcomings should
be considered:

- a reliable infrastructure must be provided to deliver the data across systems;
- the delivery of data must be guaranteed, to be able to maintain consistency across systems;
- events must be identified, and their effect on the enterprise evaluated and documented;
- the number of systems to be integrated results in a large number of data items to be
delivered;
- due to the different meaning of enterprise data for different systems, there is a need for
data interpretation and translation services; and
- the programming effort may be considerable for a large organisation.

Centralised vs. Distributed EI

In general, centralised approaches to enterprise integration such as the ones provided by an
all-encompassing system — ERP, application integration suites — are usually risky due to
their complexity and size. In addition, these integration projects require an ‘all or nothing’
approach that makes their implementation and deployment very difficult. Even minor changes
to the organisation require extensive systems re-engineering, therefore making it very hard to
accommodate any enhancements. A centralised integration model will experience also tech-
nological conflicts, as systems based on very different technologies that were not designed to
share information — e.g. legacy systems — are now required to do so. Besides, this approach
imposes a rigid target structure, so new applications to be incorporated require major changes
to adapt them to the existing framework.

On the other hand, distributed integration strategies — point-to-point or event-driven — often
suffer from complexity and scalability problems due to the intricacy and diversity of a large
modern organisation IT structure. When two applications exchange information, either by
sharing data or by remote invocations, they require understanding of each other. If only by
data sharing, the format and semantic meaning of the data must be established. If it is an
invocation-based exchange, details such as the number and type of the parameters, module lo-
cation at a network node, and return values must be known and correctly interpreted. Although
modern middleware software, such as the Web Services Framework, provides the infrastructure and support to resolve these calls in a flexible and dynamic way, still the resolution of these aspects of remote communication is a difficult task, since the necessary invocation information must be stored and managed in a way that keeps it readily available.

In addition, distributed integration is based on remote communications, which are much more involved than local ones, since they rely on layers of technology — middleware, network — to execute a call. The response time of a remote invocation is quite different to a local one, so design decisions have been found to significantly affect the performance of an application [103]. The reliability of a remote invocation is also influenced by the reliability of the communication infrastructure [102]. Hence, the characteristics of each interaction type supported by the middleware have to be taken into account when designing an interaction, otherwise there will be performance and reliability implications. For example, a service-based invocation has higher coupling\footnote{Coupling is formally introduced in Section 4.4.1.} than an information only exchange, and therefore this approach has difficulties scaling up as the number of invocations increases [124]. Hence, a distributed EI strategy must take the infrastructure into account and design and implement accordingly.

In summary, centralised integration strategies entail some risks because of their all-or-nothing, protracted implementation, and their lack of flexibility. On the other hand, distributed strategies tend to become unwieldy due to the proliferation of interacting interfaces, the lack of common syntax and semantics and their heavy reliance on the communication infrastructure. To succeed, a distributed strategy must address these shortcomings.

\section{2.2 Facets of Organisational Knowledge}

An integration strategy aims to make possible that groups of people and their supporting systems, which are in charge of the execution of organisational processes, share enterprise information. We will be assuming here that these clusters of people and systems are mostly self-sufficient, and that they naturally share their local data and terminology, and the understanding of their practices and methods. This might be the result of having a common purpose (the Sales Department, Warehouse, a company branch), running the same applications, sharing a common physical space, or belonging to the same technical group. Due to this common day to day communication and to their personal interactions, these units naturally share the same conceptualisations of their data. Members of these groups are in continuous contact with each
other, and in their frequent discussions they establish the necessary understanding to be able to successfully exchange information. However, this approach to knowledge sharing is not easily transportable to the whole organisation. The size and complexity of a modern enterprise, the physical distance between groups and the mostly sporadic nature of their interactions makes it difficult to maintain the consistent dialog required to be able to successfully integrate the many complex human and technological facets of knowledge.

2.2.1 Semantic Heterogeneity

As a result, in most enterprises there is no homogeneity in the meaning of the data exchanged, or the operations shared, among systems. Data items, tasks and processes are often interpreted differently by the various groups. In addition, different applications are very likely to follow different business rules related to operations such as accesses or updates. For example:

- a sale could be seen as ‘complete’ by the Sales application when the customer fills out the order, but Head Office may have the view that the sale is complete only when the corresponding invoice is issued and the bill paid; furthermore, Warehouse may see the sale as complete only when the goods are dispatched;

- total turnover could be reported in different currencies or using different scale factors (e.g. dollars versus thousands of dollars) for branches in different countries;

- a rule specifying that a customer may have only one contact telephone number may be valid for one application, but invalid for another application that may accept several such contact numbers;

- a salary can be before or after tax, in Australian (AUD) or NZ (NZD) dollars, or weekly for one application and fortnightly for another.

The problems presented by this lack of semantic consistency are exacerbated by the fact that very rarely in an organisation it is possible to find comprehensive information about the data items and processes managed by the different applications. Data usually reside in local files or databases, and the knowledge about their exact meaning is buried within the applications themselves, and the people running them. Even when data items are described, such as in a database, typically this description is syntactic rather than semantic — the format of data items as a string, an integer, or a real number — since this is necessary in order for the computer systems to manage the items properly. In small or highly localised enterprises where the business context is understood, the users themselves provide the required missing knowledge: users know whether a salary is weekly or fortnightly, in New Zealand (NZD) or Australian
(AUD) dollars, or before or after tax. However, for a large organisation, especially one in which previously independent applications are required to share knowledge, these disparities have to be identified and resolved before meaningful sharing can occur.

The sale and telephone numbers examples above exemplify that not only data may have different interpretations, but that there may be also important semantic discrepancies in the interpretation of rules and business process relevant to the whole enterprise. Traditionally, the meaning of an enterprise process, such as a New Order, New Customer or Renew Insurance Policy, is contained within the application in charge of the execution of the activity\(^2\). In an integrated enterprise, however, a business process may transcend an individual application, system or group, requiring the cooperation of several applications within different enterprise units for its successful completion. If this is so, business processes and rules will suffer from semantic heterogeneity problems similar to the ones afflicting the data.

We intend to provide models, mechanisms and protocols to be able to integrate the different aspects of enterprise knowledge. Although we will be assuming that the enterprise is federated in the way described in Chapter 4, most of our discussion in this chapter applies without any change to any IT-mature, large organisation, regardless the structure of its systems architecture.

The following sections discuss how the different aspects of enterprise knowledge have, independently, been considered in the research literature.

\section{2.3 Data Integration}

\subsection{2.3.1 Data Integration vs. Database Integration}

The problem of heterogeneity of corporate data is akin to the problem of database heterogeneity, which has been discussed extensively in the research literature [80, 113, 114, 120]. The discussion centres on the different types of heterogeneity that may occur when databases schemata are developed independently, and what are the consequences of this for integrated and multidatabase systems. Similarly, enterprise data integration needs to overcome heterogeneity present in the different systems, so it is important to compare and contrast the characteristics of both problems.

\footnote{In this dissertation, the notion of activity is more generic than the term task, which is associated with an activity supported by an individual application. However, the literature often uses both terms interchangeably, and we shall do the same if it does not lead to confusion.}
Database Integration: The problem of integrating heterogeneous databases exhibits the following characteristics:

- The focus of the significant effort invested in the area of database integration research is to provide support for the posing of database queries in heterogeneous environments, and for the correct interpretation of query results.

- Heterogeneous databases start from the representation of data items for a particular (typically small) set of applications. The range of data is narrow, restricted to the schemata of the component databases, and the intention is to correctly map or translate the schemata to a coherent set of views. This approach attempts to integrate the constituting systems, intending to make sure that the semantics of the applications remain intact after the translation.

- Database heterogeneity must be resolved at the level of the queries to be issued to the integrated system: the system should appear as unified to the users. Traditionally, the approach is to either properly represent all the data within a single integrated global schema against which the queries would be put (shared schema, tightly integrated databases), or to provide transparent, efficient, query translations between the different schemata (loosely integrated, multidatabases).

- Database systems are based on certain technologies, such as relational or object-oriented, the characteristics of which have an impact on the structure and the operations allowed on the integrated data. For example, since the relational model is not expressive enough to represent the is-a relationship between manager and employee, applications are forced to make up for these shortcomings by the careful management of keys by the application.

- Database operations are controlled by a set of database rules such as integrity constraints and business rules governing use of the data, which also have to be integrated.

- In the component schemata, the same real world entities are very likely to exhibit conflicts in several orthogonal dimensions [80, 113]. These conflicts are often cumulative; thus, several components schemata are likely to show a combination of these.

- The integration process is often laborious, mostly performed manually by Database Administrators and users (sometimes supported by appropriate software), to solve the arising conflicts.

- The integration process must proceed at least one database component at a time, restricting a gradual incorporation of components.

- The database integration process typically covers all schema attributes for all components, whether they are shared or not. A notable exception — relevant to our discussion here
— is the case of federated databases, in which local systems expose only a subset of their entities (their Export Schema), which is then shared with other components [95].

- Tightly integrated databases are usually not extensible, and they hard to maintain since the consequences of schemata and attribute changes are difficult to contain.

**Data Integration:** Although the problem of enterprise data integration shows obvious similarities to the problem of database integration, it also presents some significant differences:

- Enterprise integration in the broadest sense presented here involves not only data but also process and rules. Since very often organisational businesses overarch departments and applications, enterprise integration transcends individual database constraints and application business rules. Thus, a higher level, all-encompassing integration strategy is required.

- A full integration of enterprise data needs to involve only the terms and concepts shared by the enterprise. Items relevant only to an application or a data store need not be part of the integration strategy. Similarly to federated databases, only elements shared by heterogeneous parties need to be considered. This has positive consequences for scalability, since the effort of creating and maintaining interoperation grows inordinately with the number of shared items [57, 72]. Scalability is one of the most important problems plaguing enterprise integration, and maintaining tightly coupled shared schemata has been recognised as a major system bottleneck [95, 113].

- Although many terms would be associated with an individual data item in a database, some others could be derived in the sense that they do not possess real existence within a data store but are a conceptualisation (e.g. Customer Rating, Personal Details), as a result, for instance, of a calculation or the execution of an application business function.

- Database integration aims to provide applications and users with a unified, transparent, consistent access to the data stored in the various database components. Data in the components are well specified (at least syntactically), documented and available. In contrast, enterprise data are dispersed throughout the organisation, perhaps undefined or loosely defined by a group of users, and accessible only locally via the execution of a method, function or stored procedure. The integration scheme must ensure that relevant local data is made available to the whole organisation.

- The number of users, the variety of applications and the dynamic nature of business in a modern organisation imply that it is not feasible to complete the integration process in one clean sweep. Applications not considered in the first instance or new applications will
2.3: Data Integration

have to be integrated at a later stage. Rules and procedures change, and new ones are introduced continuously. Therefore, a fundamental goal of enterprise integration should be to provide users and systems with accurate knowledge of the data, processes and rules of the enterprise, so they can make consistent use of these. In this way, the integration makes possible the controlled evolution of, and the incorporation of new, enterprise knowledge.

- Although enterprise integration may involve some database integration, it is not necessarily so. Data can be shared between applications without necessarily providing a unified view of the data in a database-like fashion. This can be achieved, for example, by allowing replicated data fragments and keeping their copies synchronised, with individual systems in charge of the required re-interpretation and mapping [125].

- Very importantly, there is no query requirement. The need for running queries against an integrated database poses very difficult technical problems, not because of the data interpretation, but mostly because of the required response time and the need for concurrency control to preserve basic database attributes such as the ACID (Atomicity, Consistency, Isolation, Durability) properties [75, 99]. This requirement is a consequence of the current state of database technology, not a necessity of enterprise data integration.

2.3.2 Metadata

In order to overcome the semantic diversity between components, some systems — such as the Global Information Resource Dictionary (GIRD) proposed by Hsu et al [70, 71] and the Intelligent Thesaurus of Kershberg and Weishar [79] — use metadata to store information about enterprise data. The most important feature of these models is the inclusion of contextual knowledge to make explicit semantic information that is missing or implicit in a data model. This contextual knowledge refers mainly to problem domain metadata, that is, specific knowledge about the entities of the problem domain of concern. The intention is to make domain data semantics explicit, to be able to detect mismatches and, when necessary, create meaningful mappings between data items [33, 34, 66, 77, 107].

A major problem with the metadata approach to semantic augmentation is the determination of what information about a certain data item must be provided to describe it completely. Only after establishing what about a data item is required, it is possible to elicit the required information, and subsequently store it and retrieve it to use it properly. This suggests a three-step approach:

- It is necessary to be able to elicit the information from data stores, applications and users.
2.4 Rules Integration

The central concern here is to extract the knowledge in a systematic rather than haphazard way: the analysts should know what about a data item or entity should be discovered in the data stores, the applications using the data, and the users. As a consequence, a more complete, unambiguous description of the data may be provided.

- It is necessary to store and retrieve the metadata information. The semantic information has to be stored so it can be readily retrieved when a transformation or interpretation is required. An information exchange will use the explicit information thus provided to implement mappings for local applications. The data contextual knowledge can be extracted from the metadata storage, and the mappings created for internal consumption. Naturally, different applications may require different translations to be performed.

- It is necessary to establish how the information is to be used by the business process. Not only the data exchange must be meaningful, but also it must properly and correctly contribute to the business of the organisation. In particular, data integrity is a central concern.

Attempts at defining data semantics through annotations attached to individual data items have been suggested in several instances [70, 111]. However, these tend to be rather verbose and cumbersome, and consequently this approach may be overwhelming for a large organisation. Rather, the model utilised should be scalable, and flexible enough to be able to accommodate the great number of items and processes involved. Consequently, some research in the area has focused in trying to reduce the number of items and the amount of information that it is required to manage about each item by using ontologies agreed upon by the parties exchanging information. Enterprise modelling [47], integration of Web resources and services [19, 30, 116], enterprise integration, cooperation and interoperability [25, 44, 120], have all benefited from an ontological approach.

2.4 Rules Integration

The partitioning of an enterprise into autonomous organisational units intends to reflect the way in which a large organisation operates. Tasks are concentrated within applications or groups of applications within these units, managing their own data and following their own business rules. Therefore, although the research and practitioner literature has covered extensively the topic of business rules and their management, the focus is on the rules for an application or a closely related group of applications [63, 100, 101]. That is, the scope of business rules analysis, creation, maintenance and management in the literature is always narrower than the one we
propose here, which intends to cover a whole enterprise.

Ideally, a complete set of business rules would be provided, stored and made available for all the tasks and processes of the enterprise. However, severe shortcomings may be mentioned against the provision of an all-encompassing business rules repository:

- As with data integration, it would be an insurmountable task to provide an enterprise-wide business rule model for a large organisation. The number and variety of applications makes it unworkable the development and upkeep of such a business rule repository.

- Such an approach would be really inflexible, given that the business rules of any new application would very likely clash with established organisational rules, and therefore it would need to be heavily customised, if at all possible, before it could be incorporated into the enterprise systems suite.

- All applications, regardless of their data model, internal business rules, and technology would have to be made to comply with the global schema.

- Typically, it is not even possible to change the business rules of an application. The IT structure of large enterprises always includes a significant number of off-the-shelf applications, the business rules of which are often embedded into the application logic. Because this logic is not always amenable to modification, off-the-shelf applications are customisable only to a certain extent:
  
  - Firstly, rules related to data updates — e.g. foreign keys and other data integrity constraints — are normally related to the individual application data model, and therefore not possible to alter other than minimally.
  
  - Secondly, in many cases for efficiency or technological reasons some constraints are not enforced by the Database Management System (DBMS) (e.g. a relational database may not be completely in normal form to avoid costly joins), but they are enforced by the application. This situation is very hard to discover in the code, and requires an intimate knowledge of the application that is not usually readily available.
  
  - Thirdly, in most cases either the source code is not available, or there is no consent by the vendor to be able to modify it; but even when the code is available and consent is granted, modifying the application to a significant extent is usually outside the expertise and the budget of most organisations.
  
  - Finally, even when modifications are possible, most enterprises are reluctant to implement substantial changes to an application, since this results in a version of the application
2.5 Process Integration

Over the past few years, workflows applications have received significant research attention. Several issues have been central to the discussions: workflow models and architectures, standards, completeness and correctness of workflows, transactional behaviour of workflows and implementation of workflow systems are some of the most significant ones [12, 13, 52, 76, 87]. In the vast majority of cases, the discussion centers on an application that completely controls the workflow, using a centralised database to store all the required data and knowledge about the process. This is so because traditional workflows are usually centred on tightly integrated processes, such as in a manufacturing environment, where the system manages a well-defined procedure by keeping all the threads of control.

In organisational workflows, however, if the integration strategy intends to maintain the autonomy of local systems to be able to control complexity, the same model is not immediately applicable. The occurrence of a new local system event — a new insurance policy, a new order, a change in a customer’s credit — may transcend the local system and result in a sequence of information exchanges with other remote systems. Still, it is necessary to ensure its correctness, the type, format and payload of the messages exchanged so to comply with the definition and structure of the corresponding enterprise business process. The integration strategy must

- sharply divergent from the one supplied, making it very difficult to manage upgrades and enhancements by the original vendor.

- The global schema would have to be adhered to by all applications at the same (cut-over) time, therefore making impracticable the roll out of the global business rule repository.

The process of business rules modification for all applications to comply with a global schema is not realistic.

Instead, we espouse here an approach that dispenses with the need for an overarching business rule schema. We argue that local systems should remain independent and work autonomously, following their own business rules for accesses and updates. We establish a stepwise methodology in which only the rules related to a relatively small number of interoperating units, and their shared data and processes are considered for integration, thus breaking down an intractable problem into more manageable segments. In addition, this makes it unnecessary that all systems follow a particular model, in order to make possible a gradual integration strategy.
provide the necessary services to take care of the generation of a proper message sequence to ensure the correctness and integrity of the business process.

Through the workflow layer, the integration must guarantee the overall correctness of the business processes. Although these services might appear similar to the ones provided by traditional workflows, they have significant differences:

- Traditional workflows are centralised, and this facilitates data management, consistency and coordination.
- The existence of autonomous processing units and their local autonomy constraint results in the need of the implementation of a distributed workflow.
- If they are to maintain their autonomy, individual systems should not be aware of the overall workflow. They should only be concerned with their internal processing and with executing the steps dictated by the workflow protocol.
- Traditional systems are able to define, manage, initiate, control and bring to a close the execution of a workflows via their internal engines. Enterprise distributed workflows should able to provide similar services, despite their lack of a central controlling structure.
- The integrated workflow protocols should be as loose as possible, to properly function in a distributed environment in which remote, heterogeneous systems interact.
- Despite being distributed, integrated workflows still should possess desirable characteristics such as correctness, failure atomicity and backward and forward recovery.

The next section presents a perspective in which all these three aspects of enterprise integration are considered.

### 2.6 Knowledge Integration

Although our discussion suggests that an enterprise-wide data, business rules and processes repository is prima facie not possible, the objectives of enterprise integration still require that these facets of enterprise knowledge are considered by the strategy. Even though it is not possible to create a central repository, the strategy must identify the aspects of enterprise knowledge present in an integration focus, and provide appropriate descriptions of these aspects and support for their involvement in the integration. This requires that the meaning of data, rules and processes is made clear, since:
• Information being the focus of the system exchanges, the meaning of enterprise data must be properly specified, in a manner understood by all participating parties. This description must include not only syntactic aspects, but also semantic descriptions that make the meaning explicit. Although it would be possible to annotate all data items, this would be equivalent to a central repository of data descriptions, which we have argued against due to its size and complexity.

• The business rules relevant to an instance of integration must be understood and complied with by all participants. This should be so despite the high number of business rules present in a large organisation and their probable close relationship with the running applications and, hence, their likely contradictions. Thus, business rules should be valid only for a particular context, and it should be possible to overwrite or supersede a rule from one context to another.

• All participants cooperating on an enterprise workflow must be able to follow its specification. Hence, the description of the workflow must be clear and unambiguous, including what each participant is supposed to do, what are the required inputs and outputs, and what are the constraints, such as integrity constraints on the execution of a workflow step. Assuming that an enterprise workflow is distributed, support for commit and abort protocols, and roll-back and roll-forward capabilities should be provided that do not rely on a centralised engine keeping all the threads of control.

To control the size and complexity of the integration strategy, it is imperative that at all times the principle of parsimony is applied. By reducing the number of items to be considered at one given time, centralising semantic descriptions, designing and implementing in stages, and basing systems interoperation in a low-coupling, scalable software architecture it should be possible to minimise the effort that the strategy entails.

The following chapter discusses related work individually focused on these three facets of enterprise integration.
Chapter 3

Background to Enterprise Integration

This chapter discusses research efforts closely related to the three aspects of enterprise knowledge considered by this dissertation. The chapter focuses on an analysis of ontology-based systems, a discussion of centralised and distributed workflow systems and their characteristics, and an analysis that results in a classification of business rules as data semantics or procedural.

3.1 Data Integration: Metadata and Data Interpretation

3.1.1 An Ontological Approach

With the aim of solving integration problems, very significant research effort has been directed to provide modeling constructs for the data and processes of a whole enterprise, to provide a common interpretation of the meaning of enterprise data. Hence, most proposals about information and knowledge integration, both centralised and distributed, concentrate on data integration. These assume that a comprehensive enterprise model may be developed, involving the creation and maintenance of some type of enterprise metadata, text-based, data model-based or ontology-based. These include traditional dictionary systems [4], and the efforts by the NISTs (National Institute of Standards and Technology) Information Resources Dictionary System (IRDS) [58]. These proposals are based on the provision of comprehensive descriptions of enterprise knowledge, with the intention of presenting an integrated environment for the organisation as a whole [70]. Standardisation efforts led to the development of reference models, such as CIM-OSA [39] and the Workflow Reference Model [69], recommended by some international bodies as frameworks to develop enterprise-wide information systems with a specific orientation.
However, to date the integration successes have been restricted to focused application environments such as Manufacturing and Supply Chain Management [46, 72, 128]. This is so because although these attempts deal with some of the issues — comprehensive documentation, unified language, shared conceptualisations — they do so at the price of imposing a complex, inflexible structure, feasible only for restricted environments. Furthermore:

- As the experience of Distributed Databases shows, the provision and maintenance of a global data and knowledge model is a major task, in fact overwhelming for a large organisation [71]. Consequently, the resulting integration strategy tends to be inflexible, and very costly to develop and maintain.

- Such an integration strategy tends to change the way the organisation operates, to be able to comply with the all-encompassing, integrated system. The so integrated organisation is required to change, with the consequent disruption to its processes.

- In a large IT-enabled enterprise, the meaning of organisational terms is often unclear. Different groups and applications often have different understanding of organisational terms and procedures. Because of this heterogeneity, it is not feasible to provide a data model that represents the totality of organisational knowledge.

- Even if a comprehensive enterprise model could eventually be produced, an initiative showing any degree of further heterogeneity would have to be accommodated into the model, making enterprise evolution very difficult.

- In addition, as it requires considerable local development involvement, this strategy may be very disruptive to roll out, making it likely to be resented by the different enterprise groups [93].

**Ontologies**

Ontologies were used originally in the Artificial Intelligence field to specify the context of shared knowledge. An ontology is an explicit specification of a conceptualisation, including a vocabulary of terms in a domain, and a specification of their relationships [121]. The objects and concepts of interest and their interrelationships form the basis for an abstracted simplified view of the world [62], and every knowledge base is committed, explicitly or implicitly, to some conceptualisation. Over the last few years significant research effort has been invested in using an ontological approach to achieve cooperation among heterogeneous data sources. For example, in heterogeneous databases ontologies have been used to assist in the specification of the concepts included in component database schemata [56, 77]. An ontology of the problem
domain, therefore, may be used to properly specify the meaning of the terms used in an organisation.

3.1.2 COIN: Ontologies, Contexts and Mediation

The Context Interchange (COIN) approach has been discussed by Goh et al [57] and Bressan et al [23] for achieving interoperability among heterogeneous data sources and receivers. They present a strategy for representing semantic knowledge, and demonstrate how it is possible to make inferences to facilitate the resolution of semantic conflicts by mediation. Their approach prescribes that all data sources and receivers explicitly describe their ontological contexts with respect to a collection of shared ontologies. The concept of *semantic domains* is central to their discussion: they represent the concepts of the world — the Universe of Discourse in database parlance — that they intent to represent. Examples of semantic domains are *MonetaryQuantity* and *MonetaryUnit*.

![Semantic Domains Diagram]

In their research, a *semantic value* is an instance of a semantic domain, as in *TotalSales* is an instance of *MonetaryQuantity*, and *US Dollars* is an instance of *MonetaryUnit*. Relationships between semantic domains can be established, and they can be labeled: *unit* may be the labeled link between *MonetaryQuantity* and *MonetaryUnit* in Figure 3.1. They also introduce the idea of modifiers, to allow for multiple interpretations of domain values depending on the
value of the modifier.

Semantic domains and their values are organised in a taxonomy, much like an object-oriented class hierarchy. To overcome the limitations of binary relationships, the notion of entity sets — *reified links* — is also allowed: for example, `CurrencyExchange` may represent a 3-way relationship between `Source`, `Target` and `ExchangeRate`. Entity sets also serve to provide aggregation of several semantic domains to construct a real-world object: an ontology can depict the name, location and type of business of a company, but without aggregating all these concepts as the notion of a company. An entity set may be used for the purpose.

For the purpose of our research, one of the most important constructs presented by Goh et al is the notion of *context*. A context is defined as a collection of formal statements that establish the meaning of entities within a given environment [57]. A context is related to a particular environment, and it makes assertions about elements in an ontology to define concepts unambiguously with respect to that environment. For example, the element `salary` in a shared ontology may have no decimal places in one environment and two decimal places in another, or perhaps be expressed in different currencies. Thus, assertions that are true for a particular environment may cease to be so for another.

Significantly, this introduces naturally the notion of context inheritance and overriding. A context assertion that holds for an outer environment can naturally be inherited by any inner environment, thereby enabling context inheritance. However, a context may disagree with some inherited conceptualisation, and decide to retract some of the inherited assertions and replace them with others suitable for the inner environment.

In most situations, specifying a context within an environment augments what is already established by the ontologies in outer contexts. A context may define new entities or provide modifier values of semantic domains, such as specifying the value of currency as US Dollars (USD). An inner environment, through overwriting, can then redefine the value of currency as Australian Dollars (AUD). In their work, ontologies and contexts take the same representational form of Horn clauses [54]. This allows for reasoning about the data semantics and their contexts, specifically to afford the knowledge required for context mediation, i.e., the mapping between contexts required for automatic detection and reconciliation of conflicts between different environments.

The presentation of the Context Interchange — introduced as the COIN framework — is based
on a deductive object-oriented data model and formal language [23]. This work focuses on the
detection and resolution of semantic conflicts by a Context Mediator, based on a declarative
specification of how data is to be interpreted in sources and receivers and how detected conflicts
should be resolved.

Importantly, the COIN framework supports knowledge-level queries, that is, queries involv-
ing the semantics of data. These are particularly relevant to solve heterogeneity problems,
which arise primarily out of incompatible assumptions on the interpretation of data from dif-
ferent sources. Answers to these queries are intensional rather than extensional. A traditional
database answer to a query is extensional, returning a collection of facts-sets satisfying certain
criteria. An intensional answer provides a characterisation of the answer set, without necessarily
supplying its constituting elements. This is very often more useful and more informative, since
it describes the answer rather than simply enumerating the elements and letting the receiver
provide a unifying characterisation.

3.1.3 GeM and WeBUSE

Another proposal close to this research is GeM (Generic Metadatabase) and WeBUSE (Web-
based Uniform Schema-browsing Environment), an ontology-based augmented data dictionary
to represent database information and assist users in understanding the semantics of remote
databases [110]. This approach does not intend to provide seamless access to disparate database
systems, but to aid users with the understanding of remote database schemata and the inter-
pretation of database query results.

The Generic Metadatabase Model (GeM) introduces the use of ontologies to represent database
schemata. Of the three ontologies for creating a generic metadatabase introduced by GeM, two
are relevant to this dissertation:

- An ontology for semantics of entities and associations: describes the possible types of seman-
tic information in entity definitions and inter-entity associations.

- Ontologies for problem domains: provide and exhaustive classification of terms used in the
  application domains

The GeM approach describes the context of a database attribute, as a sequence of value-pairs
associated with the attribute. The definition is recursive, so there is a clear requirement for
primitive elements, that is, elements that are clearly understood regardless of the context. El-
ements such as Centimetres, Day, US Dollars are assumed to have the same unambiguous
meaning in any context, and are used to anchor the recursive stack.

GeM concentrates primarily on identifying the kind of information that is missing from, or implicit in, existing database schemata, and focuses on how to provide a uniform representation of this information for the widest possible variety of modeling techniques. The GeM approach is that the representation ought to include information about the data model, since this has a very significant influence in the final description of a schema. A major difficulty with this approach is the existence of several data models, of different level of expressiveness, to specify a database schema. The proposed model is deemed able to store enough information so that the characteristics of the data elements can be determined. The intention is to make syntactic and semantic information explicit, to be able to access databases even when the terms of the Universe of Discourse (UoD) are not known.

The GeM approach relies on the implementation of a standard information resource dictionary — a metadata repository — that takes into consideration the existence of the different modeling techniques. An ontology is defined here as a formal description of the elements of a particular domain, i.e. what it is usually known as a domain ontology. If an ontology of the problem domain is established, it is possible to compare heterogeneous application data elements by linking them to the elements of the shared domain ontology, thereby establishing an agreement on the terminology of one application to that of another. It is indicated here that additional annotations can be used to further clarify meaning.

A distinguishing feature of the GeM approach is that it includes the structure of the data model within the ontological framework as well (this is the third ontology referred to before). That is, not only the problem domain is described in an ontological manner, but also the structure — relational, object-oriented, etc. — of the data model representing the information is described using the same approach. To this end, the semantics of a database application are divided into the semantics of the data elements themselves, and the semantics of the structure of the database. An ontology for the semantic information represented by database schemata is suggested, introducing a classification of schema elements as depicting abstraction or values. Semantics of value refers to the elements of the problem domain. The classification includes the representation of schema elements such as inheritance, association, aggregation, rules and operations.

After the contextual semantic information has been determined for each shared non-primitive element in the organisation, the information has to be stored so it can be easily retrieved to
facilitate its interpretation. In [110], a relational database (WebUSE) is introduced to store the context of individual data items. A set of standard queries to make possible the storage and retrieval of all the information pertaining to an item is also presented. The set of standard queries described is sufficient to manage the context of a data item that has been described by the ontology scheme.

Semantic modeling of a database schema is the process of specifying the structure of the data to produce a Conceptual Schema, i.e. a description of the structure of the entities of the universe of discourse and the relationships between them. Some of these structures are imposed by the particular data model employed, others by the modelers when they try to explicate the structure of the data. Although this is usually a very comprehensive process, the experience of researchers and practitioners involved in interoperation indicates that data modelers experience significant difficulties when specifying database schemata. Their requirements are far more extensive than the level of detail afforded by the standard modeling techniques.

Not all data models are able to represent all of the constructs above. Relational models, for example, do not directly support taxonomies or associations. The standard technique is to first use a semantically rich model, such as EE/R or NIAM [75], to represent the UoD, and then convert this representation to the semantically poorer relational model. Some loss of higher order semantic information such as taxonomy and affinity will be incurred in this process, when it is transformed into a lower order semantic form such as aggregation. The semantic information will need to be reconstructed when required. The schemata ontology is then linked to domain-specific ontologies depicting values and strong aggregations. However, use of this latter category is discouraged in the problem domain to avoid the perception incompatibility that different applications can have of the aggregation and its constituent elements. The GeM approach advocates that schemata should be broken down into their constituting elements, and extra information about these schema elements should be provided in the form of metadata. Since it is the most commonly supported by the different data models, the fundamental construct in GeM is the aggregation construct. This allows for the uniform representation of the semantic information by the use of element composition. This concept includes not only data elements, but also binary links such as inheritance and associations, rules and operations.

The use of such a flat model allows the storage of metadata information in a relational schema [111], and the reification of links makes possible the storage and retrieval of complex relationships between data elements.
3.1.4 Ontology Clustering for Information Integration

Tamma and Visser [114] present a structure of multiple shared ontologies with the intention of integrating heterogeneous sources. Their approach is to investigate an architecture that is closer to a human conceptual model, is easier to implement and has better prospects in terms of maintenance and scalability. Ontology-based resource clustering (ontology clustering) is an approach where resources do not commit to one comprehensive ontology, but they are grouped together by the similarities they show in the way they understand and explain the common domain. The authors attempts to overcome the most important limitations of other strategies, namely:

- The integration of heterogeneous resources without an intermediate ontology — the one-to-one approach — where for each ontology a set of mapping functions is provided a la OBSERVER system [91]. This approach is feasible only if there are only a few ontologies to map, since given N ontologies it may be necessary to have \( N(N - 1) \) mapping functions. In addition, if a new resource is added to the structure, additional \( 2N \) mapping functions may be required — providing a total of \( (N + 1)N \) functions — so scalability is a real concern.

- The use of a single shared ontology, for example InfoSleuth [17] or KRAFT [60, 61]. The problem with this approach is that it is necessary to define a standard ontology that is suitable for all potential resources. Defining such a standard is always lengthy and cumbersome, and compliance engenders resistance from users who are forced to sacrifice their autonomy to conform with the standard. In addition, a standard ontology is resistant to be easily changed, and thus it is hard to maintain. The provision of a shared global ontology is comparable with the provision of a Shared Global Data Dictionary in distributed databases. As we discussed previously in Section 2.3, the difficulty of this approach has been well documented in the research literature.

In this model, ontologies are then organised hierarchically in a tree-like structure, with the most generic concepts rooted at the top of the hierarchy. When some concepts are not shared by the agents (users) of the system, there is a reason to create a new ontology cluster and incorporate it in the hierarchy. Children clusters specialise inherited concepts, providing more precise definitions than their parents. Similarly to object-oriented hierarchies, this approach allows the specialisation of a more general concept that inherit all the attributes from its parent concept. This allows the passing of information down the hierarchy through inheritance of attributes.

\[ \text{Note that given N heterogeneous resources, the optimum is having to provide } 2N \text{ functions, if all map to a standard format. This is linear with the number of resources } N \text{ (i.e. } O(N)) \text{, rather than with the square } O(N^2). \]
and relations.

