INSTITUTO SUPERIOR TÉCNICO
Universidade Técnica de Lisboa

A Software Framework for Enterprise Architecture Modelling

Pedro Miguel Carvalho Pinto

Dissertação para obtenção do Grau de Mestre em
Engenharia Informática e de Computadores

Júri

Presidente:	Professor Doutor José Manuel Nunes Salvador Tribolet
Orientador:	Professor Doutor Alberto Manuel Rodrigues da Silva
Vogal:	Professor Doutor André Ferreira Couto e Vasconcelos

Outubro 2009

Acknowledgements

I would like to express my gratitude to everyone who helped me throughout this year. Their support was of the utmost importance for achieving the final work presented at this dissertation, which allowed me to step into a new stage of my personal and professional life.

For his guidance and investigation, I will start by thanking my supervisor Prof. Alberto Silva, whose expertise and assistance were essential for taking this work into completion. I would also like to thank to David Ferreira and João Saraiva for all the ideas brought into discussion when planning and building this solution.

I would also like to express my gratefulness and appreciation to my mother, father and sister, whose presence, support and encouragement were paramount for getting me this far. For that, and all other sort of things, I thank them all.

A huge thanks to my friends, whose understanding and friendship allowed me to endure the hardships of writing this dissertation.

Finally, I would also like to thank the Instituto Superior Técnico for the learning opportunity and for preparing myself to the challenges of my work field.

Lisboa, October 21, 2009
Pedro Pinto

Resumo

Os Quadros de Referência Arquitecturais desempenham um papel importante na definição de princípios, recomendações, linguagens, vistas e pontos de vista nas descrições arquitecturais. O crescimento dos Quadros de Referência nas Arquitecturas Empresariais tem sido acompanhada pelo desenvolvimento e evolução de várias linguagens de modelação, que abordam um ou vários domínios empresariais. Estas Linguagens de Modelação são utilizadas na representação de informação através de um conjunto de conceitos e de relações. No entanto, a heterogeneidade dos domínios empresariais restringe a integridade e as relações entre estes elementos, ao longo dos modelos nos quais foram definidos. Actualmente não existe uma linguagem que consiga ultrapassar com eficácia tais limitações. Neste trabalho tais limitações são abordadas através da definição de um meta-modelo independente do domínio e de um conjunto de meta-modelos específicos ao domínio, construídos sobre uma infra-estrutura baseada num conjunto de normas utilizadas na descrição de Arquitecturas Empresariais. É por isso proposta uma solução que utiliza como referência o quadro de referência TOGAF, a linguagem UML, a notação BPMN, e a norma IEEE 1471-2000.

A solução proposta consiste numa abordagem prática a múltiplas limitações no conhecimento de uma Arquitectura Empresarial. São reforçadas as relações entre elementos arquitecturais (como vistas, modelos ou pontos de vista), assim como as relações entre entidades definidas nesses mesmos Modelos. Deste modo é possível obter uma visão holística do sistema a partir de quaisquer entidades, existindo indireção múltipla entre os elementos arquitecturais. No âmbito do projecto TEA4U foi desenvolvido um *toolkit* sobre o Sistema de Gestão de Conteúdos WebComfort.

Abstract

Architectural frameworks have an important role in establishing principles, recommendations, languages, views and viewpoints for architectural description. The growth of enterprise architecture frameworks has been followed by the development and improvement of several modelling languages, addressing one or more enterprise domains. Modelling Languages are used to represent information through a set of concepts and relationships. However, the heterogeneity of enterprise domains constraints the integrity and relationships between those elements, across the models on which they are defined. Currently, there isn't a language capable of effectively surpassing such limitations. In this work, both issues are addressed through the definition of a domain-independent metamodel and a set of domain-specific metamodels extended from the former, built upon a standard-compliant infrastructure. A solution is proposed that uses as a reference the TOGAF framework, both the UML language and BPMN modelling notation, and the IEEE 1471-2000 architectural framework standard.

The proposed solution is a practical approach to several knowledge limitations of an Enterprise Architecture. The relationships across architectural elements (such as Views, Viewpoints and Models) are enhanced, as well as the relationships across the entities defined within those Models. By focusing primarily on those entities, a holistic view from the modelled enterprise can be achieved from any entity, making available multiple indirection mechanisms across the architectural elements. By using the WebComfort Content Management System, a toolkit was developed for the TEA4U project.

Palavras Chave
Keywords

Palavras Chave

Quadro de Referência Arquitectural
Linguagem de Modelação
Meta-modelo
Ferramenta de Modelação Empresarial

Keywords

Architectural Framework
Modelling Language
Meta-model
Enterprise Modelling Tool

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The need for reliable structures as a starting point for building complex systems has emerged in many knowledge areas, either scientific or social. In the business world, an enterprise relies on a more or less formal enterprise architecture [51], which holds a coherent whole of principles, methods, and models, which are used in the design and making of an enterprise's organisational structure, business processes, information systems and infrastructure [21]. Each subject, or domain, can be described through software tools which use one or more modelling languages for representing enterprise concepts and relations. The coherent grouping of those elements forms a view upon a portion of the Enterprise being modelled, conforming to a set of rules and concerns addressed by different stakeholders.

1.1 Context

A number of frameworks for designing and organising system architectures have been proposed, being mostly oriented for software [44]. However, some of them have been adapted, and newer ones have emerged in order to deal with a wider scope. Those frameworks have an important role in establishing principles, recommendations, languages, views and viewpoints for architectural description.

The growth of enterprise architecture frameworks has been followed by the development and improvement of several modelling languages, addressing one or more enterprise domains. These languages distinguish themselves by the expressiveness of their concepts and relationships, their conformance to well-established standards and their ability to integrate and interrelate their elements across enterprise views [21]. However, their heterogeneity is an obstacle for the convergence for a flexible and integrated solution.

Enterprise architecture standards also arose in recent years [18] for establishing a set of commonly accepted architectural elements, defining the rationale and the relationships for elements such as Views, ViewPoints or Models. Nevertheless, their recommendations are not yet widely apprehended within the different architectural description practises. Therefore, the knowledge of the relationships between all enterprise elements becomes limited.

1.2 Problem

As it will be further detailed in the forthcoming chapter, there is a great number of domains, views, modelling languages and notations for describing an enterprise architecture. Their adequacy to the enterprise which is being modelled is largely determined by the industry, the compliance to standards and the intended detail level of architectural description. By the same vein, the ability of several modelling languages to model their target domains must also be assessed. Their expressiveness and adequacy is

also determined by their compliance to standards, by their defined elements for each target domain, their underlying syntax and semantic, and their ability to provide coherent and integrated views. However, there is not an explicit concern on tracking their modelled elements across architectural artifacts. One reason for such limitation is mostly due to the lack of a common structure. As a result, the maintenance of integrity and communication between elements across views is greatly impaired. Nevertheless, tracing those elements cannot be fully achieved only through the usage of a common structure between models. The need of an infrastructure is paramount for mapping the relationships not only between modelled elements, but also between the architectural elements.

Given the above situation there are four questions to address:

- Which enterprise domain division should be used for an industry-neutral approach?
- Which views and viewpoints should be used to address the most common enterprise domains?
- Which concepts and relationships should be used in the representation of different elements across enterprise domains?
- How to address a stronger communication between elements, and across views, while maintaining their flexibility and extensibility?

1.3 Thesis Statement

This thesis states that it is possible to enhance both the integrity and the relationships between elements across multiple enterprise domains, by means of a domain-independent meta-model and a set of domain specific meta-models extended from the former, built upon a standard compliant infrastructure.

The solution must satisfy the following requirements:

- To be industry independent.
- Promote flexibility and extensibility.
- Have enough expressive power when compared with the available modelling languages or architectural description languages.
- Enable navigability across elements and views.

This work proposes a set of meta-models that fulfil these requirements, as well as an infrastructure for an enhanced knowledge of the usage and state of the enterprise elements.

1.4 Methodology

The evaluation of the proposed solution will be done according the Design Research methodology, which states a five step process for the following assessment [47]:

- **Awareness of Problem** - Addressed at section 1.2, where it is stated the lack of integration and communication between elements and across views represented through modelling languages.
- **Suggestion** - Addressed at section 1.3, and followed by a tentative design in Chapter 3, where are presented a domain-independent meta-model and its elements, along with a set of domain-specific meta-models realising it. For those meta-models are proposed a set of elements, a core set of views, and a solution for managing the relationships between the defined elements. The final solution will address the following aspects:

1. The integrity of each defined element, i.e., the avoidance of element overlapping across views;
2. The ability to navigate across elements through all the defined relationships;
3. The ability of switching across views;
4. An enhanced knowledge regarding the state, values, and relationships of those objects, independently of the context in which they are defined.
5. The conformance of the meta-model with real enterprise data;
6. The ability to extend the organisational awareness about its enterprise objects;

- **Development** - This step will be carried out by the development of a prototype in WebComfort, which is a Content Management System. An overview of the prototype's workbench is shown in Figure 1.1. The workbench is composed by the following sections: *Management*, for the creation and edition of entities across architectural artifacts; *Analysis*, for the definition of tools which will appropriately output the content of each view for the relevant stakeholders; *Work Area*, for operation and visualisation of the solution; and a *Graphical Representation*, where will be shown the relationships of the selected element or view.

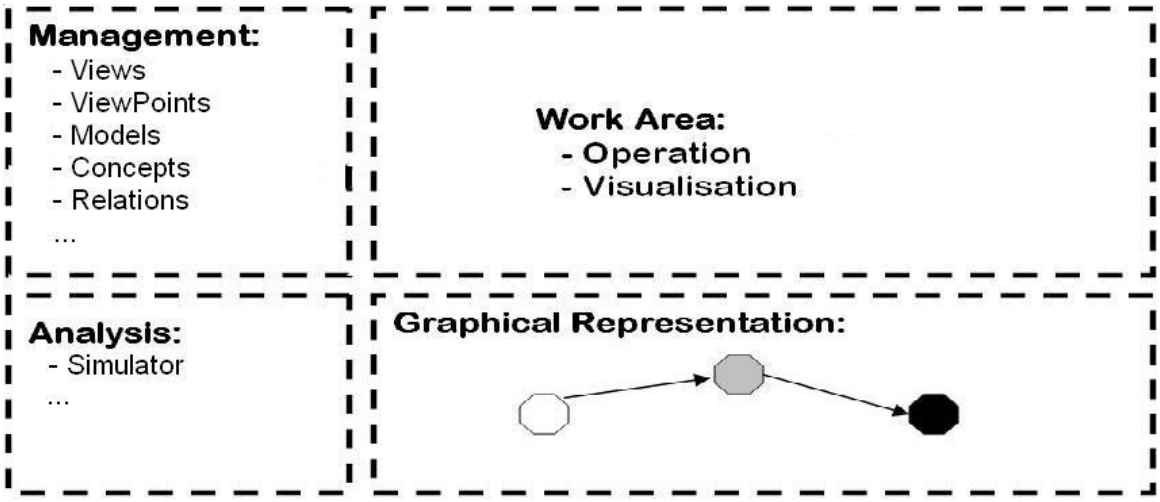

Figure 1.1: Prototype's Workbench Overview

- **Evaluation** - The evaluation will be done at two levels. Firstly, by the design and implementation of the prototype described above. At this stage will be carried on the accomplishment of the aspects 1, 2 and 3. Secondly, by the design, definition and analysis of a business scenario tackling real enterprise concerns, for achieving the remaining aspects.

- **Conclusion** - Consolidation of results and adjudgement of the evaluation.

This methodology is iterative and allows the repeatability of previous steps. Therefore, new knowledge is generated and accumulated, which allows to refine the solution for the achievement of all the proposed goals.

1.5 Outline

This dissertation has the following organisation:

- Introduction - The present chapter, which introduces the subject of study, the problem, the thesis statement and the methodology for validation.

- State of the Art - Covers the work and literature related to the subject of study.
- Solution Overview - Provides a high level description of the proposed solution and describes the motivation and goals behind each component.
- An Enterprise Architecture Prototype - Presents the prototype developed for this dissertation's purpose.
- Validation - Presents a simulation of a case scenario within the developed prototype, showing how the knowledge limitations across enterprise elements can be surpassed.
- Conclusion - Summarises the main contributions of this dissertation, and provides some insights for future work.

Chapter 2

State of the Art

This paper delves into a problem that has been addressed in multiple areas within Enterprise Architectures. Hence, it's necessary to perform an analysis of the related work which can be useful to accomplish the proposed solution. This analysis covers all the work discovered during the reading process. To my knowledge, no other works are aligned with the scope and objectives proposed in this thesis. Therefore, the following analysis covers all the work which contributed for the development of the aforementioned solution. For the sake of clarity, this section will be broken down into several sections. Each section will account the differing elements which should be taken into consideration, also featuring a summary of the main lines of discussion.

In section 2.1 the most popular Architectural Frameworks are presented . Taking into account the meta-model requirements, a summarised description of each framework is presented, along with an analysis which underpins its architectural description, with its viewpoints, views and recommendations, and also the framework's ability for being industry-independent.

Section 2.2 presents Modelling Languages. It's not intended to cover the available tools, but to analyse their underlying languages. Therefore, the strengths and weaknesses of several languages are covered, with the purpose of studying their expressiveness in representing elements in their target domains.

Section 2.3 presents Architecture Description Languages (ADLs). Their practicability and their ability in achieving a coherent enterprise modelling is analysed.

Finally, in section 2.4, the traceability solutions which can be applied to the meta-model are described, in order to study different ways of managing and improving relationships between elements.

2.1 Architectural Frameworks

Architectural Frameworks identify architectural views and viewpoints, along with a number of modelling techniques. They do not directly provide the concepts and relations to model, but define conceptual domains and aspects, which in turn link them closely to certain modelling languages. Therefore, the choice of which Architectural Framework to use undoubtedly affects the expressiveness and usefulness of the meta-model presented in this paper. Several architectural frameworks have been proposed in recent years, such as the Integrated Architecture Framework [5] from Capgemini, the Enterprise Architecture [30] from Microsoft, the Enterprise Architecture Framework [13] from Gartner, or the CEO Framework [41] [48] [49] from INOV. For the scope of this dissertation, the domains, views and viewpoints on widely known and used architectural frameworks from the study of [44] will be analysed. The modelling languages referred in this section are further studied in Section 2.2.

One of earliest works regarding information system architectures is the Zachman Framework [52] [42].

It is a widely known and used framework, with focus on the analysis and modelling of enterprise architectures. The framework establishes six types of perspectives - scope, enterprise, logical system, technology, detailed representations and functioning enterprise -, each one representing a different enterprise level. Within each perspective, the information is characterised by answering six interrogatives - what, how, where, who, when and why - which identities its aspects - data, function, network, people, time and motivation. The crossing between the perspectives and the aspects results in an architectural design element, used as a view for a particular group of stakeholders.

The framework's classification and categorisation allows a simple and comprehensive knowledge of the enterprise. However, its large number of views leads to applicability issues. In addition, it doesn't directly specifies viewpoint representations, nor modelling languages or notations. Thus, it doesn't explicitly support non-functional requirements or architecture evolution, doesn't distinguishes between modelling from design activities, and lacks an architectural description process.

From the Zachman Framework several architecture-oriented frameworks have emerged. One of them is The Open Group Architectural Framework (TOGAF) [46]. TOGAF provides a method and a set of tools which support the development of enterprise architectures. Its main goals are the building, design and evaluation of enterprise architectures. The framework is composed by the following components: the TOGAF Architecture Development Method (ADM), which specifies a iterative sequence of steps for developing enterprise architectures; the Enterprise Continuum, which is a virtual repository of models, patterns and architectural descriptions; and the TOGAF Resource Base, which specifies a set of resources, templates and guidelines for using TOGAF.

TOGAF identifies several architectural viewpoints which can be depicted in two major categories: Business Architecture, which addresses the system user's concerns, describing the business processes and information; and Technical Architecture, which addresses the technical concerns regarding the development, acquisition, and system operation, encompassing system and software engineers, system administrators, system managers and procurement staff.

TOGAF has well-defined activities which address both enterprise and individual system levels, being comprehensive and standard-compliant. Regarding the representation of views and viewpoints, TOGAF recommends several modelling languages for each domain.

As the US Department of Defence had also the need for a framework capable of addressing its business operations and processes, it developed the Department of Defence Architectural Framework (DoDAF) [9]. DoDAF has architecture development techniques aiming at the specification of processes for analysis and documentation of architecture objectives, requirements definition, scope definition and data collection. Its architectural domains focus on operations, systems and technology, which resemble the ones defined in TOGAF. The architecture description is handled by the Core Architecture Data Model (CADM), which is a taxonomy that defines views and their elements, and by architecture products, describing views with text and UML.

Through CADM, DoDAF provides traceability of the operational requirements and design decisions. Its views and viewpoints are mainly represented through tables, Entity-Relationship Diagrams, IDEF and UML. However, some design processes are dependent of the defence operations domain. Additionally, the framework doesn't tackle software configuration modelling, and lacks support for non-functional requirements modelling.

By the US Federal government the Federal Enterprise Architectural Framework (FEAF) [7] was developed, aiming at the interoperability of processes, and information sharing among government entities. The framework has as its main goal the architecture planning, thus being organised in four hierarchi-

cal levels: the first addresses architecture drivers, external stimulus and strategic direction; the second deals with the architecture business and design drivers; the third focus on business, application and data views; and the fourth uses enterprise architecture planning methods with the categorisation of the Zachman Framework. Within FEAF, three architectural domains are addressed: Enterprise, with strategy concerns and aimed for a wider audience; Segment, regarding business and business owners; and Solution, regarding the operational outcomes for users and developers.

One main characteristic of FEAF is the support of architectural evolution. Some modelling tools enable the representation of FEAF views and viewpoints, such as the Troux Transformation Platform, from Troux Technologies, which uses BPMN and UML. However, it doesn't fully describe non-functional requirements nor supports software configuration modelling.

The need for better specifications for large distributed systems led to the development of The Reference Model for Open Distributed Processing (RM-ODP) [33], which defines a framework that integrates a set of ODP international standards. Its main goal is the accomplishment of information services in an heterogeneous IT environment, crossing multiple organisation domains. It is composed by four parts based on the object oriented paradigm, namely: Reference, Foundations, Architecture and Architecture Semantics.

The Architecture part of the system defines five viewpoints: the Enterprise Viewpoint, which deals with high level enterprise requirements; the Information Viewpoint, which focuses on information structures and semantics; the Computational Viewpoint, which addresses system decomposition, object constraints and interactions; the Engineering Viewpoint, which deals with functions and mechanisms that supports interactions between distributed objects; and the Technology Viewpoint, which dwells in the technological artifacts that support the system's infrastructure.

The RM-ODP is a standardised, complete, consistent, but complex model, for the specification of system architecture design. Its views and viewpoints are represented through UML diagrams. However, it doesn't address change in the enterprise architecture, nor describes an architecture process and the detail level for architecture modelling.

Another framework for modelling software architectures is the 4+1 View Model of Architecture [20]. Its main goals are the architecture analysis and software systems modelling. The architecture models are represented through four views: the Logical View, supporting functional requirements; the Process View, focusing on non-functional requirements, such as design's concurrency and synchronisation aspects; the Development View, addressing software modules and their environment; and the Physical View, which focus the software's deployment. Additionally, there is the Scenario View, which supports discovery and verification of the above views through use cases.

The 4+1 View Model of Architecture uses an iterative approach for analysis and decomposition of design issues into smaller ones. Thus, its focus addresses key development issues of the system being built. Each view is represented through specific notations, where the Booch notation is included. However, as a framework, it doesn't describe the detail level for architecture modelling.

Summary

The most relevant aspects from the above mentioned architectural frameworks are summarised in Table 2.1.