An interesting feature of this approach is the suggested navigation from one concept in the hierarchy to another in a different hierarchy. This involves two kinds of steps: up the hierarchy to a more generalised term (hypernym), and down the hierarchy through specialisation (hyponym). Naturally, given a concept in an ontology there is no assurance of the existence of a synonym in another, and therefore there is no guarantee that the mapping will preserve meaning. In that case, the origin concept is mapped to a more general, abstract one, with the resulting loss of accuracy.

### 3.1.5 Summary of Data Integration

The discussion of the previous section highlights the difficulties of the early attempts to enterprise data integration. Due to its size and complexity, we have argued against an all-encompassing metadata repository, and we are proposing instead that the strategy aims to reduce the number of items to be described to a minimum and to reuse descriptions by careful consideration of contexts, scope, inheritance and default values. In addition, the strategy should minimise the involvement of local applications in the data sharing mechanism and data mappings, to make it as simple as possible for sources and receivers to interoperate. The integration should consider important factors such as flexibility, to be able to accommodate different types of applications and technologies, and provide for scalability, extensibility and ease of transition to the integrated structure. Specifically, the strategy should aim to:

1. avoid a strategy that covers all the data in the enterprise at once, since that would be overly complex for a large enterprise;
2. document centrally data semantics to avoid misinterpretation, and provide support for local units to identify semantic and syntactic conflicts;
3. accept conflicts as a natural occurrence, and reconcile them at the local level;
4. provide a data-mapping service, so applications are not responsible for the data translations;
5. provide a data exchange infrastructure, so applications are not responsible for the sharing mechanism;
6. plan for flexibility, to be able to accommodate different types of applications and technologies;
7. plan for scalability by reducing strong dependencies between data sources and receivers;
8. plan for extensibility by supporting a standard interaction format;

9. facilitate transition to the new structure by making possible a gradual commitment to the strategy; and

10. minimise the amount of information required to be stored and managed by the integration by:

   • using semantic repositories associated with semantic local contexts;
   • containing local changes by restricting semantic scope;
   • providing economy of expression by using defaults as much as possible;
   • allowing semantic and context inheritance to enable reuse.

In order to achieve the objectives stated above, we advocate in Section 5.1 the use of a central facility related to a given context to store semantic descriptions of shared data. We propose the adoption of a corporate data format associated to each integration space, and a corresponding ontology to store its semantic descriptions. This enables sources and receivers to map local semantics to and from the corporate format. We advance that a combination of standard and enterprise-based ontologies might be used to describe the semantics of the entities involved in the integration.

While on the surface such an ontological approach may seem the same as the one provided by commercial solutions such as SAP’s Master Data Management (MDM), and indeed both share some common features and goals — data consolidation, maintaining consistent information, centralised data descriptions — it must be noted that ontologies are not simply data models. A data model describes the data in an organisation as a collection of entities and their attributes, and these descriptions may be used for a data integration strategy as in MDM. An ontology is a particular form of data model, in which each attribute is described by its semantic meaning rather than its meaning referring to a particular entity, and so this description has different implications. For example, although in a standard data model Salary and Total Sales may have different formats if they belong to different entities, in an ontological description these attributes would be both Money Quantities, and thus they would share this meaning within the context of concern.

The approach we espouse here is based on a federated architectural form. Within a federated framework, the division of an enterprise into a relatively small number of units of processing,
the provision of a corporate ontology-based data format, and the support of sharing by federated structures aims to achieve the data sharing objectives just described.

3.2 Workflows: Models, Architectures and Applications

3.2.1 Introduction

Over the past few years there has been a keen interest on workflows and workflow applications to support the increasing complexity of processes in modern organisations. Workflows are an attempt at closing the gap between organisational processes — people, document or computer based — of a corporation and the computer processing of its information systems infrastructure. Developments in technology have produced significantly cheaper and more powerful computers. Hence, the notion of integrating all the process information resources of an organisation into an execution environment where they can be controlled and monitored has become a possibility. The concept of workflow embodies the automation or semi-automation of procedures where the required information is passed between participants to achieve, wholly or partially, a common business goal. Workflows are mostly concerned with the definition, modeling, management and control of the component activities of a process. For business processes, the objective is to systematise them by managing the sequencing, execution and completion of its constitutive tasks, calling resources as they are required and ensuring the correct completion of the various activity steps.

Substantial effort has been devoted to the development of workflow systems, both in academic and industrial environments. Due to the variety and complexity of existing organisational processes, such systems can be conceived and implemented in many different ways, using different techniques within a variety of operational environments. The efforts have concentrated along several directions, all of them addressing the three functional areas presented by the Workflow Management Coalition (WFMC): build-time functions, run-time control, and run-time interactions. Among others, research and implementation efforts include workflow systems architectures, the provision of models for workflow systems and workflow repositories, the extension of transactional behaviour to workflow systems, provision of run-time support for workflows, distribution of tasks and information among participants and workflow evolution. These include:

- The Workflow Management Coalition endeavour to standardise workflows by specifying terms such as event, process, pre and post conditions, state transitions and roles [69]. This is also the way followed by much of the current workflow research, which concentrates on the specification of execution semantics such as ordering, parallelism, alternative, iteration,
3.2: Workflows: Models, Architectures and Applications

conditional, and nesting [82, 83].

- The introduction of a higher cognitive order of workflow modeling at the conceptual level, which includes characteristics such as temporal constraints [14, 15].

- The provision of architectures for workflow systems, both centralised based on a server providing most of the functionality, or distributed across several servers [5, 74].

- The extension of advanced database transaction models to Workflow Management Systems. In particular these include support for long lived transactions, atomic operations, forward recovery and backward recovery, and distributed workflow commitment protocols [5].

- The provision of repositories and repository management tools for workflows, covering issues such as team support, version tracking, change propagation, and task reuse [89].

3.2.2 Transactional Workflows

The concept of database transaction originated to relieve application programmers of the burden of dealing with issues such as concurrency control, failure detection and recovery and data consistency. In a traditional database environment, the transaction management system provides the well known ACID (Atomicity, Consistency, Durability, Isolation) properties [99]:

**Atomicity:** A transaction should either commit or not in its entirety, in this way behaving to an outsider — user, process — as an atomic operation. If any intermediate operation fails the transaction is rolled back.

**Consistency:** Ensures that any changes to data by a transaction are consistent with changes to the same data by other transactions, taking the database from one consistent state to another.

**Isolation:** Each transaction should execute as if it was the only transaction running in the system, so operations of one transaction do not interfere with operations of another.

**Durability:** The changes introduced by a committed transaction are permanent in the database.

These properties are considered a safe way to ensure that operations correctly update the database in the context of concurrent executions. Various transaction models have been proposed to extend these traditional concepts to more advanced applications, including workflows [37].
With the growing interest in workflows, there is also a growing interest in systems that support operations in distributed heterogeneous environments, much like those addressed by advanced transaction models. These systems intend to provide a real-life working environment, where human actors and computer applications assist in each others work, with support for flexible operations and long-lived transactions. However, it has been pointed out that although the goals of advanced transaction models are very similar to those of Workflow Management Systems, these deal with a much richer set of requirements. Although transactions are standard in the database area, Alonso et al argue that workflow models have in general richer semantics that advanced transaction models, and that they are more suitable for commercial applications because of their greater expressibility [5].

An enterprise business process can be defined as a set of scheduled activities that cooperate to achieve a common goal [5, 12, 69, 82]. This concept of a business process is central to the notion of workflows, since workflow constructs may be used to link and manage the flow of data and control information among the activities comprising a business process. Some of these require human intervention, some are completely executed by computer applications, some others require the cooperation of both humans and computer systems. Human tasks are not naturally transactional, and even computerised tasks need not be necessarily transactional or transaction-based. A successful workflow implementation has to be flexible, and has to be able to deal with the long-lived activities that a businesses process typically comprises. This does not translate well from the short-lived, tightly controlled database transaction models, where operations are under the jurisdiction of the database management system, and elements can be locked or time-stamped to effect concurrency control.

In spite of these limitations, it is highly desirable to be able to extend transactional behaviour to workflow systems, since this results in the preservation of at least some advantageous characteristics of transactional environments. Even if some of the ACID properties may have to be ultimately relaxed, the basic notion of using transactions — long-lived transactions — as the starting point results in business processes with better defined concurrency, failure and recovery semantics. The introduction of formal dependencies, recovery and compensation would make possible the running of more complex business processes in distributed heterogeneous environments. Even though the main goal of workflow systems is not the same as those of databases, endowing workflows with transactional capabilities enhances them to be able to handle more complex scenarios in a safer manner. The goal should be to attempt to apply ideas and solutions derived from transactional environments to workflows, rather than translating database concepts to the workflow arena. As it has been pointed out, workflow systems provide extensive
support for aspects not in the realm of databases, and these features form the basis of their success [5].

3.2.3 A General Model for Transactional Workflows

One of the main transactional aspects of interest for workflow systems is the commit protocol. A database system does not finally write any changes until there is certainty that the updates can be committed; all database operations must be correctly terminated by reaching one of two states: commit or abort. Kuo et al [82], introduce a general model for nested transactional workflows (TWF), including the notion of correctness for a workflow encompassing traditional transactions, operations, human activities, or even another TWF, therefore enabling the nesting of workflows. A TWF system does not deal with the semantics of individual tasks, but manages the execution of the various tasks comprising a workflow.

In their definition of transactional workflows, Kuo et al relax the notion of failure atomicity by classifying tasks into critical and non-critical and requiring different termination semantics for both types [82]. Non-critical tasks are those for which there is no need to abort the workflow if the task fails. An unsuccessful critical task, on the contrary, results in the workflow aborting and needing backward recovery. This implies rolling back or compensating tasks that require undoing. Hence, the authors introduce two major properties of tasks: compensatability and forcibility.

**Forcibility:** A task is **forcible** if the underlying system can guarantee that the task will eventually succeed, even in the presence of failures. For example, although the sale of 100 books can fail if there is not enough stock available (non-forcible), it may be assumed that if the stock is available once the sale has taken place the issue of an invoice is a forcible task (in the absence of fatal errors).

**Compensatability:** A task is termed **compensatable** if it can be undone after it has committed. This does not necessarily mean that the TWF is in charge of defining the compensation of a compensatable task, but it may be the responsibility of the programmer defining the task to specify a compensation for it. This is the approach taken by Rusinkiewicz et al [104], where each task must have a compensation specified as part of the definition of the task. In contrast, rather than having the task compensation defined as part of the task description, it is also possible to accept that the compensation may be actually dependent on which TWF executed the task. Therefore, Kuo et al [82, 83] allow for the task compensation to be specified within the TWF itself.
All tasks in the Kuo et al. model have an initial state (before it is invoked), a set of commit states and a set of abort states. (This is to be able to properly model contingencies and alternatives, which can result in more than one commit state.) If a task has successfully executed, it transitions to one of the commit states; if a task fails, it transitions to one of the abort states.

In addition, the authors allow the construction of TWFs with a collection of constructs combining individual tasks: ordering, contingency, alternative, conditional and iteration.

**Ordering:** The specification of the order in which the tasks may be executed. It can be represented as an acyclic directed graph, with tasks as the nodes and edges indicating an ordering constraint.

**Contingency:** It allows the specification of a contingency task for a given (main) task. The TWF will execute the contingency task if the main task aborts. This implements XOR semantics; that is, one of the tasks is executed, but not both. For example: “If the book is not available in Australia, it should be sourced from New Zealand”.

**Alternative:** An alternative is composed of any number of tasks. The TWF is free to choose any of the alternatives to execute, probably depending on the properties of the task. Unlike contingency, there is no requirement for a task to abort before an alternative can be chosen. This implements OR semantics; that is, at least one of the tasks is executed, possibly more than one. For example: “To fulfill an order, books may be sourced from either country Australia or New Zealand”.

**Conditional:** They are composed by a condition and two tasks. During the execution of the TWF, if the condition evaluates to true one task is executed; if false, the other task is executed. For example: “If the customer is a preferred customer, an order with special discounts will apply. Otherwise, a normal order should be raised”.

**Iteration:** An iteration contains a condition and a subtask encapsulated as a single task; while the condition evaluates to true, the subtask is executed. For example: “An order can be partially fulfilled until complete”.

An important feature of the proposed model is the commit protocols for the different tasks and constructs, since this enables the relaxation of failure atomicity. To this end, a null task is defined as a task with no effect that always commits. Thus, a non-critical task is represented as a contingency with a null task as the contingency task. Since the null task always commits, aborting a non-critical task results in the execution of its contingency (null) task, which commits, therefore committing the contingency as a whole. A task that does not require undo can be
modeled as a task with null compensation. Since the various types of tasks have been correctly modeled with the above constructs, it is now possible to define that a TWF commits if all the tasks are in one of the commit states and it aborts if at least one of the tasks is in the abort state.

Nesting of TWFs is introduced by the mapping of a TWF to a task, by representing it as a finite state machine. In order to better manage the, possibly large, number of states in the newly created task, the authors describe and provide criteria to decide which states are useful (visible), and discarding from the model the non-visible states. In addition, a relation of equivalence between states is introduced, thereby reducing the number of states by using a single state to represent a set of states.

### 3.2.4 Exotica/FMQM

Workflow Management Systems are used to automate and coordinate the various tasks comprising a business process. Most systems rely on a client/server implementation, where the clients mainly manage the interaction with the users, while most of the functionality is provided by the server. Although this is very convenient from the overall system simplicity point of view, it is not clear how this approach will scale up to very considerable number of business processes in the case of large organisations. Besides, a centralised server is always a bottleneck and a single point of failure. Thus, an architecture based on a centralised server is unlikely to meet the requirements of an organisation relying on such a workflow system to support its critical business processes.

Alonso et al argue that it is possible to eliminate the need for a centralised database and server by proposing a workflow system based on a set of autonomous nodes [5]. The nodes function independently, their interaction reduced to an exchange of persistent messages that coordinate the steps of the different tasks. The nodes are responsible for completing their tasks, and communicate to other nodes their termination and the necessary data and termination conditions so the flow can continue. This type of mechanism hides the complexity of the tasks contained in the individual nodes, and the heterogeneity of the different platforms and the underlying protocols. This model resembles the model of a distributed computation, where a process is executed in a truly distributed manner with each node holding only the information it needs to perform its job as part of the computation. In this case each node executes as part of the overall workflow job, and synchronises by using a persistent message mechanism and a minimal amount of persistent storage at each node. A process is therefore executed with the cooperation of the nodes involved. A node task aborting leads only to the failure of the activities
executed at the node, but this does not prevent other nodes to execute different parts of the same or some other workflow. The communication mechanism suggested is an asynchronous and connectionless messaging service, so the temporary unavailability of a node will not cause the overall workflow to abort.

Exotica has been implemented on top of an existing workflow system, and a persistent queue-based messaging middleware. FlowMark follows closely the reference model of the Workflow Management Coalition, and thus the natural mapping of FlowMark features may be naturally mapped to the concepts introduced by the model. FlowMark acts as the overall controller of the workflow navigation but, rather that relying on a central back-end database server, several build-time clients, run-time clients and FlowMark servers are allowed to execute concurrently. The creation and definition of a process is done in a build-time client — the process definition node — an independent module executing in any node that can provide an appropriate graphical user interface and can check for inconsistencies and errors. The run time nodes are nodes at which process instances are executed.

Once a process has been defined, the process information is compiled, the information corresponding to each node is established, and the activities bound to the node where they will be executed. At each node a node manager communicates with the process definition node to create and maintain a per-process table with the static information concerning the activities executing at that node. The node manager can then start the process thread, a thread that creates and periodically inspects a message queue where messages from other nodes are received. When an activity is ready, an activity thread is triggered to manage the execution of that activity.

Several data structures and commands such as GET and PUT, assumed to be part of the middleware API, are involved in the execution of an individual process instance. A combination of activity states — inactive, executing, executed, terminated — input and output data, control connectors constrained by conditions and filters to extract relevant data are used to manage the different activities. A terminal activity signals the termination of a process to ensure proper completion.

3.2.5 Information Carriers: INCAs

To implement truly distributed workflows, Barbara et al [11] introduced the notion of information carriers (INCAs). In their model workflows execute under the control of various au-
tonomous processing stations, which take turns to process the different steps of the workflow. The INCA model is grounded on several principles that underpin it:

- Processing stations are autonomous, they appear as opaque to the other workstations and to the workflow manager. Other workflow actors are free to ignore what happens inside one of these stations and, consequently, no single person or entity has complete knowledge of the workflow. The execution of the workflow is the result of the interaction between processing stations.

- Dynamic flow of control: stations are allowed to alter the control and data flow of the workflow, in an unpredictable manner. Changes on the step sequence are actually decided upon and managed by the individual stations, even if these result an alteration affecting other stations. Therefore, it is not possible to establish a priori the flow of execution, and in this sense workflows are not deterministic.

- Partial automation: several workflow models attempt to incorporate manual or semi-automated step into the process sequence. In INCA there is no assumption of fully automated processes within a station, or that a station has full communications support. File transfers, email, faxes, etc., are all acceptable means of communication between stations in INCA.

- The underlying INCA computation model does not assume that stations are connected all the time. In other models it is assumed that a network connection is permanently established, a disconnection a temporary occurrence. However, INCA accepts stations that are disconnected most of the time, and may connect to the network only when necessary, if at all. Thus, in the INCA model network partitioning is the norm rather than the exception, and it is even possible that a station may be connected only briefly to the network if the communication is enabled by some other means (e.g. a smart card carrying the required information).

In INCA, workflow computations are carried out by submitting information carriers to the processing stations. These INCAs contain the required information to execute the service, rules to govern the flow of data and control, and the failure atomicity requirements for the computation. Upon receiving an INCA, a processing station executes the required service according to the INCA data and rules, as well as its own data and rules. Since during the running of the service the station can modify the INCA data and rules, and therefore alter the next INCA destinations, the station itself must determine the next set of destinations for the INCA. When the execution is finished, the station forwards the INCA — including the flow of control, data and failure atomicity requirements — to the appropriate destinations. Due to the fact that processing stations can modify the internals of the INCA, the path of the computation and the steps
to be performed at each station cannot be known in advance. Instead of a centralised thread of control, the INCA model require the processing stations to understand the INCA formats and participate in the protocol of receiving and INCA, performing the required operations and forwarding the INCA.

In the model, an INCA is an object representing an activity or a complete workflow, which includes private data and a set of rules. The private data that an INCA includes defines the global context for the computation, that is, the data shared by the different steps of a given activity. The set of rules defines the control and flow of data between the steps of the activity. These belong to the INCA, and they are carried along with it as the various steps are performed by the processing stations. In this way, these INCA data describe the global context of the workflow.

In turn, each station also possesses a set of local data, a set of local rules and the sequence of local procedures to be performed while the INCA is located there. If during the execution of this sequence it is deemed necessary to change some of the private data or rules associated with the INCA, the processing station does not overwrite the INCA but creates a new version of the INCA to be able to perform a partial roll-back and restart its computation to a given time in the execution. Processing stations may perform the required procedures in an automated, semi-automated or completely manual way.

Nesting of workflows is naturally supported by the model by allowing a parent INCA to spawn a child INCA computation and wait for it to execute before regaining the thread of control. The child INCA returns to the originating site, and it is used to make modifications to the parent INCA which resumes execution according to its own data and rules.

The combination of the INCAs own rules and the local rules may permit multiple steps of a computation to be performed concurrently, thereby enabling the parallel execution of workflow steps. Since the steps of an INCA computation may modify the original data, the model imposes the restriction that the concurrent copies of the INCA have to return to the originating station after they finish their respective tasks. The data must be then merged somehow, using a merging policy that must also be specified within the originating INCA.
ACID Properties of an INCA Computation

To ensure the atomicity of an INCA computation in the presence of process, station or communication failures, the model prescribes that it should be possible to test whether a local procedure terminated successfully, and that it is possible to reverse the effects of a local procedure by executing a compensating step (testability and reversibility). In the event of a failed execution, the exact semantics for forward recovery depend on the type of failure, when exactly the failure occurred, and the status of the procedure and its output parameters at the time of the failure. In general terms, the system either behaves as if the INCA was never received or the local procedure has been aborted, or as if the procedure has committed and the actions should be undone. Even in the case of communication failures (e.g. a lost INCA), it is always possible to reconstruct the INCA since each processing station maintains a persistent copy of the changes made to the it during execution of the workflow. In the INCA model there is no difference between normal and compensating steps, i.e. the execution of a compensating step is requested by sending the appropriate INCA to the processing station. This in particular implies that the protocol to undo an activity has its own sequence and, unlike the sagas model, the compensating tasks are not necessarily executed in the reverse order, and they can even be executed with some degree of parallelism.

In the process of executing an INCA computation, it is possible that parallel task executions modify common data at one of the processing stations or modify replicated data over the processing stations involved in the concurrent execution of the workflow. Therefore, consistency of INCA computation presents two aspects: consistency of the INCA data, and consistency of the local processing stations data. If the consistency constraints are restricted to a single processing station, the same isolation semantics as in the saga model can be used here to ensure that there is no loss of consistency of the processing stations data.

In the case of data constraints spanning multiple stations it may be necessary to group local operations of an INCA computation into an atomic isolated unit that spans those boundaries. A few possible solutions — imposing conditions over reads and writes to make the schedule equivalent to a serial one, allowing inconsistent executions that are ultimately merged, partitioning the data — are suggested, but a definitive approach is not taken.

As for durability, since the model requires that the effects of all successful partial steps are durable, the durability of the INCA computation is automatically guaranteed.
3.2.6 Web Services Based Workflows

The IBM Software Group has introduced a model for the implementation of Web Services-based workflows [87]. The model includes descriptions of how to achieve a business goal by a composition of Web Services, and the specification of interaction patterns between collections of Web Services. The research presents the Web Services Flow Language (WSFL), an XML language appropriate for the description of Web Services compositions. WSFL is layered on top of the Web Services Description Language (WSDL), using WSDL to describe the service points where the business services can be executed, and the description of their interfaces and their protocol bindings. The principle behind the specification is to implement business processes as a composition of a set of Web Services. In [87], a WSFL flow model is presented as taking into account the business processes, the business rules (for sequencing of steps) and the flow of information between the steps.

A business process is defined by a set of activities, each describing a processing step, and data and control links representing the sequencing rules and flows of information between the steps. In keeping with the Web Services philosophy, late binding is allowed between a service request and the operations offered by a service provider. That is, the model allows for the discovery and late association between activities in the model and services offered by providers. To be able to satisfy these requests, WSFL defines service providers locators to specify criteria for selection of a particular provider and to associate the functions on offer in the provider with the requests of the flow interface. Despite this flexibility, these requests are not sent to any provider, but the mapping between an operation and a particular function of a provider is specified by a plug link (<plugLink> element).

In order to sequence the activities in the flow, two types of connections are defined, a control connection — control link — and a data connection — data link element. The first type indicates flow of the end of an activity to the start of another. The second indicates a data flow from the preceding activity to its successors.

The WSFL Workflow Model

As with other workflow systems, this model is described by an acyclic directed graph, where the nodes are the activities to be performed as a single step contributing to an overall business goal. Each activity may have an input message and an output message (and multiple error messages). Activities are linked via control links, directed edges that prescribe the order in
which the activities are to be performed by associating a given activity with all the possible
downstream activities. These flows are controlled by Boolean expressions — i.e. true/false tran-
sition conditions — associated to each link. At the completion of an activity, the conditions
of all the conditions emanating from the activity are evaluated, and the flow follows the links
of the ones that evaluate to true to the actual successors. At most one control link is allowed
between two activities.

The WSFL model supports the following constructs:

- Forks: an activity with more than one outgoing link is called a fork activity. Given a fork
  activity, which of the follow-on activities will be executed is determined by the truth values
  of the outgoing links. As a consequence, parallelism might be achieved by a fork activity
  with more than one outgoing true-valued link, since all triggered sub-graphs will be executed
  in parallel.

- Due to its potential for ambiguity, looping is supported only in a controlled manner, in which
  an activity might be iterated until its terminating condition is met. Therefore, cycles must
  be implemented by separate ‘do …until’ flows based on exit conditions.

- Joins: an activity is a join activity if it has more than one incoming link. This construct
  might be used to encapsulate AND and OR behaviour as follows:
    - AND: the activity is executed only when all the incoming links evaluate to true.
    - OR: the activity is executed when at least one link evaluates to true. In this case, there
      might be a different synchronisation behaviours, one stronger (the flow waits until all
      the links have been evaluated), one weaker (the activity is executed as soon as one link
      evaluates to true).

- Start and end activities: activities with no incoming (outgoing) link are called start (end)
  activities. When a flow is instantiated, all its start activities are executed in parallel, and
  at the completion of each start activity the downflow activities are triggered according to
  the truth value of its outgoing connectors. Normal flow proceeds through the graph, until
  it reaches an end activity. When an end activity completes, the flow sub-graph stops since
  there is no successor. When all end activities in the graph are completed, the output of the
  flow is determined and the workflow finishes.

Aside from control links, a second type of link — a data link — is allowed in the model. These
specify that a source activity delivers some data to the flow engine, which in turn passes some
of the data to the target activities. To avoid situations such as trying to consume data that
has not been produced yet, a data link can only be specified if the target can be reached from
the source through control links. Although it is not required that data is always passed from
a source to its successors, an activity may have many incoming data links. This permits ag-
ggregation of data, determined by map specifications. The model allows for multiple mapping
specifications for the same map target, since there might be alternative paths and data needed
downstream might be produced along any of these paths.

Given that workflows might be encapsulated recursively, flows are allowed to have inputs and
outputs. A flow source and a flow sink elements are provided to represent the input message
and the output message of a flow respectively.

3.2.7 Workflow Patterns

Van Der Aalst et al [117] study a large collection of workflow requirements, and describe them
in the form of workflow patterns. This work is not concerned with the description of a workflow
system or model in particular, but instead concentrates on establishing a set of desirable generic
workflow constructs — patterns — and discussing to what extent workflow models and systems
used in industry support them. Their investigation addresses a comprehensive set of workflow
scenarios, thus providing a framework for a comparison of workflow models and products, based
on their modeling and routing capabilities. In particular, the authors are interested in the ex-
pressive power of the models, and their suitability for the implementation of enterprise flows.

The patterns discussed in this work are design patterns, that is, the discussion centres on ac-
tivity coordination, ranging from the most common standard constructs, to more sophisticated
situations not usually supported by workflow products. They establish that since workflows
are created to enact business processes, workflow systems should specify, execute and monitor
these processes and, thus, they assume the existence of a workflow engine. As in other models,
an activity is an atomic unit of work, and a workflow process definition is to be specified to
determine what activities must be executed in what order. They consider the data layer as
placed on top of the activity layer, which includes control and production data.

Van Der Aalst et al do not present a new modeling language, but instead they determine the se-
manetics of various business cases — always within a well-specified context — from the workflow
point of view. The discussion includes the notions of workflow instances, transition conditions,
parallel and alternative executions, conditional routing, execution states and thread synchro-
nisation. The concluding section summarises the capabilities of the most important industry
workflow products with respect to their support for the patterns. The results indicate that all
the products support the basic patterns indicated by the Workflow Coalition, but that support
for more advanced patterns is sparse.

Although there is no specific mention, it may be seen from the discussion in [117] that all the
products include a workflow engine that controls the execution of the workflow. Hence, in this
sense all workflows are centralised. In addition, even though the models may include cancella-
tion or disabling of activities, there is no provision for transactional behaviour.

### 3.2.8 Summary: Workflow Models for Large Enterprises

The workflow models presented in the previous section all present the flow characteristics es-
tablished by the Workflow Coalition [69]. Table 3.1 summarises the characteristics and the
structures supported by the models. Since the work of van Der Aalst et al covers several prod-
ucts, a ‘*’ has been used when it is not possible to give the precise truth value of the variable
for all of them.

<table>
<thead>
<tr>
<th>WF Attributes</th>
<th>INCA</th>
<th>WSFL</th>
<th>Kuo et al</th>
<th>Exotica</th>
<th>Van Der Aalst et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Node autonomy</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Process oriented</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Processing only within nodes</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Message-based</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>Asynchronous processing</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Support start-start semantics</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Cycles/Loops</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>Joins</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Conditionals</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alternatives</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Transactional behaviour</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Logic associated to links</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>*</td>
</tr>
<tr>
<td>Dynamic connections</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
</tr>
<tr>
<td>Support for embedded WF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

All the models present desirable characteristics to support workflow-based processes. However,
they all have shortcomings when supporting the low coupling enterprise processes that are the
focus of this research.
• Most workflow models are centralised. This may be appropriate for small, process oriented workflows where all the threads of control may be managed by the central system. However, the size of a large enterprise and the heterogeneity of its processes would make it very hard for this approach to scale up to very considerable number of business processes in the case of a large organisation. Besides, a centralised server is always a bottleneck and a single point of failure.

• Some models — such as WSFL and Kuo’s — associate logic with workflow links. Although this might be a desirable characteristic in some situations, it is not appropriate for a large enterprise in which the separation of concerns between the processing nodes and the workflow is so important for scalability and reusability. The seepage of processing logic between the workflow and the processing nodes space is undesirable because it greatly reduces flexibility. Firstly, if the flow of information between nodes is governed by a workflow application, this has to be modified every time the flow must be even slightly altered. Secondly, if a node participates in more than one workflow there is an associated link logic and message with each workflow and therefore no possibility of message reuse. Thirdly, as we shall see in Chapter 5, this reduces the possibility of dynamically extending a workflow by adding new nodes.

• In order to loosen the tight links of traditional models, INCA and Exotica present genuine distributed processing modes. However, their approach results in workflow models that would not be acceptable in an enterprise. In INCA for example, stations are allowed to alter the control and data flow of the workflow in an unpredictable manner, making impossible to establish a priori the flow of execution. In other words, INCA workflows are non-deterministic. In addition, the INCAs carry their own rules to the processing stations, and they must be automatically reconciled with the local rules, even in the presence of ambiguities and contradictions. Besides, the INCA model accepts processing stations that might be disconnected as the norm rather than the exception, and the communication enabled by some other means. Very diverse means of communication are acceptable for INCA — email, fax, file transfer — and this provides great flexibility. However, since enterprise business depends on the timely and correct execution and guaranteed completion of the processes embodied by the workflows, this approach would not be acceptable by the organisation.

In contrast, Exotica truly establishes a workflow and distributes processing, but once the workflow is compiled it may not be altered without redefinition. Thus, messages are set and cannot be reused. Again, this rules out the possibility of dynamically extending a workflow by adding new nodes. In addition, it is not clear what transactional characteristics are
supported by the model.

- Kuo et al propose a model with strong transactional characteristics, in which the commit conditions are relaxed to accept a more flexible model. However, they still advocate a centralised workflow model, with the negative connotations discussed above.

Instead, we will present here in Chapter 5 a workflow model appropriate for a large enterprise based on autonomous nodes that act as processing stations. The model does not attach any logic to the links, but instead to progress a workflow a node publishes the required information for downstream nodes to consume. Similarly to the model by Kuo et al, the workflow model possesses transactional characteristics.

### 3.3 Business Rules

Business rules intend to control or shape the way a business operates. That is, “a business rule is a statement that defines or constrains some aspect of the business” [63] (pp 4). They do so by stating the structure of business aspects (known facts, or relationships between facts), constraints on data or processes, and inferences from known facts to derive new ones. With reference to information systems, “a business rule pertains to the facts that are recorded as data and constraints on changes to the values of those facts” [63] (pp. 5). The rules contemplated in this research do not refer to entities or policies such as Equal Opportunity or Smoking policies, which lie outside the information systems of the organisation that are the focus of this dissertation. Rather, in our context a business rule expresses constraints on the creation, updating, and removal of data in any of the information systems of the enterprise. We share here the position of the Business Rules Group, which allocate their study to row three of Zachman’s classification depicted in Figure 3.2.

In other words, the rules of concern for this dissertation are the ones referring to the way enterprise information is managed, and that are in some way reflected on enterprise systems. These are the business rules of concern for this dissertation.

### 3.3.1 A Classification of Business Rules

Following the Business Rules Group [63], in a general setting rules can be classified as follows:

\[\text{For a complete description of Zachman’s framework, please refer to [129]}\]
### ENTERPRISE ARCHITECTURE - A FRAMEWORK

<table>
<thead>
<tr>
<th>SCOPE (CONTEXTUAL)</th>
<th>DATA</th>
<th>FUNCTION</th>
<th>NETWORK</th>
<th>PEOPLE</th>
<th>TIME</th>
<th>MOTIVATION</th>
<th>WHY</th>
</tr>
</thead>
<tbody>
<tr>
<td>List of Things Important to the Business</td>
<td>List of Processes the Business Performs</td>
<td>List of Locations in which the Business Operates</td>
<td>List of Organizations Important to the Business</td>
<td>List of People Significant to the Business</td>
<td>List of Business Goals Strat</td>
<td>End-Mean = Major Bus. Goal Critical/Success Factor</td>
<td></td>
</tr>
<tr>
<td>Planner</td>
<td>ENTITY = Class of Business Thing</td>
<td>Function = Class of Business Process</td>
<td>Node = Major Business Location</td>
<td>People = Major Organizations</td>
<td>Time = Major Business Event</td>
<td>End-Mean = Major Bus. Goal Critical/Success Factor</td>
<td></td>
</tr>
<tr>
<td>ENTERPRISE MODEL (CONCEPTUAL)</td>
<td>e.g. Semantic Model</td>
<td>e.g. Business Process Model</td>
<td>e.g. Logistics Network</td>
<td>e.g. Work Flow Model</td>
<td>e.g. Master Schedule</td>
<td>e.g. Business Plan</td>
<td></td>
</tr>
<tr>
<td>SYSTEM MODEL (LOGICAL)</td>
<td>e.g. Logical Data Model</td>
<td>e.g. &quot;Application Architecture&quot;</td>
<td>e.g. &quot;Distributed System Architecture&quot;</td>
<td>e.g. Human Interface Architecture</td>
<td>e.g. Processing Structure</td>
<td>e.g. Business Rule Model</td>
<td></td>
</tr>
<tr>
<td>Designer</td>
<td>Entity = Data Entity</td>
<td>Role = Data Relationship</td>
<td>Proc. = Application Function</td>
<td>Node = IS Function</td>
<td>Link = Data Characteristics</td>
<td>People = Role</td>
<td></td>
</tr>
<tr>
<td>TECHNOLOGY MODEL (PHYSICAL)</td>
<td>e.g. Physical Data Model</td>
<td>e.g. &quot;System Design&quot;</td>
<td>e.g. &quot;System Architecture&quot;</td>
<td>e.g. Presentation Architecture</td>
<td>e.g. Control Structure</td>
<td>e.g. Role Design</td>
<td></td>
</tr>
<tr>
<td>Builder</td>
<td>Entity = Segment/Table,etc</td>
<td>Role = Interface,sys Para</td>
<td>Proc. = Computer Function</td>
<td>Node = Hardware/System Software</td>
<td>Link = Data Specifications</td>
<td>People = User</td>
<td></td>
</tr>
<tr>
<td>DETAILED REPRESENTATIONS (OUTOF-CONTEXT)</td>
<td>e.g. Data Definition</td>
<td>e.g. &quot;Program&quot;</td>
<td>e.g. &quot;Network Architecture&quot;</td>
<td>e.g. Security Architecture</td>
<td>e.g. Timing Definition</td>
<td>e.g. Rule Specifications</td>
<td></td>
</tr>
<tr>
<td>Sub-Contractor</td>
<td>Entity = Field</td>
<td>Role = Address</td>
<td>Proc. = Language Syntax</td>
<td>Node = Addresses</td>
<td>Link = Protocol</td>
<td>People = Identity</td>
<td></td>
</tr>
<tr>
<td>FUNCTIONING ENTERPRISE</td>
<td>e.g. DATA</td>
<td>e.g. FUNCTION</td>
<td>e.g. NETWORK</td>
<td>e.g. ORGANIZATION</td>
<td>e.g. SCHEDULE</td>
<td>e.g. STRATEGY</td>
<td></td>
</tr>
</tbody>
</table>

---

Zachman Institute for Framework Advancement

Copyright - John A. Zachman, Zachman International
• Structural assertions: define concepts, terms, or state facts related to aspects of organisational structure. A term refers to a specific business concept. By assembling terms in different ways, facts capture relationships between concepts. “A manager is an employee of the company”, “A book belongs to one and only one of the categories Technical, Medical or General”, “Name is an attribute of Customer” are examples of structural assertions.

• Action assertions: constraint statements, to limit, restrict, confine and in general control actions of the enterprise. These express dynamic aspects, conveying authorisations, integrity statements or conditions, such as “A customer cannot raise an order unless a credit limit has been allocated”.

• Derivation assertions: statements of knowledge derived from existing knowledge within the enterprise. For example “The discount for a customer is calculated from the customer type, the total of the order and the method of payment”.

There is a significant conceptual difference between the first type, structural assertions, and the other two:

• Structural assertions describe a static aspect of the business, by stating known things or how known things relate to each other. In this sense, structural assertions relate to the structure of the terms used in the enterprise, defining terms and placing them in a hierarchy, or stating relationships between concepts, including properties of a concept, a concept falling within the scope of another, and participation facts such as association, aggregation and roles played. These are the rules typically captured by data models expressed in formalisms such as UML (Unified Modeling Language) or Entity/Relationship Diagrams.

• Action and derivation assertions, in contrast, refer to dynamic aspects of the business:
  – An action assertion is expressed in a declarative form, imposing conditions or constraints on the results that actions can produce. “An invoice must be dispatched together with the goods”, or “For an order of a new customer to be valid, the customer must open a credit line first” are examples of action assertions.
  – A derivation is either a mathematical calculation or a logical inference. “The Interest Rate of a Preferred Customer will be 0.5 less than the normal standard rate for a Normal Customer”, or “The status of a customer as a Preferred Customer is due to the customers Total Yearly Business exceeding 1,000,000, or the customers profession being one of the Preferred Professions”.

Although the concepts are clearly established in the research literature, the definition of atomic and composite business rules differ slightly depending on the author. Sandy defines an Atomic
Rule as a “declarative statement of an organisational rule containing one unit of logical thought”, and a Compound Rule as a “declarative ‘statement’ of two or more atomic rules” [105] (pp. 72). In contrast, the Business Rules Group defines a Business Rule Statement as a “declarative statement or constraint which the business places upon itself or has placed upon it” [63] (pp. 11). For the Business Rules Group, each Business Rule Statement may be related to other Business Rule Statements, thus making for composite statements. That is, for BRG a Business Rule is a Business Rule Statement that cannot be broken down further.

Regardless of the nomenclature, the literature shows that there are two types of rules: complex rules, relating several simpler business rules, and simple (atomic) rules, which cannot be decomposed further. For this report, we take the following definitions:

- A business rule is a declarative statement or constraint on organisational business. If the rule can be decomposed further into a higher level of detail (i.e. two or more statements), we call this a composite business rule.
- When a business rule cannot be decomposed further, it is referred here to as an atomic business rule.

The following sections describe the different types of business rules, following the discussion by the Business Rules Group [63].

### 3.3.2 Structural Assertions: Terms and Facts

A term used in the organisation is the simplest, atomic rule. A term is a word or phrase with meaning for the organisation. Terms can be divided into two types:

- **Common terms**: words in the everyday language that are used in the organisation. These terms do not require an organisational definition, their meaning is generally understood and agreed by the people and systems. Concepts such as Person, City, Fire, are (likely to be) common terms in an insurance company, for example.

- **Business terms**: words that have a specific meaning for the organisation, within a designated context. Terms such as Agent, Insurance Policy, Invoice, and Commission might be business terms in an insurance company.

The essential difference between common and business terms is that common terms are not explicitly defined for the organisation, whereas business terms are specifically defined by their participation in one or more organisational facts. An organisational fact (or simply a fact)
establishes an association between terms by expressing a relationship between them within the context of the organisation. For example, “An insurance agent receives a commission for each insurance policy sold”, establishes a relationship (a fact defining the relationship) involving the terms insurance agent, commission, insurance policy and sale. Terms such as these appearing in facts can be either common terms or business terms, but a business term must appear in at least one fact to be such. In other words, common terms can, but they are not required to, appear in a fact, while each business term must be used in one or more facts.

Intuitively, business terms are the concepts that have special meaning for the organisation, while common terms are terms of common use that the organisation accepts as generally understood and need not be defined explicitly (they were called primitives in the GeM model). It is important to note that what is a common term and what a business term depends on the organisation itself: customer can be a business term for an organisation selling insurance policies — the term is involved in one or more facts, it has a specific meaning as a person who has taken up at least a policy or it has a record on a computer system — whereas it can be a common term for a small retail that does not specifically keep track of their customers. The organisation decides which terms require a deeper definition — to ensure proper interpretation, to be used correctly in their business interactions, to precisely define what is the beginning or end of a operation — and these are called here business terms.

Facts assert an association between terms. That is, a fact expresses a relationship between two or more terms, and a term may be involved in more than one fact. A fact is not necessarily binary, but it can involve several terms, such as “The selling agent receives a commission for each insurance policy sold” that defines the fact “Insurance Agent Commission on Sale” in which the three business terms involved are Insurance, Agent Commission and Sale. The intervention of a business term as part of a fact is termed a role of the term in the fact. For example, “An insurance policy corresponds to one customer” is an expression that a customer plays a (semantic) role as part of the fact and, of course, that each insurance policy plays a mirror role in the same fact.

Facts can also be classified in two different ways:

**Base Facts and Derived Facts:** a base fact is simply asserted (“An Insurance Policy must be either Home, Car or Life Insurance”), while a derived fact is an assertion constructed from other assertions via a derivation following some rule/s (“The agents commission is calculated based on the total premium and the form of payment”). The inference of derived facts can
be based on an inference (“The premium for a Car insurance policy is based on the category to which the driver belongs”), or a mathematical calculation (“Agents Commission = 5% * Total Premium + Payment Bonus”).

**Attributes vs. Generalisation vs. Participation:** facts can also be classified as:

- A term — a type — can be an attribute of another term, as in “Model is an attribute of car”.
- A term — a type — can be a generalisation of other terms, as in “A Manager is an Employee”. This is expressed by saying that Employee is a supertype of Manager.
- Terms can participate in a fact, such as “An insurance policy corresponds to one customer”.
  A participation can be of three types:
  - An aggregation: the part of/composed of relationship
  - A role: defines a term as an actor in a fact, as in “A person can be the beneficiary of a policy”.
  - An association: “Insurance policies are issued to customers”

Another way of classifying terms is based on the difference between an abstraction representing a set of instances (close to the idea of an object-oriented class as a template describing a set of instances), and literals representing instances such as “100” or “US Dollars” [63]. A type is used to signify an abstraction of a set of instances, such as “Customer” or “Insurance Policy”, while a literal is a specific value or instance\(^3\). Although more often business rules refer to types “An insurance policy corresponds to one customer”, they can also include literals as in “A policy must be paid in full within 7 days of the date of issue”.

### 3.3.3 Action Assertions

Action assertions specify the constraints of some aspects of the business. Consequently, they refer to the behaviour rather than structure of the business. Contrarily to structural assertions, which are static in nature, action assertions are dynamic. Structural assertions define possibilities, action assertions constrain organisational activities. These constraints are expressed in a declarative way — “A policy must have a responsible agent” — rather than in a procedural way.

An action assertion is anchored on a business rule (the anchor object), related to one or more correspondent objects. The action assertion is the property of the anchor object; a business

\(^3\)Note that this covers the relationship *is-instance-of* common in object-oriented modeling.
3.3: Business Rules

rule may be the anchor of one or more action assertions. For instance, in the rule “A policy must have a responsible agent”, the term (structural rule) “policy” is the anchor, and “has a responsible agent” the correspondent object. If the rule is rewritten in its equivalent form “If it is a policy, then a responsible agent must be recorded on it”, the anchor is the rule after the ‘if’ and the correspondent object the rule following the ‘then’.  

In an action assertion:

- The anchor object can be a structural assertion (“A policy...”), but it can also be another action assertion (“A new policy must register the agent’s commission”, where “new policy” is the assertion “A policy issued to a previously not existing customer, or to an existing customer on a different insurance area, is a new policy”).

- A correspondent object can be another business rule or some specified action. Both these are called constructs in [63]. In this way, the correspondent object in an action assertion is always a construct.

Action assertions are divided into classes, are of many possible types, and are either controlling or influencing assertions:

- Action assertion classes: action assertions can be classified as:
  - Conditions: if a certain condition is true then another business rule will apply. “If a customer does not exist then execute NewCustomer()”
  - Integrity constraints: must always be true. A rule such as “A customer must have a Customer ID” precludes a new customer from being created without an ID, or to be given a NULL ID value on a database system. Business processes must ensure that these constraints are met at all times.
  - Authorisations: defines a privilege of an active agent (person, process) with regards to a construct (business rule, action). For example “Only Head Office can create invoices” or “Only the manager can authorise recreation leave”.

- Action assertion types: There are many possible types of action assertion rules (for a comprehensive list see [63]). The types relevant to this research are:
  - Enablers: an anchor action assertion which truth value allows (enables) the existence of the correspondent object (the creation of a new instance, enables another action assertion, triggers the execution of a procedure). If the anchor is an integrity constraint (i.e. it must always be true), it directly leads to the existence of the correspondent object; if the

---

4Note that we are not including in this discussion what form the description of rules takes; this is covered extensively in the literature. Any of these forms may be used for this purpose.
anchor is a condition, then it leads to the existence of the correspondent object only if the condition evaluates to true.

- Executives: a type of action assertion that causes the execution of one of more actions. The rule “If a customer does not exist then execute NewCustomer()” is an example of the combination of a condition “If a customer does not exist”, and an executive action assertion “execute NewCustomer()”.

- Controlling vs. Influencing: All of the examples discussed here can be categorised as controlling assertions (i.e. they determine what must or must not happen). However, is it possible to use action assertions to define what should or should not happen (“Postgraduate students should not have more that 10 hours a week of casual teaching work”); these are action influencing assertions. Although these are a kind of guideline not typically ‘hard coded’ into systems, they can be used to outline a course of action, or even to request a specific action to take place if the guideline is violated, as in “Postgraduate students should not have more than 10 hours a week of casual teaching work, unless specifically authorised by the Head of Department”.

### 3.3.4 Derivations

As discussed in Section 3.3.2, a derived fact is a particular type of fact created by an inference or a mathematical calculation from other business rules (terms, facts, action assertions, etc). Each derived fact must be derived using one derivation, and each derivation must be used to derive at least one derived fact. A derived fact is managed just like any other fact, it derivation is invisible. A derivation is a business rule, either:

- A mathematical calculation: produces a derived fact following a mathematical algorithm. “Agents Commission = 5% * Total Premium + Payment Bonus” is an example of such a derivation.

- An inference: produces a derived fact by logical reasoning: “The Total Premium of an Agent is calculated by using the individual Policy Premiums for which the agent is responsible”.

### 3.3.5 Summary of Business Rules: Semantic vs. Procedural Rules

The classification in the previous sections establishes that with respect to the data in the organisation, the business rules either relate to the structure or to the behaviour of the organisational entities. The purpose of this section is to determine which business rules are related to the
morphology of the data entities, and which to the procedures or executable actions of the organisation. Morphological (data structural) rules are the rules that define the structure of the data, usually expressed as a data model such as E/R or NIAM, and they define the meaning — semantics — of the business terms in use in the organisation. A business rule that specifies known entities, or state how entities relate to each other is the case of a rule that defines the semantics of data through typing, inheritance and aggregation. We call these rules here \textit{data semantics, semantics related} or simply \textit{semantic} rules. Business terms and their structure, and all base facts are simple, non-calculated assertions about organisational terms, and therefore relate to the structure of the data entities of concern. Hence, they are data semantics rules. Many structural rules belong to this category. In contrast, derived facts — also structural assertions — are complex assertions constructed from other assertions via some logical inference or calculation, and hence they are not directly related to the structure of the data. Table 3.2 summarises.

<table>
<thead>
<tr>
<th>Rules</th>
<th>Data Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business terms</td>
<td>Y</td>
</tr>
<tr>
<td>Base facts</td>
<td>Y</td>
</tr>
<tr>
<td>Derived facts</td>
<td>N</td>
</tr>
</tbody>
</table>

Action assertions rules may also be directly related to the structure of the data, as in the action assertion “A customer must have an address recorded”, which indicates that the business term customer possesses the mandatory attribute address. Although the classification of action assertions as Conditions, Integrity Constraints and Authorisations is dependent on the anchor objects, whether an action assertion is related to the structure of the data really depends on the type of correspondent object. That is, if the correspondent object is a data semantic rule, then the action assertion is a data semantics rule, as in:

“An employee must have an address recorded”

“A new customer must be allocated a credit limit”

Otherwise, if the correspondent object is an action, the rule is not directly related to the structure of the data, but it produces the execution of a procedure, as in (See Table 3.3):

“If the customer is new, then execute \texttt{NewCustomer()}”

“For each new order, execute \texttt{NewInvoice(OrderNo)}”
It is important to note that in some cases the same rule expression may be modeled either as a data semantics or a procedural construct. For example, the expression “An insurance policy has a premium” may be equally validly modeled as an attribute of the policy, or as the execution of the premium calculation. This is a design decision that is in the hands to the analysts driving the modeling process, based on their knowledge of the business.

The action assertion most likely to be related to the structure of the data is an integrity constraint. This is because an integrity constraint is a rule that must always hold true, and it is therefore an appropriate class of rule to specify mandatory type and composition of organisational entities, as in “A Department has a Manager”, or “A book must have its ISBN recorded”.

Conditions (“If a customer has been with the company for more than 5 years, a special discount will apply to each purchase”) and authorisations (“Credit Managers may authorise a Credit Limit”) may modeled by optional attributes of an entity if they lead to optional data structure rules.

The correspondent constructs — e.g. “execute NewCustomer()” — of other action assertions are executives, and some conditions such as “If a new policy is issued then execute WelcomeNote()” do not directly relate to the structure of the data. These are the rules for which the correspondent construct is a specified action (e.g. “execute NewCustomer()”).

As it is the case with derived facts, derivations do not relate to the structure of the data, since they represent logical inferences or calculations that are to be performed to arrive at the value of the rule. Thus, derivations cannot be modeled as part of static data structures, and they must be mapped instead to a function or method. Table 3.4 summarises:

Since composite rules may be decomposed into a sequence of atomic ones, the analysis of the following section is restricted to atomic rules, that is, rules expressing one unit of logical thought.
Table 3.4: Business rules and data semantics

<table>
<thead>
<tr>
<th>Rules</th>
<th>Data Semantics</th>
</tr>
</thead>
<tbody>
<tr>
<td>Business terms</td>
<td>Y</td>
</tr>
<tr>
<td>Base facts</td>
<td>Y</td>
</tr>
<tr>
<td>Derived facts</td>
<td>N</td>
</tr>
<tr>
<td>Action assertion, data-related construct</td>
<td>Y</td>
</tr>
<tr>
<td>Action assertion, action/executable construct</td>
<td>N</td>
</tr>
</tbody>
</table>

3.4 Summary: Data, Business Rules and Workflows

The notion of constructs as a general term for a correspondent object of an action assertion is central to both the structure of the data and the execution of business workflows. As stated before, rules are declarative rather than procedural, so by allowing the correspondent object to be either a data semantics related construct or a specified action, it is possible to include within the rules framework the semantic of data and execution of procedures.

Thus, this chapter presents data semantics as part of the business rules of the enterprise. That is, the meaning of organisational data may be described by business rules expressing the is-a and has-a relationships. If the organisation decides on the use of ontologies to be able to meaningfully share data, the structure of a domain ontology may be described by a set of business rules. The rules defining the ontology represent the meaning of the data in a semantic way, not simply in a structural way, and these semantic descriptions may be used to map data items across the enterprise. As discussed in Section 3.3.5, these rules are semantic rules that refer to the structure of the data.

On the other hand, when the corresponding object is a procedure-oriented construct rather than a semantics-oriented construct, the truth value of the anchor assertion will trigger the execution of the correspondent activity, thus resulting in the execution of an enterprise workflow. See Figure 3.3.

Although some type of business rules are always data semantic rules, many times what kind of rule a given rule is is ultimately a design decision stemming from the analysis of the business process. This is so because often the same expression may be interpreted either as a data semantics or a procedural rule. That is, in some cases it is not possible to establish a priori whether the corresponding object is a data attribute or a procedure, and it may be modeled either way. For this dissertation, when the execution of a workflow is the corresponding object of an action assertion, we use here the form:
if condition then execute procedure().
Chapter 4

The Federation

This chapter introduces the principles that guide our approach to enterprise integration. It presents a particular view of an enterprise as a federated organisation, and introduces the federation as a supporting structure for enterprise integration. The discussion includes the types of interactions possible between interacting systems, and the conditions and constraints for a federation-based information systems structure.

4.1 Principles of Enterprise Integration

The problems of disconnection, common to many organisations, are compounded by the pressure on a modern enterprise to be more flexible and to adapt quickly to changes trying to keep pace with new developments, whilst attempting to contain their typically high IT expenditure. A modern corporation needs to be able to react quickly to changes in the business environment, such as changes in regulations, mergers and takeovers, new competitors and new technologies [106]. Although an enterprise needs to be constantly searching for new ways to better meet its objectives to keep the business competitive, it should always respond to the challenge by making the best possible use of its existing people and resources [29, 64]. Since we contend that for a really large organisation the creation and maintenance of a coherent set of centralised organisational knowledge would be an unfeasible task, we argue that to achieve the objectives stated in Section 1.5 the enterprise integration strategy should:

- divide the integration effort into manageable components, by grouping and layering;
- reuse as much as possible existing processing and systems to reduce costs and disruption;
- reduce the number of knowledge elements to be shared;
- provide a minimal, consistent, properly documented standard description of the knowledge required by the integration;
• establish an appropriate framework and technical infrastructure for safe and effective knowledge sharing;

• provide an environment and mechanisms to overcome enterprise heterogeneity;

• provide structures to support the correct execution of enterprise processes;

• enable the gradual integration of processes and business flows;

• provide a progressive strategy to move to an integrated enterprise, minimising upheaval;

• implement a flexible, capable of controlled evolution, integration architecture.

Hence, we maintain that a successful enterprise-wide integration strategy should recognise and show consideration for the existing elements of enterprise structure and behaviour, while at the same time provide a framework within which meaningful knowledge sharing can occur. Our intention is to reuse as much as possible the existing infrastructure in terms of people, knowledge and systems to meet the new challenges, developing a strategy to provide the right conditions within the organisation to exploit new opportunities. Such an approach respects the existing organisational structure by identifying existing enterprise processing components, enabling them to share knowledge in a standard, consistent manner. New processes and systems might then be incorporated to an existing structure, making enterprise IT evolution a smoother process.

4.1.1 Following the Structure of the Enterprise

Thus, we argue that the integration strategy should avoid as much as possible conflict between local systems and the organisation as a whole. We try to leverage off the work of existing organisational units as they carry out their tasks, with the following characteristics:

• A bottom-up — i.e. from the units up to the organisation as a whole — integration strategy to take into account not only the data, processes and rules of the enterprise, but also its existing organisational divisions. These groups are natural reservoirs of organisational knowledge and of semantic and syntactic coherence, since they share the same language and conceptualisations.

• Centralised, semantic data descriptions to provide consistent, minimal, documented storage of enterprise knowledge.

• Respect for existing units, which are already working groups in charge of enterprise tasks, supported by their own local systems and hence technologically coherent — albeit not necessarily homogenous. Consequently, they understand their own data and processes, and if
properly supported they should be able to provide mappings between external knowledge and their own.

- Implementation of a corporate information structure and mechanisms to replace point-to-point exchanges between systems by point-to-structure exchanges, reducing the number of required contact points.

- Units to keep control over their own processes, and consequently over their systems and the information they manage, making them less likely to resent the integration strategy.

- Local applications still providing the necessary support for carrying out the tasks of the enterprise. In this way, their processing capacity is reused, resulting in less need for significant added computational infrastructure.

As a consequence, it should be possible to more readily accommodate initiatives — such as following a merger or takeover — given that new incorporations may be considered as another unit or set of units to be associated with the sharing strategy. It should also be easier to absorb changes in organisational structure, since they are likely to involve only some of the units and then these are the only reorganisations to be contemplated. In addition, this also allows the progressive introduction of individual components and integration services, so the integration may be implemented gradually, avoiding the disturbing implementation and rollout typical of other strategies.

This approach requires, though, an overarching structure to be put in place for local processing to be able to contribute to the common goals. The overall integration effort should respect the autonomy of the units but enable them to contribute to the enterprise-wide, integrated operation. In this way, most of the existing knowledge and local processing power is reused by the integration strategy.

4.2 The Federated Organisation

4.2.1 Introduction

Recent advances in distributed computing technology were thought to offer a good deal of promise to the enterprise to make the best of existing systems. The development of fast, reliable network technology has made it possible to actually distribute processing across network nodes, thereby opening many new possibilities for distributed computing. Today, distributed computing has grown to be one of the most important IT issues affecting the enterprise and
the way it operates. However, software engineering principles and guidelines for distributed systems have not always kept pace with the increase in popularity of distributed computing technologies [35, 122]. Distributed systems have often been found to be expensive to maintain and enhance, and they are generally unresponsive to business needs for change [18, 31, 85]. The technology is often taken as driving the software engineering process, rather than the reverse.

In order to overcome the difficulties that enterprise systems are currently facing, it is necessary to establish principles and techniques to design distributed systems that avoid the pitfalls of the early developments, and to develop and deploy them effectively. To this end, some researchers have focused on extending existing analysis and design techniques to the distributed domain [32, 45, 67, 90, 108], while others have focused on patterns of relationships between distributed modules to provide an architectural solution to specific distribution problems [1, 124]. However, despite that advances in middleware technology have provided a reliable platform for the distribution of data and computation across the different nodes of a network, the engineering of distributed systems has not taken advantage of the distinctive aspects of distributed computing [42, 124].

In a centralised environment all the modules of a running application are located at a single platform, under the control of the application, which is typically managed by a particular enterprise group. In contrast, modern communication infrastructure makes it possible for an application to execute in a really distributed manner, with components invoking remote services of other components located at a different network location. Components interacting remotely in this way are likely to be under the control of different enterprise groups, in which case the requester of a remote service does not have complete jurisdiction over the call. Consequently, a poor performing module, or a component outage for a significant period will affect not only the direct group involved but also all other groups invoking the service. Since the technology allows callers to invoke a remote service connected by slow or unreliable technology, it is not safe to assume that the connection will be fast enough, reliable enough, or will be operating at full capacity all of the time. Thus, the design and implementation of enterprise-wide distributed systems should contemplate not only a certain organisational configuration but also the availability, capacity and reliability of the supporting technology. In addition, the necessary use of middleware to enable the communication between the running components implies that the characteristics of that middleware infrastructure also have to be considered. Rather than concentrating solely on establishing which modules are interacting, a designer also has to consider how the distributed modules will interact, and what requirements the different types of interactions place on the groups comprising the enterprise, their applications and the communication infrastructure.
Typically, in an organisation of a considerable size there are a large number of business processes, many centres of knowledge, significant user groups who value their authority and autonomy and a dispersed geographical distribution of organisational activities [29]. As a result, a large enterprise IT structure is built out of applications that support the work of these groups, often upon heterogeneous platforms. This naturally results in tightly integrated software clusters supporting closely coupled business activities related to an organisational unit, such as Sales, Warehouse, or Head Office. To complete business processes, often these relatively autonomous units have to co-operate with each other, their running processes exchanging agreed information at specific times. These groups are responsible for parts of the enterprise processes and information management. Thus, they are used to manage and control, and feel that they own, the information for which they are responsible, and they are likely to resist losing any jurisdiction over it.

In addition, these groups are very often led by a manager of some standing and considerable influence within the organisation, who may perceive as dangerous the exposure of the group’s information, and therefore can make the sharing rather more difficult. Often these groups also have some aspects of direct competition with each other, such as the different groups and different branches of AllBooks. This presents similar difficulties to the case of distributed databases. Due to the high significance of data integrity in a database system, it may be hard to get agreement from database stakeholders to expose their data to external agents to be able to create and maintain a complete integrated data schema. Thus, a model that controls the extent and the mode of the collaboration, and minimises the effort required for the sharing is more likely to succeed than one suggesting unconditional, unconstrained cooperation.

4.2.2 Federated Frameworks for Information Systems

Over the last decade, enterprise federations have been proposed as a significant model to ameliorate the problems of disconnected organisations [24, 28, 130]. The main goal is to reuse as much as possible existing systems, and extract as much commonality as possible from them, thus reducing the complexity of information systems implementation, management and evolution for large corporations. A federated model recognises the existence of relatively autonomous groups that agree to cooperate to achieve a common enterprise business goal. Ideally, such a federated enterprise structure should be reflected on its application software, by matching its information system structure to its organisational structure, the systems reflecting the organisation and the way it operates [42]. Although these groups might be quite heterogeneous, they nonetheless
must communicate — often outside the standard established communication channels — in different ways and by different means to achieve cooperation. In addition to their advantages from the technical point of view, federated architectures are also convenient from the point of view of organisational politics, since the involvement of the different components is clearly delineated and controlled, therefore reducing the likelihood of friction between the different groups.

As a result of the individual efforts, there is no unique view of the federated architectural style, and several models have been investigated. While some approaches are only concerned with enterprise data, other approaches intend to interconnect existing systems to be able to share data or services. Zachman’s Federated Architecture model proposes a more comprehensive strategy based on the development of primitive enterprise models [130]. Although all these approaches are termed Federated Architectures, they are really addressing different problems, and they pursue different goals:

- **Federated Databases** — a federated architecture comprised of database components — have been implemented to solve the problem of integrating heterogeneous data sources without having to provide a single Global Schema and Data Dictionary [95]. A federated database approach reduces the complexity of the data integration by keeping local systems operating autonomously, but agreeing on which data they are going to share with other component systems. In particular multidatabase systems keep local schemata and provide mapping functions to support the interpretation of data. Although this is relevant for traditional systems structure, it may be even more important for modern systems — such as the ones managing Web-based content — since the demand for flexible access and correct interpretation of data from disparate repositories is even greater [51].

- The USA Federation of Government Information Processing Councils/Industry Advisory Council (FGIPPC/IAC) [44] strategy only intends to interconnect completely heterogenous systems to be able to share services across different government organisations. The need to consolidate information from disparate systems to support mission critical services of government agencies drives this effort. Central to their strategy is the sharing of service; that is, the strategy is function centric, rather than system or data centric. To make sense of the data that traverse across shared interfaces they consider imperative the establishment of a register of shared data elements. The initiative proposes the use of open declarative mechanisms and standard documentation languages such as XML, WSDL (Web Services Documentation Language) and ebXML (e-business XML).

- Based on Zachman’s Framework for Enterprise Architecture, his federated architecture proposes an architecture based on the concept of ‘sameness’. That is, the identification of
commonalities, standardisation, reusability, integration among the business units’ primitive models. This sameness is to be found both horizontally and vertically, attempting to integrate every cell in the Framework, including data (Column 1), functions (Column 2) and rules (Column 5) (Refer to Figure 3.2). The architecture stems from clustering around common segments to form Sameness Templates, which are the main constructs for the integration.

The model we espouse here differs from the above along several dimensions. Firstly, our approach includes aspects of enterprise integration — business rules and processes — aside from data and services. This results in a strategy that includes the fifth column of Zachman’s Enterprise Architecture (Motivation). Secondly, the scope of our integration is more reduced than the FGIPPC/IAC, since it is restricted to the knowledge and supporting systems of an individual enterprise; hence there is no need for the discovery of services, and the rules and procedures of the enterprise may also be included in the strategy. Thirdly, our approach does not prescribe the identification of commonalities in the different business units, but intends to be the vehicle of shared knowledge by supporting the loose interoperation of heterogeneous enterprise groups that still maintain their complete autonomy. Finally, our approach is event-driven, as opposed to request/reply-driven, with positive consequences for flexibility, scalability and extensibility as discussed in Section 2.1.

Our federated approach allows a large, IT mature enterprise to support a model of interaction that mirrors existing business interactions, thereby minimising the architectural mismatch between the software architecture and organisational structure. As a consequence, our strategy is better posed to follow the path of organisational evolution. Table 4.1 summarises.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Federated DB</th>
<th>FGIPPC/IAC</th>
<th>Zachman</th>
<th>Fernandez et al</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data oriented</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Service oriented</td>
<td>N</td>
<td>Y</td>
<td>N/A</td>
<td>N</td>
</tr>
<tr>
<td>Event-driven</td>
<td>N</td>
<td>N</td>
<td>N/A</td>
<td>Y</td>
</tr>
<tr>
<td>Local autonomy</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Interfaces</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Sameness</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Scope</td>
<td>Enterprise</td>
<td>US Government</td>
<td>Enterprise</td>
<td>Enterprise</td>
</tr>
</tbody>
</table>

A system federation as described in [124] consists of clusters of applications and technology — called *domains* — supporting the work of the organisational groups. Although within each domain tightly integrated sets of applications are allowed to exist, clear domain boundaries
are defined so domains exhibit the lowest possible coupling between them. The same type of people-based interactions that take place in the organisation are reflected by the information systems-side interactions. A great deal of congruence is then possible between the system architecture on the one hand and the organisational structure and politics on the other: highly coupled interactions are restricted to within domains, while only lowly coupled interactions only are allowed between domains. As IT people working in large organisations can attest, the development and upkeep of enterprise-wide distributed applications usually involves disparate parties at the different stages of the software life cycle. Hence, from the technical point of view a federated systems architecture is likely to impose less administrative and maintenance overheads when compared with other architectural forms [42].

The previous paragraphs describe an organisation as a collection of clusters of business activity, each supported by appropriate local IT infrastructure, loosely cooperating with other clusters to achieve a common goal. Fernandez and Wijegunaratne call this a federated organisation [42, 124]. In such a federation, inter-cluster interactions are to be reduced to passing only information across cluster boundaries, in order to reduce as much as possible distributed systems coupling\(^1\). Thus, no inter-cluster processing is allowed, thereby preserving as much as possible the autonomy of the clusters in terms of their processing and administration. An appropriate federal information bus — a middleware-based infrastructure — is specified to support inter-cluster traffic. In this way, clusters are free to conduct their local business as they see fit, as long as they keep their obligations to the organisation as a whole. These characteristics determine a federated organisation, or more precisely, an organisation based upon a federated organisational style.

### 4.3 Software Systems Architecture

The architectural level is the highest level of representation of software structure. In this report, this represents the structure and behaviour of the major components of the information system infrastructure of a given enterprise. As such, this layer:

- is the fundamental organisation of a system embodied by its components, their relationships to each other and to the environment and the principles guiding its design and evolution [73];

- deals with enterprise system-level properties and organisation [109];

\(^1\)Ideally. In practice a small number of simple inter-cluster processing may be allowed.
• refers to the high-level structure of a software system as a collection of processing elements and the description of their interactions [50];

• defines the structure and relationship among the major components of its software systems [97];

• deals with a system structure comprising active modules, the mechanisms to allow interaction among these modules, and the set of rules that govern that interaction [22].

There has been considerable research interest in the architecture of large software systems in general [22, 50, 109]. Models, frameworks and patterns have provided guidelines for the design and implementation of large software systems. However, only recently the attention has focused on software architectural considerations for the whole of an organisation, or a group of organisations [24, 43, 44, 130]. In this dissertation the focus of the architecture is to cover the computer-based systems of an enterprise’s information technology infrastructure, laid out in a specific architectural form. The term “software architecture” is used here to refer to the macro level structure and organisation of the computer-based systems that support the operations and processes of a whole enterprise. We base our architecture on a coarse-grained breakdown of the organisation into units and their supporting systems. We are specifically interested in enabling knowledge — data, rules and processes — integration of the different units in an enterprise structured upon a federated style.

The term software or application architecture used this way also establishes an architectural form or style. Similarly to the concept of an object-oriented class as a template to define a family of instances, an architectural style defines the form and structure of a family of architectural instances [2, 50, 96], and establishes conditions and constraints for compliance with the style. In other words, an architectural style is the representation of a generic structure, defining a distinctive form associated with a set of structural and behavioural principles for the members of a family of software instances.

An architectural style is, thus, related to the idea of software patterns, a concept initially established to describe reusable software elements. The pattern community has recognised the value of codifying elements of systems and organisations into recognisable patterns [26, 45, 48], to help describe and understand them and their functioning. Considerable research effort has been invested on the discussion of organisational and software patterns, since a pattern strategy takes a more abstract view and hence is able to capture and describe the essentials of structures and the way they interact, without getting lost in the details. This includes recognising,
specifying, classifying and recording repeatable system components and how they interact with each other. Once defined, patterns may then be identified and re-interpreted, to be able to be applied in a wide variety of situations. Although not completely formalised as yet, pattern descriptions use languages and methods that have been accepted by the research community, and as they become more widely accepted the patterns themselves are better understood by implementers and practitioners.