Regarding scope, the RM-ODP and 4+1 View Model Architecture are aimed particularly at software architecture development, being RM-ODP most focused on large distributed systems. Zachman, TOGAF,

Table 2.1: Overview of Architectural Frameworks

Name	**Scope**	**Views and/or Domains**	**Recommended Viewpoints Representations**
Zachman Framework	Industry-independent enterprise architectures	View for each architectural element (cell)	None
TOGAF	Industry-independent enterprise architectures	A collection of Business and Technical Views	Several, such as BPMN or UML
DoDAF	US Defence enterprise architectures	A collection of Operational, Systems and Technical Views	Tables, ER Diagrams, IDEF and UML
FEAF	US Government enterprise architectures	A collection of Enterprise, Segment and Solution Views	BPMN or UML
RM-ODP	Software architectures for Large Distributed Systems	Enterprise, Information, Computational, Engineering and Technology Views	UML
4+1 View Model of Architecture	Software architectures	Logical, Process, Development, Physical and Scenario Views	UML

DoDAF and FEAF focus on enterprise architecture planning and system's interoperability. However, the Zachman Framework doesn't focus on enterprise architecture evolution, and DoDAF and FEAF scope is restricted to the US defence and government domains.

Concerning their views and viewpoints, all frameworks have defined them accordingly to their scope. The difference between domains from one framework to another stems from the amount of information that each domain encompass. However, in its essence, these domains share similar concerns. Hence, the difference relies more in their clarity and easy of use.

Accordingly with the representation of views and viewpoints, the Zachman Framework doesn't specify any language or notation, whereas the remaining frameworks recommends a set of languages and notations for one or more domains.

2.2 Modelling Languages / Notations

Each architectural framework deals with the domain division which fulfils its purpose the most, and it remains flexible enough to adapt itself to the system being modelled. This system is represented through modelling languages, whose main purpose is the definition of concepts and relations in one or more domains. Therefore, an analysis of widely known modelling languages for multiple domains is henceforth presented.

The need for a language for general purpose modelling led the US Ministry of Defence to develop the Integration DEFinition (IDEF) languages. IDEF is a family of modelling languages [29] [26] [32], which originated from the US Ministry of Defence, and that covers a wide range of general purpose modelling and diagramming techniques. There are sixteen IDEF methods, but the most commonly used are: IDEF0, for functional modelling; IDEF1/IDEF1x (the core) for information and data modelling; and IDEF3, for process description.

As stated in [21], the IDEF0 functional model is composed by: the activity, which is represented by boxes, inputs, outputs, and controls; and mechanisms, which carry out the activity. The inputs, controls,

outputs and mechanisms are referred as ICOMs, which can be decomposed into more detailed levels of analysis.

The decomposition is also used in the IDEF3 process model. This model leads with process description by means of two diagram techniques: process flow, which captures the information of how things work through one or more processes; and object state transition network, which captures the transitions available for an object in a process.

IDEF supports the modelling of multiple architectural views. However, it doesn't provide communication mechanisms between models, doesn't have a formal meta-model description nor formal semantics.

Within the business architecture modelling area, it is noteworthy the Architecture of Integrated Information Systems Toolset (ARIS). It covers process design, workflows, management and application processing [36], and describes the dynamics of business processes, by focusing in the organisation and business modelling.

ARIS provides charts, diagrams and a clearly defined graphical notation. It also provides a modelling language called Event-driven Process Chains (EPCs), which is a ordered graph of events and functions, that allows decomposition, but not extensibility. The ARIS Toolset specify several process charts and diagrams: the value-added chain diagram, the organisational charts, the interaction diagrams, the function trees and the EPCs.

The EPCs have a formal syntax definition, however they lack a precise formal semantic [21], as well as a precise meta-model description. The multiple views provided by ARIS have integration issues between them, thus being limited for architectural design.

The existence of several modelling languages and notations led to the development, by the Object Management Group, of The Unified Modelling Language (UML) [4], which is one of the most important standards for general purpose modelling. UML aims at the specification, visualisation, construction and documentation of artifacts in software systems. However, its use has expanded through other areas such as architecture modelling [21].

The reason for such expansion is explained by the simplicity of its constructors. It specifies a component called object, and a connector called link, capable of representing anything in any domain. The objects have a static part, regarding its structure, and a dynamic part, as regards its behaviour. The links represent any kind of connection between objects.

The newest version of UML specifies a collection of thirteen diagrams, which can be classified as: structural, where are defined diagrams for packages, classes, objects, composite structures, component and deployment; or behavioural, where diagrams for use cases, states, sequences, timings, communications, activities and interactions are defined.

Each diagram represents a view and has a specific notation. The views are interrelated with the UML meta-model concepts, leading to an overlap of concepts across different views, which promotes a richer expressiveness, but also readability issues.

UML also offers the Object Constraint Language (OCL), which is a language for setting constraints between model elements. It allows for the extensibility of their elements through stereotypes, and of their properties through tagged values. Additionally, it allows for the creation of profiles, which aggregates components and connectors with precise meanings for specific problems. UML has a formal syntax, but lacks a well-defined semantic.

For the establishment of a notation for business modelling, the Business Process Management Initiative (BPMI) [31] - now part of the Object Management Group (OMG) - developed the Business Process Modelling Notation (BPMN). BPMN is a standard which aims for the process design, business process

modelling techniques and business modelling tools. BPMN isn't a modelling language, but a notation which led several languages to review their elements and notation. Such examples are the ARIS EPCs, IDEF and even UML Activity diagrams.

The BPMN has a uniform graphical notation for modelling business processes in terms of activities and their relationships [21]. However, it only defines a formal syntax, lacking semantic and a formal meta-model.

Summary

The most relevant aspects from the above mentioned modelling languages are summarised in Table 2.2.

Table 2.2: Overview of Modelling Languages/Notations

Name	**Scope**	**Views**	**Formalisms**
ARIS	Business and Processes	Business, Application and Technology	Formal syntax, without formal semantics and a well-defined meta-model
IDEF	Business and Processes	One view per defined language	Without a formal syntax, as well as without semantics and a meta-model
UML	General Purpose	Multiple views through each diagramming technique	Formal syntax, without a well-defined semantics and a well-defined meta-model
BPMN	Business	Only for Business Behaviour	Formal syntax, without formal semantics and a meta-model

Regarding scope, the BPMN, ARIS and the most used IDEF languages aim at the organisation and business process modelling. UML is primarily focused on software systems, but has turned to be an expressive general purpose modelling language.

Concerning their domains, the BPMN, ARIS, IDEF1 and IDEF3 encompass several perspectives and their correspondent diagrams within process modelling, being all of them well suited for their intended purpose. UML diagrams allows not only process modelling, but also the modelling of any possible domain with structural and behavioural concerns.

In the matter of formalisms, most languages have a formal syntax, but none of them tackle a strong semantic definition. A formal meta-model is presented in UML, but doesn't have a clear separation from the elements found in its views. However, it is more refined than the meta-model found in ARIS, and better than the inexistent one at BPMN and IDEF.

2.3 Architecture Description Languages

The Architecture Description Language (ADL) refers to a formal language which describes a software architecture in general terms [21]. ADLs define generic level concepts, which can be more focused in structural or in behavioural aspects. Their high abstraction can be an obstacle for their applicability, however their formal basis make them suitable for the definition of specific concepts.

ADLs tackle components and connectors regarding their nature, properties and semantic. The ADLs can be split in two categories: software architecture, or enterprise modelling.

Within software architecture, the ADLs aim at the description of components and connectors to software developers and engineers. In [28] several languages have been compared and discussed. Some of the ADLs described are: ACME, which aims at architectural information interchange [12]; C2 [27], which

focus on architectures for evolutive, distributed and dynamic systems; Darwin [25], for architectures regarding distributed systems, but with more formalisms; and Rapide [23], focusing the architecture for describing system's behaviour. These ADLs have a high formalism and abstraction level, thus are more directed for technical users. Its nature renders them unsuitable for communication at the organisational level.

Within the enterprise modelling community, there isn't yet any well-defined architectural description language. However, there is a modelling technique which is half-way of being an Enterprise ADL: the Archimate.

In [19] is described the Archimate, which is an enterprise modelling technique from the Open Group. It presents an integrated architectural approach which describes and visualises the different business domains and their relations. It uses a common language, based on UML, for describing business processes, organisational structures, IT systems and technical infrastructure. It's based on the IEEE 1471 standard, and aims primarily to the alignment between domains (another subject of great study, as can be seen in [40]). It states that domains are not approached in an integrated way, which places obstacles for an impact analysis of design choices and changes. Therefore, it proposes a language and visualisation techniques for tackling enterprise domains and their relations.

Archimate has an architectural framework inspired by the Zachman Framework, presenting three aspects and three layers. The aspects are structure, behaviour and information. The behaviour is shown by structural elements, which in turn exchange information. The three layers are business, application and technology, which are an enterprise common layered approach. The crossing between the perspectives and the aspects encompasses one or more domains.

The conceptual domains proposed are: Product, which describes products or services; Organisation, which describes actors and roles; Process, for business processes description; Information, tackling the relevant information from a business perspective; Data, for information suitable for automated processing; Application, for software applications; and Technical Infrastructure, for hardware platforms.

In order to achieve its goal of domain integration, Archimate proposes an inter-layer relationships. It is used the concept of "service", which acts as an interface between architectural domains.

2.4 Traceability Approaches

In general terms, traceability can be defined as any relationship that exists between artifacts involved in the software-engineering life cycle [1]. The concerns on traceability have its origin within the requirements management community. Therefore, some of the aforementioned architectural frameworks approach traceability within the context of the mapping of operational requirements to design decisions.

The most basic solution consists on the simple linkage between artifacts [1]. However, this solution doesn't take into account semantic, which doesn't allow for an effective trace analysis. This problem can be circumvented by using predefined relationship types. In [34] two reference models are proposed, aiming at different types of users: the low-end users, which are provided with a set of link types, and the high-end users, which are provided with a richer set of link types. In [22] the use of traceability meta-models has also been discussed.

Regarding linkage information storage, several solutions such as [45] or [16] opt to save the information within the artifacts. This leads to maintenance and consistency issues when performing link updates. In [39] another approach is found. The information is saved in a metadata repository, which allows for the mitigation of the issues described above. However, it is required that all artifacts be uniquely identified.

Regarding linkage creation, some research has been done which aims at the automatic creation and maintenance of linkage information, ranging from information retrieval techniques [14] to inferred relationships [39]. However, their manual creation and maintenance is usually necessary, specially when dealing with enterprise changes. Several approaches have been attempted. In [8] an event-based solution is presented, in which artifacts are linked through publish-subscribe relations. The artifacts subject to change take the role of publishers, whereas the dependent artifacts take the role of subscribers. An event server it used to widespread the notifications to the appropriate receivers. In [11] a consistency management solution is proposed, by means of reference objects and consistency relations. Each artifact has an associated reference object. Each reference object has consistency relations defined with other reference objects, with all the semantic needed. There are also configuration management solutions which tackles the complex task [50] of artifact versioning.

2.5 Enterprise Architecture Support

The architectural frameworks and the languages studied in the previous sections have shown different approaches for dealing with different enterprise contexts. The recommendations given by the former and the characteristics of the latter aim for the most reliable and truthful representation of the artifacts which compose an enterprise architecture. However, for achieving a high degree of relationships between the artifacts modelled, neither the best domain division nor the most appropriate modelling language are enough.

The existence of an appropriate infrastructure, which supports the enterprise architecture modelling activity, is needed. Its planning and design must be taken into consideration not only for the extensibility, flexibility and expressiveness of the artifacts being modelled. Those artifacts must also present characteristics useful for the future users, the stakeholders. Therefore, a set of additional requirements must be taken into account:

- Navigability across artifacts. Not only in an hierarchical fashion, crossing different enterprise domains, but also across models, and views. Therefore, it must support both vertical and horizontal navigability.
- Knowledge of the artifact's usage in an enterprise-wide context. The enterprise architecture must give not only an holistic view of the system, from a top-down perspective, but also from a bottom-up perspective, taking as a starting point a single artifact.

Such requirements are not directly addressed by modelling languages. The enterprise architecture frameworks do not tackle them either, because those features are not in their main concerns. The support of such features is only taken into account when analysis and manipulation activities are required upon the enterprise entities. Therefore, such concerns are only tackled within the scope of enterprise architecture tools [24].

An enterprise architecture tool often provides support for manipulation and analysis of the defined models. However, the type of support is often bound to the modelling approaches offered by the tool.

In this section the entity properties are studied, as well as the analysis and manipulation capabilities from an arbitrary set of enterprise architecture tools, taken from the study made by Schekkerman [37], which can be seen in Appendix A. After an analysis of their infrastructure, an approach aligned with the scope of this dissertation will be presented.

2.5.1 Analysis and Manipulation of Entities and Models

The support of each tool is often bound to its modelling approaches. As studied in section 2.2, each modelling language addresses one or more enterprise domains. Therefore, an enterprise architecture tool is conditioned by its language support.

Regarding the business domain, or enterprise model as described in the Zackman Framework, an assortment of tools are found. From the IDS Sheer the ARIS Toolset is presented [17]. The ARIS Toolset is a set of tools aimed at the business process management, offering functionalities for design, analysis, implementation and optimisation of business processes. This tool uses the ARIS framework language, which was studied in section 2.2. Each entity within the framework has several properties allowing performance analysis, in both time and cost dimensions, and also allowing process simulation. Nevertheless, those properties are tightly linked to the business behaviour.

For tackling not only business concerns, but also the information systems and technology domains, the Enterprise Architecture from Sparx Systems [43] is presented. The Enterprise Architect is an UML analysis and design tool which covers several stages, such as requirements gathering, design models, testing and maintenance. The entity infrastructure presents a set of properties used mainly for testing and maintenance support.

Regarding the business and information systems domains, the Rational Software of IBM [15] is presented. This tool is a software engineering process based on UML, providing an extensive set of visual modelling tools. The comprehensive feature list of this software addresses support for modelling and visualisation of business processes and system requirements, as well as database support for object-relational mapping and schema generation.

Concerning the technology domain and functioning enterprise, the toolset from BiZZdesign [3] is presented. Its toolset aggregates a set of tools for design, analysis and management of business process models. The tool supports the qualitative and quantitative assessment of properties for business process designs.

Dealing with enterprise entities are not only addressed by enterprise architecture tools. As stated in [38], repositories are used for maintaining the developed models and their underlying entities, having a significant impact on the overall functionality, scalability and extensibility of an enterprise architecture tool. For instance, the BiZZdesign solution has its own repository for tackling model management.

A more complete example can be studied from ASG-Rochade [2], which provides a meta-data repository for collection, management, control and reuse of entities. This solution provides a set of functions and applications for accessing the repository in many ways. One of the its applications allows the navigation through single artifacts, attributes and relationships, allowing further visualisation and information manipulation.

2.5.2 Entity Underpinnings

The analysis and manipulation abilities of each tool are subject to limitations by the properties defined at each entity. Depending on the architectural domain addressed by the solution, a different array of properties will be available. However, as studied in the previous section, not only the entity properties lead to particular analysis or manipulation activities, but also the way in which the entities are kept at a database. The planning and design of a repository is of the utmost importance, not only for entity access, but also for allowing complex queries upon the entities, leading to extended functionalities and interactions between them.

Therefore, for an enhanced knowledge and enhanced relationships between entities, three aspects at the planning stage of a solution must be considered:

- The architectural domain addressed by the solution settles a set of entity properties which are used for analysis and manipulation activities upon its scope.
- The choice of properties attached to each entity predetermines the available functions upon the entities modelled within the architecture.
- The design of a repository for all the enterprise architecture entities is paramount for the expressiveness and richness of the information pulled out from each entity.

Chapter 3

Overview

As stated in section 1.2, there isn't a widespread concern regarding the construction of a coherent domain-independent meta-model, with a set of concepts and relations, ready to be fully realised and extended across domains. The multiple integration and communication issues which arise between models are a good example of such omission. In this Chapter, a proposal for those problems is presented. This proposal encompasses a domain-independent meta-model, with a defined set of concepts and relations, that will be accomplished by a set of standard elements from well-known modelling languages. Therefore, a framework of reference is used to address the relevant views and viewpoints in an enterprise, as well as the logical division of its domains. The solution also presents a traceability mechanism tightly connected to the meta-model for enhancing the integrity and communication between elements and across views.

In this Chapter the overall architecture is explained. In Section 3.1 the domain-independent meta-model is presented, with a collection of generic concepts and relations. Section 3.2 will outline the assumptions that must be respected in order to achieve the proposed goals for the meta-model. Section 3.3 will present the domains and views to be tackled, as well as the domain-specific meta-models and their elements, which will realise the ones defined in the meta-model. Finally, in Section 3.4 a traceability solution to be coupled with the meta-models' elements is proposed.

3.1 Meta-model

The proposed enterprise meta-model tries to answer the six primitive interrogatives found in the fundamentals of communication, as in the Zachman Framework. These interrogatives gather the characteristics of non-human systems - what, where, when, how - and human systems - what, where, when, how, who, why. Therefore, it presents the conditions for questioning both organisational and technological domains.

Consequently, for each interrogative an answer is proposed, which can be summarised in a single word, and whose meaning can be mapped into a generic concept. Each concept must be categorised in a type, in order to represent what part of the system is being described. In Section 2.2 the different modelling languages were analysed, regarding their expressiveness and domain-scope. From all those languages, UML is the one that presented the most flexible and extendible set of elements and diagrams for general purpose modelling. Taking into account its current importance and relevance, it is chosen as the language of reference for representing the meta-model's concepts, relations and types. Therefore, the generic concepts derived from the answers of the six interrogatives can be classified as being: structural, describing a static part of the system; or behavioural, describing a dynamic part. Table 3.1 summarises the classification of those concepts.

Besides the definition of those concepts, it is necessary to establish a logical association between them. It follows the proposed associations:

Table 3.1: Answers to the six-interrogatives

Interrogative	Answer	Meaning	Type
Who	Someone or something with the ability of realising things	Actor	Structure
Why	The reason which leads to the realisation of things	Motivation	
Where	The physical or logical locations where things are done	Place	
What	What is used to perform something	Information	
How	How things are done	Action	Behaviour
When	When actions are triggered	Event	

- An *Actor* fulfils a *Motivation*.
- A *Motivation* is performed at some *Place*.
- A *Motivation* is achieved through some *Action*.
- An *Action* is performed at some *Place*.
- An *Action* produces some kind of *Information*.
- An *Action* can produce an *Event*.
- An *Event* can trigger an *Action*.
- An *Action* can consume and can produce *Information*.

These relationships can be represented through simple directional associations, with a descriptive name. These associations are intended to represent a kind of connection which can't be lost through the several levels where elements can be realised or specialised. By using the UML Class notation, the meta-model concepts and their relations are represented in Figure 3.1. Each concept above defined will constitute a sub-type for each domain-specific concept.

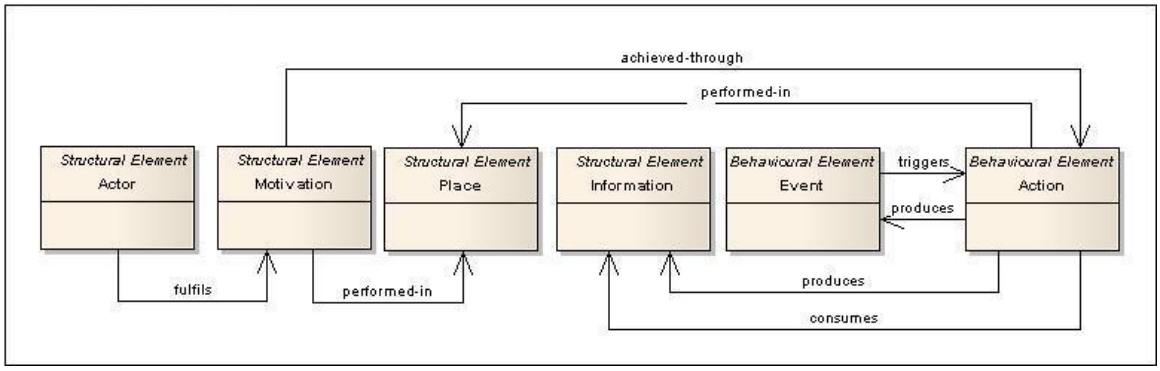

Figure 3.1: Common Meta-model

The common meta-model and the relationships described above will allow to infer relationships between elements of different models defined at different architectural domains. Therefore, the relations between those models can be established through simple associations, allowing to tackle both integrity and communication limitations.