Importantly, this approach introduces the concept of enforceability. That is, the specification of an architectural style that includes the stipulation of conformance criteria provides the necessary information to decide whether an instance conforms to the style, and hence will exhibit the declared behaviour. To do so, the specification must clearly describe the problem it solves and the proposed architectural solution, the context in which the solution may be used, the principles and rationale on which the architecture is based, and the constraints placed upon its application. In this sense, the federation presented here has been formally introduced as an architectural pattern by Fernandez et al [43].

This type of architectural work is consistent with the emerging trend to provide domain-specific, reusable, frameworks and component technologies [50, 86]. Following these efforts system architects may focus on specifying enterprise-wide systems architecture. Our intention here is to make possible the reuse, not only of code segments, modules, and applications but of complete systems. This is paramount for large enterprises, since the number of applications and their complexity makes the detailed analysis of their interactions very difficult. If only coarse-grained components are considered, the architectural work required may be divided into more manageable steps.

An architectural description indicates to the IT professional or system designer that there has been a body of work dedicated to studying the characteristics of a particular type of structure, and that the most important distinctive system features and the consequences of coarse grained design decisions have been carefully considered and documented before. This should clarify for the IT professional how to choose a strategy among the opportunities brought about by new technology. A designer can rely on this knowledge to decide whether a particular solution fits the problem at hand, and to determine the fundamental software components of a system. In addition, knowing in advance the characteristics of the system being built makes it possible to choose the technology infrastructure — the technology architecture — most appropriate for implementation. Furthermore, the description of the architectural style itself often includes suggestions as to what would be the appropriate technology or platform to be used for devel-
development. The work of the designer in this case is to establish whether the generic architectural elements fit the problem, and to what extent the recommendations of the architectural work can be adopted in a particular circumstance.

Figure 4.1 portrays our interpretation of the structure of enterprise IT, comprising four main levels: the services architecture level, the systems architecture level, the software infrastructure level and the technology infrastructure level. For the implementation of its own services, each level relies on the services provided by the level immediately below. This dissertation is mostly concerned with the top three levels:

- the service architecture level, since interactions are based on the application services exposed at this layer;
- the application architecture level, since this level is the main focus of the integration strategy;
- the software infrastructure level, since our integration approach presupposes a federated structure and a supporting middleware infrastructure possessing certain characteristics. These two layers may be naturally mapped to Zachman’s classification: the top level maps to row 3, and the second to row 4, of the Enterprise Architecture Framework.

The following sections present an overview of the fundamental tenets of a federated architecture as introduced in [124]. Section 4.4.1 presents the key concepts of coupling and dependency upon which the federation is based, and section 4.6 discusses in detail the federated architectural style.

4.4 Distributed Enterprise Information Systems

4.4.1 Coupling and Dependency in a Distributed Computing Environment

The concept of coupling and cohesion were discussed by Yourdon and Constantine [127] in 1979. Over the years, these two notions have become established tenets of software engineering (See for example [97, 112]). Cohesion was introduced as a measure of the inner coherence of a software module, that is, to what extent the elements in the module are functionally related. Cohesion is an attribute of software quality that relates to the internal structure of a module. Therefore, it is reasonable not to expect a significant difference between the notion of cohesion for distributed than for centralised systems.
Figure 4.1: Three-Layered Enterprise Architecture
Coupling, on the other hand, is related to the way software modules interact in a partitioned system. It has been established that distinct aspects of distributed systems have considerable influence on coupling [102, 103, 124]. In discussing coupling, Yourdon and Constantine intend to determine how much of one module must be known in order to understand another module. They presented the notion of coupling as a measure of the strength of the interconnection. That is, highly coupled modules are joined by strong interconnections; loosely coupled ones by weak interconnections. If two modules are highly coupled, then there is a high probability that a programmer trying to modify one of them will need to take into account the other. Their aim is to implement loosely coupled systems, systems in which a module may be maintained without having to consider other modules in the system. Their golden rule is that for software to be easier to understand, correct and maintain, a system should be comprised of modules of high cohesion, and be partitioned so that the coupling between modules is as loose as possible.

When they discussed coupling and cohesion Yourdon and Constantine were considering how to modularise a single program or centralised system. Distributed systems as we know them today did not exist at the time. In the distributed systems of today we are confronted with more complex problem domains and highly sophisticated system structures, built out of application components running on heterogeneous environments. These components are likely to be written in different languages, and to be executing at different network nodes on different machines. Nevertheless, they are required to interoperate dynamically to execute some business process. As shown in Figure 4.1, this is facilitated by a set of services provided by a communication middleware infrastructure, which makes it possible for distributed application components to interact with each other via a set of services. These services may take different forms, resulting in different possibilities for the interaction between modules, with different consequences for the structure and behaviour of the system [40]. As discussed in [124], a distributed environment has its own characteristic development, maintenance and execution issues:

- Design, development and maintenance of a distributed system often requires the concurrence of diverse groups and constituencies, likely to be geographically distributed, requiring agreement over the design of the external interfaces. Therefore, each mistake in the design of inter-module interactions becomes doubly significant, since it attracts not only a hefty redevelopment penalty, but also because the penalty may be compounded by the need for re-negotiation between groups with different — sometimes competing — interests.

- In addition to the redevelopment penalty mentioned above, changes for shared software components in a distributed environment become especially difficult, since a service is typically used by any number of components in disparate locations under the control of different organ-
isational groups. Thus, particularly changes involving the external interfaces of distributed components may have severe consequences. Hence, designing these interfaces to minimise a module’s internals exposure to remote components and maximising local autonomy and control, is of critical importance. The goal should be to design modules with minimally connected interfaces, and make systems impervious to changes that do not require modification of these interfaces.

- An executing module of a distributed application calling upon remote modules or resources can raise performance and robustness implications. Distributed systems are built upon a complex set of services connected by communication technology of varying degrees of capacity and efficiency. Due to the complexity of the transport path and the volume of data transferred, remote requests such as functions and database calls may be too inefficient to be invoked remotely, especially if the communication technology is of restricted capacity and there is a high number of calls, as it is expected in a large enterprise. Therefore, the approach of having a single central database to satisfy the data needs of the enterprise is often not be suitable, and this inevitably leads to multiple copies of the data and the integrity problems associated with this approach.

- Off-the-shelf applications may need significant re-engineering to be able to switch from their local source of data to a remote one. This may not even be possible with some applications, due to technological limitations (e.g. legacy applications), or lack of knowledge or authority to introduce the modifications.

- A poorly performing remote service may result in performance degradation for modules that blocked when invoking the service, and the unavailability of a remote resource may result in the inability of the caller to continue processing. These implications will intensify if the remote resources are beyond the jurisdiction of the owners/users of the calling module, since neither the caller nor its owners may be able to do much about rectifying the problem.

- In contrast to centralised systems, module interaction in distributed systems is typically enabled by a layer of middleware software providing support for the resolution of remote calls at the different network nodes. A middleware API (Application Programming Interface) usually provides more than one choice for the type of interaction allowed between modules. An appropriate choice of interface will result in lower overall run-time complexity. If two components, for example, are interconnected via a non-available interface — where the interaction can be take place without the invoked module being available — it is possible to take a service off line without affecting the execution of the caller: the client module keeps executing normally since the server availability is not required for the interoperation. Thus,
in a distributed application run-time behaviour is not only a consequence of the partitioning of software into modules, but also of the type of interaction the designer specifies between them. The very low coupling enabled by appropriate middleware invocations results in types run-time behaviour usually not available in centralised systems.

- Software quality attributes such as efficiency, reliability and maintainability are affected by the choice of interaction type between modules. The discussion in [102] demonstrates that a poor choice of module interfacing has negative consequences for the quality attributes of a software product, and that a good choice is indeed positively reflected on these quality attributes.

The consequences of this analysis for enterprise integration are multiple and significant. One of the main aims of enterprise integration as presented in this dissertation is to make possible the interoperation of existing, previously independent systems. Therefore, the conditions for the occurrence of the problems stated above are aggravated: the prospect of tensions between the different groups, the possibility of different assumptions about processes and data, and the likelihood of negative implications for maintenance, performance and robustness are increased when independent data sources and applications are made to cooperate. These and similar issues mean that the choice of appropriate types and levels of interaction between cooperating remote software modules in an integrated enterprise becomes critical to the success or otherwise of distributed computations within the overall enterprise framework.

With the assistance of the middleware infrastructure, executing modules communicate over a network mimicking local interaction. As indicated above, in an integrated enterprise each poorly designed or implemented remote processing interaction will attract a heavy penalty in terms of the overall reliability, performance and maintainability. An available communication (see below) requires interacting modules to be up and ready to run at the time of the call, introducing a higher system complexity than in the non-available case. A synchronous communication ties up a client module to the execution of the server module. Consequently, large integrated systems often face scalability problems since, due to the implementation of calls entrenched in the mindset of centralised computing, the opportunities for efficiency are often lost by wrong design decisions [103]. Whilst in centralised computing a synchronous function or method call is most often the chosen way for two modules to interact, current distributed computing infrastructure offers several possibilities for implementing remote interactions. Every time an interaction is designed with a communication type of a stronger pairing level than is really required, the designer is introducing unnecessary complexity in the distributed system. The goal should be
to design integrated systems in which the interactions are exactly what is appropriate for the requirement. The possible different modes of communication between two running components afforded by the middleware infrastructure have been described in [124] as follows:

- **Synchronous vs. asynchronous communication:** synchronous communication between two components occurs when one component makes a request to the other, and blocks while waiting for the reply. All the while the call is resolved the caller is not executing. In asynchronous communication, the calling component does not block but continues executing while waiting for the reply. In this case, the call is made at an earlier section of the code than the one using the result [78].

- **Available vs. non-available:** available communication requires both interacting components to be available at the time of communication, such as an RPC (Remote Procedure Call). Non-available communication can proceed even if the intended receiver components is unavailable at the time of communication. This mode is not to be confused with asynchronous communication, which requires the receiving module to be available, even though there is a difference between the time of the contact is made and the time when the return value is used.

- **Conversational vs. non-conversational:** in a conversational interaction, a session is established between the two components, typically by selecting a session ID, and allocating structures — memory buffers, tables, pointers — to keep state of the communication. During the conversation, typically both components exchange information several times; the connection remains open during the whole session. Non-conversational communication is instead reduced to a one-off exchange, after which the connection is broken. If there is a need for more communication, a fresh new connection must be established.

- **Static vs. dynamic binding:** Binding refers to the time at which some parameters of the communication are decided. Static binding ties communication parameters at compilation time, while dynamic binding leaves aspects of the communication to be decided at the time of execution. In centralised systems there is essentially only one type of binding (binding to a form/structure, see below), but three types of binding have been identified in distributed systems [124]:

  - Binding to a form/structure: the determination of the called component identifier, the composition of the parameter list required for the call, and the type of the returned value.
  - Binding to an implementation: binding a component to the implementation language of a remote component (for example, to a particular SQL dialect in a stored procedure). This
is pertinent because, as different from a centralised system, a distributed system can be heterogeneous.

– Binding to an occurrence: resolving the call to specifics of the remote communication, such as to a specific network node or network address, or to the physical characteristics of a remote server.

The notion of coupling has been extended by Fernandez et al [42] to the distributed arena. The types and levels of remote interactions can be articulated in terms of coupling, and therefore coupling assumes a critical importance in designing quality applications with distributed interactions [102, 103]. The conclusion is that, although low coupling is an objective greatly to be desired in a distributed application, the same low coupling cannot be sustained in all remote interfaces of an application. Rather, given a certain requirement designers should employ the lowest possible coupling that satisfies the requirement.

This is closely related to the notion of software dependencies discussed in the next section.

### 4.4.2 Types of Software Dependency

The notion of dependency is related to the requirement that a software module makes of another. For example, when issuing an insurance policy it is normally necessary to obtain the customer details to apply customer-specific discounts to the policy. If this is not done locally but at a host server, the local software supporting the task must send a request to the customer database, which returns the result. The policy can then be issued. The dependency between the software supporting the policy issuing task and the component providing the customer information is a *processing dependency*: the policy issuing task cannot be completed without that the customer information being returned by the call. The requester needs to use the return value for its own processing. In contrast, if the Human Resources (HR) department requires the policy information to allocate the selling agent’s commission, it would be only necessary to make available the total policy premium to it. There is no processing at, or information from, the target module that the originator needs to use: there is no need for a returned value at the caller. The software supporting the issuing of policies has an *informational dependency* — not a processing dependency — with the relevant HR software component. This information carrying responsibility can be handed over to a reliable delivery mechanism that guarantees the delivery of the information.

Moreover, ideally the task at origin does not know or should care about the recipients of the
new policy information message. This implies that the caller possesses no knowledge about
the intended recipients of the information, further decoupling the interaction. Apart from the
already mentioned HR module, there could actually be other modules or systems, such as a
MIS (Management Information System), interested in that piece of information. On completion
of the New Policy task, the source simply needs to hand over the message to an appropriate
infrastructure, which should determine the intended recipients and ensure delivery.

**Processing Dependency:** More formally, a processing dependency occurs when an applica-
tion component needs, for its own processing, some remote work to be performed. There are
two categories of processing dependency:

- **Simple Processing Dependency:** This occurs when application component A needs, for its
  own processing, some work to be performed remotely by another application component
  B. That is, application component A cannot progress or complete its processing until
  application component B performs the requested action.

- **Transactional Dependency:** A transactional dependency occurs when application com-
  ponent A needs, for its own processing, some work to be performed remotely by other
  application components (for example B, C, and D, in general at different physical loca-
  tions in the network). Furthermore, to application component A the separate work that
  components B, C, and D perform must appear as a single logical unit of work, to be carried
  out in an all or nothing fashion. Application component A cannot progress or complete
  its processing until this logical unit of work is completed.

**Informational Dependency:** In contrast to the previous case, there is no processing require-
ment, only information is required.

- As a consequence of some event within its jurisdiction, application component A needs to
  convey some information to one or more remote components. With an informational de-
  pendency, there is no expectation of processing associated with an interaction: application
  component A only needs to make known that some event has occurred, some new infor-
  mation has been generated, or some existing information has been changed. This event
  generates a message destined to, probably more than one, target remote components.

### 4.4.3 Dependencies and Appropriate Coupling

The concept of software dependency becomes crucial when addressing the question of appro-
priate coupling. In terms of attributes such as performance, robustness, maintainability and
scalability, stronger coupling is likely to result in higher integration costs to a system. It is important to remember the lessons taught by the initial experiences with distributed computing in the 90s by researchers and practitioners: in distributed computing costs break down differently than in centralised computing [35]. In the centralised case the major investments are in terms of the necessary hardware and software. In distributed computing, the major expense is associated with the implementation, maintenance and deployment of distributed software. Thus, stronger than appropriate coupling, which introduces unnecessary costs and disruption to existing systems, should be avoided. A lower than appropriate level of coupling, however, may result in integrity problems: for example, if a transactional dependency is implemented as a processing, but non-transactional, dependency, the failure of a component to properly complete is likely to produce inconsistent results. Therefore, rather than universally low coupling, system designers should aim to design with the lowest possible coupling appropriate for the dependencies at hand. When designing a system, some degree of coupling is then introduced as a result of the coupling introduced by the designer, in two ways:

1. by the choice of the interaction type between the modules; and
2. as a result of the implementation selected from among the choices supported by the middleware.

The rules for architecting and designing with appropriate coupling were presented by Wijegunanaratne and Fernandez [124]:

**Rule 1:** At the architectural level, identify dependencies and for each dependency select an interaction with the lowest possible coupling to satisfy the dependency.

**Rule 2:** Select a middleware interface type providing the lowest possible default coupling to implement the interaction.

The designer must design and build interfaces with the lowest possible coupling by exercising careful control over these discretionary elements of coupling. A common mistake is to implement informational dependencies as processing dependencies, or simple processing and/or informational dependencies as transactional dependencies. This results in poor system design. In these situations, the actual underlying dependencies must be uncovered and the interaction re-designed accordingly.

### 4.4.4 Transient Inconsistencies

Please note that the above analysis is not related to the software implementation aspects of distributed computations, but rather is related to the requirements of the system. The decision
actually hinges on a close examination of the business process. In our previous insurance policy example, if applying the premium and updating the commission are not implemented together as a transactional unit of work, there will be a transient inconsistency in the overall (business) procedure between the update of the premium and the update of the agent’s commission: the overall procedure will be ‘out of synch’ in between the two updates, because the policy premium would have been updated on the first component but the commission would not have been recorded as yet on the second. In general this is not a problem, as long as the commission update is eventually recorded against the agent, even after some (acceptable) delay. Therefore, if this transient inconsistency may be tolerated by the business process, there is no impediment to implementing the interaction in a non-processing or transactional manner. In this case, the commission-processing task can be realised with only an informational dependency with the module responsible for premium registration. Once the premium is registered, this information can be published so the HR agent administration system can obtain this information to update the agent’s commission.

Accordingly, where there are dependencies between two or more entities or processes, a decision on processing vs. informational or transactional vs. non-transactional dependency depends upon the degree of business tolerance to the likely transient inconsistencies that will occur. A transient inconsistency can be tolerated under two circumstances:

- it is possible to ensure that an inquiry, processing, or update related to the inconsistency cannot occur in the critical period; or
- the consequences of such an occurrence during the period of transient inconsistency are relatively unimportant for the business process.

Since the second situation may result in an operation reading ‘dirty’ data, it is usually not acceptable in other scenarios, such as databases. However, we focus on a situation in which inconsistencies already exist, since overlapping data is managed by autonomous applications unaware of each other. Thus, corporate processes already act on wrong data assumptions, and it is only the work of the people in the organisation that remediate the problem.

**Transactions**

The use of the term transaction in the above discussion should be considered with caution. In our case the term refers to the completion of a distinct set of operations between distributed

---

2This is closely related to the notion of *lazy* or *asynchronous* data replication strategy. See for example Bischofs et al [20].
modules in an atomic manner, in order to produce a correct and consistent result for the business process. This is an aspect of software development of significant complexity, and therefore in enterprise systems it is usually delegated to a Distributed Transaction Monitor (e.g. Encina or Tuxedo). Although the possibility of non-transactional interactions for a business process exists, such as in the example of the insurance agent’s commission discussed, still the atomic operations and database updates resulting from a process have to be correct. Thus, any database updates, for example, will be treated as transactions by the underlying DBMS.

In the case of distributed transactions it will be necessary to ensure the preservation of data integrity by the use of an appropriate transaction commit protocol, such as the 2-Phase or 3-Phase Commit protocols. Similarly, a MOM infrastructure typically offers an interface to flag a set of messages as a transactional unit — either they are all delivered, or none of them are — thereby making possible to ensure that all the messages reach all their intended recipients. This transactional behaviour is, however, exhibited at the level of the infrastructure rather than the business process. The notion of non-transactional behaviour presented here is restricted exclusively to the updates required by an executing business procedure.

### 4.4.5 Re-engineering Enterprise Systems

The process of enterprise integration is always linked to some re-engineering of enterprise systems to accommodate the integration strategy. When trying to determine module interactions, it is not always easy to discern the actual dependencies, since these interactions are often not properly documented, and they are hard to discover in the code, even when it is available. Although the use of middleware affords designers several interfacing possibilities, as a result of a centralised mindset even when system diagrams and such like exist they typically depict the existing interactions, without necessarily specifying their type. The de-coupling of these interactions requires a careful examination of the business requirements, establishing whether there is sharing of data, whether the interaction may result in a transient inconsistency and, if that is the case, whether this may be tolerated by the business process. Therefore, note that this decision belongs entirely to the realm of business requirements: the decision as to how to classify each requirement is the responsibility of the business analyst, not of the software engineer. System designers should be careful to properly match the given requirements to the type of interaction though, avoid architecting lower-coupling requirements as higher-coupling interactions, and vice versa.

Existing systems that interact via clearly defined, asynchronous interfaces — a file or a database table written by one application and read by another, for example — are obvious candidates
for informational dependencies. This is so because, due to the time lag between writes by a producer application and reads by consumer applications, it is very unlikely that such communications correspond to processing interactions. Firstly, the business process clearly tolerates the transient inconsistencies arising, since there is no synchronicity between the producer and the consumer. Secondly, it is very unlikely that such an interaction corresponds to a processing dependency, given that the time lag would be just too big for a caller to be waiting for a response to finish processing. Hence, for these the resolution is almost always automatic: they can be categorised as informational dependencies.

In general, when previously independent applications are required to interact, two possible situations may arise:

- A module within an application requires some processing from another module in another application before it can finish its processing. We argue that, although possible, this is very rarely the case since the caller module did not need the processing when it was designed in the first place. Only in some circumstances — such as the synchronous use of a new service remotely available — the interaction really needs to be implemented as a processing dependency. However, in practice it is common to see an implementation of the processing type when two previously independent modules are made to cooperate, even though this is not really necessary. We consider this to be an error of design since it adds unnecessary coupling to the system.

- A module needs to be informed by another module of an event, such as a data update. This is the most common situation, and it corresponds to an informational dependency.

The next section discusses two different interaction modes between application modules, and their consequences for the structure and behaviour of enterprise systems.

### 4.5 Service-Oriented Interactions vs. Information-Oriented Interactions

The discussion of the previous sections highlights the new possibilities for the interaction between two modules afforded by the middleware. We argue here that each interaction should closely align with the requirements of the dependencies, as determined by the business process. Each dependency will be mapped to one of the two types of interactions, as follows:

**Information-oriented interactions:** correspond to informational dependencies, and are the result of an event occurring, most likely as the result of a data-updating event, within an
4.5: Service-Oriented Interactions vs. Information-Oriented Interactions

Figure 4.2: An Information-Oriented Interaction

originating module. The module dispatches a message to propagate the update to other remote applications with which it shares data. Hence, this propagation is a consequence of, and results in, data replication. Figure 4.2 shows an information-oriented interaction, where the execution of UpdateCustomer(ID) in the left-hand-side application triggers a message with the updated customer data, and consequently the execution of UpdCustDetails(ID) on the right-hand-side application. Note that there is a local service at each side of the call — UpdateCustomer(ID) and UpdCustDetails(ID) — in charge of updating local data. The data relevant to the exchange may be the same in both applications, or may only have some overlap. In all cases, the data fragments shared by both applications are to be kept consistent.

Service-oriented interactions: correspond to processing dependencies. In this case, a module requests a service to be provided by another module or modules, typically via the execution of a function, procedure or method under the control of the invoked module. Service-based interaction patterns of this type are most common, and have been discussed in the research literature. (See for example Barros et al [16].) Since the interaction maps to a processing dependency, the values returned by the invocation are used by the caller to complete processing. There might be, but not necessarily so, a data update at the caller side using these values, but there need not be data replication. Figure 4.3 shows a data entry module invoking a remote method to retrieve customer data and get the new customer credit limit. Only when the return value has been received the caller is allowed to terminate its execution.

Both types of interactions have advantages and drawbacks. Information-oriented interactions imply data replication across systems, and hence there is a need to put in place mechanisms to keep copies of data properly coordinated, which is not always necessary for service-oriented interactions. On the positive side, although data replication introduces problems for updates, it is very convenient for read-only operations, since the module may access a local copy from
4.5: Service-Oriented Interactions vs. Information-Oriented Interactions

which to read. In addition, information-oriented interactions are essentially asynchronous — even non-available if such a middleware interface has been used for implementation — in the time dimension, and therefore introduce much less coupling than service-oriented interactions, provided that the information can be reliably exchanged. Since this interaction is not initiated by clients requesting information, probably putting the server under pressure to respond within a certain time, this type of exchange is much more resilient to scaling up with the number of interactions. The server (producer) of the information may be made (almost) impervious to the number of recipients by reducing server involvement to its lowest possible when designing the interaction. It has been demonstrated that placing the details of the data exchange on a reliable transport mechanism rather than using a request/reply mechanism has measurable beneficial effects on the performance of the interaction [40, 103]. Since data updates are always executed at the local sites, following the local applications own business rules, there is a very low probability of a failed data update, reducing the likelihood of errors. In addition, the amount of administrative information that interacting modules have to keep of each other is minimal — it may be made almost zero — thereby simplifying application logic.

In contrast, service-oriented interactions may avoid the duplication of data and services (i.e. it is possible to have only one copy of the data, and only one service), but at the price of introducing a higher level of coupling in the system. This higher level of coupling may result in an unacceptable level of system stress and system scalability problems due to the repeated execution of the same service if the provider is affected by a high number of invocations. Additionally, independent applications have been designed to interact exclusively with their own back-end databases, hence the type and the level of granularity of the information on the server application is likely to be different from the client. Consequently, it is highly unlikely that a service-oriented interaction may be appropriate to replace local interactions. Only when a client module requires information not available locally is that a service-oriented interaction
may be indicated. In this case, the information must be found, searched for, requested and received as part of the interaction. This includes interpretation of the format and semantics of the server module by the client, which needs to have readily available a significant amount of administrative information such as argument list, types and formats of the arguments and location of the server module. If this information is resolved at compilation time, flexibility is greatly reduced since the parameters of the call are hard to change when required. If it is resolved at execution time, performance will surely be affected. Hence, for standard, recurrent interactions it is more advisable to make a copy of the required data at the client side to have it available locally. This should really then be implemented as an information-oriented interaction. Table 4.2 summarises.

Due to the centralised computing mindset, service-oriented interactions are widely used in industry implementations, without proper regard for the business requirements. As discussed, however, for two pre-existing applications that independently implement separate business functions and are now required to cooperate, an information-oriented interaction is most likely to suffice.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>SOI</th>
<th>IOI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Replicated data</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Reads immediacy</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Scalability</td>
<td>Poor</td>
<td>Good</td>
</tr>
<tr>
<td>Probability of local failures</td>
<td>Higher</td>
<td>Lower</td>
</tr>
<tr>
<td>Synchronous</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Non-available</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Coupling</td>
<td>Higher</td>
<td>Lower</td>
</tr>
</tbody>
</table>

It is then good design practice to reengineer service-oriented interactions as information-oriented interactions if appropriate to the requirements. The rule to use is:

**SOI vs. IOI** A service-oriented interaction between caller and called modules may be re-engineered as an information-oriented interaction if the return value of the call is only an acknowledgment, which is not necessary for the caller to complete processing.

In other words, if the interaction is essentially one-way, except perhaps for an acknowledgment, it may be implemented as an information-oriented interaction.
4.5: Service-Oriented Interactions vs. Information-Oriented Interactions

4.5.1 Service Orchestration and Choreography

The above discussion should be referred to the new technologies and standards available for implementation of system interactions. Web Services technologies are becoming more prevalent for the integration of disparate systems, mainly because they are based on open, XML-based standards. In this approach, the services to be part of a common business goal are exposed as Web Services to form part of a higher-level, orchestrated process. This is usually controlled by an ‘orchestration engine’, which is in charge of controlling, dispatching and bringing to a close the overall process. This provides a clean separation of concerns between the systems exposing the services, and how these services are combined to provide new processes. This orchestration strategy results in the possibility of reuse of the services so exposed, because each service may be part of several so orchestrated processes.

Several specifications have attempted to address the requirements of service orchestration: XLANG, WSFL (Web Services Flow Language, already discussed in Section 3.2.6), BPL4WS (Business Process Execution Language for Web Services), WSCI (Web Services Choreography Interface) and BPML (Business Process Management Language) may be mentioned. All of them provide a layer on top of the Web Services layer to compose collaborative processes, with varying support for sequencing and flow of control of activities, support for data exchange and persistence, support for transactions and exception handling. Importantly, they all address the need for asynchronous interactions — with call-backs and time-outs — and long-lived transactions, mainly by compensation.

Referring back to our discussion of Section 4.4.2, it is important to note that even though all the strategies outlined here support asynchronous processes, the interaction type is always a processing interaction. That is, all interactions are of the request/reply, service-based type in which the caller is executing a service and waiting for the reply, closely resembling an asynchronous RPC. As discussed, this corresponds to processing dependencies.

In contrast, we are advocating here the suitability of informational dependencies for the implementation of many business processes currently based on processing dependencies. In Chapter 5 we introduce FEW, a workflow model exclusively based on publish-and-subscribe message passing, in which all interactions are non-available — rather than simple asynchronous — and hence all dependencies are assumed to be informational dependencies. We do not prescribe any specific capabilities of the middleware infrastructure other than guaranteed message delivery and sequencing, and the possibility of treating several messages as a single unit of work. Modern
message-oriented middleware products all exhibit these characteristics, and as we posit in our Conclusion section we envisage that the Web Services framework will provide these services in the near future.

4.6 A Federated Systems Architecture

The federated architectural style presented in this section is grounded upon several principles. Firstly, as discussed before, it makes certain assumptions about the enterprise, recognising and respecting some of its generic structural and organisational features. Secondly, it assumes a certain distributed infrastructure capability — based on existing middleware — that makes possible for distributed application components to communicate and interoperate. There is no implication here that the wider aspects of the enterprise — such as people-based procedures, informal groupings and their communication [55], organisational politics, power and its impact [28] — form part of the software architecture [27]. However, it has been recognised that an enterprise application architecture that acknowledges and supports these facets of organisational reality is better suited to the enterprise than one that does not [42]. The system infrastructure becomes then the software counterpart of the enterprise, its architecture in context [49] reflecting its physical and organisational boundaries and behaviour.

A domain is defined as a cluster of organisational processes and their supporting systems. Each domain may contain a single application or a related group of applications concerned about the processing of one or more closely related tasks. For example, a domain can be determined by a process such as General Insurance, an organisational group or structure such as Human Resources, or by a geographical criterion such as the New Zealand branch. A federated enterprise is comprised of relatively autonomous, co-operating domains each connected to a middleware-based Federation Highway that makes their communication possible (See Figure 4.4).

A major objective of this architecture is to keep domains as autonomous as possible, since in a large organisation there are obvious benefits for a domain in being able to grow and change with relative freedom. This basic principle of autonomy should be accompanied with a commitment from domains to keep their interactions with the rest of the federation stable.

Our basic premise is that applications housed within a domain should exhibit the least possible coupling with applications in other domains. The discussion in [42] argues that the highest
degree of autonomy arises where domain boundaries are demarcated such that only information is exchanged between them. In addition, since message-oriented middleware (MOM) is best suited to implement informational interactions, the federal highway suggested implementation is by connecting each domain to a message-oriented middleware infrastructure. Domains communicate with each other exclusively via sending and receiving messages, with the federal highway being responsible for guaranteeing delivery. Incoming and outgoing messages represent a domain’s only means of communication, the only interface to the rest of the federation. The rules — introduced by Fernandez and Wijegunaratne [42, 124] — governing domain interactions are:

Rule 1: The information source (message sender) does not require any processing from a remote module; it merely needs to make available the information to the federation.

Rule 2: Source domains publish information, to which authorised domains may subscribe. This is the only mechanism for information exchange, although as noted some exceptions may be allowed.

Rule 3: Information about recipients is not to be maintained at application level. The lowest coupling is achieved when such information is maintained at the infrastructure level.

Rule 4: Since the infrastructure contains an assurance or guarantee of delivery, applications need not be concerned with message acknowledgement or tracking (thus, further reducing coupling).

A domain needs not know about the other domains interested in a piece of information. It only needs to:

- publish via the federal highway items of information generated as a consequence of some
event within its boundary — such as a new customer, a new order, a change in the insurance
premium — that is of more than local interest;

• get from the federal highway items of inter-domain traffic of interest to the domain.

Therefore, an event occurring within a domain triggers a message exchange in the federation,
as long as the information generated by the event needs to be shared with other domains. As
discussed in Section 1.3, this is typically the result of the interaction of a human (customer,
salesperson, representative) with an enterprise application housed inside a domain that results
in some information being updated within the application. The application then must place
the required information on the federation structure; it is the responsibility of the federation
service to multicast the update to all intended recipients.

4.6.1 Federated Architecture and Organisational Structure

The way domains are partitioned largely depends on the individual circumstances of the enter-
prise. The major partitioning objective is to confine closer dependencies within a domain; thus,
the systems architect should investigate different candidate boundaries to ascertain the best
possible way forward. In partitioning the domains, there are several competing candidates:

• Organisational boundaries, such as boundaries of:

  – organisational units, departments or divisions, such as Head Office or Human Resources;
  – organisational functions [53] or processes [64] — Home Loans, General Insurance, Credit
    Cards — which may or may not coincide with the above;
  – individual applications that support a defined process or task such as Order Processing or
    Financials (a very common case);
  – existing organisational boundaries, or intended ones where the software architecture is
    built to support an anticipated organisational change.

• Geographical boundaries, such as physical sites where distinct parts of the enterprise are
  located.

• Technology boundaries, such as the ones demarcated by existing platforms or application
  groups.

It is natural to partition along organisational process lines for organisations with a process ori-
etnation, and along functional lines for organisations that are functionally organised. In global
organisations it may be more appropriate for domains to be determined on a geographical basis
— different enterprise branches, even in different countries, for example — since this might be the best avenue to isolate closer dependencies within a domain and dissect looser dependencies across domains. Of course, a geographic, process, or function-based partitioning approach needs to be tempered by the structure of existing systems, which may contain closely coupled interactions across these boundaries. Equally, as described before, the improperly designed structure of existing systems — loose coupling where close integration is needed, tight coupling where loose integration is indicated — may well be a very important motivating factor to organise the enterprise systems as a federation. In order to maintain data integrity, in centralised systems the tendency of analysts is always to design with tight coupling. It is then common to see loose requirements designed and implemented as tight interactions. In these cases, some redesign of existing systems to uncouple these tight links across domains may be required. In contrast, design with loose coupling when tighter coupling is be required is much less common. As discussed, in all cases these decisions hinge exclusively on the requirements imposed by the business process, they are not a based on a technical design decision.

Regardless of the criteria used for domain demarcation, two issues are significant for enterprise integration:

- Different domains may be decided upon according to different criteria. It is possible, indeed very likely, that a domain can be determined by one given criterion, e.g. a specific task such as an application controlling a complex machine, while others are decided upon some other criteria such as geographical or functional lines. In this sense, domains need not be necessarily coherent; it is indeed very likely that in an organisation domains will be demarcated responding to different lines.