For achieving the goals described above, it is necessary that each element is unequivocally identified. For that reason, the following tuple is proposed:

Id := (level, domain, type, name, context)

The *level* corresponds to the abstraction level where the elements are defined. Taking into account that the proposed meta-model elements are at level n, a realisation of one of its elements will be at level $n + 1$. The *domain* corresponds to the architectural domain where the element is defined. The *type* identifies an element of being structural or behavioural, as well as its sub-type. The *name* corresponds to its meaning, and the *context* identifies the instance of such element in a view. This identifier can be fully or partially filled, which will distinguish between elements defined in meta-models, models and in particular instances.

3.2 Meta-model Assumptions

The specificities and goals of the generic meta-model lead to the definition of some assumptions, in order to guide its development. The assumptions are as follows:

1. An element must only realise one element from the meta-model. If multiple realisation is allowed, the intended semantic of each element loses effectiveness.

2. Each element that realises one meta-model element must maintain the relationships defined for its parent, i.e., if a Business Actor realises an Actor, and a Process realises an Action, the established relation between Actor and Action will be inherited, thus requiring that a relation be established between the Business Actor and the Process.

3. Each realised or extended element must have knowledge about its parent.

4. Each element must be unequivocally identified.

3.3 Domain Mappings

In section 2.1 different ways of organising enterprise information were analysed. Despite the variety of domains and views, it was seen that they overlap across frameworks. There isn't an ideal choice, thus other aspects must be taken into account: the framework's adequacy to current standards; its ability of being cross-industry; the modelling languages that can be applied to its domains. From the analysis in section 2.1, the framework to be used as reference is the TOGAF framework, due to its expressiveness, widespread support and well-defined views and viewpoints.

TOGAF addresses three major hierarchical architectural domains, which are (from top to bottom):

- **Business** - Defines the business strategy, the structure and the business processes for the organisation;

- **Information Systems** - Defines the structure and interactions of application systems, along with the relationships to the business processes, as also the logical and physical data describing business assets.

- **Technology** - Defines the infrastructure (software, hardware, networks) which supports the applications.

For each one of them, it will be realised a set of concepts from the ones defined in the domain-independent meta-model, thus defining domain-specific meta-models for each architectural domain.

TOGAF recommends a set of languages for modelling each domain, such as BPMN or ARIS ICOMs for Business, or IDEF for Information Systems. However, one language is evidenced at any domain: UML.

As was analysed in section 2.2, UML is one of the most used standards for general purpose modelling, due to the richness of its concepts, relationships, and diagrams. Therefore, the use of UML elements from UML diagrams throughout each architectural domain is proposed.

UML has thirteen diagram types for modelling the structure or the behaviour of the system. However, several concepts and relationships are overlapped through several diagrams. The choice of which diagrams to use depends on the abstraction degree for the system being modelled. Therefore, for the intended propose of this thesis, it will be taken into account the elements with a higher abstraction degree. The addressed diagrams are shown in Table 3.2.

Table 3.2: UML Diagrams addressed

Domain \Type	**Structure**	**Behaviour**
Business	Class Diagram	Activity Diagram
Information	Class and Component Diagram	Activity Diagram
Technology	Deployment	

The realised elements for each domain are presented in Figure 3.2. As was stated in Section 3.2, each element that realises one meta-model element must maintain the relationships defined for its parent. Therefore, for the sake of clarity, those relations aren't shown.

For Business modelling, UML offers most of the concepts for answering the proposed interrogatives. However, due to its main purpose of modelling system architectures, it doesn't offer adequate structural elements for representing the Business domain. Nevertheless, its flexibility and extensibility allows the definition of those concepts through its Class elements. For Information Systems modelling, there isn't an acceptable realisation for the meta-model concept *Motivation* . Therefore, it is used one more time the extensibility of a Class element to present one with appropriate significance. For Technology modelling, none of the behavioural concepts was realised, due to the level of abstraction that is taken into account for the solution.

Along with the defined concepts, it is necessary to define a set of relationships. The structural and behavioural diagrams offer a set of relations adequate for each domain. Therefore, those relationships are shown in Table 3.3. The Structure aspect of the Business and Information Systems doesn't have specialised relations, due to the actual expressiveness of the common relationships.

Table 3.3: Enterprise Relationships Addressed

Domain \Type	**Structure**	**Behaviour**
Business		Control Flow, Information Flow
Information		Control Flow, Information Flow
Technology	Assembly, Communication Path, Deployment	
Generic	Associate, Realise, Generalise, Aggregate, Compose	

In the Zachman Framework, each crossing between a perspective (or domain) and an aspect (answer to an interrogative) results in a view. Therefore, a similar approach is taken for the proposed solution. The TOGAF Framework proposes three architectural domains, thus three viewpoints can be addressed for different groups of stakeholders. For each viewpoint, it will be addressed two views, concerning the structural aspect and the behaviour aspect of the system. Therefore, the core views proposed for this solution are shown in Table 3.4.

The elements for each view can be depicted through the specialisation of the elements from the

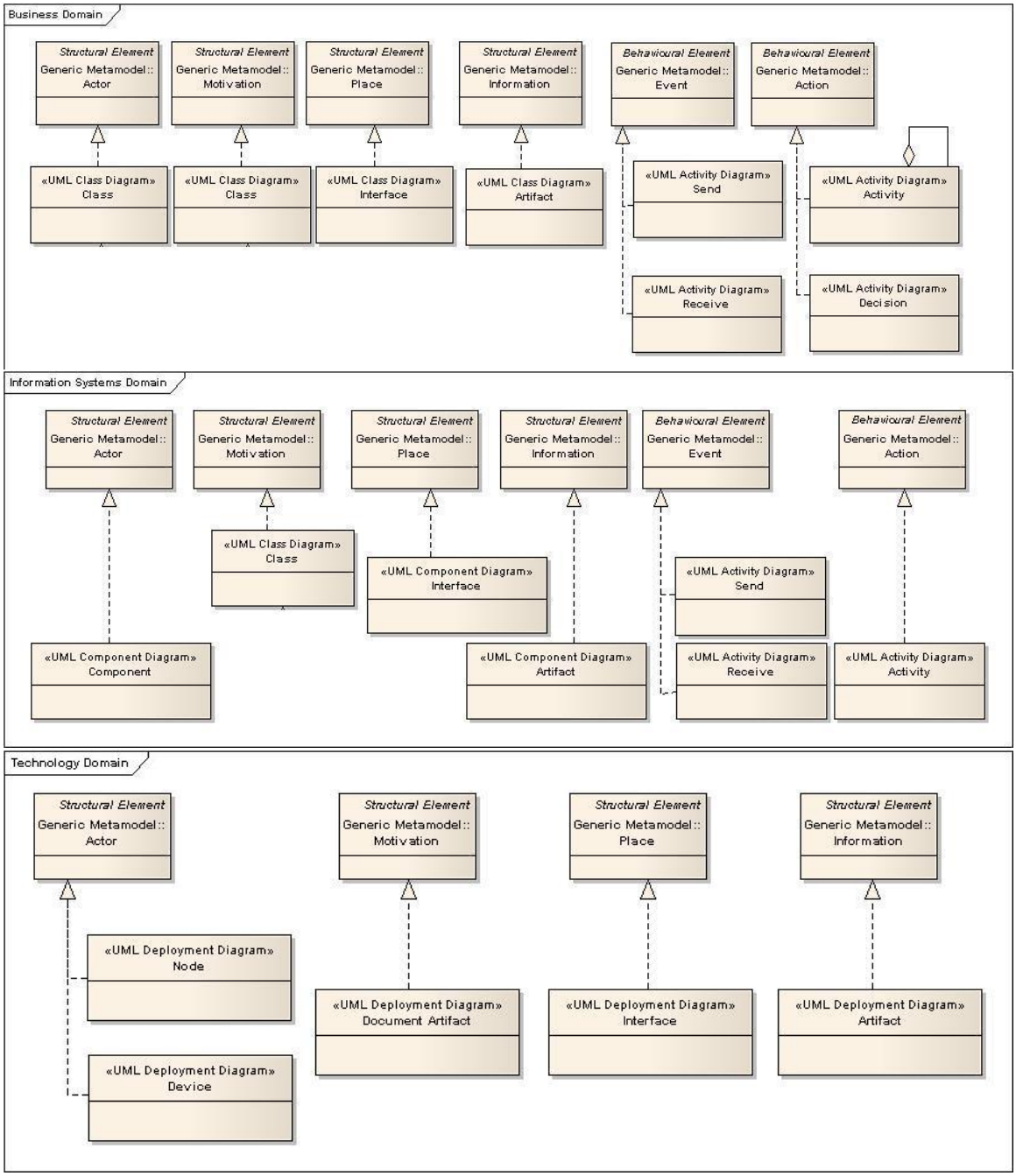

Figure 3.2: Mapping between the meta-model concepts and UML

Table 3.4: Enterprise Views Addressed

Domain \Type	**Structure**	**Behaviour**
Business	People View	Business Process View
Information	System View	Data Flow View
Technology	Infrastructure View	

meta-models defined for each domain. Therefore, the core elements, as well as the core views have been proposed. Throughout the dissertation, those elements will be further developed as well as additional views.

3.4 Traceability Solution

As mentioned in Chapter 2, modelling languages do not display an explicit concern with the tracing the modelled elements. They define a formal syntax but they hardly tackle semantic. Most of them lack a meta-model, therefore increasing the integration and communication problems between elements and views. For that reason, some languages address these issues by means of elements which act as interfaces between domains. However, such approach has a limited effectiveness, when compared with the potential of the relationships between elements derived from a meta-model. Therefore, in Section 2.4, multiple traceability solutions were analysed. The proposed solution doesn't aim for the best traceability solution for managing the relationships across elements, but only one with suitable characteristics for the meta-model requirements.

The intended traceability solution must be able to address the following aspects:

- Management of the linkages between models from different architectural domains, as referred in Section 3.1.
- Management of relationships between elements of the same type.
- Management of relationships between instantiated elements.

Given the ability of extensibility of each one of the defined elements, and the high number of possible relationships between them, it is necessary a solution that doesn't represent a storage burden. For that reason, a solution based on a repository will be addressed.

Using a repository will allow to save all the relationships between elements in a central place, avoiding the burden of creating and updating linkage information directly into the elements. It is then used a repository where it is saved all the element types defined above, with rules to enforce linkage semantic, thus enhancing the integrity of the solution.

This traceability solution will be further detailed throughout the dissertation.

3.5 A Generic Approach

The solution proposed in this chapter aims for a domain-independent approach for modelling and representing enterprise entities. Therefore, the solution shall surpass the limitations imposed by modelling languages, entity designs or repository structures, which are more or less connected at each architectural domain.

One of the solution's goal is to provide an enhanced knowledge and enhanced relationships between entities, e.g., entity relationships across models, views, viewpoints or even architectural domains. For

achieving such navigability, the repository design must be addressed. As studied in section 2.5.1, the way on which a repository is designed affects the functions and applications to be applied upon entities and relationships. Therefore, the design must take into consideration this kind of navigability through elements representing models, views or viewpoints.

Chapter 4

An Enterprise Architecture Prototype Tool

For tackling the issues studied so far and proving the practicability of such approach, a prototype was developed. The solution was implemented within the WebComfort framework [6], a Content Management System (CMS) which structures and manages entities within an organisational environment. By taking advantage of its content management capabilities, the prototype will offer enhanced knowledge from each entity defined within the system, as well as enterprise architecture modelling capabilities.

In this Chapter a WebComfort overview will be presented, following by the design and implementation stages of the prototype within the framework. Therefore, the different tiers of the prototype will be presented, describing the design decisions and constraints taken throughout the solution.

4.1 The Framework's Technology

The WebComfort [6] is a Web Portal and a Content Management System Framework developed and supported by technologies such as ASP.NET 2.0 (C-Sharp) and SQL Server 2005. The framework supports content management tools and mechanisms through web browser access.

The WebComfort Portal is composed by a flexible number of dynamic pages based on the ASP.NET master pages with an hierarchical organisation, offering variable content structures through predefined module types, wrapped up by ASP.NET user controls.

In WebComfort the content is presented through information modules, while the presentation of such content is configured by a layout management mechanism. The platform supports extensibility, allowing new module types for managing and displaying information, and supports the development of new module logic design by means of a well-defined API.

The WebComfort Portal comprises multiple pages, denominated as tabs, which are accessible through browser navigation. Each tab aggregates a configurable number of vertical containers which hold the modules for each page. Modules are elements with content management and display capabilities within a tab. An example of a WebComfort page is presented in Figure 4.1, with five tabs and two vertical contents, one for the Login and Calendar modules, and the other for the WebComfort corporate information module.

The WebComfort platform supports several content operation features, allowing:

- Content referencing between modules, enabling automatic content changes throughout multiple modules.
- Programmatic support for content copy between different modules.

Figure 4.1: A WebComfort Portal

- Content Import and Export through XML.
- Multi-language through the definition of Language Packs in XML.

As a result, the choice of WebComfort for developing this dissertation prototype relies on the features made available by the framework, providing full content and presentation separation, extensibility through new modules, content operation capabilities and content search support.

The recent developments of the platform support the concept of toolkits, allowing the creation of installable module packages which are deployed on the platform without particular intervention from the user. By taking advantage of such architecture, as presented by [35] in Figure 4.2, the above mentioned prototype was developed and deployed.

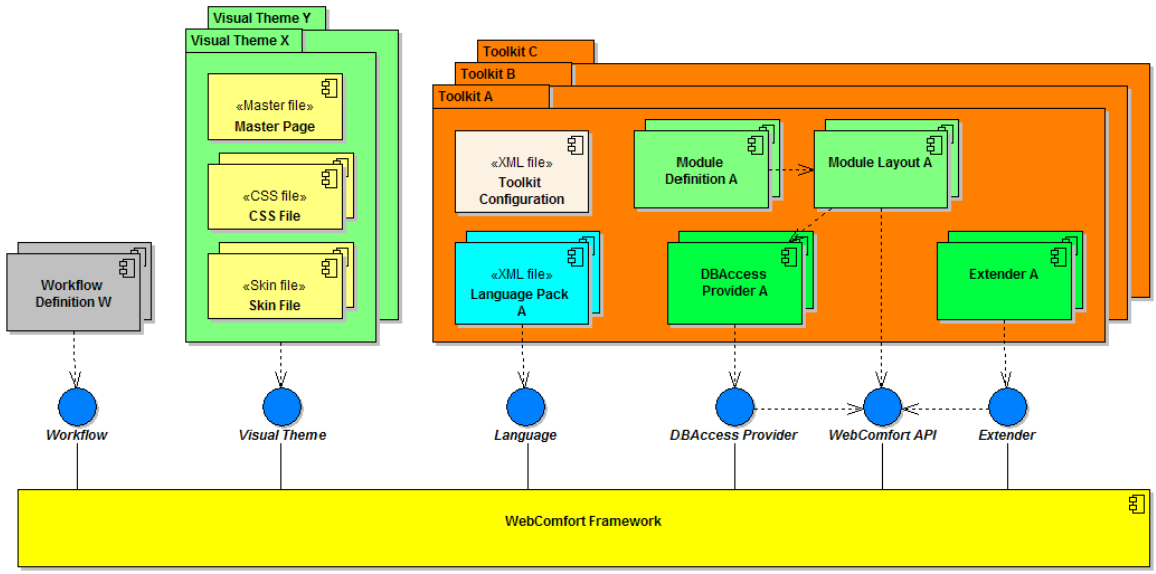

Figure 4.2: WebComfort Overall Architecture

4.2 Prototype Overview

As explained in the previous section, the prototype's architecture derives from the WebComfort's toolkit architecture presented in Figure 4.2. Therefore, the prototype overall architecture can be depicted in Figure 4.3.

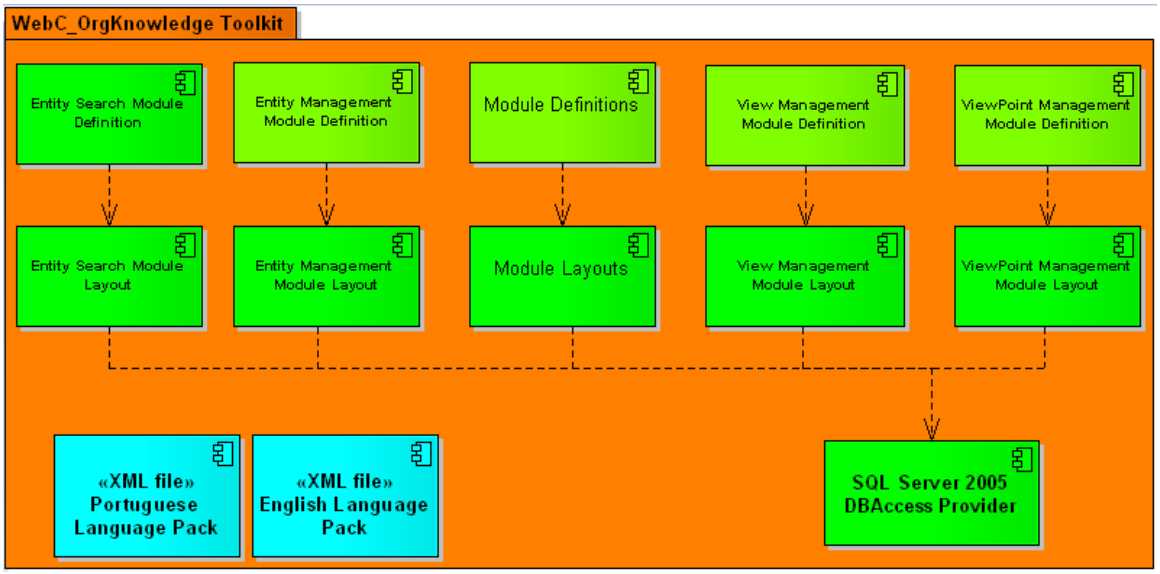

Figure 4.3: Prototype Overall Architecture

The toolkit, called WebComfort Organisational Knowledge (WebC-OrgKnowledge), has a set of modules which are responsible for a distinct management of entities, models, views and viewpoints, and has also an additional module for searching purposes. The toolkit offers two language packs in Portuguese and English, for a more convenient user interaction. In the following sections, a detailed description of each module is presented.

4.3 The Data Infrastructure

For achieving the enhanced relationships between entities as discussed in Chapter 2.5, an appropriate infrastructure design must be achieved. Therefore, the repository design for this dissertation's solution derives from the conceptual model used from a commonly accepted standard for enterprise architecture frameworks. The standard is the IEEE 1471-2000 [18], which is now published as an international standard with the designation of ISO/IEC 42010.

The repository design is focused primarily at the triple (Viewpoint, View, Model) which can be depicted in Figure 4.4. The design takes into account the standard recommendations regarding the representation of entities across the elements defined at this triple. Therefore, the main database scheme of the repository is shown in Figure 4.5. For the sake of clarity, some tables and table attributes are not shown.

The database design was built within the context of architectural description as defined in the IEEE 1471-2000 conceptual framework. Therefore, it meets the following requirements:

- An architectural description is organised into one or more constituents called architectural views.
- Each view addresses one or more of the concerns of the system stakeholders.
- View is used to refer to the expression of a system's architecture with respect to a particular viewpoint.