- The federation is a recursive pattern. Since the internals of a domain are opaque to the rest of the federation, a domain itself may be organised as a federation. It is therefore possible to recursively implement a federation inside a higher-level federation domain, so that there may be several levels of federation as shown in Figure 4.5. This feature becomes essential when it is necessary to consider global enterprise processes, as discussed in Chapters 5, 6 and 7.

From the organisational point of view, Davenport et al [28] reason that a model that admits politics and conflict better reflects reality than a model of unstinting co-operation and unfettered exchange of information. They suggest federalism, a model where potentially competing or previously non-co-operating parties are brought together by negotiation, as a preferred archetype. The proposed federated architecture is well placed to support this type of behaviour:

- Firstly, no process belonging to another domain will exert control over a local process or
resource (e.g. locking records during a distributed transaction), since only information in the form of messages is allowed in the inter-domain arena.

- Secondly, the performance of a module will never be hampered by the poor response time of another.

- Thirdly, since processing reflects organisational boundaries, the existing processing power of domains remains intact.

- Finally, apart from the window into it represented by the domain interface determining the domain’s runtime behaviour with respect to the federation, a domain is opaque to the federation. Hence, the nature and extent of the domain’s dealings with the federation can be negotiated, and the internal workings of the domain may be exposed to the federation possibly only to the extent with which the managers of the domain feel comfortable.

This federated architecture aims to minimise the conflict between the software architecture and organisational reality by designing systems following the contours of the enterprise. The term “architectural mismatch” — originally proposed by Garlan et al [49] to cover assumptions in the technical domain that conflict with reality — can be naturally extended here to take account of mismatches between the software architecture and organisational reality. With the domains mirroring the most significant organisational boundaries, and their behaviour able to support established organisational behaviour, the federation can become the software counterpart of the
4.6.2 A Layered Organisation

The federation has been presented as a recursive architectural pattern [43]. The only opening available into a domain is via its single entry point, an interfacing software module — a Message Adaptor — which communicates, but also insulates, the domain from the federation.

Hence, a federation is only aware of its own domains and the messages they exchange, and it is oblivious as to the final destination of the information exchange within a domain. Trans-
internal domain consumption, to the appropriate application, or even to an inner federation housed within the domain. These details are transparent to the outer federation, which only has information about domains Message Adaptors and the type of messages exchanged among them. This naturally results in a layered structure, for the enterprise, with inner federations housed within domains of outer federations, as depicted in Figure 4.7.

We shall argue in Section 5.2 that such an organisation determines the separation of data, rules and processes in natural layers as well. A domain inner federation may contain data formats, business rules and workflows different to the ones defined for the outer federation. The outer federation is oblivious to this, since compliance with its own structures and protocols is its only concern. Moreover, since the rules of the federated architecture prevent the seepage of domain business rules and processes into the federal arena (see Chapter 6), only information-carrying message payloads are to be re-interpreted across a domains boundary. In this case, the Message Adaptor of a domain that is itself a federation is in charge of translating data and their formats in and out of the domain, by converting messages from the outer federation format into the inner federation format and viceversa. Chapters 6 and 7 analyse the impact of this layered structure in our proposal for enterprise integration.
4.7 Domains

4.7.1 A Domain’s Obligations to the Federation

The operation of a federation is based on the controlled, regulated cooperation of the domains involved. The responsibilities and obligations that a domain has to the federation must be established, and complied with at all times. These obligations are defined by a contract — not a contract in the legal sense of the word — a commitment to act according to the rules and protocols of the federation. This approach is absolutely necessary to ensure the proper flow of information through the federation. Among others matters, a contract may specify:

- Data (and the formats) originating in the domain of more than local interest; data (and formats) originating elsewhere that are of interest to the domain.

- Business rules associated with the above categories of data, such as rules about the treatment of inter-domain data. For example, a domain may be entitled to subscribe to a message carrying data, but it may not be allowed to modify it. (These rules will be discussed further in Section 5.6).

- Although a domain may unilaterally discontinue its subscription to a message, a publishing domain is not allowed to cancel the publication of information relevant to the federation. That is, the cancellation of a message publication must have the agreement of all domains subscribing to the publication.

- Performance guarantees that may apply in certain implementations. For example, that messages will be published by the originating domain immediately after the event, or that a certain type of messages may be batched and put onto the highway within a certain time of the event generating the message.

- Protocols to deal with errors and recover from failures. Although the delivery of messages is guaranteed in the federation, there is always the possibility of out of the ordinary errors, such as full queues, of long time network unavailability. A well-understood set of rules for domains to follow when recovering from failure is needed to ensure the integrity of the federation.

- Security mechanisms such as authentication and encryption: confidentiality and integrity at domain level may be desirable. Hence, security management of messages on the federal highway may be an important issue. Federation structures are to maintain that information.
• A self-describing message, where this is permitted by the middleware, involves an even looser
degree of coupling, by preceding the data with a section describing it. The semantic content
of each message type still needs to be agreed upon, but the domain will have discretion on
the order, length and type of the data in a message. For example, a New Customer message
may still contain the ID, Name, Delivery Address, Contact Name, Contact Address, and
Contact Telephone Number, but the order of this information and the field lengths can be
described in the meta-data at the front of the message.

Elements of the contract are incorporated into various software components of the federation
structures. The most important of such components, the Messaging Adaptor, supported by the
Integration Service (IS) is responsible for the domain’s interaction with the federal highway. As
a consequence of its contractual obligations and based on the information contained in the IS,
a Messaging Adaptor must:

• Receive from the applications within the domain the information intended for inter-domain
  travel, correctly prepare and address a message for multicast via the middleware.

• Select those messages relevant to this domain from inter-domain traffic, and deliver them on
to the appropriate application component within the domain.

• Act as an agent for the applications to implement federation-based protocols.

• Manage federal activities on behalf of domain applications.

• Contribute to federation error recovery.

The Message Adaptor responsibilities may be carried out by either one or more application
components, or be spread among application and infrastructure levels. The actual implementa-
tion of Message Adaptor functionality is contingent upon the MOM chosen for the federal
highway, the intra-domain middleware, and their capabilities. Nonetheless, stemming from the
case studies experience — see Section 4.8 — a generic structure for a Message Adaptor is pre-
sented in Section 6.6.

4.7.2 Inside a Domain

So long as the obligations to the federation are preserved, a domain is free to organise itself and
evolve in any manner it sees fit. A domain will have its own data and process model, and its
own technology platform. Apart from the data-only interfaces and contractual obligations, the
domain is opaque to the federation — its internal structure, organisation and implementation
need not be visible. Within a domain, all dependencies (processing, simple and transactional,
plus informational) are permitted, and there may be distributed applications or application components or a mix of legacy and distributed applications. As discussed, a domain may be organised itself as a federation.

Different domains within a federation can be differently organised. A domain’s internal organisation need not be influenced by that of any other. Where a domain contains distributed applications, it may choose any of the prevailing types of middleware — SQL, distributed TP monitor, object broker, MOM, an RPC product — for intra-domain communication. Since inter-domain communication is restricted to informational dependencies, all transactional behaviour is constrained to within a domain.

If, for example, inter-domain interfaces were processing dependencies (implemented perhaps with RPC-type interactions), then, using the coupling principles in [42], the degree of coupling will be higher, since:

• for each RPC interface the calling module needs an understanding of what (not how) the called module does, possibly resulting in less scope for self describing data;

• knowledge of location and usage is likely to be vast and more complex, almost unmanageable for an enterprise of a certain size;

• there is a need for availability of both calling and called modules at execution time; and

• the efficiency of a call may be severely affected by the underlying communication infrastructure.

New developments have started coming on line that may contribute to addressing some of these problems. For example, the use of the Extended Mark-up Language (XML) for the description of shared information and the Web Services framework may help to solve the first two points above, by providing standard mechanisms for the description and discovery of services, making possible system interoperation. However, we argue that for efficiency and functionality reasons a framework oriented to dynamically discovering and requesting unknown services from remote applications is better suited for other type of integration problems — such as the case of the FGIPPC/IAC discussed before — rather than for an enterprise where the operations belong to standard organisational processes.

Demarcating domain boundaries is an important issue in designing a federation. This process must identify the type of dependency that business operations imply, since these are required for
separating out domains so that the dependencies among them are solely informational ones. It is also important to determine the candidates in the enterprise along which domain boundaries may be demarcated. As it may be seen in the next section, in the case of merging enterprises the most likely candidates, at least for an initial demarcation, are the original enterprises themselves. A new dominant TBS enterprise application would also be an important source of enterprise information, and therefore also a clear domain candidate.

4.8 Case Studies

Two examples of real-life federated enterprise integration implementations with the characteristics discussed previously are discussed here. These case studies include the determination of enterprise domains and the middleware infrastructure, and experience with the development of Message Adaptors. These examples illustrate the importance of adhering to the principles of the federation.

The first enterprise is in the Australian energy business, which went through a period of transition due to the deregulation of the energy market in several states of the country. As a result, energy enterprises acquired, took over and/or merged existing businesses, previously under state control. The pressures associated with the full contestability of the newly deregulated market forced these companies to re-evaluate their existing information systems, and to attempt to restructure their IT operations to more flexibly support their business. The second case study is a Health Insurance operator, which customised the implementation of a new off-the-shelf Customer Relationship Manager system to be able to integrate it to their existing IT systems infrastructure.

4.8.1 Case 1: An Australian Energy Company

This Australian energy company originally operated a gas retail business in the state of New South Wales (Australia). In 1993, the State of Victoria (Australia) embarked on the privatization of their electricity business, then under the control of the State Electricity Commission. The company bought into the Victorian electricity business, and therefore found itself with two parallel IT systems infrastructures, one to serve their gas operations in the State of New South Wales, and another to support their electricity operations in the State of Victoria. In both states their IT systems infrastructure comprised numerous applications that satisfied their needs as they stood. In addition to the standard HR, financial, marketing, customer relation-
ship, purchase and billing applications, their systems included applications specific to these energy businesses. These included meter reading, connection management, energy trading and monitoring, and contractors’ management. As part of the evolution of its operations fuelled by the new contestability requirement, new applications — a Call Centre, Energy Trading, Works Management — were under development at the time of the IT architecture revision.

To compete successfully in the emerging deregulated utility marketplace, the company decided on a strategy that would attempt to maximise the existing investment in their information systems. A federated architecture was proposed and accepted, and an appropriate middleware infrastructure, which provided a good functional fit for the federation and a good commercial fit for the organisation, was selected. Some enterprise applications were selected as candidate domains, due to their importance in enterprise operations and the amount of information that they exchanged with other IT systems. These applications were to be brought into the federation in stages by developing the required Message Adaptors, so that subsequent interactions would be guided by the rules of the federation. The first two in the development effort were a GIS (Geographic Information System) application — to be integrated with a CIS+ (Customer Information System) application (running under an OS/390 environment). A third application (SAP) was to follow suit. The initial implementation hinged upon the implementation of two adaptors: a GIS adaptor and an EntireX adaptor.

In January 2000, a review of the project was commissioned, right after the first deliverables were completed. The revision project found some shortcomings with the integration project:

- A separate company had produced the interface specification for the GIS/CIS+ interaction previously, and this did not conform to the interaction model of the federation. These interfaces consisted of seven inquiries and six informational exchanges. In particular, the inquiry exchanges of the specification were based on a client/server model rather than a message-based model. These interactions exposed remote functions of GIS to CIS+, contrary to the tenets of the federation since they induce knowledge of one application in another,

- For certain transactions where GIS was not available the sending application CIS+ would record the need for resynchronisation. This also resulted in an unwanted level of coupling, since the sending application CIS+ is concerned with the receiving GIS application availability.

- Error processing involved sending a message back to the source of the message. This again results in higher-than-required coupling, because it means that the sending application must be concerned with acknowledgements from receiver applications.
• No work was undertaken to establish a corporate format. The adaptors implemented were developed mainly independently of each other, and that resulted in different transformation requirements. For the GIS/CIS+ interface there were no transformation requirements, so no transformation layer was included. For the second adaptor and the third under production at the time of the review, the need for message transformation was implemented merely to translate between the source and destination applications.

• XML metadata was used to describe message structures and set the configuration parameters for both the adaptors. The latter contains the information necessary — such as publishers and subscribers and log information — for the configuration of the adaptor. The former includes all the message structure information, which was passed as a data object to the middleware infrastructure. It was found that the implementations did not follow the model established in the federation framework design, which took into account the envisaged growth of the federation.

• The GIS adaptor was developed to execute in client memory. A main concern with this approach is that client-based processing was perceived (correctly) as less stable than server-based processing. For example, a client workstation is likely to be switched off and rebooted relatively frequently, and this was seen as a possible cause of problems. In addition, responsibility for updates was shown to be difficult, since there were multiple concurrently executing instances to which it could be apportioned.

• Since the development of the adaptor for the OS/390 environment was managed in a more generic way around the EntireX Broker, which handled all communication with the environment, this development was perceived as a possible foundation for the implementation of a common reusable adaptor.

• However, since this adaptor includes the communication between GIS and CIS+, it included the seven interactions that have been designed as request/reply. This decision was taken because of the limitations of the EntireX broker to implement these interactions in any other way. However, the review stresses that in order to improve the interaction model and develop a better interface the original decision should have been resisted [3].

• The approach of mapping a single broker service to handle all the functions of a target application (e.g. the GIS application) was seen also as a shortcoming. If all functions of an application are made available via the same service, this introduces an unnecessary degree of coupling. In contrast, an unbundled service can be reused by several target applications, and a service may be assigned and reassigned as necessary.
• The initial implementations did not consider support for mapping of entity keys between interacting applications. Some interactions did not required key mapping, while others for expediency reasons were implemented at the local level, with a database to map the key pairs.

4.8.2 Case 2: An Australian Health Insurance Provider

In the year 2002, a main Health Insurance provider in Australia decided upon the installation of a new Customer Relationship Management package to manage all customer interaction. Although the initial objective of the strategy was to integrate the new CRM system with an existing mainframe application, a further goal was that this integration effort would reusable across other systems. An important consideration of this development was the need to accommodate both request/reply and send and forget types of interactions. The source of request/reply interactions was the CRM application, with destination in the mainframe system. On the other hand, the mainframe systems were the origin of ‘send and forget’ interactions, destined to the CRM and all other interested systems. The initial analysis established the need for three types of interfaces:

• ‘Send and forget’ interactions, supported by guaranteed delivery capabilities of the middleware of choice. There were 65 of these message-based interactions, generated by:
  – an on-line user carrying out a business transaction on a mainframe system;
  – as a consequence of a request/reply interaction, as described above;
  – as a consequence of a batch job run in the mainframe.

• Request/reply, made possible by the near-real time efficiency of the middleware. There were five of these interactions, initiated by on-line users carrying out a business transaction.

• Scheduled data transfer: an extract, transform and load is used to apply high volume batches to the CRM.

All these interactions have their own end-to-end path between the CRM and the mainframe. In addition, there is an error management system for production support staff to manage manually. All messages emitted or consumed by CRM are in XML, while all messages produced or consumed by the mainframe are in COBOL since this is a legacy application.

4.8.3 Case Studies: Summary

The experience with the case studies illustrates some of the benefits of a federated approach, and how it is possible to implement a simple federation that may be progressively extended. It
is clear from the discussion that in both case studies the federated approach was found to be appropriate for the integration strategy, and that clustering and layering, and the experience with the adaptors, were found to be very valuable for the integration.

The second case study illustrates a slightly different aspect of the federated approach. In this case, a dominant CRM application is to be integrated with a mainframe legacy application. Adopting a federated approach made it possible to reuse the ‘send and forget’ messages with the rest of the organisation, which includes other interested systems. This case exemplifies that the efficiency of the middleware also allows the support of the request/reply interactions that were required by the business process.

However, also some of the problems with the federated approach are also apparent in the case studies. Although the first case study implements a federation at a basic level — Level 0 as described in Section 6.3.1 — the post-implementation analysis highlights the lack of a corporate format as one of the shortcomings of the strategy. This highlights the technical and enterprise political difficulties to achieve the necessary consensus to establish the corporate format. The implementation of the federated approach needs a strong technical and organisational support, since the analysis of the business requirements is highly dependent of the particulars of the enterprise. In addition, despite the federated architecture having been adopted, for different reasons the management of the project was not always closely related to the principles of the federation. Any outsourcing must be carefully orchestrated to ensure that it fits within the federation framework. As the review highlights, any departure results in higher levels of coupling than necessary, difficulties of reuse of the Message Adaptors development, and lack of consistency of the interfaces. In all cases, the report indicates that every mistake has resulted in negative consequences, and the need to return to the original design.

4.9 Summary: Rationale for the Federation

Many attempts in industry, mainly in large enterprises, have attempted to quarantine systems from the effects of other systems; unfortunately, these often have been ad-hoc, employing no clear generic underlying principles. ANSA [9] specifies a useful software model for trading and federation, which recommends a context-relative naming scheme (rather than a global hierarchical one), and traders (components who specialise in knowledge of services provided by other components), and autonomy for members of the federation. However, this model still uses the processing interaction paradigm for communication, with the negative consequences (as argued
earlier) of tighter coupling than the model presented here.

As discussed, an RPC interaction type has an undesirable effect: a given domain resources are utilised at the behest of another domain request, and the requesting domain waiting upon response by another, over which it has little or no control, to progress its work. In addition, often a polling paradigm is needed to implement event-driven information flows in older systems (for example, the warehouse domain polling the order processing domain for new orders) resulting in unnecessary network traffic and waste of processing cycles.

The federated architectural form presented here has the following advantages:

- Inter-domain dependencies, being only informational dependencies, may be satisfied with looser coupling than others types of dependency. Therefore, this architectural form imposes less administrative and maintenance overhead when compared with other types of dependency between domains, and consequently it is easier to operate and maintain.

- The architecture is scalable to enterprises encompassing several remote sites, in different locations or even different countries. Although we have focused on a single enterprise, this architecture is also suitable:
  - to accommodate the interaction of two or more enterprises that co-operate (for example, in a customer-supplier relationship);
  - to provide an effective way forward for merging businesses;
  - to serve as a model for multinationals.

- New applications may be readily incorporated, since the extent and nature of the interaction of an application with the federation are clearly established.

- Legacy applications can be integrated relatively easily into this framework, by having a whole or part of a domain consisting of legacy applications. Limiting inter-domain dependencies to informational dependencies means that the scope of effect of legacy applications can be contained within a domain. Furthermore, in general, message-oriented middleware offers very good connectivity to legacy applications.

- With informational dependencies and MOMs, synchronicity of sender and receiver does not matter. Therefore, messages can be sent as the events occur, or as consolidated information at designated times. Messages can be read as they arrive or at convenient designated times, thus providing a great deal of flexibility in the timing and scheduling of inter-domain communications.
• A great deal of congruence is possible between the architecture on the one hand and the organisational structure, as well as the politics of organisational behaviour on the other. This results in the information architecture being aligned with the business organisation. This should be the preferred paradigm, given that the development and upkeep of enterprise-wide distributed applications usually involves disparate parties at all stages of the software life-cycle. The concept of domains embodies the idea of independent organisational units brought together by cooperation and negotiation.

Within a federation, it is possible to respect the existing organisational structure, reflecting in the application software the present interactions between groups. If domains are properly demarcated, the extra system stress introduced by implementing a coupling stronger than necessary is eliminated. Moving to a federation will probably require the de-coupling by re-engineering of some processes, currently exhibiting too strong a coupling, to reflect the work of the people in the organisation.
Chapter 5

Knowledge-Based Enterprise Integration

This chapter discusses the characteristics of each of the three main aspects of organisational knowledge — data, business rules and business workflows — and the relationships among them of interest to the federated integration strategy. This section also presents the semantics of data with an ontological approach as a particular case of structural business rules, and how rules relate to the structure and execution of enterprise processes. An illustrative example for the AllBooks books wholesaler is included. The FEW workflow model is also introduced, and its operational semantics and transactional characteristics are discussed.

5.1 Ontologies and Data Semantics

In Section 2.3 we discussed the similarities between the problem of database heterogeneity and the general problem of data integration. The main impediment for information exchange in both cases is that data sources and receivers are likely to have different assumptions about the meaning of the data. Since in almost every meaningful exchange the source and destination must use some information of their own to clarify the existing ambiguities in the description of the data, this is not easy to overcome. A data item in a datastore — database, data file — such as salary, with a value 50,000 is a very imprecise description (what is the currency?, is it monthly?, is it yearly?, is it before or after tax?). Typically, the context within which the item is managed provides the required missing knowledge.

For example, consider two independent applications, a gas and electricity customer invoicing applications, billing the same customer for an energy company. The company wants to offer a discount to customers who achieve a certain total level of monthly energy use, so the
applications are now required to share customer usage information. The applications may exhibit heterogeneity along several dimensions, such as customer information their periodicity of billing (e.g. bimonthly vs. quarterly), their discount status (e.g. discount customer vs. standard customer), monetary units (e.g. New Zealand vs. Australian dollars) and precision (e.g. with vs. without fractional part). When the information is exchanged by, or extracted from, these applications with the purpose of coalescing the gas and electricity bills, it is first necessary to resolve these disparities to make sense of the data. Figure 5.1 describes a section of a domain ontology, illustrating the different dimensions along which the two bills may diverge.

From this ontology segment it is possible to establish the attributes necessary to semantically describe the Bill data item. In this way, the ontology provides a template for the contextual information that needs to be elicited for the item. If this information is subsequently made available to the different applications sharing the data, the correct meaning of the term Bill may be determined, and the correct monthly amount calculated. For example:

- if it is known that the gas bill is issued bimonthly and the electricity bill quarterly, it is possible to determine the monthly energy use, and whether the customer is above the minimum required for the discount;

- if the minimum must be calculated before discounts, it is possible to ascertain the discount status of the bill amount in both applications before the calculation is made.
it is also possible to map different currency and formats — scale factor, precision — to properly interpret bill data.

A problem-domain ontology may therefore be used to establish an exhaustive semantic specification for a data item; that is, the information that supports a unique semantic interpretation of the term as it is used in a given environment. For example, the diagram in Figure 5.2 depicts the semantic specification for the concept of a term Bill. This knowledge may be used as the basis for comparison and interpretation, as illustrated above, between data elements belonging to independent applications. If the domains commit to the ontologies — show ontological commitment [62] — by consistently adhering to the terms agreed upon in an ontology, it is possible for an organisation to prescribe a corporate format based on the agreed ontology, and this may then form the basis for applications to map their concepts to the corporate format. Any change to an application or any new application can be then mapped to the standard formats, thus making the changes opaque to the rest of the organisation.

This ontological approach allows the addressing of the three-pronged approach presented in Section 2.3.2, as follows:

- Elicit the information from data stores, applications and users: the ontology provides the description that the analysts can use to extract the information from the data stores.

- The central repository stores the semantic descriptions of the data items, so the data contextual knowledge may be extracted and the necessary local mappings produced.

- The specification provided by the ontology ensures that the data is properly described. It is therefore possible to develop protocols to properly use the data and ensure its integrity.

Note that for this approach to be valid, an ontology of the problem domain must be available, or be constructed for the purpose. It is likely that appropriate established ontologies — such as financial or manufacturing ontologies [7, 98] — already exist; if possible, standard ontologies should be preferred to in-house ones because:

- widely accepted ‘standard’ ontologies are the result of the work by problem domain specialists, and therefore more likely to satisfy desirable established design criteria such as clarity, coherence, and extensibility [62];

- any development will be a better fit with others, which are more likely to be following the same standard ontology;

- this approach will facilitate interoperability in business-to-business interactions with external systems.
However, due to the particular data interpretation of each organisation, it is not always possible to completely follow the semantic descriptions of standard ontologies, and some enterprise-semantics may be required. Hence, the elements to be compared will not necessarily belong to the same ontology, even though they may be part of the same data entity. For example, the term **Monetary Amount** might belong to an **Financial Instruments Ontology** rather than the (AllBooks) **Enterprise Ontology** as illustrated in Figure 5.2. Other terms may be unknown outside the enterprise, or may have a different meaning, and may belong to a private enterprise ontology developed for the purpose. Ontologies particular to an organisation may be developed in-house and linked to standard ones if necessary, as it may be seen in Figure 5.2 [23, 57].

![Figure 5.2: An Ontology Segment for AllBooks Australia](image)

The provision of an ontology-based corporate format provides the necessary support to achieve the data integration objectives discussed in Section 3.1.5, as follows:

- the strategy avoids considering the whole of enterprise data at once; instead, it covers only the data shared within that context (addresses 1 of Section 3.1.5);
- there is a semantic repository for a given context, documenting data semantics and making possible local mappings (addresses 2, 3 and 4);
- reduces dependencies between sources and receivers by providing a central repository of semantic information (addresses 8);
• the amount of information stored by the integration is minimised, since:
  
  – there is only one semantic description of an entity rather than this be repeated every time
    the entity appears in a shared instance (addresses 9 and 11);
  
  – there is reuse of semantic specifications by inheriting them from more general to more
    restricted contexts (addresses 7 and 11);
  
  – there is reuse of semantics specifications by inheritance in semantic types (addresses 11);
  
  – defaults semantics are provided by the repository, although they may overridden (addresses
    9 and 11);
  
  – local changes are controlled, since the semantic scope of each ontological specification is
    its integration context (addresses 7, 9 and 11);

• it makes it possible to implement a supporting flexible data exchange infrastructure that
  uses the publicly available context ontological information.

5.2 Business Rules and Integration contexts

Business rules are declarations of facts that control the way an organisation operates [63]. They
establish conditions for the management and interpretation of enterprise data and processes,
and help define the processes themselves. Business rules are specified in relation to the entities
of interest to the organisation; that is, they express relationships between the concepts relevant
to the enterprise. We discussed in Section 3.3 that structural rules are used to describe spe-
cific, static aspects of the business, and that for a term to be a business term — rather than
a common term — it has to be a part of a business fact, that is, it has to participate in the
specification of at least one business fact. Hence, data entities such as Commission or Total
Sales must be part of an organisational fact to be considered a business term.

We consider here the declaration of the semantics of terms used in an organisation a particular
case of business rule. If we establish that AllBooks International would use AUD as its cur-
rency overall, then all money quantities would be expressed using that currency for messages
exchanged across its top-level federation. This is described by the representation schema in
Figure 5.3.

However, the schema may be also presented as the following collection of rules:

• “Total Sales and Total Dispatched are instances of Financial Information”
5.2: Business Rules and Integration contexts

Figure 5.3: An Ontology Segment

- “Financial Information is a Monetary Amount”
- “Currency is an attribute of Monetary Amount”
- “Currency is a String”
- “The String Currency has the value “AUD””

It is then possible to ascertain the impact that an ontology may have in the rules that describe the data: an enterprise or a domain ontology may be used to describe the meaning of the data, and therefore, establishes structural rules as discussed in Section 3.3.2. More precisely, the ontology relates to the notion of Semantic Rules introduced in Section 3.3.5. This includes a formal explicit description of the business terms (concepts), their attributes (properties) and restrictions on the attributes, such as cardinality, for example.

The close relationship between data semantic rules and the enterprise ontology is made apparent by the similarities between the methods used to represent them. Entity/Relationship, UML (Unified Modeling Language) and Object-Oriented diagrams are most often used in order to represent structural business rules [57, 63, 111, 114]. This convergence is due to the fact that these constructs are able to represent the same relationships between the concepts under discussion: that a concept is-a generalisation of another (more specific) concept and that an entity or concept is an attribute of (is-part-of) another. Both in the case of business rules and ontologies, it is intended to be represent the structure of the data as the organisation understands it.

These structural definitions are only valid for the enterprise context of concern. The BRG report expresses this by stating that a business term “is a word or phrase that has a specific
meaning for a business in some designated context” [63] (pp. 15). Thus, we would like now to extend the definition of a semantic specification for a data item, a rule or a process, as the information that supports the unique semantic interpretation of any of these business terms in a given environment\(^1\).

A semantic specification may persist being inherited, or it may be overridden, by a more restricted environment. It may also be redefined for other environments, as a result of using a different business terminology, a different business process, or a different procedure. Term semantics are thus associated to a given environment, and business rules and process specifications are only valid for a prescribed environment.

These definitions lead to our definition of an integration context, as an enterprise environment in which the semantic specification of the terms in use is established and consistent. In other words, an integration context is an enterprise environment where the validity of the semantic specification of the terms in use within that environment is shared and understood. Consequently, integration contexts are the candidates to be the focus of an integration cycle.

Since integration contexts are environments for which the meaning of a set of terms is established, and we accept different specifications for terms in different contexts, it may be necessary to map terms from one context to another. As mentioned above, the term may be inherited — i.e. the mapping is the identity — or it may be overridden or redefined, in which case an explicit mapping is required. This loose integration of contexts follows an approach closer to ontology mapping (see for example [114]), rather than the provision of a unified ontology to serve the whole of the enterprise which, as discussed in Section 4.1 it would be an unsurmountable task for a large organisation.

5.3 Business Processes and Business Workflows

The following rule illustrates that not all the entities that participate in a business rule correspond to data items in organisational use:

“A new Customer must be created first at Head Office before it can raise a new Order at Sales”.

\(^1\) A similar concept was called context in the COIN project. In this dissertation we use the word context with a different meaning, as seen below.
This rule establishes a relationship between Customer, Head Office and Sales in relation to the raising of a new Order. The terms Head Office and Sales are identifiable as organisational units, and the term Order refers to a corporate entity that has, at least partial, existence in the Sales Department because one of its fragments is stored as such in a Sales database. However, the term New Customer does not describe an existing entity (that would be Customer), not even the result of a function invocation. The term describes the execution of an enterprise process that — according to the rule — is housed and executed at Head Office. New Customer is what we have called before an enterprise workflow — albeit it may comprise a single task — to implement an enterprise process.

Although structural assertions may be appropriately described by an ontology, other types of rules should rather be described in terms of a different type of business rules. A rule specifying an action may be better described using the action assertions and derivations discussed in Section 3.3 [63]. This is because the models used to describe the structure of data are not capable of capturing and documenting all types of rules, such as business rules that deal with the use of information — when, and how it should be used — rather than its structure. Business processes are, thus, closely linked to action assertions since an action assertion determines when a business process is to be executed — ‘if Customer does not exist’, ‘when Order is complete’ — as a consequence of some event. In other words, a business process is the corresponding object in an action assertion with an executive construct, as presented in Section 3.4. Consider again the above example, rewritten to conform to the if ... then syntax:

“If a customer raising a new Order at Sales does not exist, then the New Customer procedure must be first executed at Head Office”

The action assertion captures the event ‘customer does not exist when trying to create an order’, and relates it to the execution of the New Customer process in Head Office.

It is important to make here a distinction between a business process and a business workflow as we use them in this research. A business process is a higher-level construct, a declaration closely linked to the specification of some of the context business rules. In Zachman’s classification [129], this type business process would correspond to the Enterprise Model (row two, conceptual, the owner view). A workflow, in contrast, is a lower level construct: it is the instantiation of a business process as a sequence of tasks, which corresponds naturally to the System Model (row three, logical, designer view). Hence, a workflow implementing a business process may take different forms, as long as the specification of the workflow properly implements the process.
and complies with all the business rules in the set constraining the process. That is, given a set of business rules governing a business process, there may be more than one workflow implementation of the process that satisfies the rules. Consider for example the process of sending the invoice and delivery docket to a customer together with the goods in AllBooks Head Office:

**Business Process:** “Appropriate purchase information documentation must accompany the delivery of goods”

**Business Rules:** “Goods dispatched to a customer must be accompanied by documentation that includes order, invoice and delivery information”

**Workflow:** This process may be implemented by the following, not identical but nevertheless correct, business workflows (informally described here):

- “Sales produces an Order, and sends it to Head Office and Warehouse.” ⇒ “Warehouse produces a delivery docket to dispatch with the goods.” ⇒ “Head Office produces and sends an invoice to Warehouse to be attached to the dispatch.”
- “Sales produces an Order, and sends it to Warehouse.” ⇒ “Warehouse produces a delivery docket and an invoice to be dispatched with the goods.”
- “Sales produces an Order, and sends it to Head Office.” ⇒ “Head Office produces a delivery docket and an invoice, and sends them to Warehouse to be dispatched with the goods.”

The correct implementation will ultimately depend of which applications have responsibility for producing the delivery docket and the invoice.

As it may be seen in the above discussion, although business rules, business processes and business workflows are closely related and each plays a component role in the definition of the other, they are conceptually different [63, 100, 101]. A business rule is a declaration describing possible states of an entity or system, whether the state is desirable, suggested, required or prohibited. It may even be conditional, as seen above. A business rule does not describe the steps to be taken to effect the transition from one state to another, or the steps to be taken to forbid a transition [63]. A business rule refers to business terms, which may include enterprise processes, as in “The NewCustomer() procedure must be executed for any new customer”. A business process is a high level specification of the procedure required to enact an action or actions to achieve a business goal.

On the other hand, a workflow is the procedural implementation of a distinct enterprise process, to implement such a process sequence and properly carry it out. A workflow-aware
specification of the above business rule may be: “If a Customer does not have an approved Credit Limit, then execute \texttt{GetCreditLimit(CustID)} at Head Office”, although as seen before there may be alternative implementations. This rule triggers the execution of the workflow \texttt{GetCreditLimit(CustID)} whenever an order is raised for a non-existing customer. The workflow specification includes specifics such as the function to be executed and the actor that is to execute it. We express this relationship between process and workflow by saying that the business process defines the workflow, and that the workflow implements or instantiates the business process.

The notion of workflow correctness is determined by the compliance of the workflow with the business process defining it and with the rules referring to it. A workflow specifies the tasks (activities) required to enact an enterprise process, their legitimate states, their sequence, their transitions and their commit protocols. The sequencing of the workflow must ensure that states, sequences or transitions do not violate any business rules related to the business process that the workflow implements [101].

In general, the close relationship between tasks, workflows and business rules can be further established by the following statements:

- The execution of a single activity is the simplest workflow. We consider here an activity an indivisible, atomic workflow.

- The specification of a business rule may include the execution of workflows. Conversely, the specification of a workflow is governed by the business rules referring to the business process that defines the workflow.

- A workflow may include — recursively — other workflows to be executed; in particular, a workflow may include atomic workflows (activities) as stated above.

- The completion of a workflow within another workflow is perceived as an atomic unit of work.

The following section discusses business workflows in the context of enterprise integration. Since it is clear in this context, we use the expressions workflows, business flows and business workflows as equivalent.
5.4 A Step-by-Step Business Rules Methodology

To be of interest to the integration effort, an enterprise term should be a part of an integration context business rule. That includes all business terms defining data structure — data semantic rules — and processes known to that context. Conversely, the behaviour of an enterprise process is constrained by a rule or a set of action business rules valid for that context. These rules refer to business terms related to the same business process.

Unfortunately, for a large enterprise the set of business rules present and active at one given time is very large and complex. The number and variety of running applications, each with their own set of rules, results in an inherent heterogeneity of organisational processes. Thus, the problem presents the same characteristics and difficulties than data integration. In fact, the discussion of the previous section indicates that since the semantics of a business term may be expressed by business rules, this latter problem includes the former. Eliciting data semantics is a very difficult task; therefore, elicitation and representation of business rules therefore is at least as, and probably more, complex. This is usually the work of knowledge domain experts, with a deep understanding of the organisation and the way it operates.

Several methodologies — including graphical representations, Decision Tables, Semi-Formal English, Rules Templates [100, 101] — have been proven to be successful for eliciting and expressing business rules. However, all these methodologies are based on the assumption that domain experts are in charge of the analysis and, therefore that the number of rules under consideration must be manageable for the experts. Hence, regardless of the methodology used, the previous section indicates that to be able to successfully elicit and express the business rules, the size of the problem must be contained, and the analysis approached in a systematic manner.

Business rules determine the structure of the data and the reaction to business events. As a reaction to events, a rule may cause other rules or processes to be triggered. As discussed in Section 3.3.5, whether a term is data or a process is often determined by the analysis and design phase. Hence, unless the rules are written following a specific convention, this may be difficult to establish from the discourse. A convention might be adopted, such as the one adopted for this dissertation of preceding the process with the word execute (. . . then execute GetCreditLimit(CustID)”). This suggests that the enterprise adopts a standard convention to make this distinction clear.

\footnote{Which particular method is used by the enterprise to represent these rules is not relevant to our discussion; thus, it is not a focus of this dissertation.}
We establish here a stepwise sequence:

1. Reduce the problem size by the careful determination of integration contexts. Each context should comprise a relatively small number of domains — probably less than twenty — where data semantics, rules and processes are to be shared.

2. Gather all business rules referring to the terms used in the integration context. These include all data semantics and process related rules, and actors.

3. Determine whether each term is a common term or a business term.

4. For each common term, confirm that there is a common understanding of the term (e.g. the spelling of a city name).

5. Determine the data related terms in use in each integration context (that is, terms referring to data shared by the domains).

6. If a data related business term, collect and document semantic information related to the term. Augment the term information with available metadata. Establish the term semantic specification.

7. Model the facts related to the data semantics terms above. When possible, link them directly to a standard ontology. If not possible, include them in a custom made ontology and link this ontology to a standard ontology as shown in Figure 5.2.

8. Extend the model by considering the terms not directly related to data semantics. For each term, confirm that the term refers to a procedure. For each procedure, define the workflow that implements this procedure.

9. Rewrite the rules in a standard way.

This generic methodology will be revisited and specialised into data and process oriented methodologies in Sections 7.1 and 7.3, within the framework of the Federation.

The next section further clarifies the relationship between business rules, business processes and business workflows.
5.5 Characteristics of Enterprise Workflows

A business workflow has been described as an implementation that tries to minimise the mismatch between business processing and the corresponding information systems [12]. This approach has been successful in modeling and processing information flows in areas such as manufacturing and document processing. As seen in Chapter 3, in general, Workflow Management Systems (WFMS) have been used to specify and execute workflows that support process execution semantics such as sequencing, looping, alternatives and synchronisation. Although much of the effort has been directed to collaborative processing, the reality of many organisations is that their business workflows may be classified as administrative workflows [52], that is, workflows supporting the appropriate sequencing of administrative tasks and flow of information between nodes, but not requiring remote processing for the completion of an individual activity within the workflow.

As a result of the integration, nodes initially autonomous — processing stations — may be required to cooperate on a business workflow in the manner just described. Since processing stations precede the definition of the workflows, we argue that, similarly to Exotica and INCA, it is possible to circumscribe all processing to the nodes, which already possess the data and the applications necessary to execute the required activities [5, 11]. There is a high likelihood of data shared between collaborating nodes, and hence it will be necessary to propagate data-updating events via the workflow links. Although it is possible that no data flows through a link between nodes — e.g. there is no shared data — the flow is more often determined by the data sharing patterns between the nodes. Thus, when an activity executed within a node results in a data update, it is necessary to propagate the update to all nodes possessing a fragment of the data.

The case studies in Section 4.8 — and also the case discussed in Section 5.7.2 — illustrate the following characteristics of business workflows of large enterprises:

- These workflows are ‘bottom-up’ (i.e. based on existing processing applications, or resulting from the introduction of new dominant application) administrative workflows. These are simpler than processing collaborations. This is shown by fewer links between nodes, and simpler rules governing the flow of control and information.

- These flows are based on information-oriented interactions, with all the processing already taking place within the nodes.

- The vast majority of interactions may be resolved by asynchronous exchanges.
chronous exchange is necessary, this may be implemented as a simple message sequence in which the sender blocks waiting for the reply message. That is, only the simplest one-request/one-reply processing dependencies need to be enabled between the nodes. This rules out ‘conversations’ between nodes where a session and its session state are maintained, but it still allows for simple, connectionless request/reply interactions.

- In enterprise systems, some applications tend to dominate as sources of information, and hence spawn workflows. Transactional Based Systems (TBS) such as a Customer Relationship Manager or an Energy Trading application continuously generate events, and hence a quantum of information flow at the occurrence of each event. Other systems produce a continuous flow of information by reading meters, valves or other devices. The flow is always out from these applications towards other enterprise systems recipients of the information.

- As it may be seen from the case studies and Figures 1.2 and 1.3, workflows consist of only a very few steps. Preponderantly, a workflow consists of only two steps with one node producing information for other nodes to consume only to keep their data synchronised.

**Workflow Suitability Principles**

The uncertainties that have resulted from many extensions to workflow modelling motivated Barros et al [12] to include a framework of suitability principles, as follows:

**Organisational Embedding Principle:** relates workflow models to organisational elements;

**Scenario Validation Principle:** requires that business process scenarios to be declared, so workflow validation is well understood;

**Service Information Hiding Principle:** which decouples any business service requests from any knowledge of the process that satisfies the request;

**Cognitive Sufficiency Principle:** incorporates into one workflow model extra aspects, such as messaging, interaction points, and temporal aspects;

**Execution Resilience Principle:** requires the explicit management of errors, which should include commit points, rollbacks and rollforwards.

These principles provide a target framework and a set of goals for workflow models to achieve. They are highly desirable characteristics of a model, and they have been presented and empirically validated in the area of government administration in the state of Queensland, Australia [13]. This work illustrates how workflows may improve business transaction processing, and what are the existing core concepts that contribute to the implementation of workflow solutions.
The case studies presented show business workflow models covered by the classification of administrative workflows, and show a close similarity to the model we are presenting here. They are characterised by the need for the coordination of tasks rather than collaborative processing, and the need to correctly manage the flow of information between nodes rather than for process collaboration by the provision of services to each other. In addition, they are supported by ‘black boxes’ of functionality that are naturally mapped to business services. Nonetheless, from the behavioural point of view these low-coupling enterprise workflows follow the same established basic modeling constructs than other workflows, and the same principles and basic coordinating constructs apply to them.

The following section introduces *FEW* (*F*ederated *E*nterprise *W*orkflow), a workflow model appropriate for a loosely coupled, federated integrated enterprise.

### 5.6 *FEW*: A Message-Based Publish-and-Subscribe Workflow Model

Similarly to other workflow models [5, 12, 13, 52, 87, 76], in *FEW* a business workflow is represented by a directed connected graph, where nodes represent activities, and directed edges the transitions between them (See Figure 5.4). In our approach all processing is concentrated within the nodes — ultimately federation domains — which are responsible for executing the activities that comprise the workflow. Although all the activities are concentrated in the nodes, the relationship between activities and nodes is not one to one, but many to one. That is, the same node may be visited several times as part of a workflow, each time the workflow executing a single activity housed in the node.

Nodes are assumed to have the processing and storage capabilities to execute the required tasks, and to be able to permanently store and manage their local versions of the data. In this sense, the *FEW* workflow model is completely distributed, with nodes that maintain their autonomy but agree to cooperate with other nodes on the completion of specific business flows, (within the structure of a federation, as we shall see in Chapter 7). No processing or logic capability is allocated to a workflow infrastructure or engine. There are, however, federal data structures to support the workflow, such as the already mentioned Message Adaptors, and the Data Integration Service (DIS) introduced in Section 6.3.

To the outside, node activities seem atomic, even though they may consist of several tasks or
the execution of a local workflow. When all the nodes in a flow have been visited and all the activities successfully completed, the business flow is complete and can commit. A business flow is specified by traversing this directed graph as the activities are completed. Thus, a workflow graph consists of a lattice with a single origin of discrete business tasks concentrated on the nodes, each contributing to an overall business procedure. In \textit{FEW}, a workflow instance is always spawned by the execution of an activity in one of the nodes — the originating or start node — initiated by the execution of a business task within the node. An end node is a node where the flow terminates, that is, there is no outgoing message to progress the flow. In general, a \textit{FEW} workflow may have several end nodes. Under certain circumstances, for convenience a workflow may be coerced to having a single end node (See section 5.7.6).

The directed edges in the workflow indicate the transfer of control from one node to all its successors in the graph as events, such as the completion of an activity, occur within the node. In \textit{FEW}, this is implemented by a publish-and-subscribe mechanism. Each connection outgoing off a node represents a message generated that the node publishes when a workflow-related event occurs. This ‘send and forget’ semantics completely decouples the sending node from the receiving nodes, provided that the message delivery is guaranteed.
Nodes successors to a node in the workflow graph subscribe to the predecessor’s message to obtain the information they need to execute the next step in the flow. There might be more than one node successor to a given node, and all of them subscribe to the message to be part of the workflow. To be precise, this subscription is what determines the existence of a link from a node to any of its successors. The same message may be subscribed to by many successor nodes, belonging to the same or different workflows. This in particular means that the same message may be reused as part of several workflows. Each workflow has a unique type that identifies it — e.g. New Order — within a given enterprise context, and each instance of an executing workflow has a unique ID used to distinguish between different executions (instances) of the same workflow. (More on this in Section 5.7.6). Each message has a unique type — e.g. Insert New Customer — within a workflow to identify it within that workflow. The combination of workflow type and workflow instance ID, and message type constitutes a unique identifier of the message in that workflow instance within that context.

Since shared data fragments often need to be updated by a workflow, update responsibility for a fragment must be allocated to one and only one node of the workflow — the primary node — to avoid data corruption. This results in circumstances when a given node may not have authority over a data fragment that requires updating; hence, an appropriate message requesting the fragment update must be published. In these cases, the primary node for the fragment is made a successor to the requesting node by subscribing to the request message.

Each node declares an interface that includes the specification of the messages it will produce as reaction to events. Sending and receiving messages is a particular type of node event. Since the flow of data and control follows a publish-and-subscribe mechanism, the generating node has no knowledge or control over which nodes subscribe to its outgoing messages. As in INCA, each subscribing node knows how to react to an incoming message, according to their own rules depending on the type of message and the values of the data included in the message. Hence, any node interested may subscribe to a message, whether or not it was originally a part of the workflow. In this way, new nodes are allowed to become part of a workflow, even if they were not originally considered. These nodes are most likely to be terminal nodes, where the message flow terminates for that workflow branch, i.e. there is an incoming message related to a remote event, but not outgoing ones. Similarly, a terminal node is allowed to unilaterally unsubscribe to a message and disconnect from a particular flow. This situation, with nodes subscribing or unsubscribing to information flow from an authoritative source but not requiring to further progress the flow is overwhelmingly the most common in our case studies.
In contrast, if a new node to be incorporated into a workflow is not a terminal node — i.e. there is a need for the workflow to progress further — the new node must be formally incorporated as part of a different, extended workflow. Thus, only terminal nodes are allowed to dynamically connect to, and disconnect from, a workflow.

### 5.6.1 Activity Coordination

As different from most other workflow models [5, 12, 13, 52, 76, 87], in FEW there is no logic directly associated with a transition link. Upon receiving an incoming message, a node executes to completion the activity associated to the message, determines the parameter list and constructs and publishes the output message that is required to produce. Subscribers consume the message. Since the edges connecting nodes have no logic directly associated with them, no transition conditions are evaluated to determine the possible successors that are to receive control in the next step. Thus, all subscribers to an output message are the successors and receive the output message. In this way, the necessary coordination logic is distributed across the nodes, rather than be associated with the links\(^3\). Since the on-flow logic is not controlled by the publishing node, and the logic as to how to react to an incoming message is contained within the successor nodes, messages are reused as needed by the process. FEW provides a flexibility not afforded by any other model, since new nodes may be included in an established workflow without any modification to the already existing nodes.

In FEW, a workflow instance is always initiated by the execution of a business task within the originating node. As a result of this event, the new workflow instance receives a unique identifier, and the input required by the triggering task is the input for the workflow to start. Typically, the task introduces new, or changes the value of some, data within the node, and the data update is propagated to all its successors in the workflow graph as part of a published message. Since the reaction to messages is contained within the subscribers, a successor node is free to ignore the data payload, and act only on the knowledge that an event has occurred. It is also possible that the result of the task produces no data update, or that only local, non-shared data is updated, so there might be no data payload in the outgoing message. That is, a data-less message is published, propagating the event but not a data update. This results in a transfer of control, but no data, to the next steps in the workflow. This loose approach is suitable for enterprise workflows as described at the beginning of this section.

---

\(^3\)This behaviour may be naturally modeled with Petri nets, as it may be seen in [117]. This, however, is outside of the scope of this research.
In some cases it may be necessary to suspend a partially completed activity, waiting for a request for additional information, or authority to proceed with it. In this case, the originating node must request the task to be executed remotely, and wait for the result of this execution to proceed. This is the case of simple request/reply stated above.

The FEW model supports different activity sequencing strategies:

- An outgoing message is produced at any time before an activity completes — \textit{anytime-start semantics} — because the output of the completed activity is not required downstream. This may happen at any stage of the execution of the activity. In particular, an output message might be produced at the start — \textit{start-start semantics} [76] — of the preceding activity. This increases concurrency, since it makes possible the parallel execution of an activity and its successors. As well, this allows for pausing a partially completed activity while progressing the flow (such as the \textit{Please} message discussed in Section 5.6.2), since the activity does not need to terminate to send an outgoing message.

- The preceding activity needs to complete — \textit{end-start semantics} [76] — before its successors can start. In this case, almost without exception, the outgoing message will contain a data payload signaling a data update; at least some of the successor activities need the data provided by the message to start execution.

Upon receiving a message with data payload, a node will interpret the incoming message and map the message data contents to the structures of its own applications. This has been called a \textit{map} in [87], but in our model is a consequence of the federal management of corporate data (discussed fully in Chapter 6) and usually results in a local data update. The node possesses the necessary knowledge to act upon the message and execute the corresponding activity and, if it is not terminal for the workflow, it will in turn publish an outgoing message to its successors.

In addition, FEW supports several additional coordination primitives:

- Branching: a natural consequence of the publish-and-subscribe mechanism. All successors to a node in the graph are assumed to be subscribers to the outgoing message, so branching is naturally supported by our model. Several logical branching operations are supported:
  - Sequence: the normal operation supported by the publish-and-subscribe. All the successors receive the message from the predecessor node and act upon it.
Joining (AND semantics): a node may receive more than one message related to a given workflow. The node itself is responsible for the coordination and management of the incoming messages. Joining also admits several logical operations, according to the messages required to start a successor activity: AND Join (all predecessors messages should be received), Exclusive Join (only one predecessor message is required), OR Join (a subset of messages is required).

Contingency (Exclusive Alternative, XOR semantics): only one of two successors $N_1$ or $N_2$ is to execute the successor activity, implemented by the logic $N_1 \otimes N_2 \iff (\neg N_1 \Rightarrow N_2)$. That is, the activity is allocated to node $N_1$. $N_2$ is successor of $N_1$. If $N_1$ does not execute the activity (e.g. the activity fails), it passes control to $N_2$ to execute it. This may be extended naturally to an arbitrary number of successors.

Alternative (Non-Exclusive Alternative, OR semantics): either one of two successors $N_1$ or $N_2$ is to execute the successor activity (OR operation), implemented by the logic $N_1 \parallel N_2 \iff (N_1 \Rightarrow \neg N_2)$. That is, the successors are node $N_1$ and $N_2$, which have the choice of executing the activity. If either node does not execute the activity, it passes control to the other node to execute it, in case it has not done so. The receiving node may choose to ignore the message if it has executed the activity already. This may be extended naturally to an arbitrary number of successors.

Looping: As discussed in [87], support for arbitrary loops may result in ambiguous situations, so our model does not support them in general. Limited looping is supported, a) a workflow may visit the same node several times, although every time a distinct individual activity is executed; b) looping restricted to cycles occurring within a node, or controlled by a node, where the controlling node spawns individual workflows until a certain exit condition is met.

The following are additional FEW constructs:

- **Sub-workflow**: a sub-workflow is a subset of a workflow that itself satisfies the definition of a workflow.
- **Workflow segment**: any connected sub-graph of a workflow graph.
- **Join node**: is a node with more than one incoming link of a given workflow.
- **Checkpoint node** (or simple checkpoint): is a node $N$ such that all other nodes are either predecessors or successors of $N$. This implies that the node $N$ collects all the predecessor links of the workflow graph, and generates a sub-workflow originating in the node. Thus, $N$ is the end node of the predecessor workflow and the start node of the successor workflow.
In order to enforce some correctness criteria for transactional behaviour (see Section 5.7.4) we introduce the following definition:

A relay link pair is created by a node subscribing to a message only to relay it by publishing it unchanged. This is usually with the intention of replacing or duplicating the original link. Figure 5.5 depicts how a link is related to a relay link pair.

Note that there is no need to in any way change the operational behaviour of FEW, since a relay is implemented by two new subscriptions, perhaps unsubscribing to one message if the link is replaced.

The model presented here includes all the constructs specified by the Workflow Management Coalition [69], so it conforms with the established standard for Workflow systems. This workflow model complies strictly with the low coupling principles enunciated in Section 4.4.1 and the discussion of Section 4.4.3, reflecting the principles on which a loosely integrated enterprise is based. These semantics are defined to ensure that business workflows of such an enterprise are properly represented by the model.

In addition, the FEW model conforms to the principles enunciated in [12, 13]. Firstly (Organisational Embedding Principle), the model directly relates nodes to organisational elements and links to information exchanges between them. This makes possible to locate processes, actors, data repositories and data exchanges and naturally relate them to the workflow. Secondly (Scenario Validation Principle), our model responds closely to the type of scenarios illustrated
by the case studies, and provides a set of modelling constructs appropriate for those scenarios. Thirdly (Cognitive Sufficiency Principle), in our model a workflow may be enacted without any extra assumption about the semantics of the business process. Lastly (Execution Resilience Principle), transactional behaviour and non-fatal error recovery are included in our model. (See Section 5.7.)

However, there are also several significant differences between the models discussed in [5, 12, 13, 52, 76, 87], and the FEW specification:

- Traditional models require nodes to be aware of other nodes. For example, in [13] a node processing an Application object should have views sought from Stakeholders, embedding application-type logic within the node. This is contrary to the principle of low coupling advocated in this dissertation, and it severely restricts the possibility of message reuse. In our model, a New Application workflow would be started, and acted upon by all nodes according to their own rules. This polymorphic behaviour has the same advantages as polymorphism has in object-oriented programming, making possible overloading the message and hence allowing its reuse.

- Traditional models do not accept nodes flexibly becoming part of, or detaching themselves from, a workflow. Support for this behaviour greatly simplifies the analysis, since it is not necessary to consider all the nodes in the first instance, but it is possible to include a core number of nodes in a workflow and incorporate terminal nodes later. As well, we argue that this flexibility may help overcome the resistance to enterprise integration strategies, because terminal nodes may be added — or may even ‘add themselves’ — when they are ready to join the workflow.

- The first two points above about traditional workflows are a violation of what we call here the The Need to Know Principle: the less a node needs to know about other nodes, the lower the coupling between them and the higher the possibility of message reuse. That is, the FEW model not only satisfies the Service Information Hiding principle — which hides service implementation details — but it further decouples a request from the service provider by hiding from the sender any knowledge of the receivers of a message.

- A FEW workflow is ultimately related to an integration context, and therefore it only needs to deal with a reduced number of actors, data items, rules and procedures.

- Existing models support synchronous, conversational interactions. This exposes remote functionality to the requesting node, the sending node being aware of the availability or otherwise
of the receiving node for an extended period of time. This type of interactions result in an unwanted level of coupling, with the negatives consequences discussed in Section 4.4.

- Only information flows through the links, so interactions are informational interactions. The vast majority of these are asynchronous, non-available interactions, with the positive consequences discussed. Simple request/reply interactions are the exception.

- Current models are often concerned with processing entities at different levels of detail and abstraction, and the depiction of objects usually not present in an enterprise model, such as database and file objects. They may even model interaction points with the workflow [12, 13] — including human dialogues — which introduce a large number of entities into the workflows making it harder to manage complexity.

- The level of granularity of the processing actors is much smaller that then one we consider here. They often model individual task-oriented applications, such as Application Lodgment and Investigation, increasing the number of number of entities to be considered in the analysis at any one time. Instead, the large organisational entities that are the focus of this research are a few, and they are easily identifiable as organisational actors and processing units.

In summary, even though it shares some constructs with other models, there are important differences between traditional workflow systems and the FEW model. The FEW model:

- Is distributed rather than running under central control.

- Is based on the exchange of information rather that services (services are encapsulated within the nodes).

- Is based exclusively on the exchange of messages among processing nodes.

- Completely decouples nodes by decoupling requests and replies.

- Allows the dynamic incorporation and detaching of nodes.

- Maps nodes responsible for processing to existing processing organisational units.

- Respects the autonomy and privacy of each processing node.

- Is based on a publish-and-subscribe mechanism.

- Does not associate any logic to the connections links.

- Allows message reuse.
• Allows no seepage of business rules between nodes and the workflow space.

• Supports embedded workflows within a workflow.

• Allows looping under the control of a node.

Table 5.1 summarises.

<table>
<thead>
<tr>
<th>WF Attributes</th>
<th>INCA</th>
<th>WSFL</th>
<th>Kuo et al</th>
<th>Exotica</th>
<th>FEW</th>
</tr>
</thead>
<tbody>
<tr>
<td>Distributed</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Node autonomy</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Process oriented</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>N</td>
</tr>
<tr>
<td>Processing only within nodes</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Message-based</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Asynchronous processing</td>
<td>Y</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Support start-start semantics</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Cycles/Loops</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Joins</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Conditionals</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Alternatives</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Transactional behaviour</td>
<td>N</td>
<td>N</td>
<td>Y</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Message reuse</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Logic associated to links</td>
<td>N</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
<tr>
<td>Coupling</td>
<td>L</td>
<td>H</td>
<td>H</td>
<td>L</td>
<td>L</td>
</tr>
<tr>
<td>Dynamic connections</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>N</td>
<td>Y</td>
</tr>
<tr>
<td>Support for embedded WF</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
<td>Y</td>
</tr>
</tbody>
</table>

5.6.2 Operational Semantics

The following describes the operational semantics of our workflow model. The simplicity of the FEW model results in a greatly simplified specification of workflow semantics:

1. As a consequence of an event within an originating node — related to a node application — a new workflow is spawned.

2. If as a result of a workflow activity within a node there is no requirement for a data update, a message with no payload is prepared; go to step 6.

3. If a workflow activity requires a data update, and the application has no update authority over the affected data, go to step 4. Otherwise, the application executes the update and prepares a message with an appropriate payload; go to step 6.
4. At any time, a node that requires a data update but for which the corresponding application has no authority over the data, publishes a *Please* request message. Since the data update authority may affect more than one node, there might be more than one possible reply. However, as discussed, only one the primary node is given authority over each fragment of the data. The originating node subscribes to the replies, and acts accordingly: if all the responses to the request are positive, the node assembles the responses and executes the activity.

5. Upon receiving a *Please* request as above, a primary node publishes its data update as an outgoing message. All subscribers — not only the originating node — receive and act upon the message updating their data fragments.

6. The workflow node progresses the workflow by publishing an outgoing message.

7. The subscriber nodes interpret the message and its payload, and acting according to their own (application) business rules execute the corresponding tasks. When and if appropriate, each node publishes an outgoing message.

8. A join node — such as the originating node above — waits to receive the required messages before processing them. Internally, the node might require that all or only some of the incoming messages arrive to trigger the event, but this logic (AND) is hidden within the node.

9. The first of two nodes involved in XOR semantics must be subscribed to the link message. The node makes a decision as to whether to execute the task. If not, it publishes an outgoing message for the other node/s involved to execute the task.

10. A loop is only permitted by the model when the iteration is under the control of a node. In this case, the controlling node repeats a message sequence until the terminating condition of the loop is satisfied.

5.7 Transactional Workflow Behaviour in *FEW*

5.7.1 Transient Inconsistencies Revisited

The notion of transactions allows database applications to concurrently operate on data and properly manage failures without having to deal with concurrency and reliability issues directly. The transaction manager of the database ensures that operations on the data are correct by providing an execution environment in which some correctness criteria — such as the ACID properties — are supported. Distributed database environments are more complex, since there
is a need for coordination of the different transaction management systems of the component databases that manage the local sub-transactions. This resulted in the implementation of distributed commit protocols such as the 2-Phase Commit (2-PC) and 3-Phase Commit (3-PC) protocols. In order to enhance the sequence and coordination capabilities of the models, one of the main goals of modern workflow research is to furnish workflows with transactional aspects similar to the ACID properties. In this way, it is possible to achieve, at least to a certain extent, the degree of correctness of database transactions, including commit protocols, rollbacks and rollforwards.

If transactions were dispatched and committed in a serial manner, the ACID properties are relatively easy to support. However, due to the number of transactions executing at any given time this is unacceptably inefficient for the DBMS system, and therefore operations of different transactions are usually allowed to interleave. That is, this is a direct result of the requirement of a certain level of throughput and response time, rather than a correctness requirement. As a consequence, most systems manage concurrency by enforcing the serialisability criterion, aiming to ensure that interleaved operations produce the same effect on the database as a serial execution [38, 75]. This is traditionally implemented by mechanisms such as locking or time-stamping.

However, all the theory and practice of Database concurrency control is based on short-lived transactions. This is so to avoid undesirable effects such as locking records for an extended period (when locking), or having to abort and restart a high number of transactions (when time-stamping). Hence, the underlying assumption of concurrency control strategies is that the time between the start and the commit of a transaction is very short. In contrast, workflow activities can be of very long duration and, hence, the same techniques have limited application. To better manage long-lived transactions, for example, the work of Kuo et al [82, 83] relaxes the commit semantics of a workflow. In their work, the workflow model is modified to accept critical and non-critical tasks, with only critical tasks requiring aborting the workflow in case of failure. In INCA, a saga approach [37] is suggested for concurrency control within a processing station, but nothing definitive is developed for executions spanning station boundaries.

5.7.2 Quasi-Consistency

One of the fundamental tenets of the FEW model is that between nodes there may exist only informational dependencies. In Section 4.4.4, we established that this is only possible if transient inconsistencies that occur as a consequence of inter-domain exchanges may be tolerated by the business process, and discussed the conditions in which the business process is able to do
5.7: Transactional Workflow Behaviour in FEW

so. However, the transient inconsistency period should be reasonably short, because an inconsistency that may be tolerated for a period of seconds or minutes is likely to be unacceptable for a period of a month or a year. Thus, the FEW model refines this specification, as follows:

A transient inconsistency may be tolerated if:

1. it is possible to ensure that an inquiry, processing, or update related to the inconsistency does not occur in the critical period; or

2. it is possible to reduce the likelihood of the above occurrence to near-zero; and

3. the consequences of such an occurrence during the period of transient inconsistency are still tolerable for the business process.

In the FEW model, this new condition (condition 2) has important implications for ensuring the correctness of enterprise workflows. In the context of transactional workflow behaviour the first condition above implies that concurrency may be managed to avoid the inconsistency. However, the second and third conditions indicate that in some circumstances it is possible to relax the strict condition of atomicity and isolation of inter-node flows, and allow an activity to read dirty or inconsistent data. In FEW, such dirty operations occur when an event in node A has triggered a message to which node B is subscribed, but before the message is received and processed by the receiving local application, the affected data fragment at B is read by another process. As discussed in Section 4.4.4 nodes — i.e. federation domains — have been determined such these inconsistencies can be tolerated. The FEW model assumes that all the links of the workflow are within the acceptable transient inconsistency period. That is, the transition between activities lies within the acceptable inconsistency limits. Although this time lag would not be acceptable for a database, it is a most common occurrence in enterprise systems when there is overlapping data across already existing autonomous applications, since in a non-integrated enterprise there is no way to control the updates. It is precisely this lack of consistency that our approach to enterprise integration intends to ameliorate.

The lack of consistency of enterprise data is a natural consequence of the absence of update coordination stemming from applications being brought in without a concerted strategy. Although in the FEW case there is some propagation latency of the information across the nodes of the flow, our experience with the case studies shows this delay to be quite short (near-zero latency, typically less than a second). This will surely have a significant impact on the decision by the business analysis to decide whether a transient inconsistency may be tolerated by the business process. The near-zero latency condition indicates that the likelihood of an inconsis-
5.7: Transactional Workflow Behaviour in FEW

tency actually occurring is very low and therefore, if the consequences are not catastrophic, that the inconsistency may be acceptable. We call this *quasi-consistency*.

**Quasi-consistency**: If as a consequence of an update on a data segment there exists a transient inconsistency with the following characteristics:

- the propagation time of the update is close to zero;
- the consequences of a dirty operation on the data are tolerable for the business process.

we say that we are in the presence of a quasi-consistency.

The difference between consistency, quasi-consistency and transient inconsistency is very significant. For example, the analysis might indicate that a wrong update as a consequence of a quasi-inconsistency should be treated as an error by the process, and that this should be fixed manually. This might be cumbersome, but if the likelihood of it occurring is low (quasi-consistency), it should still be acceptable. The same situation might be unacceptable in the case of a transient inconsistency of long duration, since the number of dirty operations may become too high during the inconsistent period. If an activity within a node results in a need for actual transactional behaviour, a database-like scheme — data locking or time-stamping — may be used to control the activities when the subscriber nodes are reading or updating affected data. As discussed, this is highly undesirable in long transactions such as these, so they should be reduced to a minimum. Section 5.7.6 revisits this issue and discusses the *FEW* transactional behaviour.

Since the nodes are established so that there is only information dependencies between them and transient inconsistencies are tolerated, all updates are quasi-consistent — i.e. the transient inconsistency possesses the characteristics of the quasi-consistency definition — if the infrastructure is efficient enough. The only condition to be imposed to the exchange between the two nodes is that the update is eventually correctly performed, regardless of the interleaving of tasks at each node.

### 5.7.3 A Case of Quasi-consistency

The previous section makes clear that the definitions of “near-zero” and “tolerable to the business process” depend on the analysis of the process itself, and it may change according to the circumstances. In a recent case, some of the systems and corresponding interfaces — more than fifty — of a major Australian bank, had to be duplicated and kept synchronised remotely
due to regulatory reasons. The architectural analysis contemplated the problems associated with the project, and established that:

- the information flow was exclusively data, that is, there was no remote processing required;
- the information flow was only one-way, in a source/receiver configuration with the exclusive intention to keep the receiver applications synchronised;
- the information flow between applications is not one to one, but in several cases is one to many;
- there were several interfacing methods, such as direct database reading and writing, direct writing to target application space, file transfers, proprietary methods, and different types of middleware, which had to be replicated;
- even for data flowing the same way, some interfaces worked following a pull model (initiated by the interface), while others followed a push model (initiated by the application), which also had to be replicated;
- not all the duplicated systems had to keep the same running interfaces; thus some interfaces would disappear on the replicated systems;
- due to the regulations, the replicated information was to be no more than thirty seconds out of synchronisation from the source.

As discussed in Chapter 4, these conditions are highly suitable for a federated solution as outlined; a) the one way, one to many, data only flow matches the publish and subscribe model; b) the federated approach provides a consistent interfacing method rather than the many different existing ones; c) the federation is based on a single push model; and d) following the federated model interfaces may easily disappear by unsubscribing.

However promising it may appear at face value in this case, the thirty seconds condition is determinant for the adoption of a federated solution. In this case an external agent — the regulatory authority — clearly established the quasi-consistency interval to be less than thirty seconds. If the middleware of choice for implementation is capable of delivering, and the applications are capable of processing, the messages within the latency determined by the authority, a federated solution would be advisable. Otherwise, although highly suitable from the model point of view, a federated solution could not be adopted.
5.7.4 Transactional Behaviour in FEW

In FEW, tasks are standard for the interoperating applications, and hence are to be successful provided that they have been properly specified and that the conditions for their execution have been satisfied. Nonetheless, there may be failures due to lack of proper data or system unavailability, so the workflow should include termination and error recovery protocols. As presented in Chapter 3, the discussions in Kuo et al [82, 83] define the task classes forcible, compensatable and undo-not-required:

- A workflow may consist of critical activities and non-critical activities. An activity is critical if its failure forces the abort of the workflow, whereas if a non-critical activity fails the workflow does not need to abort.
- Undo-not-required activities do not require undo if a workflow aborts.
- An activity is forcible if it can be guaranteed that it will eventually commit once it has started.
- An activity is compensatable when its effects can be undone after it has committed.

According to these definitions, atomicity correctness specifications depend exclusively on the type of tasks [82, 83]:

- Commit correctness: a workflow may commit if all critical tasks also commit.
- Abort correctness: a workflow may abort if all undo-required activities abort.

The vast majority of tasks in enterprise workflows belong to these categories, because the applications in charge of the tasks normally include robust protocols for task commitment, and for dealing with compensation when tasks fail in the normal operation of the application. This in particular implies that most tasks are forcible if the correct parameters of the execution are provided, and that enterprise tasks can be undone by compensation by the application that supports it.

5.7.5 Transactional Protocols in FEW

The FEW model takes these attributes of enterprise activities into account to establish workflow inter-node protocols. The following definitions and rules related to FEW constructs are required for determining transactional behaviour:

Data: Definitions and rules on data links.
• An application with update authority over a data fragment is called the primary application for the fragment (See Section 6.3 for a formal definition). Therefore, following the definition in Section 5.6.2 the primary application is housed in the primary node for the fragment.