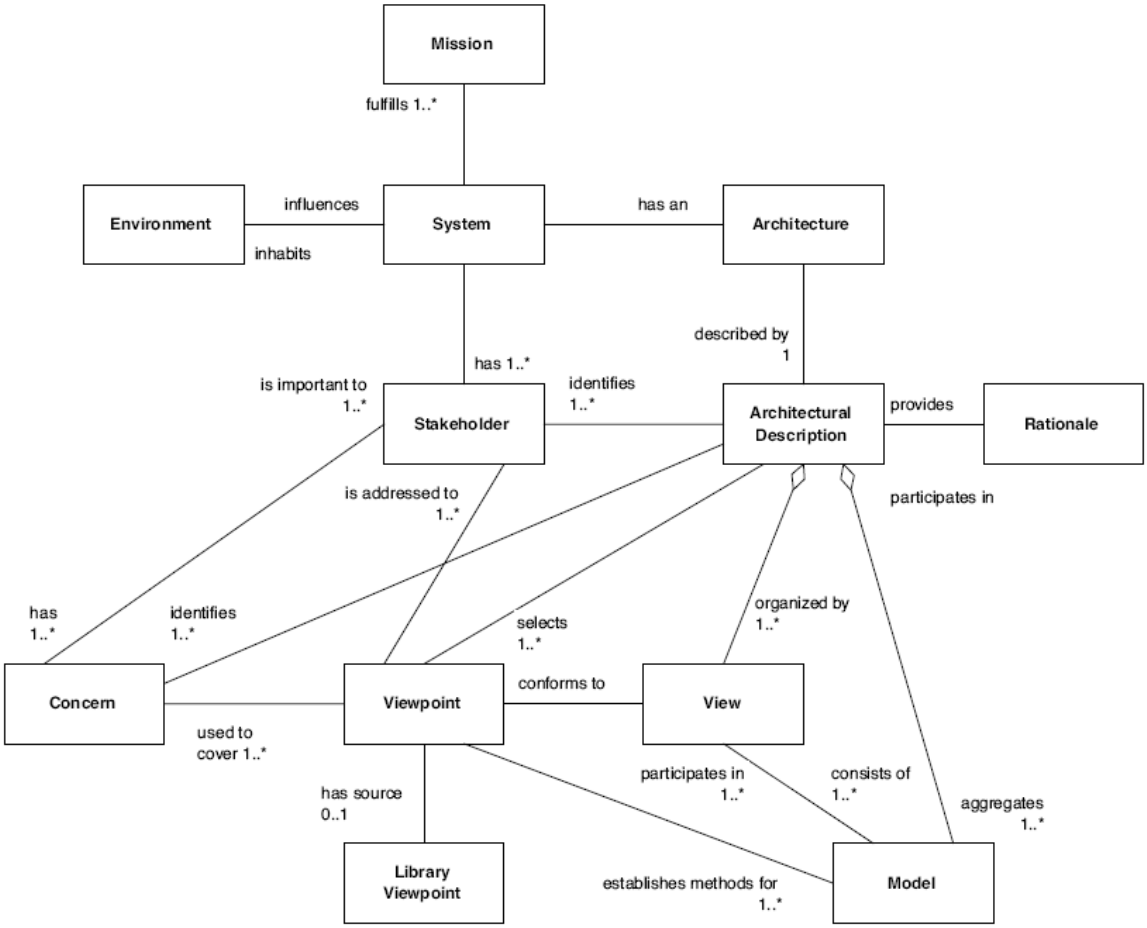

Figure 4.4: IEEE 1471 Conceptual Model

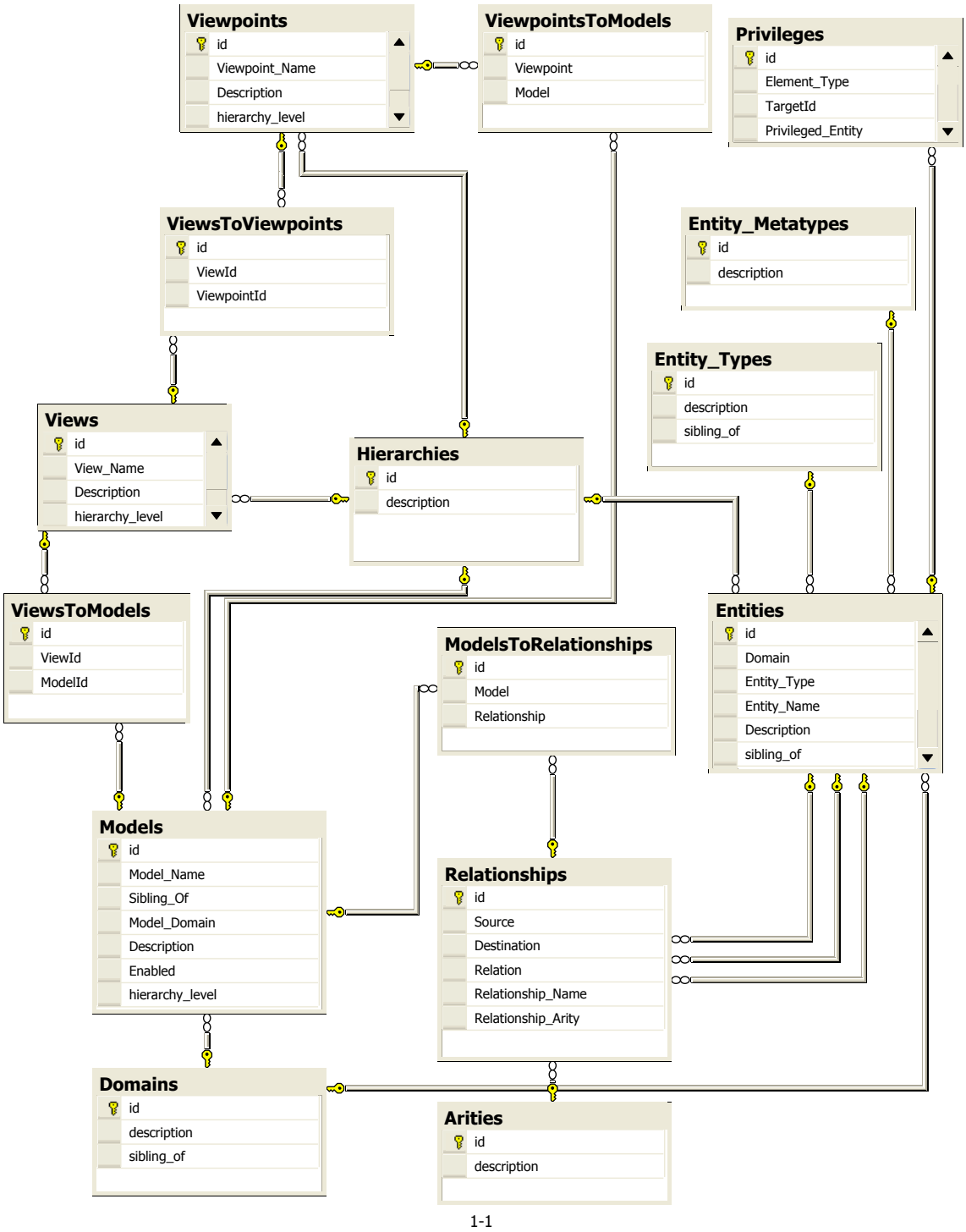

Figure 4.5: Database E-R Scheme

- A viewpoint establishes the conventions by which a view is created, depicted and analysed. Therefore, a view conforms to a viewpoint.
- An architectural description selects one or more viewpoints for use.
- A view may consist of one or more architectural models.
- Each architectural model is developed using the methods established by its associated architectural viewpoint.
- An architectural model may participate in more than one view.

As described in section 2.5, the properties defined for each entity are also paramount for a tool's analysis and manipulation capabilities. The definition of such properties are often linked to the architectural domain where the entities are specified. The solution proposed in this dissertation aims for a domain-independent approach, which imposes some limitation on the properties being defined. However, the extensibility of such properties is also intended, allowing the application of specialised tasks related to particular domains. Therefore, the defined properties are intended to be used or extended by any domain, regardless of the type of analysis or manipulation task. For such reason, the properties address the following aspects:

- State or life cycle. Each entity must have a property regarding its current condition or state within the system. The possible values of this property are tightly linked with the entity's semantic.
- Privileges amongst entities. The basic privileges for creation, access, update and delete must be defined between entities across models, views and viewpoints, enforcing the right usage of each entity defined within the system.

4.4 The Business Logic

From the lower tier it is possible to formulate complex queries regarding one or more elements, given the relationships defined across tables. In order to enhance the information retrieval from the database, multiple views, stored procedures and functions were implemented. By doing so, information regarding entities and their usage throughout the enterprise architecture is more easily retrieved. Detailed information regarding these elements can be consulted in Appendix B.

The updates and the information retrievals from the database are accomplished by applying the Data Transfer Object (DTO) design pattern, which guarantees a transparent data access between domain objects and tabular data. Each DTO have two different mappings: one DTO for each individual table, whose columns are matched within object attributes; and one DTO for each view, enabling the aggregation of several table sources into one object. However, DTOs for views are only used for reading operations.

As presented in Figure 4.6, the DTOs are directly used by a layer which presents the toolkit's available functions, e.g., the toolkit's API. This layer is composed by a set of classes which define the interfaces for manipulation of enterprise architecture elements, as well as their implementation. Therefore, the create, access, update and delete operations for the elements represented in the previous section are made available by the API. Detailed information regarding the toolkit's API can be consulted in Appendix C.

As presented in section 4.2, the toolkit is composed by several module layouts and module definitions where the business logic is defined. The modules' definitions make use of the defined API for achieving their proposed goals, and the modules' layouts are defined through ASP.NET (C-Sharp) controls, which are tightly linked with the presentation layer.

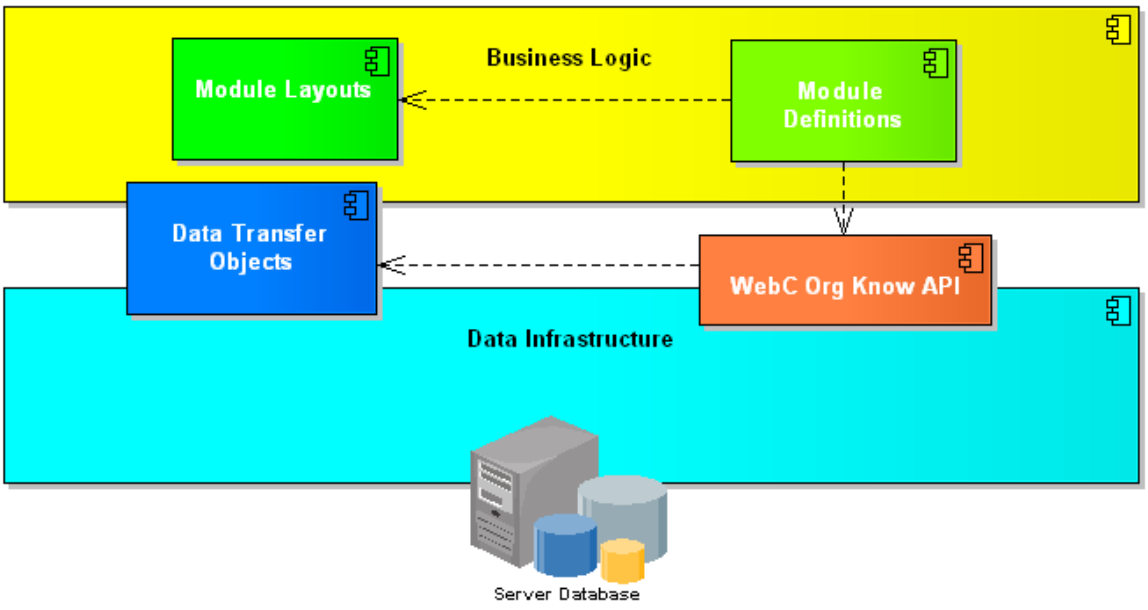

Figure 4.6: Prototype Layered Architecture

4.5 The Presentation

The toolkit's presentation is made through *aspx* files, defining ASP.NET controls which render HTML. Each module has its own layout, which are integrated on an ASP.NET masterpage. The toolkit's layout can be depicted from Figure 4.7.

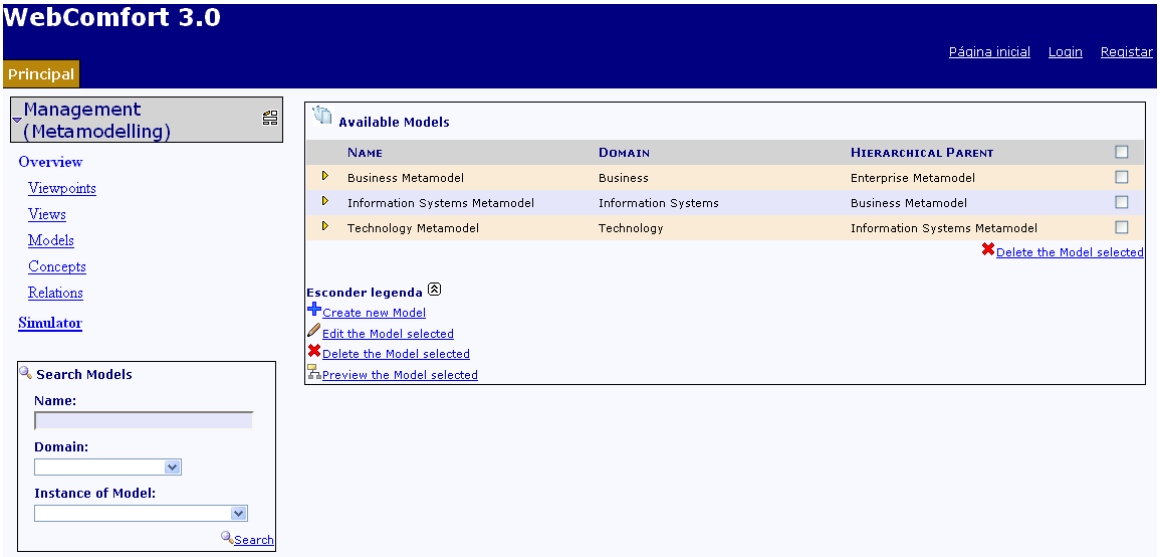

Figure 4.7: Prototype Layout

The prototype's layout is as follows: on the left side are gathered the management operations regarding the enterprise architecture elements, followed by a search module which changes its target according to the management choice. On the right side the available architectural elements are shown, as well as the CRUD operations upon them. Additionally, if a model management option is chosen, a preview operation is also presented.

Only one tab is used for all the toolkit's modules in order to offer a centralised view upon the available options. The overall menus maintain an uniform organisation throughout the entire system. By selecting

any management option, a grid view is presented on the right pane, as shown in Figure 4.8.

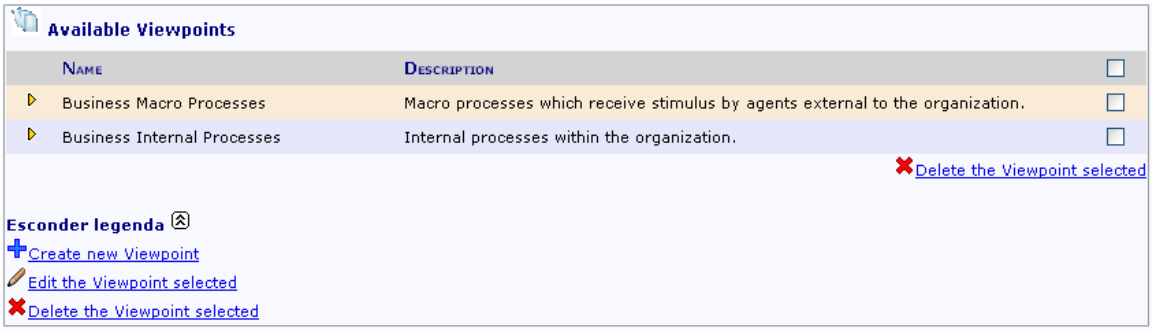

Figure 4.8: Management option selection example - Models

The management module offers one particularity. An action button is presented for choosing the appropriate enterprise architecture context. This button is presented on the right side of the module's title, and by clicking it displays the page presented in Figure 4.9.

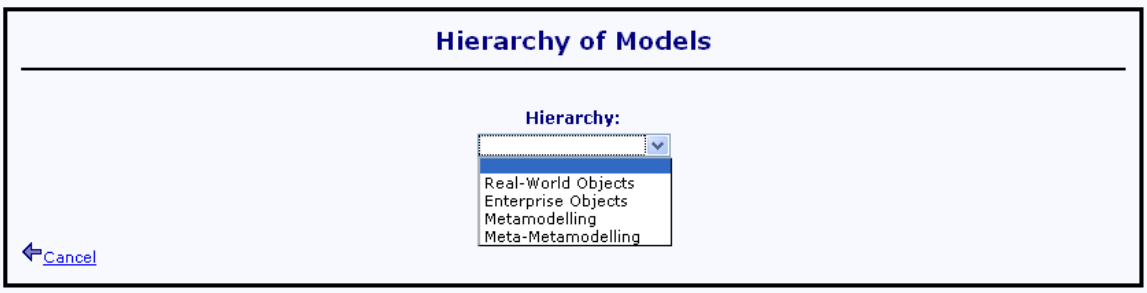

Figure 4.9: Hierarchy of Models

The available options are:

- Meta-Meta-modelling - Displays the most generic level of enterprise modelling. This option is used for implementing the generic model and the domain-independent entities described in section 3.1.
- Meta-modelling - Represents a level where the previous model and its entities are instantiated. On this level the domain-specific meta-models and the entities chosen from well-known modelling languages are implemented.
- Enterprise Objects - Represents models and entities for a specific enterprise context. This is the level where the Enterprise Architect builds a solution which match the needs of its client.
- Real-World Objects - Represents processes in execution and entities with life-cycle states within the enterprise context.

A detailed description of all the management options and intended usage are presented in the following sections.

4.5.1 ViewPoint Management

When the Viewpoint Management option is selected, a grid view displaying the available viewpoints for a particular modelling context is presented, with a structure and layout similar to the one presented in Figure 4.10. The available operations for Creation, Access/Edit and Delete are presented at the bottom of the grid view.

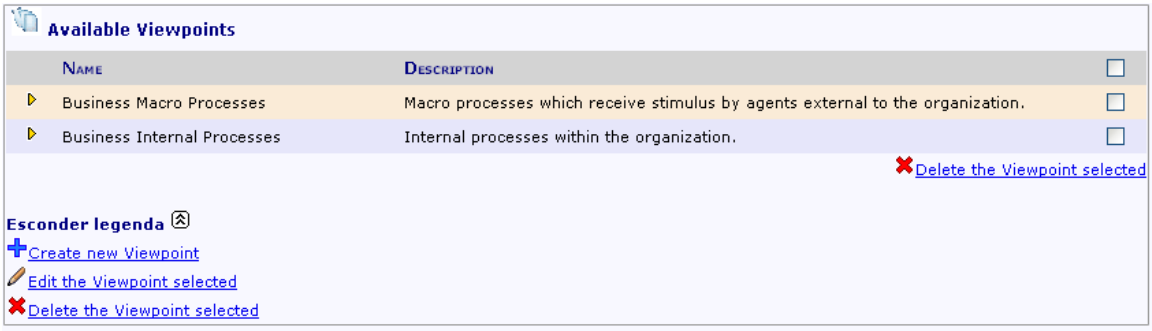

Figure 4.10: Management Option selection example - Viewpoints

Each architectural element (e.g. Viewpoint, View, Model or Entity) has the necessary menus for allowing their standard compliant definition. Therefore, the Viewpoint defines which models are within its scope, as presented in Figure 4.11, allowing their further formalisation at the definition of different Views. Such operation is described at the following section.

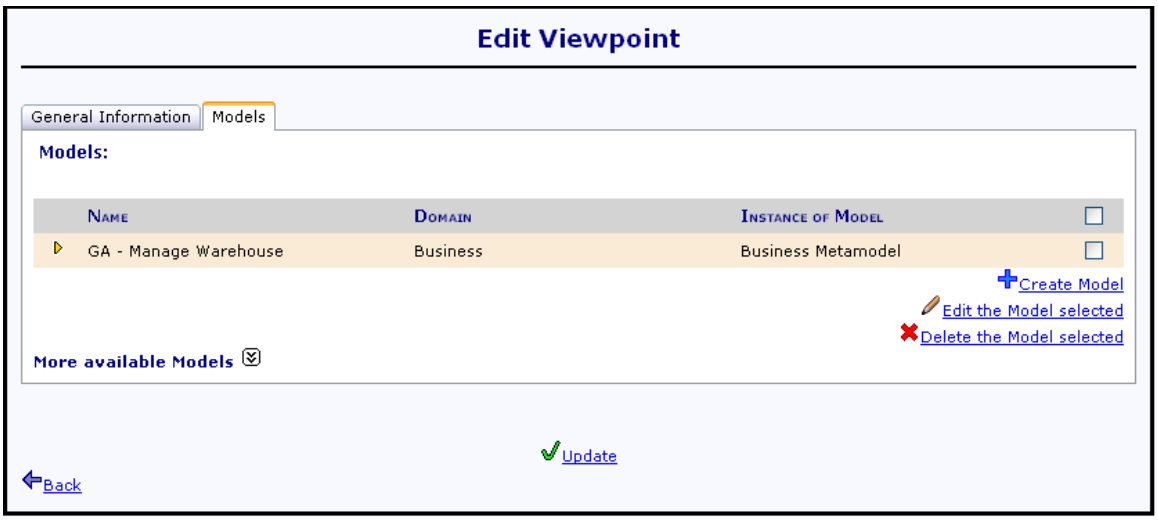

Figure 4.11: Models referenced by a Viewpoint

4.5.2 View Management

The menus from element displaying and CRUD operations from the View Management option are similar to the ones presented at the previous section.

The definition of a particular View is enforced by the standard compliant requirements explained in section 4.3. The Viewpoint references the models which are aligned with its definition. Therefore, a View must be defined accordingly to the rules defined at a particular Viewpoint, thus referencing a subset of its models. From Figure 4.12 and Figure 4.13, such behaviour is enforced by first defining which viewpoint to conform, and then defining which models to approach.

4.5.3 Model Management

Apart from the menus for creation and model edition, which are similar across the management options, one additional operation is available. The user can make a graphical preview of the model, similar to a graph. In Figure 4.14 a preview is presented. Each entity is represented by an image, which can be

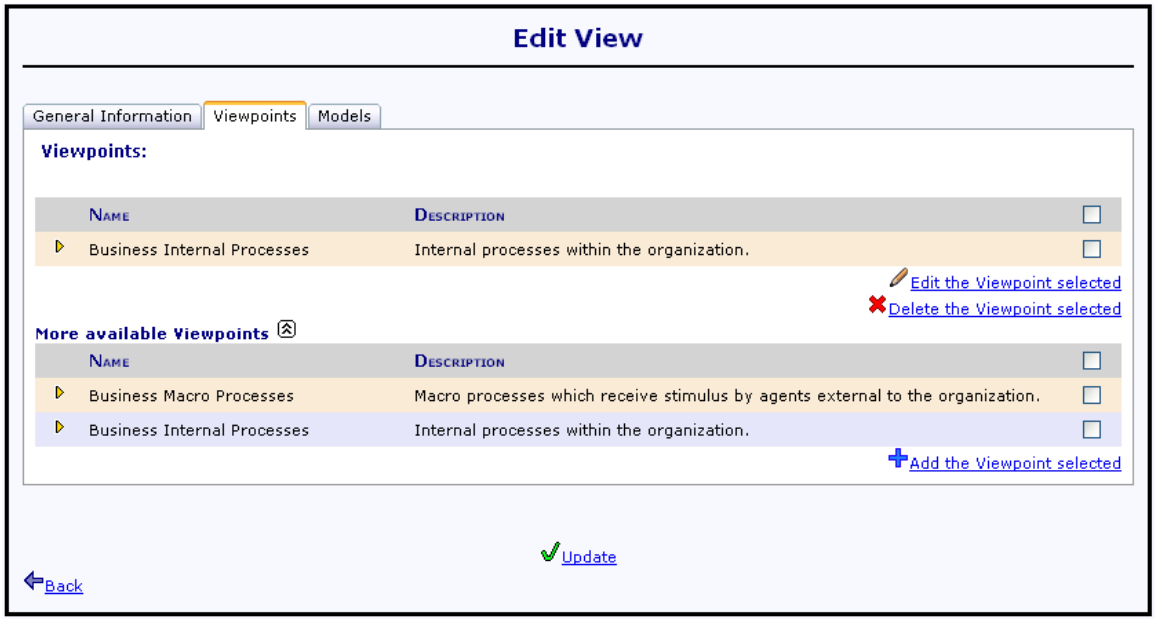

Figure 4.12: Viewpoint referenced by the View

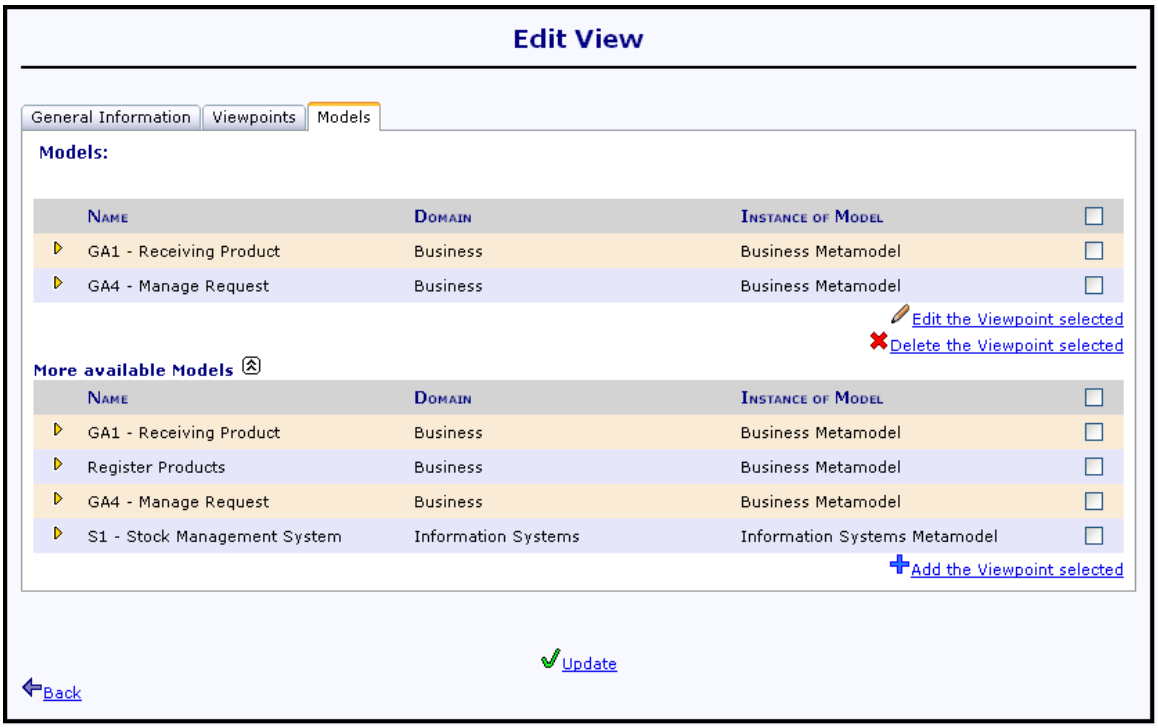

Figure 4.13: Subset of Models approached by the View

changed accordingly to the entity's semantic. However, such extensibility is relegated for future work, and currently the image is represented through a coloured circumference.

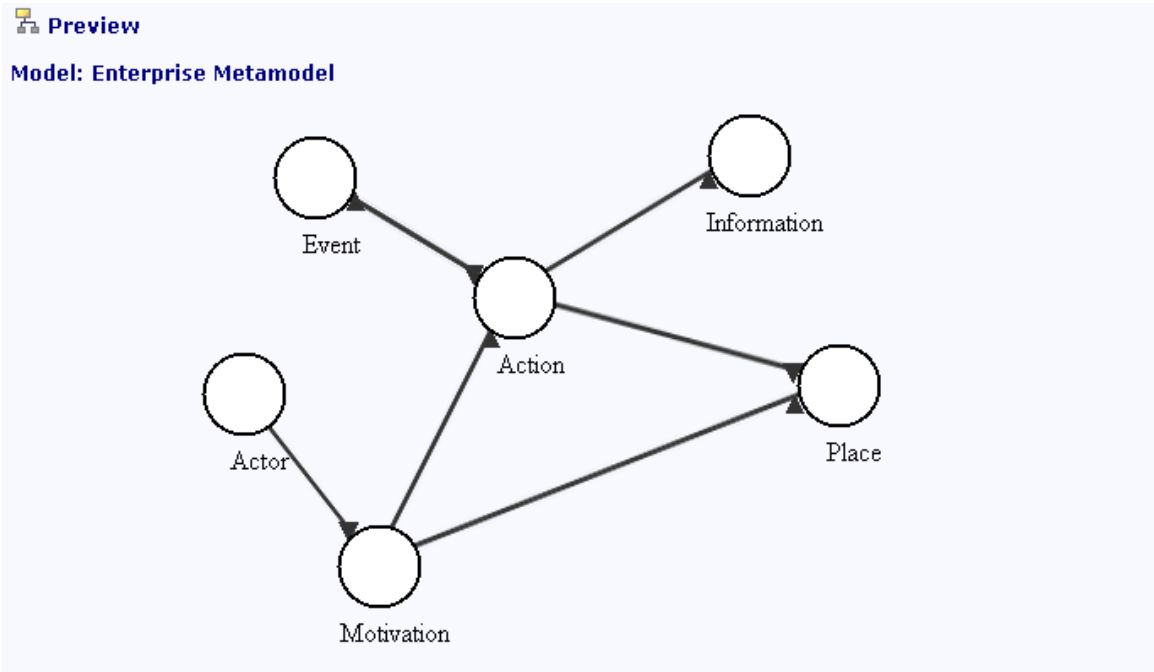

Figure 4.14: Model Preview Example

For each entity, their related relationships are represented. The relationships only describe the existence of a certain directed relation between two entities, thus not describing the arity nor the semantic of the relation. For more details regarding those relationships, each node can be double-clicked for detailed information regarding their usage. The information regards not only the entity itself, but also both its income and outcome relationships. Further details are presented in the following section.

4.5.4 Entity Management

The CRUD operations are similar to the operations presented at the last sections. Entities are the elements with bigger focus within the system, because it is through them that an enterprise architecture is represented. Therefore, the entity element has additional operations regarding their usage throughout the system. The usage can be depicted from Figure 4.15.

Through the entity usage panel, it is possible to analyse both information made available directly by properties defined at each entity, and by the relationships defined within the repository. Therefore, knowledge regarding entity participation at views and models is made available, as well as the entity instantiations across the system. Additionally, information regarding the hierarchy of an entity within the several architectural domains composing the enterprise architecture is also known.

The entity usage panel provides an aggregation of several enterprise architectural sources at one place, thus allowing an enhanced information regarding entities defined throughout the system, as proposed in this dissertation in Chapter 3.

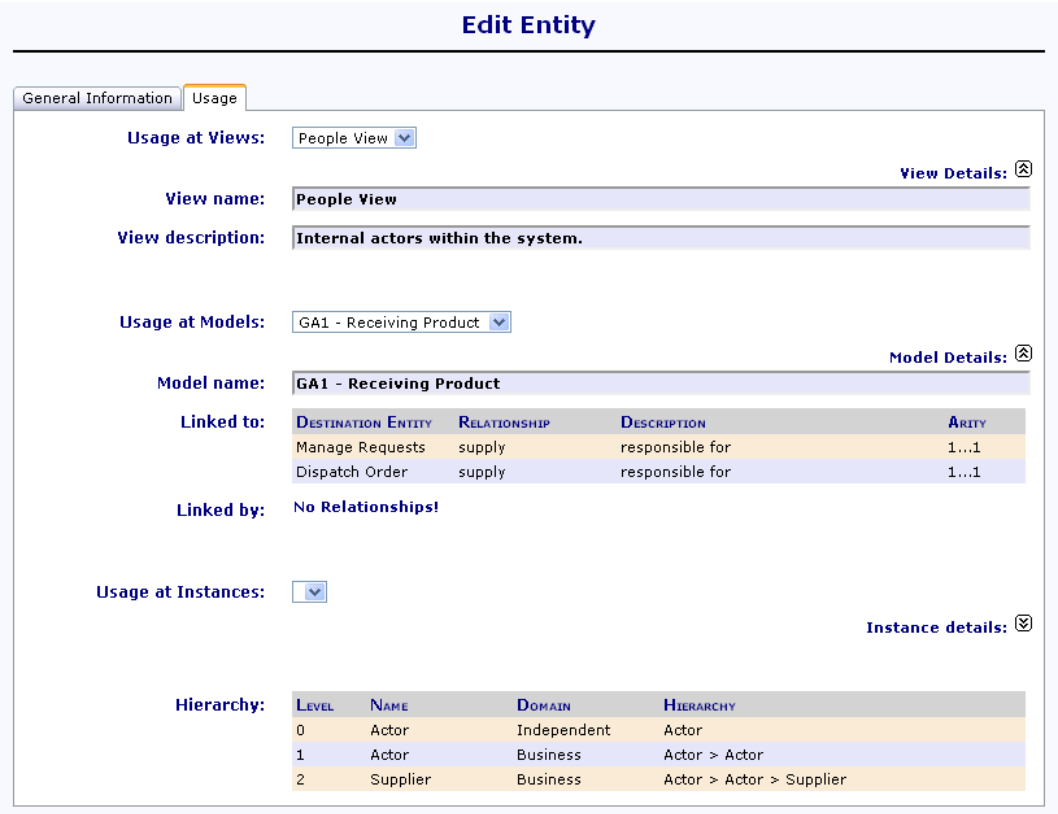

Figure 4.15: Entity Usage Screen

Chapter 5

Validation

In section 1.2 the problems of enterprise architecture modelling were introduced. In the following chapters a solution was proposed, and a prototype was developed. In this chapter, the solution is validated by presenting an enhanced knowledge of the defined entities thorough models, views and viewpoints.

This chapter presents a business case scenario focusing a particular business concern. The prototype presented in Chapter 4 will be used for the evaluation, and the accomplishment of all the aspects presented in Chapter 1.4 will be analysed.

5.1 Evaluation Overview

The case scenario for evaluating the proposed solution is recalled from the 2007/2008 project of the *Architecture, Processes and Tools for Information Systems* course, where the development of an Enterprise Architecture was required. The evaluation will fall into some of its processes, information elements and applications.

The solution's will be compared against the Sparx Enterprise Architect 7.0 tool, due to its general purpose modelling by means of UML. The aspects being focused are the ones described at Chapter 1.4, which are:

1. The integrity of each defined element, i.e., the avoidance of element overlapping across views;
2. The ability to navigate across elements through all the defined relationships;
3. The ability of switching across views;
4. An enhanced knowledge regarding the state, values, and relationships of those objects, independently of the context in which they are defined.
5. The conformance of the meta-model with real enterprise data;
6. The ability to extend the organisational awareness about its enterprise objects;

5.2 Case Scenario: XPTO Supermarket's Warehouse Management

The case scenario focuses on the Warehouse Management process. The Warehouse Management must guarantee stock availability as also the capability for receiving, manipulate, preparing and dispatching products to its own supermarkets.

The process will be described by modelling it both in the Sparx Enterprise Architect tool and in the proposed solution. Each model is thereby analysed against the aspects recalled in the previous section.

5.2.1 Manage Warehouse process

The Manage Warehouse macro-process undertakes several elements from the business architectural domain. For an overview, some elements from BPMN are used to describe its actors, processes, artifacts, information and events. Therefore, a representation of such elements is presented in Figure 5.1.

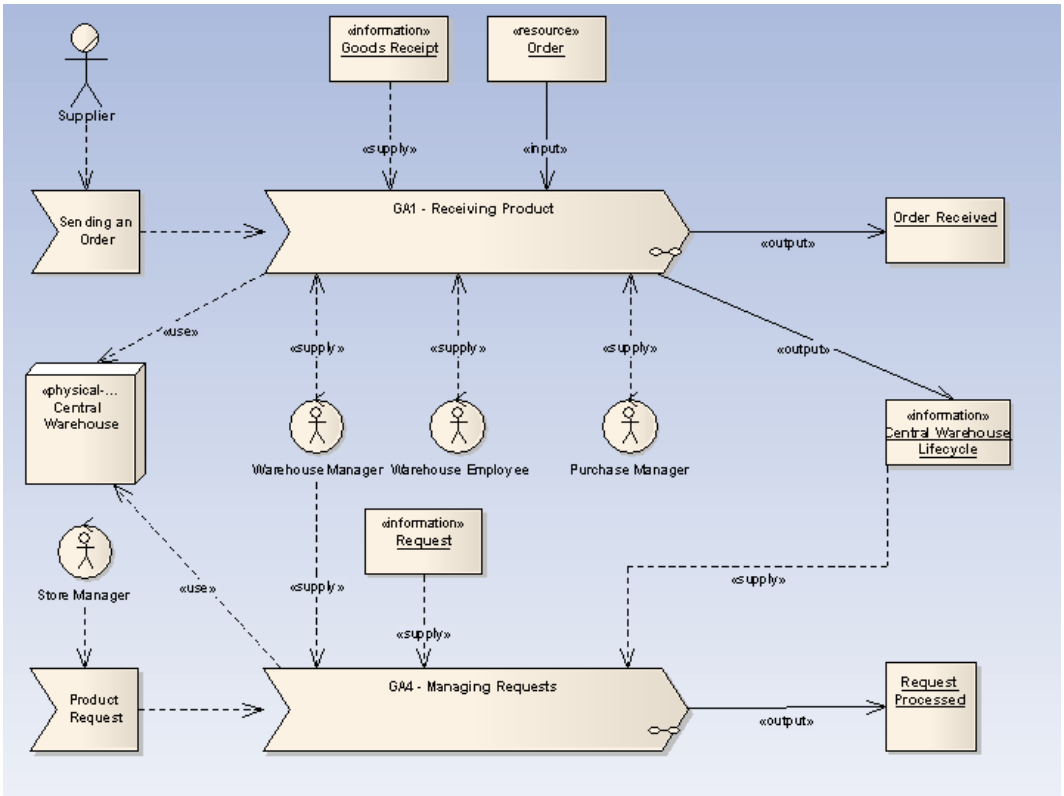

Figure 5.1: Sparx EA - Manage Warehouse

The Sparx Enterprise Architect has a well-defined graphical interface, which provides a clear understanding of its elements and relationships amongst them. From the model presented in Figure 5.1, both internal and external actors are presented, as well as the information and resources consumed throughout the process. Therefore, the tool provides an appropriate representation of each element, and a comprehensive preview of the overall process.

In Figure 5.2 the same process is presented by means of the proposed solution. Several concerns regarding the graphical representation of this process were not taken into account, as explained in section 4.5. Therefore, neither the individual representation of each element nor the relationship semantics are graphically represented, because it is out of the purpose of this dissertation. Nevertheless, a *drag-and-drop* system, based on the work of [10] is offered. This *drag-and-drop* mechanism allows the user to re-position each element, allowing the most appropriate graphical organisation within each model.