• An update of shared data has an outgoing associated message type, related to the type of update (Create New, Update, Delete) and the data fragment (e.g. Australian Customer). That is, if a shared data fragment has been updated a message of a distinct type is published.

• An activity must have authority to update a shared data fragment; hence, an incoming message that requires a data update must be received and acted upon, and then propagated by, the primary application.

• Since a message requiring a data update must be published by a primary node, a non-primary node must publish a Please request as discussed in Section 5.6.2.

• A workflow should be defined such that the primary node is the predecessor of all other nodes involved in an update. If the update is a result of a Please message, the Please requestor is subscribed to the update message from the primary node.

• Due to the possibility of message reuse, the same message may belong to more than one workflow.

Activities: Definitions and rules related to workflow activities:

• The undo of an undo-required activity and the compensation of a compensatable activity must be forcible.

• An activity with alternatives of which at least one is guaranteed to succeed is defined in FEW as jointly forcible. Note that individually each alternative individually may not be forcible.

• A workflow is called forcible if all its activities are forcible or jointly forcible. A workflow is called critically forcible if all its critical activities are forcible or jointly forcible. In general, we say that a sub-workflow or a workflow segment is (critically) forcible if all its activities are (critically) forcible or jointly forcible.

• A workflow is called compensatable if all its activities are compensatable. In general, we say that a sub-workflow or a workflow segment is compensatable if all its activities are compensatable.
5.7.6 ACID properties of a FEW workflow

Stemming from the definitions and rules of the previous section, the following correctness requirements are supported by FEW:

Atomicity: This means that the workflow behaves as an atomic unit of processing: it is either executed in its entirety, or it is not executed at all. This includes correctness criteria for commits and aborts. Due to the operations being standard, and that the message types are established by the analysis process, failure of an activity should be a rare occurrence. The following are the definitions and rules related to the atomicity (commit or abort) of a FEW workflow:

- Commit correctness:
  - If an activity is forcible, for the commit protocol it is not relevant whether or not it is critical. Commit correctness in FEW is therefore redefined: a workflow may commit if all non-forcible critical activities also commit.
  - A non-compensatable activity cannot be included in a workflow unless all activities, except perhaps its predecessors, are forcible or jointly forcible. This is because if the activity commits, the whole workflow must also commit.
  - A checkpoint may be used to partially commit a sub-workflow to that point. The partially committed workflow is perceived by the workflow model as, and has the same characteristics of, an individual activity.
  - Checkpoints in the situation above remain ready to commit until the sub-workflow spawned in the node is committed.
  - A commitpoint node (or simply a commitpoint) is a node where it is correct to commit a workflow, even if the workflow has not finished executing. This means that the results of the predecessor activities are durable and may be used. (See below.)
  - A checkpoint for which the predecessor workflow may commit and the successor workflow is critically forcible, is a commitpoint.

- Abort correctness:
  - A workflow that includes an undo-required activity that is not compensatable cannot abort. Abort correctness in FEW is therefore redefined: a workflow may abort if all undo-required activities are compensatable.
  - A checkpoint may be used to abort a workflow; if the predecessor workflow includes critical activities and aborts, the whole workflow aborts.
As it may be seen from the above discussion, the notion of checkpoints and commitpoints is central to support for atomicity in FEW. Since a checkpoint may be used to temporarily commit (the node is ready to commit) a partially completed workflow, the node serves as a starting point for forward recovery in case of failure. The workflow needs only to be restarted from the checkpoint. If the checkpoint is also a commitpoint, the workflow may be assumed to commit in its entirety, and the results of activities predecessors to the node are durable. Hence, checkpoints are very important for the FEW model; thus, relay link pairs should be used to establish checkpoints in a workflow, as depicted in Figure 5.6. The resulting workflow with a new checkpoint is depicted in Figure 5.7.

**Figure 5.6: Replacing a link (red) with a relay pair (green)**

**Isolation:** In workflow execution, an individual activity is the natural unit of isolated execution. Since in FEW scope may transcend the boundaries of processing nodes by accessing shared data, there might be update conflicts. However, in FEW the reads and writes are different to database operations. In a database, reads and writes follow a request (pull) model, where operations result in reads and writes dispatched by the system, interleaved in an unpredictable order. Instead, in FEW a write operation on a fragment is always initiated by an incoming message originated by the fragment primary application (maybe sometimes after a Please Request message). After the primary update, the update information is propagated (pushed) to the successor nodes in the workflow, which use the message content to update
their own data stores. Note that due to the condition of quasi-consistency, all ACID-type transactions are restricted internally to domains, and there cannot be processing dependencies — in particular transactions — involving more than one domain.

Similarly to database transactions, in FEW there might be two different types of conflicts:

1. Write-write conflicts: Where two activities may attempt to write to the same data fragment. In FEW this is avoided by the primary update responsibility allocated to one of the applications, resulting in all update operations initiated by the primary application. Thus, it is possible to ensure the correct sequence of update operations by adding a sequential number to the update message at the primary node. This results in a serial schedule as long as the successor nodes follow the same sequence\(^4\).

2. Write-read conflicts: Although a counter as above ensures a serial schedule for the updates, it is still possible that a read operation reads old or dirty data; however, this is taken care of by the the quasi-consistency — or at least tolerance of transient inconsistency — existing between the nodes.

\(^4\)Most middleware products support the retrieval of messages in the same order in which they were sent. This facility may be used instead.
It should be noted that the use of a sequential number for update operations on fragments is closely related to the classic Time-Stamp Ordering algorithm for the same type of conflicts in the area of databases. An increasing sequence attached to the update operation may be correctly considered a time-stamp, since the message type is determined by the operation to be performed and the affected fragment. That is, the sequence determines simultaneously the time stamp of the operation and the time stamp of the fragment (record) to be updated; hence, there is no need for a separate fragment time stamp. As a consequence, all workflow operations do not need to be sequential if this is considered detrimental for performance, but they may be interleaved arbitrarily and conflicts detected by comparing the timestamps of messages of the same type. Moreover, if the transient inconsistency may be tolerated for an extended period, it is not necessary to restart a newer operation when a conflict is detected, it may be simply delayed until the older operation has performed the update.
Chapter 6

Federated Infrastructure for Enterprise Integration

This chapter introduces the federation services that support the enterprise integration. With a ‘bottom-up’ approach, we discuss first the services provided by a lower-level federation layer, including the federation protocols that ensure the integrity of shared data, and how these support a higher-level layer that deals with global data entities and business workflows.

6.1 Introduction: Separation of Concerns

In all three aspects of organisational knowledge — data, rules and workflows — the focus is always related to the data fragments shared among the interoperating applications, thus:

- an ontology stemming from the business rules is used to describe the structure of shared data;
- the structure of corporate data must be correctly mapped to the processing stations, by translating the structure of data fragments to and from the corporate format;
- updates must maintain the integrity of corporate data;
- the workflows must properly implement business processes by exchanging data between nodes.

In order to ensure its flexibility and maintainability, the architecture of the integration should separate the concerns of the higher-level business operation from the concerns related to the integrity of the data. This is so because higher-level concerns such as business rules or processes are likely to change often to follow the path of organisational evolution, whereas the data exchange protocols that ensure the integrity of the data must be stable, and are to be maintained
oblivious to these changes. In this way, the changes to the higher-level layer that are likely to occur will not affect the functioning of the supporting infrastructure.

The federation provides the infrastructure and the required services to effect the correct sharing of data between component domains, and to guarantee the integrity and timeliness of the messages exchanged. Higher-level constructs are highly dependent on the business definitions, and therefore they are outside the scope of the federated infrastructure services. They must be provided in addition to the federal infrastructure. As a consequence, we posit that a higher layer should manage the business flows, while a lower layer implements the necessary protocols to properly implement the data exchanges. Thus, the federal corporate infrastructure should support, and provide the mechanisms for, sharing of data between the systems (domains) of the federation [42], but it should not implement high-level constructs, such as business flows. Instead, higher level constructs may rely on the services provided by the federation to carry out the enterprise workflows.

Additionally, the federation relies on the guaranteed delivery of messages among interacting domains. Although this service may be currently provided by an infrastructure layer of Message Oriented Middleware, new standard frameworks coming online may provide a more abstract, vendor-neutral — and therefore more flexible — platform on which to base the exchanges. Certainly, the Web Services framework shows promise to provide such a platform, although as indicated in our conclusion, there needs to be further development along the lines of non-available interfaces.

6.2 Corporate Entities

As a result of the integration, data residing in local systems may now have a new global scope, and the propagation of an updating local event may trigger several remote updates. This is caused by a shared global concept — customer, order, invoice — having aspects replicated in systems housed in different domains. Therefore, it is necessary to identify local data fragments that have some overlapping with other remote fragments. For example, if employee information is partly replicated across several systems — e.g. HR, email, Department database — it would be required to propagate an update on any of the overlapping segments to the others to keep the data synchronised. Thus, the different local fragments must be identified as part of a global entity, and the integrated enterprise must keep track of the required updates and propagate any local data-updating operation to all the affected domains. Following our discussion, this should
be a service provided by the lower-level federation layer, so a sequence of high-level messages exchanged among domains to carry out an enterprise flow is to be translated to a sequence of lower level federation exchanges that ensures the correct update of local data.

Consequently, enterprise integration in such a federated context focuses on data items shared by more than one domain: they are the only business of interest to the federation because they are the ones that generate federation traffic. Wijegunaratne et al [125] describe a federated approach to support the exchange of data housed within disparate systems in different geographic locations, and discuss how the scheme has been implemented in an Australian energy company. This implementation provides several services to support the local autonomy of data:

- an asynchronous messaging transport mechanism that provides guaranteed message delivery services and insulates one participating application process from another;
- limited data interpretation capability confined to syntax translation, data format transformation and key mapping services (See Section 6.3).

We call these shared data entities — such as customers, insurance policies, employees and orders — together with their associated operations Corporate Entities (CE). A domain application may have a subset (a fragment) of customer information data and operations (say, an AllBooks International Australian customer account fragment), while another application in a different domain may have an overlapping fragment of the same customer attributes and operations (e.g. as a New Zealand customer).

Even when considered in the context of an integrated enterprise, these entity fragments are independent of each other. They have been defined independently, and they are administered autonomously inside the domain housing each responsible application. However, the integration strategy needs to identify and manage these two fragments as being part of the same customer, to make sure that updates to one fragment (e.g. address details) are reflected on the other.

Corporate Entities, therefore, are the association and aggregation of their fragments residing within domain applications. Very importantly, in our approach a fragment is not necessarily an attribute, or a table a relational database, or an instance object in the object-oriented sense of the term. Some fragments may be the structure returned by a function call — such as a method or a stored procedure — which processes or aggregates information from within an application. In this sense, fragments do not necessarily have existence as a collection of attributes in a database or a file, but they may be derived in this sense. This is absolutely necessary for
enterprise integration, since in its normal way of operation a database application very rarely
lets users or processes access data directly from the database, but offers a set of services —
views, functions, stored procedures, methods — to be invoked ‘as is’ by external processes, and
these are the only entries to stored data.

6.3 Federation Services for Enterprise Integration

The integration effort requires that these independent fragments (including derived fragments)
are identified as belonging to the same Corporate Entity, and the provision of services for the set
of standalone fragments to behave as one integrated entity. The following services are required:

Corporate Format: To support the integration strategy the federation should adopt a corpo-
rate message format (i.e. a unified corporate language), to determine the syntax and format
of messages flowing through the federation (See below). In order to reduce the complexity of
the implementation, we advocate the use of a corporate ontology describing the data. Since
the ontological approach centralises them, this reduces the number of specifications necessary,
and simplifies their management. However, domains are only producers and consumers of
information in the form of messages, and an individual domain responsibility is (exclusively)
to publish the information agreed upon with the global structure. Therefore, the message
translation service required to map domain information to and from corporate format should
be provided by federation structures that have access to the ontological descriptions. This is
naturally implemented within the domain Message Adaptors.

Key Mapping: Since fragments are unaware of each others existence, a Corporate Entity
must be identified as an enterprise-wide entity. The process of creating and maintaining a
global ID, and linking it to its constituent fragments, has been called key mapping in [125].
Due to the fragments lack of awareness of each other, this Data Integration Service (DIS)
is the responsibility of the federation. It involves providing a global ID for the entity, and
maintaining the necessary cross-references for the individual fragments comprising the entity.
Following the Customer example above, the service may provide a global ID (cust8765) to
map the two local ones, as shown in Table 6.1:

<table>
<thead>
<tr>
<th>AllBooks ID</th>
<th>Australian ID</th>
<th>New Zealand ID</th>
</tr>
</thead>
<tbody>
<tr>
<td>cust8765</td>
<td>1234</td>
<td>xyz456</td>
</tr>
</tbody>
</table>

Table 6.1: Key Mapping for an AllBooks Customer
Primary Update Responsibility: The overlapping of fragments necessarily introduces update conflicts upon shared data. This is similar to the problem of replicated data in a distributed database, where one copy of the data is allocated as the primary copy, the other copies acting as slaves [38]. As discussed in Chapter 5, in order to avoid update conflicts, the federation follows a similar strategy by selecting one application as the primary application for the fragment. A primary application has update authority over the fragment, all other applications must follow. Note that this is naturally related to the notion of primary node presented in Section 5.6.2: the primary node for a fragment is the node housing the primary application. This decision is made, and its management belongs, in the federation space, but applications need to know whether they are a primary application for a given fragment, since their local update behaviour may change slightly if they are not 1. A standard sequence of messages exchanged by the corresponding domains enforces this behaviour. (See Section 6.4 for a detailed explanation).

Following the discussion of the previous paragraphs, the fact that some fragment information belonging to a domain is now shared with other domains results in the need for the implementation of federation protocols to govern the update of shared data. To be precise, when an update within a domain is a federation event because the data is shared, there is a need to follow a specific sequence of steps to ensure the integrity and consistency of the corresponding corporate entity. The intervening applications are not aware of each others existence — this information belongs to the federation space — hence the implementation and enforcement of these protocols is also the responsibility of the federation infrastructure. These federation message protocols are different, depending on whether the fragment update is initiated by a primary application or not. (See Section 6.4).

6.3.1 Federation Service Levels

Experience with enabling enterprise-wide computing has shown that regardless of the care with which overarching changes are implemented, the process is many times perceived as too disruptive [93], “too much too soon” in the words of the system architects of a major Australian bank [124] (pp 197). A federated approach may avoid these pitfalls by a careful introduction of federal structures:

- Domains may be demarcated and/or introduced a few at a time.

1It may be advisable in some cases to allocate primary responsibility for a fragment to more than one application. This is only possible if the fragment set may be partitioned in a mutually exclusive way between the application under consideration.
• Federal services may be introduced level by level (see below) rather than all at once.

• The experience with the development of initial Message Adaptors may be used to develop
generic adaptors that may be serve as framework for the remainder of the domains.

It is thus possible to implement the federated lower-level infrastructure as several levels of
services, described from Level 0 to Level 3 in increasing order of federation involvement:

**Level 0:** The Federation provides no services other than guaranteed message delivery via its
middleware infrastructure. Corporate message format is not supported by the federation.
As a consequence, message structure and format translation, update conflicts, key mapping
between domain applications are all resolved at individual applications level. Although this
violates the autonomy principle of applications, such a federation would enable the safe
exchange of information between applications. This is the simplest form of federation in-
volvement, and although it solves the problem of inter-domain communication — as our
first case study illustrates — the exchanges are still point to point, there is no possibility of
message reuse and applications must be aware of each other.

**Level 1:** Level 0 compliance is assumed, but message structure is now known to the federation.
That is, there is a corporate format for messages flowing through the federation. Therefore,
message structure and format translation to and from a corporate format are supported
by the federation. Domains avoid point to point communication, but they comply with
the corporate format. This federal service is implemented as a set of established message
types recognised and supported by the federation. However, at this level key values and
corporate entities fragments are not known by the federation (i.e. Corporate Entities are
not supported); key mapping is, therefore, not provided. Update conflict resolution is not
supported, so update conflicts must be resolved by individual applications/domains. This,
again, implies that the principle of application autonomy is violated, since applications must
be aware of each other.

**Level 2:** Level 1 compliance, but key values are now managed by the federation. Consequently,
message structure, format translation and key mapping services are all provided by the
federation. However, update conflicts still have to be resolved by individual applications.
Thus, applications have to be made aware of the existence of relevant fragments in other
applications, and they are in charge of negotiating an update protocol.

**Level 3:** Level 2 compliance, plus primary update responsibilities are now known to the fed-
eration. Therefore, structure, format translation, and key mapping services are provided by
the federation. Federation message flows and authorisation services now support the known
primary update responsibilities.
It follows that the federated enterprise integration approach advocated here requires a federation Level 3, since only this level provides the required services for data integrity. However, note that even at the highest level presented here, Level 3, the domains are not responsible for establishing or maintaining a business workflow. Although the implementation of the Level 3 service level defines a set of primitive federation message protocols to be triggered when certain events occur, these are only to take care of the consistency of corporate entity fragments in different domains. As a consequence of a New Order event, for example, using federation infrastructure information (see Section 6.4) a known message sequence will be exchanged between the originating domain and the domains subscribing to the new order information. Although this ensures that information flows properly between domains, it does not guarantee that the process is the correct one from the point of view of the enterprise business. To this end, a business workflow layer such as the FEW model introduced in Chapter 5 must be provided on top of a Level 3 compliant federation.

### 6.4 Data Integrity: Federation Protocols

As discussed previously a federated, message-based approach necessarily involves data replication. Consequently, a Corporate Entity with fragments in different domains — i.e. the applications within the domains — needs to react to a fragment-updating event within the domain. Thus, an event occurring within the application related to a fragment update initiates
a standard sequence of actions to maintain the integrity of the Corporate Entity. The sequence from the point of view of the triggering application is as follows:

1. The event is recognised within the application.
2. The event is detected and accessed by the domain Message Adaptor.
3. The event is classified by the domain Message Adaptor as being of a certain (standard) type.
4. An outgoing message of a type related to the event in 3 is created. (Full descriptions below).
5. The source local entity ID is mapped to an appropriate cross-reference (key mapping) and the message is transformed to corporate format.
6. The message is published.
7. Subscribers domains — their Message Adaptors — receive the message as an incoming message, and the cross-reference is resolved to local destination IDs.
8. The incoming message is delivered to the appropriate application/s.

Although this sequence is standard, there are some differences between the steps depending on the type of operation performed within the application, and hence the outgoing message type. In order to implement this sequence it is necessary to establish and maintain some federal structures, such as the Data Integration Service (DIS) mentioned before, which is in charge of the creation and management of the required federal-level cross-references.

Several basic protocol flows are necessary to the correct management of fragment updates:

1. New Instance (Primary) Event: The application is the primary application for a fragment, and it has created and inserted a new instance of the fragment. The application then delivers the parameters of the new instance, including the new local CE fragment ID, to the Message Adaptor. The Adaptor requests the creation of a new corporate cross-reference from the DIS to map to the new local fragment ID. Upon receiving the new Corporate Entity ID, the Adaptor updates its database, creates a new outgoing message with an appropriate header and the required parameters, includes the Corporate Entity ID in the message instead of the original ID, translates the message to corporate format and publishes the message by placing it on the federation infrastructure. A subscriber domain (i.e. its Adaptor) interprets the corporate format, translates the message to the format of the receiving application and delivers it to the application, requesting the insertion of a the new local fragment. Each application returns the new local fragment ID, which the subscriber Adaptors record against the federation cross-reference.
2. Update Instance (Primary) Event: The application is the primary application for a fragment, and it has updated an instance of the fragment. The application then delivers the parameters of the updated instance, including the local CE instance ID, to the Message Adaptor. The Adaptor maps the local instance ID to the corresponding Corporate Entity ID, creates a new outgoing message with an appropriate header and the required parameters, inserts the Corporate Entity ID into the message, translates the message to corporate format and publishes the message by placing it on the federation infrastructure. A subscriber domain (i.e. its Adaptor) interprets the corporate format, maps the Corporate Entity ID to each application local ID, translates the message to the format of each receiving application and delivers the message to the applications.

3. Delete Instance (Primary) Event: The application is the primary application for a fragment, and it has deleted an instance of the fragment. The application then delivers the parameters of the deleted instance, including the local instance ID, to the Message Adaptor. The Adaptor maps the instance ID to the corresponding Corporate Entity ID, creates a new outgoing message with an appropriate header and the required parameters, inserts the Corporate Entity ID, translates the message to corporate format and publishes the message by placing it on the federation infrastructure. A subscriber domain (i.e. its Adaptor) maps the Corporate Entity ID to the local fragment ID, translates the message to the format of, and delivers it to, the receiving applications.

In addition to the above (primary) protocols, the actions triggered by non-primary applications must also be considered. In this case the actions are not acted upon immediately, but they are only Please Requests related to non-primary applications since these do not have authority to act unilaterally on the fragment. Note that the definition of these Please Requests is closely linked to the Please workflow node behaviour defined in Section 5.6.2: the requests are initiated by non-primary applications housed in the node.

Hence, we define here the following flows:

1. New Instance (Non-Primary) Event: The application is not the primary application for a fragment, and it has created and intends to insert a new instance of the fragment. The application delivers the parameters of the new instance to the Message Adaptor, which creates a Request New Instance outgoing message with an appropriate header and the required parameters, and then translates to corporate format and publishes the message. The subscriber domain — the one housing the primary application for the fragment — interprets the corporate format, translates the message to the format of the receiving application and
delivers it to the primary application, which proceeds with the creation of the New Instance according to its own business rules. This event naturally results in the execution of the New Instance (Primary) Event above, and the corresponding flow is then executed. The subscribing domains that insert the new instance include the originating domain, although there might also be other subscribers.

2. Update Instance (Non-Primary) Event: The application is not the primary application for a fragment, and intends to update an instance of the fragment. The application delivers the parameters of the update, including the local ID, to the Message Adaptor. The Adaptor maps the local application ID to the Corporate Entity ID, creates a new Request Update Instance outgoing message with an appropriate header and the required parameters, translates the message to corporate format, and publishes the message by placing it on the Federation infrastructure. The subscriber domain — the one housing the primary application for the fragment — interprets the corporate format, translates the message to the format of the receiving application including mapping the corporate ID to the local ID, and delivers it to the primary application, which proceeds with the update of the Instance according to its own business rules. This process results in the event Update Instance (Primary) Event, and the corresponding flow is then executed. The subscribing domains that update the instance include the originating domain, although there might also be other subscribers as well.

3. Delete Instance (Non-Primary) Event: The application is not the primary application for a fragment, and intends to delete an instance of the fragment. The application delivers the parameters of the update, including the local ID, to the Message Adaptor. The Adaptor maps the local application ID to the Corporate Entity ID, creates a new Request Update Instance outgoing message with an appropriate header and the required parameters, translates the message to corporate format, and publishes the message by placing it on the Federation infrastructure. The subscriber domain — the one housing the primary application for the fragment — interprets the corporate format, translates the message to the format of the receiving application including mapping the corporate ID to the local ID and delivers it to the primary application, which proceeds with the deletion of the Instance according to its own business rules. This process results in the event Delete Instance (Primary) Event, and the corresponding flow is then executed. The subscribing domains that delete the instance include the originating domain, although there might also be other subscribers as well.

Some comments are important to understand the functioning of these federation protocols:

- These protocols are applicable only when a fragment update is taking place.
A non-primary application that requires a fragment update always initiates a Please federation protocol in the manner described above.

Primary protocols are all initiated by the application having primary update responsibility for the instance fragment. In order to maintain data integrity these flows produce quasi-mandatory actions (create, update, delete) at the receiving end. The receiving applications are required to carry on the actions — for example, when the actions are part of an enterprise workflow — and they should be able to comply with the request. Since they are not primarily responsible for the fragment, if any action is not possible at the receiving end this is considered an error, and it must be handled by the federation error mechanisms.

A non-primary application may unilaterally ignore a data update or delete upon its own copy of a fragment, if that operation is restricted to the domain housing the application. Such is the case of an application housed within a terminal node, as discussed in Section 5.6. This should be only when there is no need to maintain — temporarily or permanently — the data integrity of the fragment. To make the change permanent the domain that houses that application only needs to unsubscribe to the workflow messages related to the fragment — as described above — given that they are not needed anymore. Note that this results in the domain node disconnecting from the workflow.

All Primary Events occurring within a primary application result in related operations on other fragments in applications within subscribing domains. Although in certain situations a non-primary application may unilaterally ignore or unsubscribe to a message, a primary application must continue to publish the messages resulting from these primary events. This is an important part of the contract discussed in Section 4.7 for the subscribing domains rely on this information to keep their data up to date and properly synchronised. To discontinue the publication of any of these messages, the primary application must obtain the agreement of all subscribing domains. This is an exceptional rather than standard behaviour; it is therefore not part of the services provided by, and it must be negotiated outside the sphere of, the federation.

6.5 The Integrity of Corporate Entities

The federation structures do not hold data related to the Corporate Entities, except for their global IDs, as shown in Figure 6.1. All other CE data is restricted to the domains, and since their Adaptors must perform the required translation this must be reflected on their Message Adaptor structures. For the appropriate management of data, the domain Message Adaptors
require the following:

- A list of their own domain applications and their required related information: their location, access methods (functions), CE fragments housed within the application, events generated, messages of which is to receive delivery.

- The structure of the CE fragments for which it is responsible (e.g. Australian Customer fragment).

- Access to the ontological description of the corporate format.

- A list of the events for which it is responsible (e.g. New Order, Update Customer), and information about how these map to messages.

- A list of the message types for which it is responsible, information about how to translate to and from corporate format and to which local applications each incoming message is to be delivered.

- A MA must hold information mapping a CE ID to all the applications with a fragment of the CE, including the application ID, the structure of the fragment and whether the application is the primary application for the fragment.

Thus, from the data management point of view, a domain Adaptor

- monitors events occurring within local applications;

- extracts the application data as a result of the event;

- transforms the data
  
  - determines the type of outgoing message from the workflow description;
  
  - creates the message header (including the key mapping information);
  
  - requests corporate format information from the ontology;
  
  - translates the message payload from the application format to the corporate format (including security transformations); and

- publishes the message.

Conversely, as a result of an incoming message, the Adaptor translates the message from corporate format to the format of all the local applications that are to receive it, and delivers the message to all the intended recipients.

The next section presents a layered structure for a generic Message Adaptor.
6.6 The Structure of a Message Adaptor

The discussion of the previous section outlines the data path from an originating, publishing application to a subscriber application via the cooperation of the applications with their corresponding Adaptors and between the Adaptors themselves. Each of the steps described in the behaviour above refers to a component of the MA or the federation infrastructure.

Table 6.2: Placement of Components

<table>
<thead>
<tr>
<th>Component</th>
<th>Placement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ontology</td>
<td>Federation</td>
</tr>
<tr>
<td>Workflow Description</td>
<td>Federation</td>
</tr>
<tr>
<td>Event monitoring</td>
<td>Message Adaptor-Middleware</td>
</tr>
<tr>
<td>Data Extraction</td>
<td>Message Adaptor</td>
</tr>
<tr>
<td>Data Transformation</td>
<td>Message Adaptor</td>
</tr>
<tr>
<td>Message Publishing</td>
<td>Message Adaptor</td>
</tr>
</tbody>
</table>

The placement of components depicted in Table 6.2 indicates a layered structure for the Message Adaptor, with each layer taking care of a discrete operation on the data. As illustrated in Figure 6.2, the Adaptor moves information between the layers, from left to right in the case of a publishing domain, and from right to left in the case of a subscriber.

**Application Access (AA):** This layer is in charge of the interaction of the MA with the exposed services of the applications layer. This interaction involves extracting and returning data via service invocations. This layer cooperates with the wrappers to monitor application events and, depending on the technology and the design, extract and deliver the data.

With respect to application access, the Message Adaptors are only allowed to invoke the services provided that relate to the shared information. This approach ensures that accesses to the back-end applications conform to the applications business rules. The only exception might be the monitoring mechanism that detects events as they occur within the application, which depending on the application technology may use an event-driven or a polling approach. Although the exposed services need not be for the exclusive use of the Adaptors and may be used by other applications within the domain, the interface with the Application Access layer should remain stable — i.e., services cannot be withdrawn or unilaterally changed — since the federation relies on them to function properly.

---

2How these services are exposed is outside the scope of the MA, which assumes the existence of a wrapper (i.e. wrapper functions) with which it can interact. These are usually provided by the middleware infrastructure.
Figure 6.2: A Layered Structure for a Message Adaptor
**Message Transformation (MT):** This layer is responsible for the necessary translations between corporate and application formats. Specifically, it is in charge of identifying and building the messages, including key mapping (DIS invocation).

The Message Transformation layer must interrogate the semantic services to effect the required format translations. Although the semantic services provide the description of the data, other mappings not directly related to the structure of the data such as deletion, re-ordering, aggregation or dis-aggregation of message fields, insertion of message header fields, and management of corporate IDs may be also necessary. This layer should also be responsible for a security service if this is required by the federation (for example to ensure message confidentiality).

**Message Delivery (MD):** This layer provides an abstract layer of services to place and retrieve messages in/from the MOM infrastructure. The Message Delivery component implements a set of abstract calls (API) to publish and subscribe messages to the federation space. This insulates the Adaptors from the particulars of the middleware of choice, enabling them to assume a publish and subscribe capability, whether this really exists or it must be implemented ‘on top’ of a simple, queue-based store and deliver system. In addition, this abstraction avoids ‘hard wiring’ MOM commands into the MT layer, making easier an eventual transition to a different MOM product.
Chapter 7

Federated Enterprise Integration

This chapter presents the relationship between the federation infrastructure, the Corporate Entities and business processes and workflows. To this end, the chapter introduces the methodology to perform the required data-oriented and process-oriented analyses. It also discusses how to map a business workflow as a combination of federation flows, and how a FEW workflow may follow the path of business evolution.

7.1 A Sale in AllBooks Australia

All Books Australia has decided to adopt the federation model, and they have accepted the recommendation of establishing a federation Level 3 to be able to solve their integration problems. However, the condition was that the project had to proceed in stages, to reduce as much as possible the risks associated with the strategy. The initial analysis resulted in a federated integration context for AllBooks Australia comprising five domains: Sales, Head Office, Warehouse, Human Resources and Editorial. In their first attempt the analysts considered the process of a “Sale” for AllBooks Australia, and the set of business rules relevant to the business process of making a sale by raising an order.

Although this follows the generic outline introduced in Section 5.4, we revisit here the methodology and to reflect the different nature of the data and process analysis we divide it into data-oriented analysis and process-oriented analysis. Following this model, the analysts gathered the set of business rules related to a Sale of books, within the AllBooks Australia context.

7.1.1 Data-Oriented Analysis

As a result of the data-oriented analysis, the following was established.
**Integration Focus:** A Sale process.

**Integration Context:** All Books Australia federation.

**Ontology:** Semantics and ontology methodology:

1. Gather all business rules referring to the terms used in the integration context.

   **General rules:** For AllBooks:
   
   “All money quantities are expressed in Australian Dollars (AUD)”
   “All money quantities must be expressed with a precision of cents”
   “All prices must include Goods and Services Tax (GST)”
   “Prices are money quantities”

   **Business Process Rules:** Related to the “Sales” workflow:
   
   “Book prices are prices”
   “A sale is initiated by raising an Order at Sales on the ORDERS+ application\(^1\). A Sales Representative must be responsible for taking the Order”
   “A new order is only possible to existing customers, up to the value of their Credit Limit”
   “If a customer doesn’t exist, then execute `NewCustomer()`”
   “`NewCustomer()` is executed by the CRM application at Head Office”
   “A new customer must be allocated an initial Credit Limit”
   “Credit Limit management is responsibility of the Credit Manager. If the Credit Manager is not available, it may be authorised by the Financial Controller”
   “Sale goods must be dispatched together with a corresponding delivery docket and invoice”
   “Financial Administrators at Head Office are in charge of producing invoices”
   “For each order, the QuickWH Warehouse application must produce a delivery docket to dispatch with the goods”
   “The purchase total value must be the same in the order, the invoice and the delivery docket”
   “No partial sales are allowed”

2. Determine the terms related to actors (domains) and data semantics of the Sales process used in the integration context:

   Money quantities, Australian Dollars, Goods and Services Tax, prices, cents, goods, Sales, Head Office, Warehouse, Order, Customer, Sales Representative, Financial Administrator, Financial Controller, Credit Manager, Credit Limit, CRM application, ORDERS+,

---

\(^1\)Both the ORDERS+ and QuickWH below are imaginary applications, assumed to be in charge of the tasks.
QuickWH, invoice, delivery docket.

3. Determine whether each of the terms above is a common term or a business term.

**Common Terms:** Money quantities, Australian Dollars, Goods and Services Tax, price, cents, goods.

**Business Terms:** Sales, Head Office, Warehouse, Order, Customer, Sales Representative, Financial Administrator, Financial Controller, Credit Manager, Credit Limit, CRM application, ORDERS+, QuickWH, Invoice, Delivery Docket, Total Value.

4. For each common term confirm that there is a common understanding of the term:
   Australian Dollars are commonly abbreviated AUD.

5. From the business rules and the federation information, determine whether the business terms are of interest to the integration context (i.e. they are shared by domains in the federation of concern), or they are only local, private to a domain. Classify the terms of interest to the federation as Domains or Corporate Entities and determine which domains possess fragments of which entities. Exclude individual attributes from this analysis.

**Shared Terms:** The following refer to shared elements in the integration context:
- Warehouse, Head Office and Sales are the domains involved in a Sale.
- Customer is a corporate entity.
- Order is a corporate entity.
- Delivery Docket is a corporate entity.
- Sales, Warehouse and Head Office have fragments of Customer.
- Warehouse and Head Office have fragments of Delivery Docket.
- Sales, Warehouse and Head Office have fragments of Order.
- Warehouse and Head Office have fragments of Invoice.

**Local Terms:** The following refer to local terms, private to a domain:
- CRM application, Financial Administrator, Credit Manager and Financial Controller are private to Head Office.
- Sales Representative and ORDERS+ are private to Sales.
- QuickWH is private to Warehouse.

6. Model the shared entities related to the data semantics terms above to include individual attributes. Attributes private to a domain are not to be considered\(^2\). Link the shared

\(^2\)For example, Sales Representative is an attribute of Order, but since it is local to Sales it is not included in this step.
terms to the corporate ontology. If necessary, add their structure to a custom made ontology and link the entities and their attributes to the standard.