A description of each element for each model will not be presented. The meaning is represented by the element's labelling, which is a sufficient condition within the scope of this dissertation. Specific models are thereby presented.

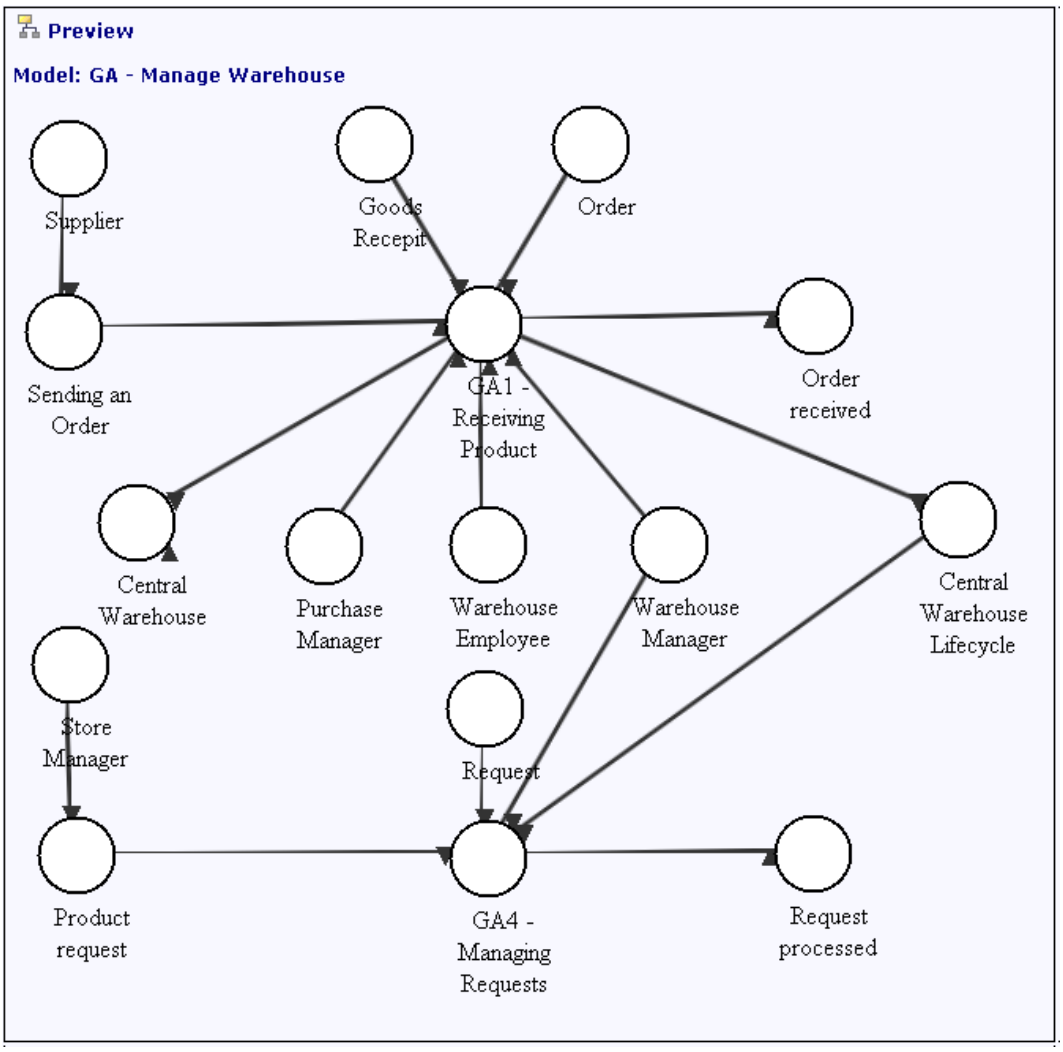

Figure 5.2: Proposed Solution - Manage Warehouse

5.2.2 Receive Product process

The Receiving Product process is part of the Manage Warehouse macro-process. As observed in Figure 5.3, the BPMN language elements are used to represent this model. For the sake of clarity, only part of the process is shown.

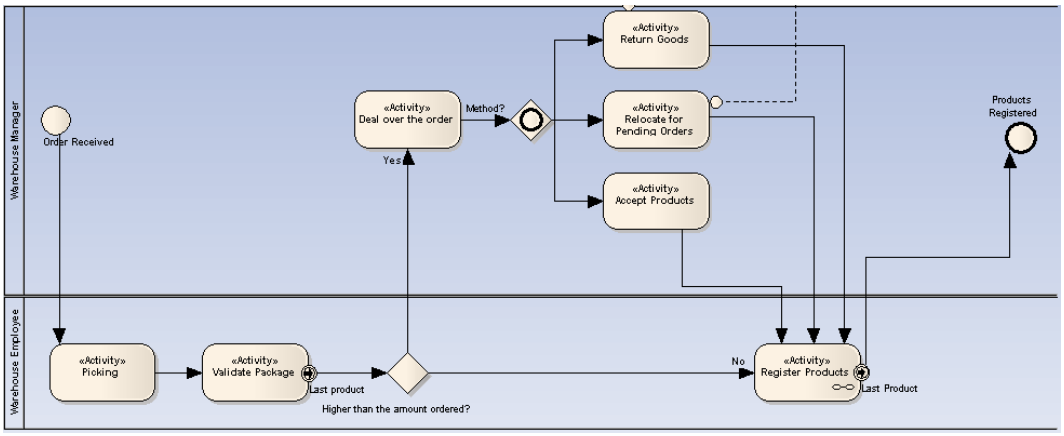

Figure 5.3: Receive Product Process

From the picture can be observed, for instance, that the actors of this process are represented through pools or lanes. The usage of different representations for describing the same element in different contexts is used in a plethora of enterprise architecture tools. However, such compromise commonly imply different elements, due to the heterogeneity of modelling techniques. This results in poor element integration, only surpassed through the burden of additional models defined specifically for mapping those differences. Therefore, a enterprise tool must have some mechanism for allowing not only the addition of different modelling languages, but also their correspondence with elements defined in other languages. The proposed solution uses elements retrieved from UML and BPMN, and its foundation aims at re-usage of such elements in different models at the same architectural domain. For different architectural domains, a new element is defined but with a relation of *realise* in order to describe a hierarchical difference. Therefore, the first aspect described in section 5.1 is thereby tackled within the proposed solution. Such integration will be further analysed in the following sections.

5.2.3 Manage Request process

The Manage Request process is also part of the Manage Warehouse macro-process. As observed in Figure 5.4, the BPMN language elements are used to represent this model.

The BPMN notation language is again used, and one more time the actors are represented through pools. The Sparx Enterprise Architect Tool allows to see the linkages established by and to a particular element. However, the information of its usage becomes limited for the facts explained in the previous section.

5.3 Analysis

The business case presented through the graphical interface provided by the Sparx Enterprise Architect Tool has allowed to observe some limitations in element integration, which is one of the tackled aspects of the proposed solution. This kind of limitation is a consequence of the heterogeneity of modelling languages for different architectural domains. Therefore, it is an issue with difficult resolution by any Enteprise Architecture tool.

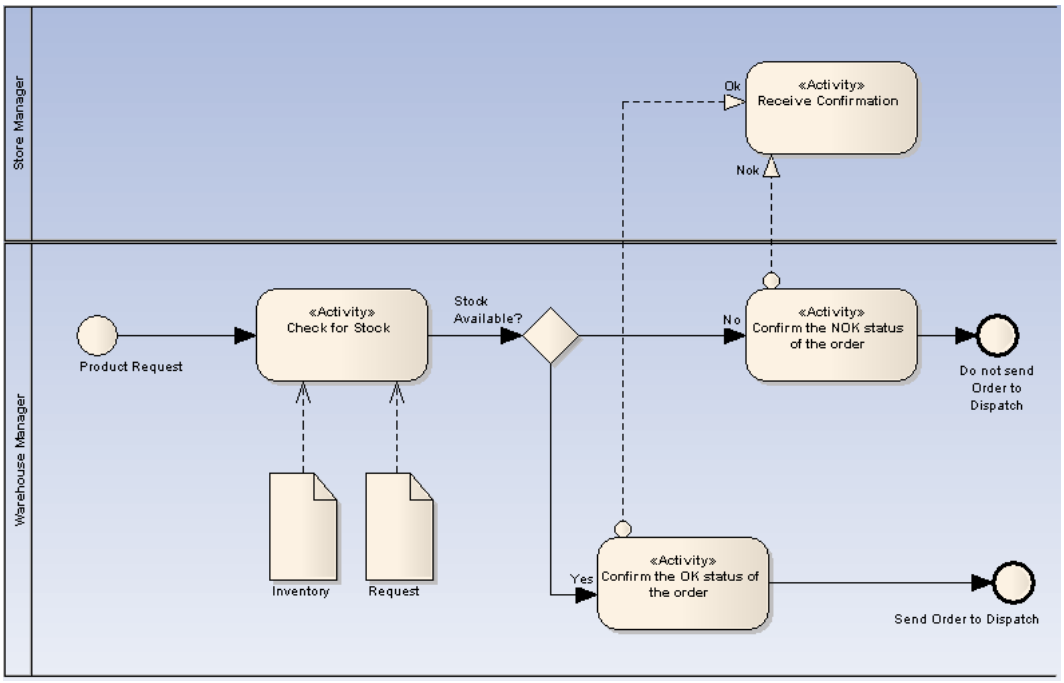

Figure 5.4: Manage Request Process

The models defined for this simple business case, in both the Sparx Enterprise Architect tool and the proposed solution, will allow the comparison of some aspects which are currently limited and with difficult resolution for the available tools, and how they can be surpassed with this dissertation's proposal. Therefore, each aspect described in section 5.1 is separately analysed.

- *The integrity of each defined element, i.e., the avoidance of element overlapping across views.*

The Sparx Enterprise Architect does not directly support Views as they are defined within the IEEE 1471-2000 standard. The tool allows the definition of different models through the elements defined by UML diagrams or by other modelling languages. However, as seen in the previous section, different modelling techniques imply different representations and different semantics for the same element. Therefore, a tool must support a mechanism defining the similarities of those elements through the different languages it supports. The proposed solution offers a prototype based on the UML language and the BPMN modelling notation, where this similarity is defined through *realise* relations assigned at the moment of creation of an element, as observed in Figure 5.5. By allowing this tighter connection between elements, further analysis can be done upon the elements defined across the system.

- *The ability to navigate across elements through all the defined relationships.*

The Sparx Enterprise Architect tool has an option where the linkages of each element can be observed at different models. However, it only presents the information regarding the relationships of such element, not describing other relevant information.

One goal in building an Enterprise Architecture is to provide an holistic view upon the defined system. Such holistic view can be achieved through a top-down approach or a bottom-up approach. The top-down approach allows to see the different architectural domains of an enterprise, allowing a drill-down of each element until achieving a specific process or application block. This approach is supported by several enterprise modelling tools, one of them the tool used for validation.

A bottom-up approach consists at navigating through the enterprise architecture from a particular element or model. This approach is not fully achieved by any enterprise architecture tool, in part due

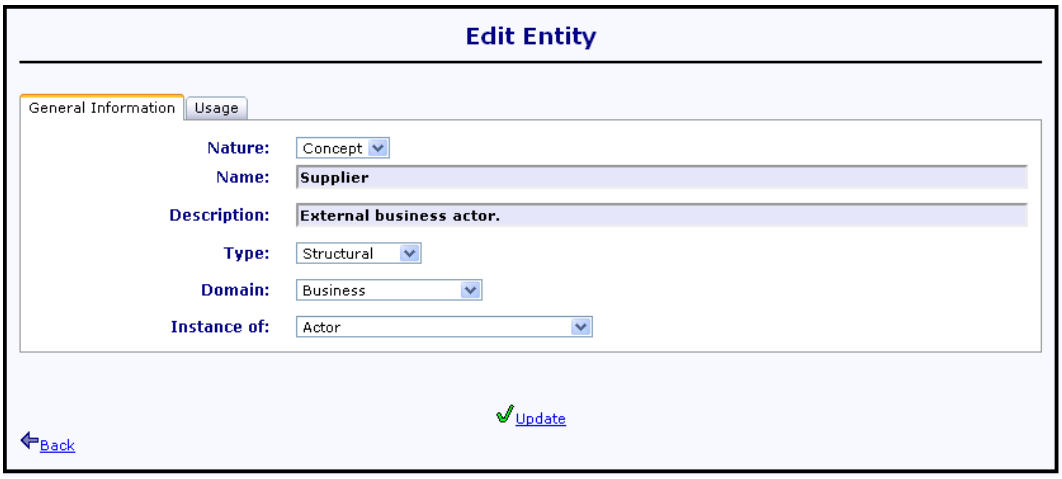

Figure 5.5: Entity Definition

to integration issues between elements. Other reasons are related to poor relationships between elements within all the other architectural elements existent in an enterprise architecture.

The proposed solution is able to surpass that limitation by making available a Usage screen, allowing to see a particular element from all the enterprise architecture perspective. Such example can be depicted from Figure 5.6.

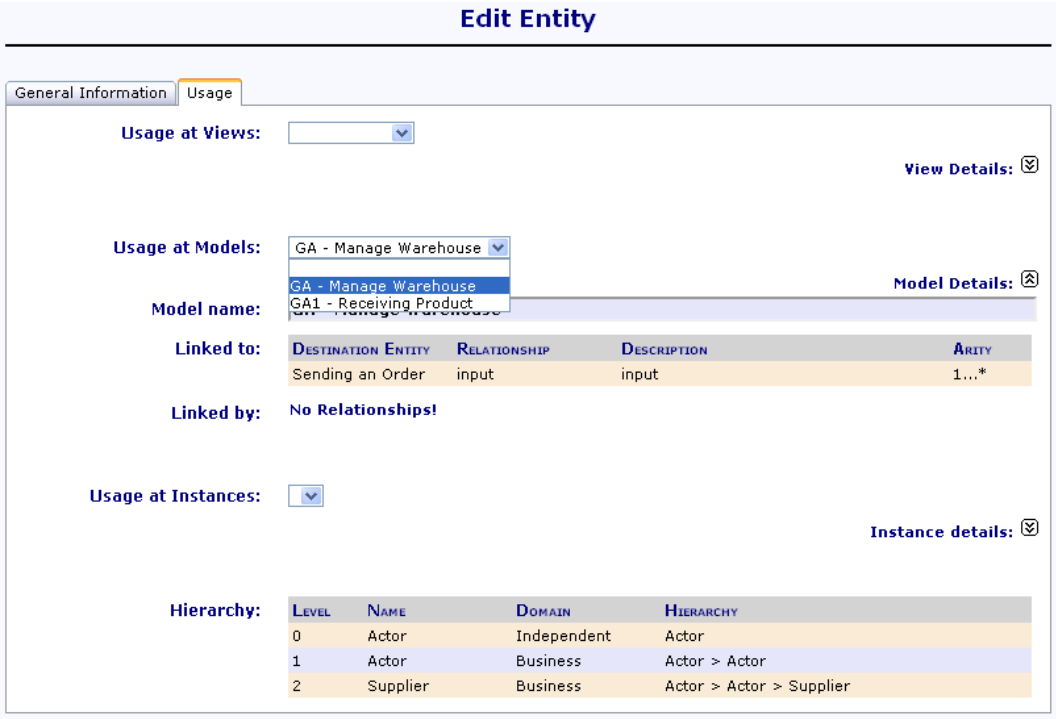

Figure 5.6: Entity Usage - Supplier

The Usage screen shows the existence of the Supplier entity within the defined models presented at the business scenario. The Supplier is used within two different models: the Manage Warehouse and the Receiving Product. For the first model, the relationships of such element are presented, where an association with the Sending an Order entity is shown. For further details, that entity can be accessed in order to see its participation on other models or views.

- *The ability of switching across views.*

This particular option is made possible with the same Usage screen previously presented. From one particular entity, it is possible to see the different models on which the entity is used of defined, as well as the different views that makes usage of models which, in turn, have references to such entities.

- *An enhanced knowledge regarding the state, values, and relationships of those objects, independently of the context in which they are defined.*

The objects described above are the instances of particular elements which are defined within different models. The enhanced knowledge of their relationships are achieved through the mechanism early described. The state and values of an element are known when such element is assigned to a given execution of such process. By taking the Manage Request model defined above, an instance is presented in Figure 5.7.

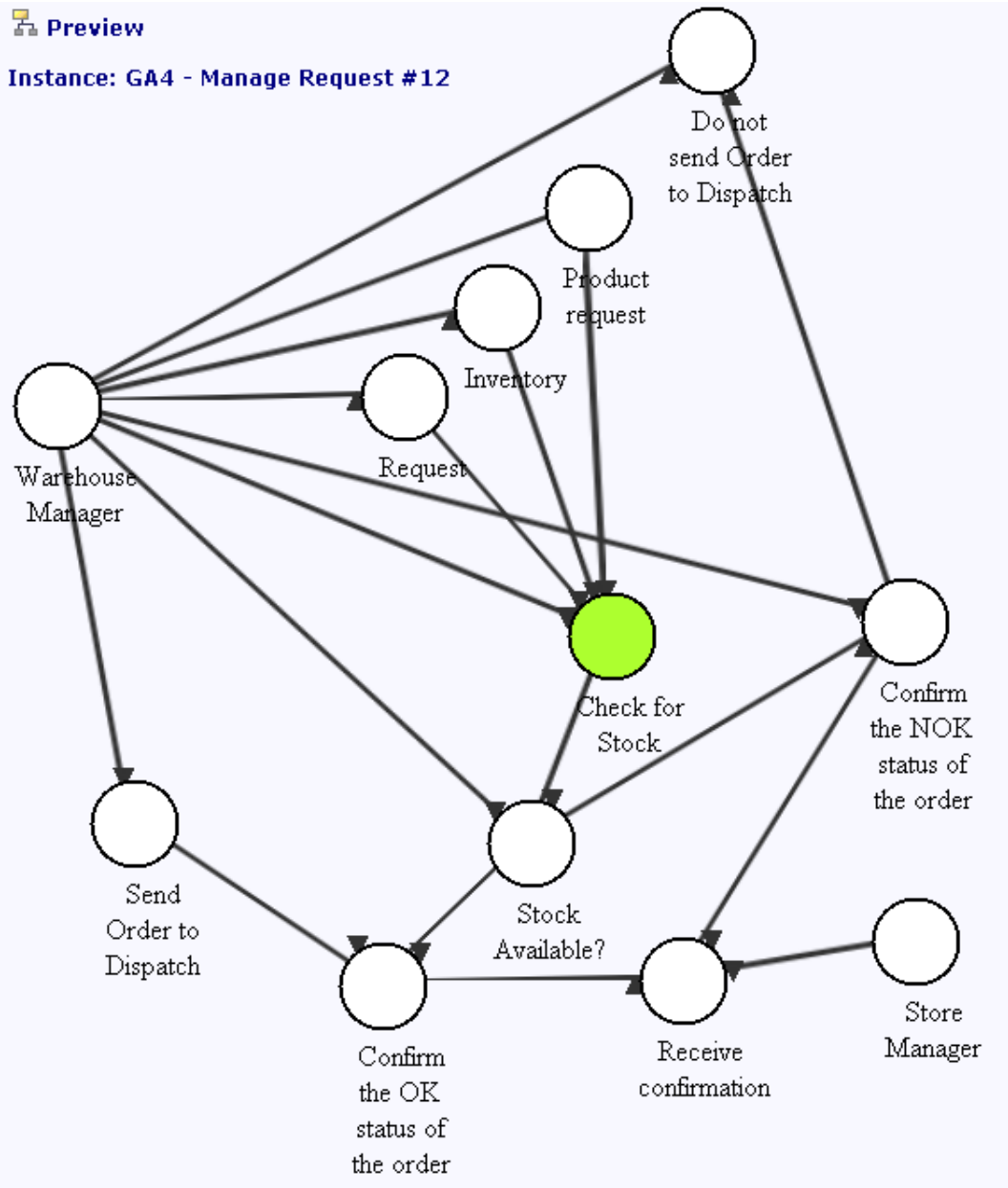

Figure 5.7: Manage Request Instance

This instance simulates a process in execution, where its active entity is the Check For Stock activity. As analysed in Chapter 2.5, an element can have different properties allowing different types of analyse

and manipulation activities upon entities or models defined within the system. As stated in Chapter 4, the proposed solution aims to a set of properties which can be used without any architectural domain restraints. Therefore, both the properties of state and privilege are assigned to each entity on the proposed solution.