![Figure 7.1: Linking Corporate Entities to the Ontology](image)

**Head Office:**

Customer: (customer_id, name, address, phone, credit_limit)

Invoice: (invoice_id, order_id, docket_id, date, total_price)

Order: (order_id, customer_id, date, total_price)

**Sales:**

Customer: (customer_id, name, credit_limit)

Order: (order_id, customer_id, date, total_price)

Order_details: (order_id, book_id, quantity)

Book: (book_id, title, author, price)

**Warehouse:**


Docket: (docket_id, order_id, invoice_id, status)

Customer: (customer_id, name, address)

Order: (order_id, customer_id)

Order_details (order_id, book_id, quantity)
7. If necessary, rewrite the business rules so that local elements are not exposed to the federation level. Any local term private to a domain — and therefore unknown outside the domain — should be replaced by the domain name.

**Existing Rules:** Considering the original set, the following rules must be rewritten because they expose local domain elements:

“A sale is initiated by raising an Order at Sales on the ORDERS+ application. A Sales Representative must be responsible for the Order”

“NewCustomer() is executed by the CRM application at Head Office”

“Credit Limit management is responsibility of the Credit Manager. If the Credit Manager is not available, it may be authorised by the Financial Controller”

“Financial Administrators at Head Office are in charge of producing invoices”

“For each order, the QuickWH Warehouse application must produce a delivery docket to dispatch with the goods”

**Modified rules:** The rewritten rules are as follows:

“A sale is initiated by raising an Order at Sales. Sales must be responsible for the Order”

“NewCustomer() is executed by Head Office”

“Credit Limit management is responsibility of Head Office”

“Head Office is in charge of producing invoices”

“For each order, Warehouse must produce a delivery docket to dispatch with the goods”

8. Determine the shared fragments of the corporate entities, and allocate primary update responsibility of the different fragments to domains.

The overlapping fragments are given by the attributes in the schema below, and depicted in Figure 7.2. The notation HO, S and W annotates primary update responsibility to the domains. (A corporate entity annotation is overwritten by a local attribute one. A ‘*’ indicates a multi-valued attribute.)

**Head Office:**

Customer[HO]: (customer_id, name, address, phone, credit_limit)

Invoice[HO]: (invoice_id, order_id[S], docket_id[W], date, total_price)


**Sales:**

Customer[HO]: (customer_id, name, address, phone, credit_limit)

Order[S]: (order_id, customer_id, Order_Line*, date, total_price)

Order_Line[S]: (order_id, book_id, quantity)
Figure 7.2: Sales data diagram
7.1.2 Process-Oriented Analysis

The second phase of the methodology is to consider the process-oriented aspects of the integration. This is necessary to be able to map domain tasks and events to the flow of messages in the integration context. The following statements determine the corresponding business process, and a workflow that implements the business process:

**Business Process:** The following statements define the methodology to establish the business process:

1. Extend the model by considering also the terms not exclusively related to data semantics. For each term, select the rules where there is a term that refers to a procedure:
   - “A sale is initiated by raising an Order at Sales on the ORDERS+ application. A Sales Representative must be responsible for the Order”
   - “If a customer doesn’t exist, then execute `NewCustomer()`”
   - “`NewCustomer()` should be executed by the CRM application at Head Office”
   - “A new customer must be allocated an initial Credit Limit”
   - “Credit Limit management is responsibility of the Credit Manager. If the Credit Manager is not available, it may be authorised by the Financial Controller”
   - “Sale goods must be dispatched together with a corresponding delivery docket and invoice”
   - “Financial Administrators at Head Office are in charge of producing invoices”
   - “For each order, the QuickWH Warehouse application must produce a delivery docket to dispatch with the goods”

2. Allocate task-related terms to domains. Determine whether tasks are local or of interest to the federation. (See Table 7.1).

3. Identify the applications in charge of tasks with federation scope, and their domains (See Table 7.2).
4. Identify federation events. Classify as primary and non-primary (Please) events. Table 7.3 summarises.

Table 7.3: Federation Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Domain</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>New Order</td>
<td>Sales</td>
<td>Primary</td>
</tr>
<tr>
<td>Order Non-Customer</td>
<td>Sales</td>
<td>Non-Primary</td>
</tr>
<tr>
<td>Insert New Customer</td>
<td>Head Office</td>
<td>Primary</td>
</tr>
<tr>
<td>Produce New Invoice</td>
<td>Head Office</td>
<td>Primary</td>
</tr>
<tr>
<td>In Stock</td>
<td>Warehouse</td>
<td>N/A</td>
</tr>
<tr>
<td>New D. Docket</td>
<td>Warehouse</td>
<td>Primary</td>
</tr>
</tbody>
</table>

5. For primary federation events, determine the federation inputs required by each domain to execute each of its tasks, and the output produced as a result (See Tables 7.4 and 7.5).

6. For non-primary federation events, determine the federation inputs required by each domain to execute each of its activities, and the output produced as a result. Table 7.6 summarises.

7. Establish the transactional behaviour of each activity: determine whether each activity is forcible, compensatable and/or undo-required. Depicted in Table 7.7.

8. Determine checkpoints and commitpoints.

- Head Office is a join node, since it ANDs together the New Order and Ready messages. Head Office is a checkpoint.
Table 7.4: Activity Inputs (Primary)

<table>
<thead>
<tr>
<th>D</th>
<th>Activity</th>
<th>Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Raise Order</td>
<td>none</td>
</tr>
<tr>
<td>HO</td>
<td>Create Invoice</td>
<td>New Order Information</td>
</tr>
<tr>
<td>W</td>
<td>Check Stock</td>
<td>New Order Information</td>
</tr>
<tr>
<td>W</td>
<td>Create Docket</td>
<td>New Order Information, New Invoice Information</td>
</tr>
</tbody>
</table>

Table 7.5: Task Outputs (Primary)

<table>
<thead>
<tr>
<th>D</th>
<th>Activity</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Raise Order</td>
<td>New Order Information</td>
</tr>
<tr>
<td>HO</td>
<td>New Customer</td>
<td>New Customer Information</td>
</tr>
<tr>
<td>HO</td>
<td>Create Invoice</td>
<td>New Invoice Information</td>
</tr>
<tr>
<td>W</td>
<td>Create Docket</td>
<td>New D.Docket Information</td>
</tr>
</tbody>
</table>

Table 7.6: Activity Input and Output (Non-Primary)

<table>
<thead>
<tr>
<th>D</th>
<th>Task</th>
<th>I/O</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Raise Order</td>
<td>O New Customer Information (partial)</td>
</tr>
<tr>
<td>S</td>
<td>Raise Order</td>
<td>I New Customer Information</td>
</tr>
</tbody>
</table>

Table 7.7: Activity Transactional Behaviour

<table>
<thead>
<tr>
<th>D</th>
<th>Activity</th>
<th>Behaviour</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>Raise Order</td>
<td>Compensatable, Undo required</td>
</tr>
<tr>
<td>HO</td>
<td>New Customer</td>
<td>Compensatable, Undo Required</td>
</tr>
<tr>
<td>HO</td>
<td>Create Invoice</td>
<td>Forcible, Compensatable, Undo Required</td>
</tr>
<tr>
<td>W</td>
<td>Create Docket</td>
<td>Forcible, Compensatable, Undo Required</td>
</tr>
<tr>
<td>W</td>
<td>Dispatch</td>
<td>Forcible, Non-Compensatable</td>
</tr>
</tbody>
</table>
• Head Office is also a commitpoint, since all downstream activities are forcible.
• Warehouse is a checkpoint.

9. Define the message types and parameters (Tables 7.8 and 7.9 summarise).

| Table 7.8: Outgoing Message Types and Parameters |
|---------------------------------|---------------------------------|
| D | Event/M. Type | Parameters |
| S | New Order | (order_id, customer_id, Order.Line*, date, total_price) |
| S | Non-Customer | (name, address, phone) |
| HO | New Customer | (customer_id, name, address, phone, credit_limit) |
| HO | New Invoice | (invoice_id, order_id, docket_id, date, total_price) |
| W | New Docket | (docket_id, invoice_id, status) |

| Table 7.9: Incoming Message Types and Parameters |
|---------------------------------|---------------------------------|
| D | Event/M. Type | Parameters |
| HO | New Invoice | (order_id, customer_id, date, total_price) |
| W | New Docket | (order_id, customer_id) |

10. Define the workflow by defining the message flow.

**Workflow:** The following workflow implements the Sale business process as a *FEW* workflow (all messages broadcast):

1. Customer initiates New Order. If customer exists, go to step 4. If customer does not exist at Sales, then publish Please (New Customer). Sale is left pending.
2. Head Office inserts new Customer, including ID and initial credit limit.
3. Head Office publishes new customer information. Subscribers (Sales and Warehouse included) update their data.
4. Sales processes order up to the Credit Limit.
5. Sales publishes order information. Subscribers (Head Office and Warehouse included) update their data.
6. (Commitpoint) Warehouse publishes a Ready message.
7. (Commitpoint) Head Office publishes Invoice information. Subscribers (Sales and Warehouse included) update their data.
8. Warehouse publishes Delivery Docket information. Subscribers (Head Office included) update their data.
9. Warehouse dispatches the goods.
10. Workflow commits.

An activity diagram in Figure 7.3 illustrates.
Figure 7.3: Activity diagram for the Sale Workflow
7.1.3 Mapping a Workflow to Federation Flows

The corresponding sequence of FEW messages — depicted in Figure 7.4 — is as follows (parameters are enclosed within [ . . . ]):

1. At Sales: if new customer, then Please Add (Customer [Partial Customer Information])
2. At HO: Primary Add (Customer [Customer Information])
3. At Sales and Warehouse: Do Add (Customer [Customer Information])
4. At Sales: Primary Add (Order [Order Information])
5. At HO and Warehouse: Do Add (Order [Order Information])
6. At Warehouse: Primary Add (Docket [Docket Information])
7. At HO: Do Add (Docket [Docket Information])
8. At HO: Primary Add (Invoice [Invoice Information])
9. At Sales and Warehouse: Do Add (Invoice [Invoice Information])

Figure 7.4: Sale Workflow for AllBooks Australia

The message sequence above is mapped to the following sequence on data fragments:
1. At Sales: if new customer, then Please Add Customer (name, address, phone); else go to 4.

2. At HO: Primary Add Customer (customer_id, name, address, phone, credit_limit).

3. At Sales and Warehouse:
   Do Add Customer (customer_id, name, address, phone, credit_limit).

4. At Sales: Primary Add Order (order_id, customer_id, date, total_price).

5. At HO and Warehouse: Do Add Order (order_id, customer_id, date, total_price).

6. At Warehouse: Primary Add Docket(docket_id, order_id, invoice_id, status).

7. At HO: Do Add Docket (docket_id, order_id, invoice_id, status).

8. At HO: Primary Add Invoice (invoice_id, order_id, docket_id, date, total_price).

9. At Sales and Warehouse:
   Do Add Invoice (invoice_id, order_id, docket_id, date, total_price).

7.2 Evolution of a FEW Workflow

Due to the autonomy of the domains in an integration context, the low coupling of their interactions, the significant number of constructs and message reuse supported by the FEW workflow federated model, it is possible for a workflow to follow the path of enterprise evolution as requirements change. After the initial integration that resulted in the “Sales” workflow, AllBooks reused the workflow to adapt to business changes, as follows:

- To keep a tally of the popularity of each author, Editorial would like to keep track of the order information relating to each author. Since are not really interested in the final result of a sale, but only on the number of requests for a given book, Editorial decides to subscribe to the New Order message as it is. Editorial is a terminal node for this workflow.

- Human Resources needs the sales information to calculate the commission that each representative receives as part of the plan of incentives. However, originally the identity of the sales representative is local to the Sales domain, so HR requests the Employee Number of the salesperson to be included in the New Order message, which now changes to:

  New Order (order_id, customer_id, Order_Line*, date, total_price, emp_no)
Since Human Resources needs to make sure that the sale has effectively taken place, it also subscribes to the `Ready(order_id)` message from Warehouse. Since all the workflow activities downstream from `Ready` — New Invoice, New Delivery Docket, Dispatch — are forcible, they do not need to wait for the completion of the workflow. To satisfy privacy considerations, the `emp_no` field of the modified message is now encrypted. Human Resources is a terminal node for this workflow.

- Originally, the rule set that defined the workflow included the rule:
  
  “No partial sales are allowed”

  However, after a trial period management considered this rule to be too restrictive for some orders, since some of the books are imported and they take some time to come in if they are out of stock. Hence, they have decided to allow partial sales, as long as each one is committed as an individual sale by the workflow. This prompted the change of the `Ready(order_id)` message to:

  `Ready(order_id, in_stock)`

  In this way all subscribers know the number of copies actually available in stock and may complete the workflow accordingly. The workflow now loops within the Sales domain until the sale is complete.

- AllBooks New Zealand is also interested in the “Sales” workflow, since they are comparing their performance with that of the Australian branch. They have subscribed to the `New Invoice (invoice_id, order_id, docket_id, date, total_price)`, and they use their own ontology to record the fact that these prices include the Australian 10% GST. Other domains in AllBooks NZ also subscribe to this information, so they have to make use of that ontology segment. In addition, since the NZ applications use a different Book ID, they invoke the federation key mapping facility to translate the Australian IDs to the corresponding NZ IDs. Since all domains in Australia share the same key, the cross reference for AllBooks International has been kept the same as the Australian. Table 7.10 depicts a segment of the key mapping database:

<table>
<thead>
<tr>
<th>ABAustralia</th>
<th>AB International</th>
<th>AB New Zealand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mat123</td>
<td>Mat123</td>
<td>Science-3567</td>
</tr>
<tr>
<td>Chem934</td>
<td>Chem934</td>
<td>Science-2987</td>
</tr>
<tr>
<td>CSc402</td>
<td>CSc402</td>
<td>Computing-1784</td>
</tr>
</tbody>
</table>
• The New Zealand branch are keen to establish a benchmark for their own sales representatives. To this end, they subscribe to the modified New Order message:

\[
\text{New Order (order_id, customer_id, Order Line*, date, total_price, emp_no)}
\]

Since the employee ID is encrypted, the confidentiality of the identity of the salesperson is assured.

• For a period, AllBooks New Zealand also subscribed to the above message in order to feed their Data Warehouse (DW) system. However, since they are now replacing the system they are not interested in this information until the new DW is bedded down, and they have unsubscribed to the message. The scope of this change is restricted to the AllBooks NZ federation, which deletes the DW from the list of subscriber applications and the fragment from the list of fragments.

• The incorporation of AllBooks New Zealand to the “Sales” workflow will be soon required, since they keep a small collection of technical books that may be used to satisfy some of the sales when there is not enough stock in Australia. So the analysts have established an OR logic between Australia and New Zealand, and when the in-stock quantity does not satisfy the requirement, a simple request/reply new message type is defined — originating in Warehouse — to request for the missing stock. As a result of this enquiry, the Ready message is produced by putting together the stock available at both branches.

7.3 Analysis Methodology:
Extends Business Rules Methodology of Section 5.4

The Message Adaptors described are responsible for the interaction with the federation, under the assumption that they know how to react to each individual event. This is a result of the exchanges of the workflow layer being translated to a sequence of messages that follow the federation protocols. In turn, a workflow specification is the result of a business process that is in compliance with the rules that govern that aspect of the business. Hence, it is necessary to analyse and document all facets relevant to the business process that is the focus of the integration in a given integration context. The analysis should first reduce the problem size by the careful determination of the integration context (i.e. federation and domains involved), if that has not been determined earlier. Each integration context should comprise a relatively small number of domains where term semantics, rules and processes are to be shared.

As a consequence of the strict separation between domains, and domains and the federation,
ultimately all rules must refer only to the business terms of interest for the federation. Any other term should not be exposed to the federation space, but it should be kept private to the domains. Federation terms must be business terms relating to data shared by the domains, to actors — i.e. exclusively the domain themselves — or very importantly, to events of federal relevance occurring within domains since they generate federation traffic in the form of messages. Thus, the concept of an Order — originated at Sales — may be known at the federation level as a recognised Corporate Entity, and the New Order event is recognised by the federation and produces a corresponding New Order message, but the details of the New Order activity are unknown outside the Sales domain. The corporate entity Order may have fragments within Sales, Warehouse and Head Office if all of them have overlapping data pertaining to the entity, but there is no New Order entity. Hence, although the term New Order has a meaning for the federation, it is not as a workflow or a corporate entity, but as an event and its associated message.

### 7.3.1 Data-Oriented Analysis

**Integration Focus:** Determine the focus of the integration.

**Integration Context:** Establish the integration context.

**Ontology:** Semantics and ontology methodology:

1. Gather all business rules referring to the terms used in the integration context, related to the focus of the integration.
   - **General rules:** Rules valid for the whole integration context.
   - **Focus Rules:** Rules related to the focus of the integration.

2. Establish the terms related to data semantics used in the context related to the process that is the focus.

3. Determine whether each of the terms above is a common term or a business term.
   - **Common Terms:** Terms that don’t need a semantic description,
   - **Business Terms:** Business terms related to the process within the integration context.

4. From the business rules and the federation information, determine whether the business terms are of interest to the integration context (i.e. they are shared by domains in the federation), or they are only local, private to a domain. Classify the terms of interest to the federation as Domains or Corporate Entities and their fragments (excluding individual attributes).
   - **Shared Terms:** Refer to shared elements in the integration context.
Local Terms: Refer to local terms, private to a domain.

5. Model the shared entities related to the data semantics terms above to include individual attributes. Attributes private to a domain are not to be considered. Link them to the corporate ontology. If necessary, add their structure in a custom made ontology and link the entities and their attributes to the standard.

6. If necessary, rewrite the business rules so that local elements are not exposed to the federation level. Any local term private to a domain — and therefore unknown outside the domain — should be replaced by the domain name.

Existing Rules: From the original set consider the that expose domain elements.

Modified rules: Rewrite the rules replacing private domain elements by the domain.

7. Determine the shared fragment of the corporate entities, and allocate primary update responsibility of the different fragments to domains.

This analysis contemplates exclusively the data semantics rules of the integration context that is the focus of the integration cycle. However, as discussed in Chapter 3, it is also necessary to deal with the rules that relate to enterprise procedures.

7.3.2 Process-Oriented Analysis

The second phase of the methodology is to consider the process-oriented aspects of the integration. This is necessary to be able to map domain tasks and events to the flow of messages in the integration context. The following statements determine the corresponding business process, and a workflow that implements the business process:

Business Process: The following statements define the methodology to establish the business process:

1. Extend the model by considering also the terms not exclusively related to data semantics. For each term, determine whether or not the term refers to a procedure.
2. Allocate task-related terms to domains. Determine whether tasks are local or of interest to the federation.
3. Identify federation events. Classify as primary and non-primary (Please events).
4. For primary federation events, determine the federation inputs required by each domain to execute each of its tasks, and the output produced as a result.
5. For non-primary federation events, determine the federation inputs required by each domain to execute each of its activities, and the output produced as a result.
6. Establish the transactional behaviour of each activity: determine whether each activity is forcible, compensatable and/or undo-required.

7. Determine check points and commit points.

**Workflow:** Defining the workflow:

1. Define the message types and parameters.
2. Define the workflow by defining the message flow.
3. Map the workflow to federation flows.
4. Map the federation flows to a sequence on data fragments.
Chapter 8

Conclusions

This chapter revisits the problems experienced by AllBooks, and discusses how the federated approach help the company solve them. Section 8.2 argues how this dissertation has addressed the research questions.

8.1 AllBooks Hypothetical Revisited

This chapter revisits the hypothetical case of AllBooks International presented in Section 1.2.1, and shows how the federated implementation has addressed their original problems. The chapter also demonstrates how the research questions have been addressed by this dissertation, and presents concluding remarks and suggestions for future research.

1. **Original Problem:** An agent’s commission is processed at Human Resources, on a slow legacy system. The original design was based on a processing dependency which often timed out and had to be restarted. Replacing the central computer system was perceived as too expensive and risky, but there is also a new requirement to connect the sales representatives mobile computers to do the data entry and issue invoices directly when they are in the field. The old system would not be able to support this functionality.

**Federated Solution:** Human Resources is now subscribed to the New Order message from Sales and to the Ready (total or partial) message from Warehouse, so when both messages are received they have the necessary information to allocate the sales representative’s commission. This de-coupling means that HR can process the commission in their own time, and the Sale workflow is free to progress asynchronously. However, this means that it is possible that a commission may not be included in the current salary cycle, but that will be included in the next pay cycle instead. Since the HR application processes incoming messages within one minute of receiving them, the managers have decided that this is an acceptable delay from the business point of view.
In addition, the representatives are now creating new orders directly in the field and publishing the information so the sales workflow can proceed. There is a delay of up to thirty seconds between the published New Order message and the reception of the New Invoice produced by Head Office, but this is not causing any problems for the issuing of invoices.

2. **Original Problem:** After the merger, it was apparent that the Australian database could not be used to support the New Zealand business, given that NZ systems had their own data models, data formats and business rules, and these systems have to be kept to support local operations. In addition, the changes required to the Australian systems to support the NZ operation were assessed as too complex and expensive, and satisfying New Zealand data operations from Australia was perceived as having unacceptable performance implications.

**Federated Solution:** The Australian implementation experience of Message Adaptors is being transported to New Zealand, so they have established a simple federation of four domains, and have started their own federal highway. Thus, AllBooks New Zealand have kept their systems intact, and they are currently gradually subscribing to the relevant messages produced by the AllBooks Australia. Very soon they will start publishing messages for the consumption of the AllBooks International federation.

3. **Original Problem:** There was apprehension about AllBooks International plans to expand into South East Asia, due to the slow and unreliable communications and lack of local support for the more advanced applications.

**Federated Solution:** Given that all interoperation between South East Asia and Australia will be based on messages of which the MOM infrastructure guarantees delivery, AllBooks have decided that when the time comes they may batch South East Asia messages locally, only to be published at appropriate times of low communication line usage.

4. **Original Problem:** Most of the existing applications were interoperating by the use of files, database tables or direct writing, so the same strategy was used to link the remaining ones. Nightly processes performed the updates. The number of these intermediate structures and their required one-to-one format translations resulted in very difficult and expensive maintenance schedules.

**Federated Solution:** The decision to adopt a corporate format supported by a corporate ontology has greatly simplified this problem, and it is now on course to a complete solution. Firstly, the creation of integration contexts meant that only a reduced number of terms, rules and processes are to be taken into account when attempting an integration cycle.
Secondly, the use now of centralised semantic ontological descriptions greatly reduces the amount of information that needs to be stored, makes it much easier to update semantics specifications because they have to be updated only at one place, and provides a well documented target at which to aim when planning the next integration step. In addition, not only the number of connections has been reduced, but now introducing a new domain or application means that only one new connection has to be developed. The experience of the initial adaptors implementation has been very useful for subsequent implementations. Since the MOM infrastructure is very efficient and provides a very fast response, there is no need to do the updates at times of low usage and the messages are published as soon as the triggering event occurs.

5. **Original Problem:** There was tension between Australian and New Zealand units since their reports of total sales were taken at different points of the sale workflow in both countries, and that made NZ totals appear lower than Australian ones.

**Federated Solution:** Now AllBooks NZ subscribe to the New Invoice message of AllBooks Australia, and they tally total sales also from those messages. They know that all activities after this message are forcible, so their Australian totals are correct, and they also discount the 10% GST to be able to compare with their totals properly. They have also decided to take the average exchange rate from the previous 3 days to map between AUD and NZD, but this is private to NZ and it is subject to revision.

6. **Original Problem:** There was a lack of data consistency and integrity throughout AllBooks. Due to the existence of overlapping data fragments in different data stores, without a coherent plan it was not possible to maintain these segments — such as staff details and customer information — synchronised.

**Federated Solution:** As part of their integration effort, All Books have embarked in a gradual strategy to identify overlapping fragments and use the federation protocols to keep them synchronised. Firstly, every time they identify a process to be integrated they take care of the data sharing in the way indicated by the methodology. Secondly, they have identified many simple workflows where one application may be clearly designated as the primary application for a fragment, while all others are keeping their fragments synchronised by subscribing to the update messages. This has quickly solved the more urgent problems.

7. **Original Problem:** The impact of a system change was always extensive, and due to the processing dependencies between different modules a system could not function without all its components.
Federated Solution: The problems with the order data entry have been ameliorated, since the order system has now only an informational dependency with invoicing, so a non-working invoicing module does not affects order data entry anymore. Sales representatives in the field are able to take orders even when the invoicing module is unavailable, although they are not able — like everybody else in AllBooks while invoices are down — to produce invoices immediately as desired.

8. Original Problem: There were autonomy and privacy considerations troubling AllBooks. Because the different groups found themselves in direct competition with each other they wanted to have control of the information they share with the organisation.

Federated Solution: The notion of the contract has gone a long way to assuage the group managers’ fears. The now know what they will share and when, and they have understood that the stress on their systems will be minor, since the MOM infrastructure provides the transport. The modifications required to their applications are also minimal, with the Message Adaptors taking care of most of the interoperation effort.

In addition, managers are now able to negotiate with top management the encryption of some of the data shared with the federation, making sure that the information is only understood by the intended recipients.

9. Original Problem: Due to the high number of requests, central services often had a very poor response time. Staff in the different units were keen on keeping a significant degree of autonomy, since this meant that they would not need to rely on central systems for their day to day tasks, but that they would keep using their reliable local systems.

Federated Solution: In the AllBooks federation processing is restricted to the domains, so processing is local.

10. Original Problem: Due to its complexity and cost, AllBooks were reluctant to invest in an ERP (Enterprise Resource Planning) solution. They perceived the ‘cold-turkey’ approach as very disruptive for the units, and they were not convinced of the extensibility and flexibility of the solution, to satisfy not only current but also future needs. For example, it is not clear how such a solution would integrate with the new CRM (Customer Relationship Manager) package.

Federated Solution: The strategy of AllBooks has been based on a gradual implementation and testing of the federated architecture. Firstly, they created only five domains in Australia, and they identified a few processes and their corresponding workflows to implement in a first instance. The experience with the first Message Adaptors was very valuable and they have now a developed a framework that they intend to instantiate each
time with little modifications. Federated services were added as required, such as the key mapping and the encryption for their exchanges with New Zealand.

As for the new CRM package, the analysis determined that in its interactions with customers, the package will originate a high number of transactions, so the current thinking is to isolate the system as a new domain in the federation.

8.2 Addressing the Research Questions

Section 1.5.2 presented the research questions that we intended to answer in this dissertation. As with the problems faced by AllBooks, our discussion has addressed the research questions, as follows:

- **Classify the problems presented by the integration of enterprise knowledge.**

  Chapters 1 and 2 discuss the problems associated with enterprise integration. The hypothetical case of AllBooks International is presented to illustrate the problems that a large, IT mature enterprise may face for lack of, or poor approach to, integration. The most pressing problem — semantic integration — is presented and discussed based on the real life situation of a major Australian bank. Based on this analysis, we present a list of objectives that this research intends to address. In addition, we introduce the different facets of enterprise knowledge that are the focus of this dissertation, and discuss the requirements and constraints associated with the integration of these facets.

- **Identify the shortcomings of the current enterprise back-end integration strategies.**

  Several important shortcomings of traditional integration strategies are identified and discussed in this dissertation. Firstly, in general strategies only address data integration, rather than including other aspects of enterprise knowledge. Only Zachman’s Federated Enterprise Framework includes other aspects, but his framework is exclusively conceptual, and it does not address any analysis, design or implementation issues. Secondly, either integration solutions are not comprehensive, or suffer from high complexity and scalability problems, due to the volume of information to be considered by the strategy at any given time, the semantic heterogeneity present in the enterprise, the number of integration points, the complexity of the development and maintenance of the shared interfaces, the difficulty of an enterprise moving to the integrated scheme, and the lack of flexibility of the resulting architecture. We posit that centralised integration strategies are not suitable for a large IT-mature enterprise
such the ones considered in this research, and that a distributed approach should be used instead.

- **Establish conditions for an enterprise integration strategy to be successful.**

  In order to address the above shortcomings, in Section 3.1.5 we establish the conditions for a distributed integration strategy to be successful. We argue that, due to its size and complexity, the strategy should not cover all the data in the enterprise at once. Instead, we present a strategy that reduces the amount of information to be managed at one given time, and controls semantic scope by using ontologies associated with semantic local contexts. We also show in Chapter 5 how it is possible to provide economy of expression by using defaults, and support reuse by allowing semantic and context inheritance. Our ontological approach ensures that data semantics are documented to avoid misinterpretation, and that the inevitable semantic conflicts may be detected and locally reconciled. We outline how to implement a data-mapping service and data exchange infrastructure. Additionally, we present a plan that provides for flexibility, scalability and extensibility, to make possible a gradual transition to a new integrated structure.

- **Present and justify a federated approach to enterprise integration.**

  Based on the concepts of coupling and dependency, the federated architecture is introduced as a low-coupling architecture suitable as a vehicle for enterprise integration. Chapter 4 introduces the notion of transient inconsistencies, and demonstrates how this notion is to be used to determine federation domains, which later are to become the building blocks of the integration contexts. Importantly, Section 4.6.2 demonstrates how the federated architecture results naturally in a layered, recursive structure for the enterprise. This, together with the notion of integration contexts, makes possible a gradual evolution to an integrated enterprise. In this context, the chapter discusses the advantages of the federated approach to support system interoperation. Two real-life implementation case studies illustrate the need for, and characteristics of, a federated approach to enterprise integration.

- **Develop a set of federation services to support the integration strategy.** We introduce in Chapter 6 a set of services to ensure shared data integrity. The notion of Corporate Entities and their fragments makes possible to aggregate individual fragments into global identifiable entities. To this end, we define Key Mapping as a service that generates and maintains cross-references from a Corporate Entity to its fragments. To ensure that shared data is correctly updated, we introduce the concept of primary applications as having primary responsibility for updates. Together, the notions of Guaranteed Delivery, Corporate Format, Key Mapping and Primary Update Responsibility are used to determine four different levels
of federation infrastructure to support application interoperation, allowing the progressive implementation of the strategy, this time by gradually adding service levels.

- **Establish appropriate sharing protocols to ensure data integrity.**
  A set of standard federation flows and integrity protocols — primary and non-primary — ensure that fragments are updated correctly when shared data is updated. These protocols form the support layer for the implementation of the message-based FEW workflow model.

- **Develop a methodology to include business rules and processes in the integration strategy.**
  In Chapter 5 we establish the relationship between the three facets of enterprise knowledge considered, and introduce the notions of integration contexts. Based on the analysis of Sections 3.3 and 3.4, we determine a business rules classification appropriate for our integration analysis, in which data semantics are a particular case on rule. As a result, in Section 5.4, we present a step-by-step business rules methodology to include business rules in the integration strategy. This methodology is later extended in Section 7.3 to include together a data-oriented and process-oriented methodologies. An example of AllBooks International shows how to use the methodologies to analyse the “Sales” business process, and how to map the business rules and business process to a FEW business workflow.

- **Demonstrate an ontology-based approach for data integration.**
  As a result of our analysis on organisation business rules in Section 3.3, we introduce the concept of data semantics rules, which are the basis of our data integration approach since they represent the structure of the data. Even at its second lowest level, Level 1, a federation implementation, prescribes the adoption of a corporate format for federation messages. We illustrate how data semantics business rules are to be linked to an integration context-related ontology, by using a mix of standard and enterprise-specific ontologies. The resulting ontology is used as a semantic repository for domains to be able to map their data to and from the corporate format.

- **Develop a low coupling workflow model with transactional characteristics to effect process integration.**
  In Chapter 7 we introduce FEW, a federated, low-coupling, publish-and-subscribe workflow model suitable for a federated architecture. The model includes constructs to provide FEW with transactional characteristics, which allows the development of total and partial commit and abort protocols, and backward and forward recovery. This chapter also introduces the notion of quasi-consistency, which extends and refines the concept of transient inconsistencies
introduced in Chapter 4, and uses it to support the *FEW* model. Since a *FEW* workflow relies on the services layer of the federation to update data fragments of corporate entities, it assumes the existence in place of a federation Level 3.

- **Show how a federated integration strategy may follow the path of enterprise evolution.**

  The AllBooks example of Section 7.2 illustrates how the federated *FEW* model and the possibility of message reuse allow the evolution of a workflow to incorporate new developments with no disruption for existing processes. New domain nodes are easily added, and new functionality is integrated, to an existing flow by simple message changes.

### 8.3 Conclusions and Suggestions for Further Research

This dissertation presents a federated approach to enterprise integration based on a low-coupling interaction between cluster components. The main advantages of this approach are robustness, flexibility, scalability, progressive integration, and simple evolution. There are, however, several areas that should be explored further:

- The integration effort depends on the skill and knowledge of enterprise analysts. This expertise is not always available, either within or outside the enterprise. More support for the integration of rules and processes should be provided, in the form of methodology-implementing tools, supporting the management of business rules and workflows within the same framework.

- The development of standard XML-based specifications such as ebXML [36] may be very significant for the development of federations of integrated enterprises. This would make the integration effort of an individual enterprise compatible with others, so research should focus on how enterprise ontologies may use, and provide mappings to, these standards.

- The development of new frameworks such as Web Services may alleviate some of the most pressing administrative problems of distributed processing, such as finding a service and its corresponding details, including the proper call format, parameter list and network node of the interaction. Although the WS framework is more often related to Service Oriented interactions, since SOAP is a one-off, text-based message-based protocol, current industry efforts are focusing on defining an open interoperable wire protocol for reliable messaging based on the SOAP protocol. It is highly probable that MOM implementations could be based entirely on one of these specifications, to be able to handle behavior such as acknowledgment of receipt, message persistence, and recovery from failure. Consequently, research effort should be
directed to the use of the Web Services framework also for Information-Oriented interactions, since this would make possible to accommodate both the request-reply and the event-driven models within the same framework and technology.
Bibliography


Appendix

Some aspects of this dissertation have been published before, in collaboration with industry colleagues:


In all these cases, my contribution has been on the theoretical and conceptual aspects of the publications, whereas my co-authors’ major contributions have been on the industry implementation aspects. I consider that the contributions have been equally shared between the authors.

The following publication is the result of my collaboration with an academic (L. Zhao) and an industry colleague:


In this last case, my contribution also has been on the theoretical and conceptual aspects of the publication; however, L. Zhao had a significant input in the area of patterns. I consider that the contribution of this publication has been equally shared between the authors.