An instance regards an entity with a given state which can evolve throughout time. Therefore, following the Figure 5.7, the Check For Stock instance information can also be consulted at the entity Usage screen, which can be depicted in Figure 5.8.

Usage at Instances: GA4 - Manage Request #12

Instance details:

Instance name: GA4 - Manage Request

Linked to:

Destination Entity	Relationship	Description	Arity
Stock Available?	Sequence Flow	Sequence Flow	1...1

Linked by:

Source Entity	Relationship	Description	Arity
Warehouse Manager	supply	responsible for	1...1
Product request	Sequence Flow	Sequence Flow	1...1
Inventory	input	input	1...1
Request	input	input	1...1

Currently active? TRUE

Current state: Waiting response to request form #12345

Figure 5.8: Check For Stock Instance information

- *The conformance of the meta-model with real enterprise data*

This aspect regards the impact of using a domain-independent meta-model, which in turn is used by each domain-specific meta-model, thereby imposing a restrain upon all the modelled elements within each domain. The proposed business case simulated a real enterprise concern, with a set of particular needs. Those needs were satisfied by the proposed solution, as observed throughout this Chapter.

- *The ability to extend the organisational awareness about its enterprise objects*

This aspect summarises all the effort put upon this dissertation. By empowering a solution with a bottom-up approach from any element in order to achieve a holistic view of all the enterprise architecture, the enterprise knowledge was enhanced without the limitations presented at other enterprise architecture solutions.

Chapter 6

Conclusion

6.1 Final Considerations

Several architectural frameworks have been proposed throughout time for establishing principles, recommendations, languages, views and viewpoints for architectural description. Along with these frameworks, several modelling languages have been developed and improved. The variety of types and notations used to represent the different enterprise elements has led to an effort for establishing well-defined standards. However, the acceptance of an ideal an unique modelling language or architectural framework is impossible given the heterogeneity of industries, domains and concerns.

Nevertheless, some modelling languages have been more successful than others. One example is the UML, which has been taken as the most expressive, standard compliant, general-purpose modelling language. However, as their counterparts, it lacks important aspects which have consequences for communication across the organisation. It lacks a well defined semantic and a well defined meta-model. Those faults lead to integrity and communication issues across views and across the elements represented through the language.

These impairments led to the development of the solution proposed on this thesis, whose goals are the enhancement of the integrity and relationships between enterprise elements and across views. Therefore, it was presented a meta-model with a proposed set of generic concepts, based on a classification scheme used by a well-known architecture framework. For setting up the most suitable domains and views for an industry-neutral approach, it was used the TOGAF framework for establishing the domains, and UML diagrams for establishing the views. It was created a meta-model for each defined domain, using elements from appropriate UML diagrams. Besides those defined elements, a standard compliant infrastructure was proposed for managing the relationships between elements, and for improving their traceability across views. With this proposal it was intended to maintain the flexibility and extensibility provided by UML, while tackling the integrity and communication limitations of enterprise knowledge.

An Enterprise Architecture prototype within a Content Management System was developed. The solution was used for a business case scenario simulation, thus showing the practicability of the proposal and the surpassing of certain limitations within the enterprise architecture knowledge.

6.2 Future Work

The infrastructure and toolkit's API made available by this dissertation can be used as a middleware tool for domain-specific solutions, thus taking advantage of the standard compliant capabilities presented. By using a standard compliant infrastructure, new analysis and manipulation techniques can be applied to

the enterprise architecture data. Therefore, this solution settles an extensible and flexible structure for new developments for exploring the enhanced relationships across architectural elements.

Within the Content Management System scope, this solution can be used for improving the relationships between content, thus allowing mechanisms for automatic information retrieval. By taking advantage of the enhanced relationships, the content can be associated to different entities and their relationships can be used for different navigation experiences throughout a website.

Bibliography

[1] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, and Y. Shaham-Gafni. Model Traceability. *IBM Systems Journal*, 45(3):515–526, 2006.

[2] ASG. Rochade. http://www.asg.com.

[3] BiZZdesign. BiZZdesigner. http://www.bizzdesign.com.

[4] G. Booch, J. Rumbaugh, and I. Jacobson. *Unified Modeling Language User Guide, The (2nd Edition) (The Addison-Wesley Object Technology Series)*. Addison-Wesley Professional, May 2005.

[5] CapGemini. Integrated Architecture Framework. http://www.capgemini.com/services/soa/ent_architecture/iaf/.

[6] J. Carmo and A. Silva. The WebComfort Project. October 2006.

[7] CIO-Council. Federal Enterprise Architecture Framework, September 1999.

[8] J. Cleland-Huang, C. K. Chang, and M. Christensen. Event-Based Traceability for Managing Evolutionary Change. *IEEE Trans. Softw. Eng.*, 29(9):796–810, 2003.

[9] Department of Defence. Department of Defence Architecture Framework, August 2003.

[10] EasyDiagram. EasyDiagram.NET. http://www.easydiagram.net.

[11] R. Freude and A. K. onigs. Tool Integration with Consistency Relations and Their Visualization. In *ESEC/ FSE Workshop on Tool Integration in System Development*, pages 6–10, Helsinki, Finland, September 2003.

[12] D. Garlan, R. Monroe, and D. Wile. ACME: an architecture description interchange language. In *CASCON '97: Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 7. IBM Press, 1997.

[13] Gartner. The Gartner Enterprise Architecture Framework. http://www.gartner.com/pages/story.php.id.2226.s.8.jsp.

[14] J. Hayes, A. Dekhtyar, and J. Osborne. Improving requirements tracing via information retrieval. *Requirements Engineering Conference, 2003. Proceedings. 11th IEEE International*, pages 138–147, Sept. 2003.

[15] IBM. Rational Software. Http://www.ibm.com/software/rational.

[16] IBM. Requisite Pro. http://www-306.ibm.com/software/awdtools/reqpro/.

[17] IDS SCHEER. ARIS Toolset. http://www.ids-scheer.com.

[18] IEEE. Recommended Practice for Architectural Description of Software-Intensive Systems. *ISO IEC 42010 IEEE Std 1471-2000 First edition 2007-07-15*, pages c1–24, 2007.

[19] H. Jonkers, R. van Buuren, F. Arbab, F. de Boer, M. Bonsangue, H. Bosma, H. ter Doest, L. Groenewegen, J. G. Scholten, S. Hoppenbrouwers, M.-E. Iacob, W. Janssen, M. Lankhorst, D. van Leeuwen, E. Proper, A. Stam, L. van der Torre, and G. V. van Zanten. Towards a Language for Coherent Enterprise Architecture Descriptions. In *EDOC '03: Proceedings of the 7th International Conference on Enterprise Distributed Object Computing*, page 28, Washington, DC, USA, 2003. IEEE Computer Society.

[20] P. Kruchten. The 4+1 View Model of Architecture. *IEEE Softw.*, 12(6):42–50, 1995.

[21] M. Lankhorst. *Enterprise Architecture at Work: Modelling, Communication and Analysis*. Springer, 2005.

[22] P. Letelier. A Framework for Requirements Traceability in UML-based Projects. In *Proceding of the 1st International Workshop of Traceability in Emerging Forms of Software Engineering*, pages 30–41, Edinburgh, Scotland, September 2002.

[23] D. C. Luckham, J. J. Kenney, L. M. Augustin, J. Vera, D. Bryan, and W. Mann. Specification and analysis of system architecture using Rapide. *IEEE Transactions on Software Engineering*, 21:336–355, 1995.

[24] R. Magalhães, M. Zacarias, and J. Tribolet. Making Sense of Enterprise Architectures as Tools of Organizational Self-Awareness (OSA). page 8, April 2007.

[25] J. Magee and J. Kramer. Dynamic structure in software architectures. *SIGSOFT Softw. Eng. Notes*, 21(6):3–14, 1996.

[26] R. Mayer, C. Menzel, M. Panther, P. DeWitte, T. Blinn, and B. Perakath. IDEF3 Process Description Capture Method Report. Interim technical report, Knowledge Based Systems, College Station, Texas, September 1995.

[27] N. Medvidovic, P. Oreizy, J. E. Robbins, and R. N. Taylor. Using object-oriented typing to support architectural design in the C2 style. In *SIGSOFT '96: Proceedings of the 4th ACM SIGSOFT symposium on Foundations of software engineering*, pages 24–32, New York, NY, USA, 1996. ACM.

[28] N. Medvidovic and R. N. Taylor. A Classification and Comparison Framework for Software Architecture Description Languages. *IEEE Trans. Softw. Eng.*, 26(1):70–93, 2000.

[29] C. Menzel and R. J. Mayer. The IDEF family of languages. *Handbook on Architectures of Information Systems*, 1:209–241, 1998.

[30] Microsoft. Enterprise Architecture. http://msdn.microsoft.com/en-us/architecture/bb469938.aspx.

[31] Object Management Group. BPMN: Business Process Modeling Notation. http://www.bpmi.org.

[32] U. D. of Commerce. *Integration Definition for Function Modeling (IDEF0) Draft, Federal Information Processing Standards Publication FIPSPUB 183*. Springfield, Virginia.

[33] J. R. Putman. *Architecting with RM-ODP*. Prentice-Hall, 1991.

[34] B. Ramesh and M. Jarke. Toward Reference Models for Requirements Traceability. *IEEE Trans. Softw. Eng.*, 27(1):58–93, 2001.

[35] J. Saraiva and A. R. da Silva. The WebComfort Framework: an Extensible Platform for the Development of Web Applications. September 2008.

[36] A.-W. Scheer. Architecture of Integrated Information Systems (ARIS). In *DIISM '93: Proceedings of the JSPE/IFIP TC5/WG5.3 Workshop on the Design of Information Infrastructure Systems for Manufacturing*, pages 85–99, Amsterdam, The Netherlands, The Netherlands, 1993. North-Holland Publishing Co.

[37] J. Schekkerman. *How to Survive in the Jungle of Enterprise Architecture Framework: Creating or Choosing an Enterprise Architecture Framework*. Trafford, 2004.

[38] J. Schekkerman. Enteprise Architecture Tool Selection Guide, 2009.

[39] S. A. Sherba, K. M. Anderson, and M. Faisal. A Framework for Mapping Traceability Relationships. In *Proceedings of the 2nd International Workshop on Traceability in Emerging Forms of Software Engineering*, Montreal, Canada, September 2003.

[40] P. Sousa, C. Pereira, and J. Marques. Enterprise Architecture Alignment Heuristics. *Journal4*, pages 34–39, 2004.

[41] P. Sousa, C. Pereira, R. Vendeirinho, A. Caetano, and J. Tribolet. Applying the Zachman Framework Dimension to support Business Process Modeling. September 2006.

[42] J. F. Sowa and J. A. Zachman. Extending and formalizing the framework for information systems architecture. *IBM Syst. J.*, 31(3):590–616, 1992.

[43] Sparx. Enterprise Architect. http://www.sparxsystems.com.

[44] A. Tang, J. Han, and P. Chen. A Comparative Analysis of Architecture Frameworks. In *APSEC '04: Proceedings of the 11th Asia-Pacific Software Engineering Conference*, pages 640–647, Washington, DC, USA, 2004. IEEE Computer Society.

[45] Telelogic. DOORS. http://www.telelogic.com/products/doorsers/index.cfm.

[46] The Open Group. TOGAF: The Open Group Architecture Framework. http://www.opengroup.org/togaf/.

[47] V. Vaishnavi and W. Kuechler. Design Research in Information Systems. URL: http://www.isworld.org/Researchdesign/drisISworld.htm.

[48] A. Vasconcelos, A. Caetano, J. Neves, P. Sinogas, R. Mendes, and J. Tribolet. A Framework for Modeling Strategy, Business Processes and Information Systems. In *Proceedings of 5th International Enterprise Distributed Object Computing Conference*, Seatle, USA, September 2001. EDOC.

[49] A. Vasconcelos, P. Sousa, and J. Tribolet. Information System Architectures: Representation, Planning and Evaluation. In *Proceedings of International Conference on Computer, Communication and Control Technologies*, Orlando, USA, July 2003. EDOC.

[50] R. Watson. Smarter Requirements Management with Intelligent Traceability. In *White Paper*, Irvine, CA, July 2003. Telelogic North America Inc.

[51] M. Zacarias, A. Caetano, R. Magalhães, H. Pinto, and J. Tribolet. Adding a Human Perspective to Enterprise Architectures. *International Workshop on Database and Expert Systems Applications*, 0:840–844, 2007.

[52] J. A. Zachman. A Framework for Information Systems Architecture. *IBM Syst. J.*, 26(3):276–292, 1987.

Appendix A

Enterprise Architecture Tools

Institute For Enterprise Architecture Developments

Risk / Strategy / Enterprise / Solution Architecture Tools Overview - 2009 v5.0

Supplier	Tool	Togaf 9 Support	Modelling Languages Support (BPMN, Archimate, UML)
Aam tech	SAMU	Not Specified	Not Specified
Acceptsoftware	Acoept 360	Not Specified	Not Specified
Adaptive	Adaptive EA Manager, IT Portfolio Manager, Metadata Manager, Project Portfolio Manager	Not Specified	Not Specified
Agilense	EA Webmodeler	Not Specified	Not Specified
Altova	Altova Enterprise Suite	Not Specified	UML
Alfabet	Planning IT	Togaf 9	Not Specified
ASG Software Solutions	ASG-Rochade	Not Specified	Not Specified
Avolution	Abacus	Togaf 9	Archimate, BPMN, UML
Bizzdesign	BiZZdesign Architect, BiZZdesigner, Riskmanager	Togaf 9	Archimate
Casewise	Corporate Modeler Enterprise Edition	Togaf 9	Archimate, BPMN, UML
CACI International	SimProcess	Not Specified	BPMN
Enterprise Elements	Elements Repository	Not Specified	UML
Forsight	Modeling & Validation Tool set	Not Specified	BPMN
Future Tech Systems, Inc	ENVISION® VIP	Not Specified	BPMN
GoAgile	GoAgile MAP Product Suite	Not Specified	Not Specified
IBM	IBM Rational Software Architect	Not Specified	UML
IDS Scheer	ARIS Business Performance Edition	Togaf 9	Archimate, BPMN
Intelligile Corporation	MAP Suite + ITAA	Togaf 9	Not Specified
Knotion Consulting	UDEF Explorer	Togaf 9	Not Specified
LogicLibrary	Logidex	Not Specified	UML
Méga International	Méga (Process, Architect, Designer)	Togaf 9	BPMN, UML
CA	NetViz Suite	Not Specified	Not Specified
Palisade	Risk & Decision Analysis	Not Specified	Not Specified
Metastorm	Metastorm Enterprise Products	Togaf 9	BPMN
Qualiware	Qualiware Product Suite	Not Specified	Not Specified
Salamander Organisation	MooD Transformation Technology	Not Specified	Not Specified
Select Business Solutions	Select Solution Factory	Not Specified	BPMN, UML
Simon Labs	SimonTool	Not Specified	Not Specified
Sparx Systems	Enterprise Architect	Togaf 9	BPMN, UML
IBM - TeleLogic	System Architect Family + Rhapsody	Not Specified	Archimate, BPMN, UML
Troux	Troux 8	Togaf 9	Archimate, BPMN, UML
Visible	Visible Advantage	Not Specified	Not Specified

Figure A.1: Enterprise Architecture Tools

Appendix B

Toolkit's Database

In this section the toolkit's database tables, functions and stored procedures are presented. Apart from the stored procedures used for inserting, updating or deleting entries at each table, some specific procedures where created for taking advantage of the database design for enterprise architecture purposes. Those stored procedures will be further described in the following sections. Therefore, only the relevant operations are described, for reviewing or future work purposes. Therefore, it follows a brief description of each database element.

Description of database tables

Table:	**DiagramInfo**
Columns:	*id* - unique ID for a particular graph *ModelId* - unique ID for a particular model *EntityId* - unique ID for a particular entity (concept or relation) *xPos* - relative horizontal position of the given entity within the graph. *yPos* - relative vertical position of the given entity within the graph.
Description:	The table keeps all information regarding the previews for multiple models. Each entry essentially describes the horizontal and vertical position of each entity for a particular model instantiation.

Table:	**InstantiatedModels**
Columns:	*id* - unique ID for a particular model instance *Instance_Of* - unique ID for a particular model *Enabled* - flag describing the element's visibility (1 for Enabled, 0 for Disabled) *ActiveStep* - unique ID representing the active entity for the given instance.
Description:	The table keeps all information regarding model instantiations, where each entry describes the instance's current state.

Table:	**InstantiatedEntities**
Columns:	*id* - unique ID for a particular entity instance at a particular model instance. *Instantiated_Model* - unique ID for a particular model instance *Related_Entity* - unique ID of the entity being instantiated *Current_State* - current entity state from the available states of that entity at a given model.
Description:	The table keeps all information regarding entities' instances used at different models instances, as well as their current values.

Table: **ModelsToEntities**
Columns: *id* - unique ID for a reference between a particular entity at a particular model
Model - unique ID for a particular model
Entity - unique ID of the entity
Entity_States - available states for the given entity.
Description: The table keeps all information regarding entities defined or used at different models, as well as their available values within those models.

Table: **ModelsToRelationships**
Columns: *id* - unique ID for a reference between a particular relationship at a particular model
Model - unique ID for a particular model
Relationship - unique ID of the relationship
Description: The table keeps all information regarding the relationships defined and used at different models.

Table: **Models**
Columns: *id* - unique ID for a particular model
Model_Name - name for a model
Sibling_Of - hierarchical parent of the model
Model_Domain - architectural domain where the model is defined.
Enabled - visibility of the model within the system.
Description - rationale of the model
Hierarchy_Level - modelling context of the model
Description: The table keeps all information regarding the models description, but does not reference its the concepts, relations nor relationships.

Table: **Relationships**
Columns: *id* - unique ID for a particular relationship
Source - unique ID for the relationship's source entity
Destination - unique ID for the relationship's destination entity
Relation - unique ID representing the relation instantiated by the relationship
Relationship_Name - name for a relationship.
Relationship_Arity - unique ID representing the multiplicity for a relationship.
Description: The table keeps all information regarding the relationships used by the different models defined within the system.

Table: **Arities**
Columns: *id* - unique ID for a particular arity
Description - rationale for an arity.
Description: The table keeps all information regarding the different multiplicities for all the relationships defined.

Table: **Domains**
Columns: *id* - unique ID for a particular domain
Sibling_Of - domain's hierarchical parent.
Description - rationale for an architectural domain.
Description: The table keeps all information regarding the different architectural domains where all the views, models and entities are defined.

Table: **Hierarchies**
Columns: *id* - unique ID for a particular hierarchy
Description - rationale for the different modelling contexts.
Description: The table keeps all information regarding the different modelling contexts available within the system.

Table: **ViewpointsToModels**
Columns: *id* - unique ID for a reference between a particular viewpoint and a particular model
Model - unique ID for a particular model
Viewpoint - unique ID for a particular viewpoint
Description: The table keeps all information regarding the models referred by a viewpoint's rationale.

Table: **ViewsToViewpoints**
Columns: *id* - unique ID for a reference between a particular viewpoint and a particular view
View - unique ID for a particular view
Viewpoint - unique ID for a particular viewpoint
Description: The table keeps all information regarding the viewpoints referred by views.

Table: **Viewpoints**
Columns: *id* - unique ID for a particular viewpoint
Viewpoint_Name - name for a viewpoint.
Hierarchy_Level - modelling context where the viewpoint is defined
Description - rationale for the viewpoint.
Description: The table keeps all information regarding the viewpoints defined within the system.

Table: **ViewsToModels**
Columns: *id* - unique ID for a reference between a particular view and a particular model
View - unique ID for a particular view
Model - unique ID for a particular model
Description: The table keeps all information regarding the models referred by views.

Table: **Views**
Columns: *id* - unique ID for a particular view
View_Name - name for a view.
Hierarchy_Level - modelling context where the view is defined
Description - rationale for the view.
Description: The table keeps all information regarding the views defined within the system.

Table: **Entity_Types**
Columns: *id* - unique ID for a particular entity type
Sibling_Of - hierarchical parent for an entity type (or subtype).
Description - rationale for an entity type.
Description: The table keeps all information regarding the different entity types (structural or behavioural) and subtypes for all the entities defined.

Table: **Entity_Metatypes**
Columns: *id* - unique ID for a particular entity metatype
Description - rationale for the different entity metatypes.
Description: The table keeps all information regarding the different entity metatypes (concept or relation) available within the system.

Table: **Entities**
Columns: *id* - unique ID for a particular entity (concept or relation)
Domain - unique ID representing the architectural domain where the entity is defined.
Entity_Type - unique ID representing the entity's type or subtype
Entity_Name - name for an entity
Description - rationale of the entity
Sibling_Of - hierarchical parent of the entity
Metatype - unique ID representing the entity's metatype (concept or relation)
Enabled - visibility of the entity within the system.
Hierarchy_Level - modelling context of the entity
Description: The table keeps all information regarding the entities described within the system.

Table: **Privileges**
Columns: *id* - unique ID for a particular privilege upon an entity, view or model
Element_Type - unique ID representing the type of architectural element to assign a privilege (ENTITY, VIEW or MODEL).
TargetId - unique ID representing the architectural element
Privileged_Entity - unique ID representing the entity to be granted with the privilege
Can_Create - flag (1 or 0) representing the creation grant
Can_Read - flag (1 or 0) representing the access grant
Can_Update - flag (1 or 0) representing the update grant
Can_Delete - flag (1 or 0) representing the delete grant
Description: The table keeps all information regarding the privileges assigned to entities across the system.

Relevant functions and/or stored procedures

Procedure: **EntityHierarchyTree**
Arguments: *entityId* - unique ID for a particular entity
Description: Returns an ordered hierarchy for a particular entity, from its root element till its current definition. The number of rows retrieved are equal to degree of specialisation for the given entity.

Procedure: **EntityUsageAtInstances**
Arguments: *entityId* - unique ID for a particular entity
Description: Retrieves a row for each instance of the given entity within the system, as well as its current state at each instantiated model.

Procedure: **EntityUsageAtInstances_ConnectedBy**
Arguments: *entityId* - unique ID for a particular entity
Description: Retrieves a row for each entity instance with a relationship to the given entity.

Procedure: **EntityUsageAtInstances_ConnectedTo**
Arguments: *entityId* - unique ID for a particular entity
Description: Retrieves a row for each entity instance referred by a relationship of the given entity.

Procedure: **EntityUsageAtModels**
Arguments: *entityId* - unique ID for a particular entity
Description: Retrieves a row for each model where the entity is referred or defined.

Procedure: **EntityUsageAtModels_ConnectedBy**
Arguments: *entityId* - unique ID for a particular entity
Description: For each model where the entity is used, a row for each entity with a relationship to the given entity is retrieved.

Procedure: **EntityUsageAtInstances_ConnectedTo**
Arguments: *entityId* - unique ID for a particular entity
Description: For each model where the entity is used, a row for each entity referred by a relationship of the given entity is retrieved.

Procedure: **EntityUsageAtViews**
Arguments: *entityId* - unique ID for a particular entity
Description: Retrieves a row for each view where the entity is referred.

Appendix C

Toolkit's API

In this section the toolkit's API is presented, featuring the list of interfaces regarding the creation, access, update and delete operations for each architectural element. Both the basic and complex operations are described, for reviewing or future work purposes. Therefore, it follows a brief description of each interface, its arguments and return values.

Reading Operations

Function: **GetDiagramInfo**
Arguments: *diagramInfoId* - unique ID for a particular graph
modelId - unique ID for a particular model
entityId - unique ID for a particular entity (concept or relation)
Description: Returns a list of *DiagramInfoDTO* objects with information regarding the relative position of one or more entities at one or more models. The arguments impose the depth of entity filtering, thus allowing null or -1 values for any arguments.

Function: **GetModels**
Arguments: *modelName* - name of a particular model
domainId - unique ID for a particular architectural domain
associatedModelId - unique ID of a hierarchical parent model
Description: Returns a list of *vModelsDTO* objects with information regarding models, its correspondent concepts, relations and links between concepts. The values assigned to each argument (which can be null) determine the number of models retrieved.

Function: **GetConcepts**
Arguments: *name* - name of a particular concept
domain - unique ID for a particular architectural domain
type - unique ID of a entity type (structural or behavioural)
Description: Returns a list of *vConceptsDTO* objects with information regarding the concept description, its architectural domain, topology and hierarchy. The values assigned to each argument (which can be null) determine the number of concepts retrieved.

Function: **GetRelations**
Arguments: *name* - name of a particular relation
domain - unique ID for a particular architectural domain
type - unique ID of a entity type (structural or behavioural)
Description: Returns a list of *vRelationsDTO* objects with information regarding the relation description, its architectural domain, topology and hierarchy. The values assigned to each argument (which can be null) determine the number of concepts retrieved.

Function: **GetViews**
Arguments: *name* - name of a particular view
Description: Returns a list of *vViewsDTO* objects with information regarding the view description.

Function: **GetInstances**
Arguments: *name* - name of a particular instantiated model
Description: Returns a list of *vInstantiatedModelsDTO* objects with information regarding the instance description, its active entity and state.

Function: **GetViewPoints**
Arguments: *name* - name of a particular viewpoint
Description: Returns a list of *vViewPointsDTO* objects with information regarding the viewpoint description.

Function: **GetModelsToEntities**
Arguments: *modelIds* - list of unique IDs identifying models within the system
Description: Returns a list of *ModelToEntitiesDTO* objects with information regarding the entities (concepts and relations) used within one or more models.

Function: **GetEntityPrivileges**
Arguments: *entityIds* - list of unique IDs identifying entities within the system
Description: Returns a list of *PrivilegesDTO* objects with information regarding the assigned privileges for a set of entities.

Function: **GetInstances**
Arguments: *instanceIds* - list of unique IDs identifying instantiated models within the system
Description: Returns a list of *vInstantiatedModelsDTO* objects with information regarding instantiated models within the system.

Function: **GetEntities**
Arguments: *entityIds* - list of unique IDs identifying entities within the system
Description: Returns a list of *vEntitiesDTO* objects with information regarding the entity description, its architectural domain, topology and hierarchy. An entity can be a concept or a relation.

Function: **GetConcepts**
Arguments: *conceptId* - list of unique IDs identifying concepts
Description: Returns a list of *vConceptsDTO* objects with information regarding the concept description, its architectural domain, topology and hierarchy.

Function: **GetConceptsFromModels**
Arguments: *modelIds* - list of unique IDs identifying models
conceptIds - list of unique ID identifying concepts
Description: Returns a list of *vConceptsDTO* objects with information regarding the concept description, its architectural domain, topology and hierarchy. The concepts retrieved belong to a given model. The values assigned to each argument (which can be null) determine the number of concepts retrieved.

Function: **GetRelations**
Arguments: *relationIds* - list of unique IDs identifying relations
Description: Returns a list of *vRelationsDTO* objects with information regarding the relation description, its architectural domain, topology and hierarchy.

Function: **GetRelationsFromModels**
Arguments: *modelIds* - list of unique IDs identifying models
relationIds - list of unique ID identifying relations
Description: Returns a list of *vRelationsDTO* objects with information regarding the relation description, its architectural domain, topology and hierarchy. The relations retrieved belong to a given model. The values assigned to each argument (which can be null) determine the number of relations retrieved.

Function: **GetRelationships**
Arguments: *relationshipIds* - list of unique IDs identifying relationships between concepts
Description: Returns a list of *vRelationshipsDTO* objects with information regarding the relationship description, its arity, the relation and the two concepts referred.

Function: **GetRelationshipsFromModels**
Arguments: *modelIds* - list of unique IDs identifying models
relationshipIds - list of unique IDs identifying relationships between concepts
Description: Returns a list of *vRelationshipsDTO* objects with information regarding the relationship description, its arity, the relation and the two concepts referred. The relations retrieved belong to a given model. The values assigned to each argument (which can be null) determine the number of relationships retrieved.

Function: **GetModels**
Arguments: *modelIds* - list of unique IDs identifying models
Description: Returns a list of *vModelsDTO* objects with information regarding models, its correspondent concepts, relations and links (relationships) between concepts.

Function: **GetModelsFromViewPoints**
Arguments: *modelIds* - list of unique IDs identifying models
viewpointIds - list of unique IDs identifying viewpoints
Description: Returns a list of *vModelsDTO* objects with information regarding models, its correspondent concepts, relations and links (relationships) between concepts. The models retrieved are referred in a given viewpoint. The values assigned to each argument (which can be null) determine the number of models retrieved.

Function: **GetModelsFromViews**
Arguments: *modelIds* - list of unique IDs identifying models
viewIds - list of unique IDs identifying views
Description: Returns a list of *vModelsDTO* objects with information regarding models, its correspondent concepts, relations and links (relationships) between concepts. The models retrieved are referred in a given view. The values assigned to each argument (which can be null) determine the number of models retrieved.

Function: **GetViewPoints**
Arguments: *viewpointIds* - list of unique IDs identifying viewpoints
Description: Returns a list of *vViewPointsDTO* objects with information regarding the viewpoint description.

Function: **GetViewpointsFromViews**
Arguments: *viewpointIds* - list of unique IDs identifying viewpoints
viewIds - list of unique IDs identifying views
Description: Returns a list of *vViewPointsDTO* objects with information regarding the viewpoint description. The viewpoints retrieved are referred in a given view. The values assigned to each argument (which can be null) determine the number of viewpoints retrieved.

Function: **GetViews**
Arguments: *viewIds* - list of unique IDs identifying views
Description: Returns a list of *vViewsDTO* objects with information regarding the view description.

Function: **GetArities**
Arguments: *arityIds* - list of unique IDs identifying arities
Description: Returns a list of *AritiesDTO* objects with information regarding the different relationship multiplicities between concepts.

Function: **GetDomains**
Arguments: *domainIds* - list of unique IDs identifying architectural domains
Description: Returns a list of *DomainsDTO* objects with information regarding the different architectural domains.

Function: **GetHierarchies**
Arguments: *hierarchyIds* - list of unique IDs identifying different hierarchical contexts
Description: Returns a list of *HierarchiesDTO* objects with information regarding the enterprise modelling context (e.g. Meta-modelling or Enterprise Objects).

Function: **GetEntityTypes**
Arguments: *typeIds* - list of unique IDs identifying different entity types
Description: Returns a list of *Entity_TypesDTO* objects with information regarding the available entity topology (structural or behavioural).

Function: **GetEntityMetaTypes**
Arguments: *typeIds* - list of unique IDs identifying the entities' nature
Description: Returns a list of *Entity_MetatypesDTO* objects with information regarding the entity nature (concept or relation).

Function: **GetEntityUsageAtViews**
Arguments: *entityId* - unique ID identifying a particular entity
Description: Returns a list of *EntityUsageAtViewsDTO* objects with information regarding the usage of an entity across the Views defined within the system.

Function: **GetEntityUsageAtModels**
Arguments: *entityId* - unique ID identifying a particular entity
Description: Returns a list of *EntityUsageAtModelsDTO* objects with information regarding the usage of an entity across the Models defined within the system.

Function: **GetEntityUsageAtInstances**
Arguments: *entityId* - unique ID identifying a particular entity
Description: Returns a list of *EntityUsageAtInstancesDTO* objects with information regarding the entity instantiations defined across the system.

Function: **GetEntityHierarchy**
Arguments: *entityId* - unique ID identifying a particular entity
Description: Returns a list of *EntityHierarchyTreeDTO* objects with information regarding the entity hierarchy across different architectural domains.

Create Operations

Function: **CreateDiagramInfo**
Arguments: *di* - *DiagramInfoDTO* object with information regarding the relative position of a group of entities from a previewed model.
Description: Creates a new diagram within the database for a given model.

Function: **CreateInstance**
Arguments: *instance* - *InstantiatedModelsDTO* object with information regarding a model instantiation, its current state and active entity.
Description: Creates a new model instance within the database for a given model.

Function: **CreateEntity**
Arguments: *entity* - *EntityDTO* object with information regarding both a concept or a relation.
Description: Creates a new entity within the system, which can be a concept or a relation.

Function: **CreateEntity**
Arguments: *modelId* - unique ID identifying the model where the entity is being created.
entity - *EntityDTO* object with information regarding both a concept or a relation.
Description: Creates a new entity within the system from the definition of a particular model. Therefore, the entity is created, and is also linked to a given model.

Function: **CreateRelationship**
Arguments: *modelId* - unique ID identifying the model where the relationship is being created.
relationship - *RelationshipDTO* object with information regarding the linkage between two concepts within a particular model.
Description: Creates a new relationship within the system from the definition of a particular model. Therefore, the relationship is created, and is also linked to a given model.

Function: **CreateModel**
Arguments: *model* - *ModelsDTO* object with information regarding a particular model.
Description: Creates a new model within the system, with the information regarding only its description. Further specification of the model is done with update operations, where its concepts, relations and relationships are defined.

Function: **CreateModel**
Arguments: *viewpointId* - unique ID identifying a particular viewpoint.
model - *ModelsDTO* object with information regarding a particular model.
Description: Creates a new model within the system, with the information regarding only its description. The model is both created and linked to a viewpoint description. Further specification of the model is done with update operations, where its concepts, relations and relationships are defined.

Function: **CreateViewPoint**
Arguments: *viewpoint* - *ViewpointsDTO* object with information regarding a particular viewpoint.
Description: Creates a new viewpoint within the system, with the information regarding only its description. Further specification of the viewpoint is done with update operations, where model references are defined.

Function: **CreateView**
Arguments: *view* - *ViewsDTO* object with information regarding a particular view.
Description: Creates a new view within the system, with the information regarding only its description. Further specification of the view is done with update operations, where both viewpoint and model references are defined.

Function: **CreateModelToEntities**
Arguments: *modelId* - unique ID identifying a particular model.
selectedIds - list of unique IDs identifying a group of entities (concepts and relations).
Description: Creates multiple references for a model and each entity defined in its specification.

Function: **CreateViewPointToModels**
Arguments: *viewpointId* - unique ID identifying a particular viewpoint.
selectedIds - list of unique IDs identifying a group of models.
Description: Creates multiple references for a viewpoint and each model referred in its description.

Function: **CreateViewToViewpoints**
Arguments: *viewId* - unique ID identifying a particular view.
selectedIds - list of unique IDs identifying a group of viewpoints.
Description: Creates a reference of the viewpoint used within a view's specification.

Function: **CreateViewToModels**
Arguments: *viewId* - unique ID identifying a particular view.
selectedIds - list of unique IDs identifying a group of models.
Description: Creates a reference of each model used within a view's specification.

Update Operations

Function: **UpdateDiagramInfo**
Arguments: *id* - unique ID identifying a particular model diagram.
di - *DiagramInfoDTO* object with information regarding the relative position of a group of entities from a previewed model
Description: Updates the information from a previously created diagram, thus updating the relative positions of one or more entities for presentation purposes.

Function: **UpdateInstance**
Arguments: *id* - unique ID identifying a particular model instance.
instance - *InstantiatedModelsDTO* object with information regarding a model instantiation, its current state and active entity.
Description: Updates the information from a previously created instance, thus updating its current state and active entity.

Function: **UpdateEntity**
Arguments: *id* - unique ID identifying a particular entity (concept or relation).
entity - *EntityDTO* object with information regarding both a concept or a relation.
Description: Updates the information from a previously created entity (which can be a concept or a relation), thus updating its description, architectural domain or its defined hierarchy.

Function: **UpdateModel**
Arguments: *id* - unique ID identifying a particular model.
model - *ModelsDTO* object with information regarding a particular model.
Description: Updates the information from a previously created model, thus updating its description, but not its references to concepts, relations or relationships (which is made through other update operations).

Function: **UpdateViewPoint**
Arguments: *id* - unique ID identifying a particular viewpoint.
viewpoint - *ViewpointsDTO* object with information regarding a particular viewpoint.
Description: Updates the information from a previously created viewpoint, thus updating its description, but not its references to models (which is made through other update operations).

Function: **UpdateView**
Arguments: *id* - unique ID identifying a particular view.
view - *ViewsDTO* object with information regarding a particular view.
Description: Updates the information from a previously created view, thus updating its description, but not its references to viewpoints nor models (which are made through other update operations).

Function: **UpdateRelationship**
Arguments: *id* - unique ID identifying a particular relationship.
relationship - *RelationshipDTO* object with information regarding the linkage between two concepts within a particular model.
Description: Updates the information from a previously created relationship, thus updating its description, arity, and both source and destination concepts.

Delete Operations

Function: **DeleteDiagramInfo**
Arguments: *id* - unique ID identifying a particular model diagram.
Description: Deletes the information from a previously created diagram, and its references to relative positions of multiple entities.

Function: **DeleteInstance**
Arguments: *id* - unique ID identifying a particular model instance.
Description: Deletes the information from a previously created instance, and its references to entities and states.

Function: **DeleteEntity**
Arguments: *id* - unique ID identifying a particular entity (concept or relation).
Description: Deletes the information from a previously created entity (which can be a concept or a relation), encompassing its description, architectural domain and its defined hierarchy.

Function: **DeleteModel**
Arguments: *id* - unique ID identifying a particular model.
Description: Deletes the information from a previously created model, encompassing its description, and also its references to concepts, relations or relationships.

Function: **DeleteViewPoint**
Arguments: *id* - unique ID identifying a particular viewpoint.
Description: Deletes the information from a previously created viewpoint, encompassing its description, and also its references to models.

Function: **DeleteView**
Arguments: *id* - unique ID identifying a particular view.
Description: Deletes the information from a previously created view, encompassing its description, and also its references to viewpoints and models.

Function: **DeleteRelationship**
Arguments: *id* - unique ID identifying a particular relationship.
Description: Deletes the information from a previously created relationship, encompassing its description, arity, and both source and destination concepts.

Function: **DeleteModelToEntities**
Arguments: *modelId* - unique ID identifying a particular model.
entityId - unique ID identifying a particular entity (concept or a relation).
Description: Deletes the reference of a particular entity within a given model.

Function: **DeleteViewPointToModels**
Arguments: *viewpointId* - unique ID identifying a particular viewpoint.
modelId - unique ID identifying a particular model.
Description: Deletes the reference of a particular model within a given viewpoint.

Function: **DeleteViewToViewpoints**
Arguments: *viewId* - unique ID identifying a particular view.
viewpointId - unique ID identifying a particular viewpoint.
Description: Deletes the reference of a particular viewpoint within a given view.

Function: **DeleteViewToModels**
Arguments: *viewId* - unique ID identifying a particular view.
modelId - unique ID identifying a particular model.
Description: Deletes the reference of a particular model within a given view.