Evaluating Information Systems: 
Constructing a Model Processing Framework 

Relatório Final

João Dias Santa de Vasconcelos Duarte

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

Júri

Presidente: Prof. José Tribolet
Orientador: Prof. André Vasconcelos
Vogal: Prof. Pedro Sousa
Acompanhante: Eng. José Alegria

Junho de 2009
Dedico este trabalho à minha família e à minha namorada, Ana.

Não teria conseguido sem o vosso apoio.

Muito obrigado.
Abstract

Enterprises are growing increasingly hungry for information: the will to empower decisions with carefully digested data, the wish of transform all implicit knowledge into clearly defined, explicit repositories that can be shared by all and the desire to monitor all aspects of their internal dynamics. In the past decade, the rush to technology has created several flaws when it comes to managing computers, applications, middleware and information systems and so organisations struggle to understand how all these elements are behaving.

Even today, as Enterprise Architectures grow in significance and are acknowledged as advantageous artifacts to help manage change, their benefit to the organisation has yet to be fully explored.

We therefore combine our desire for real-time evaluation with the benefits of architecture representation to produce an ontologically supported method of modelling, capturing and evaluating systemic behaviour. We produce a conceptual framework to performing this task, while avoiding the imprecise definitions of quality and quality attributes. According to our abstraction, transferring that responsibility to the end user is essential since such aspects are subjective and depend on the human context in which they exist. This conceptualisation was materialised in a model-eval-display loop framework, and implemented using Model Driven Software Development practices and tools.

Finally, we tested the prototype against a real-world scenario in PT-Comunicações, to observe our conceptual solution in practice.

Keywords: Evaluation, Information System Architecture, Model Processing Framework, Phenomenology
Resumo

As empresas têm vindo gradualmente a atribuir uma maior importância à informação: o interesse em potenciar as decisões usando dados cuidadosamente processados e o desejo de transformar todo o conhecimento implícito em repositórios de conhecimento explícito para o fácil acesso de todos. Contudo, na última década, a corrida à tecnologia criou diversas falhas no que toca à gestão de computadores, aplicações, middleware e sistemas de informação, que causou dificuldades nas organizações em perceber como todos estes objectos se comportam.

Mesmo nos dias correntes, em que a importância das Arquitecturas Empresariais cresce e estas são reconhecidas como artefactos valiosos para gerir a mudança, o seu benefício não está completamente explorado. Deste modo, nós combinámos o desejo de ter avaliação de sistemas em real-time com os benefícios da representação arquitectural para produzir um método, suportado ontologicamente, de modelar, avaliar e capturar comportamento sistémico. Produzimos uma framework conceptual para executar esta tarefa, evitando as definições impresisas de qualidade e atributos de qualidade. De acordo com a nossa abstracção, transferir tal responsabilidade para o utilizador final é essencial uma vez que tais aspectos são subjetivos e dependem do contexto humano onde estes existem. Esta conceptualização foi materializada numa framework de um ciclo de modelação-avaliação-visualização que foi implementado usando práticas e ferramentas de Model Driven Software Development.

Finalmente, testámos o protótipo num cenário real da PT-Comunicações, onde pudemos observar a nossa solução conceptual em prática.

Palavras-chave: Avaliação, Arquitectura de Sistemas de Informação, Framework de Processamento de Modelos, Fenomenologia
Index

Abstract iii

Resumo v

List of Figures xi

List of Tables xiii

Acronyms xv

1 Introduction 1

1.1 The world we live in and the need for Organizational Self-Awareness 1

1.1.1 Architectural Representation of Organisations 1

1.2 Monitoring Information Technology behaviour: Next steps 1

1.3 Main contributions 2

1.4 Following chapters 2

2 Related Work 5

2.1 Enterprise Architectures 5

2.1.1 Unified Enterprise Architecture Modelling Language and LEAN 5

2.1.2 CEO Framework 7

2.1.3 ArchiMate 9

2.1.4 Summary of Enterprise Architectures 10

2.2 Evaluating Information Technology: metrics and measuring qualities 11

2.2.1 Quality Meta-Model 11

2.2.2 Quality of Service and Quality of Protection 11

2.2.3 Key Performance Indicators 12

2.2.4 Software Metrics 12

2.2.5 Conclusion 13

2.3 Instrumentation 13

2.3.1 Unified Modelling Language 13

2.3.2 Eclipse Project 14

2.3.3 openArchitectureWare 15

2.3.4 Enterprise Architect 16

2.3.5 Comparison and Conclusion 16

2.4 Business Process Management 17

2.4.1 Business Activity Monitoring 18

2.5 On providing quality views over Information System Architectures 20
## List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Layered Architecture Meta-Model</td>
<td>6</td>
</tr>
<tr>
<td>2.2</td>
<td>Graphical representation of LEAN nodes</td>
<td>7</td>
</tr>
<tr>
<td>2.3</td>
<td>CEO Framework Meta-Model</td>
<td>8</td>
</tr>
<tr>
<td>2.4</td>
<td>Archimate Meta-Model</td>
<td>9</td>
</tr>
<tr>
<td>2.5</td>
<td>Archimate Performance Views</td>
<td>10</td>
</tr>
<tr>
<td>2.6</td>
<td>Quality Meta-Model</td>
<td>12</td>
</tr>
<tr>
<td>2.7</td>
<td>UML Layers</td>
<td>14</td>
</tr>
<tr>
<td>2.8</td>
<td>openArchitectureWare structure</td>
<td>15</td>
</tr>
<tr>
<td>2.9</td>
<td>The Business Process Management cycle</td>
<td>17</td>
</tr>
<tr>
<td>2.10</td>
<td>Positioning of BAM systems in relation to the stakeholder and delivery latency</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Steps in constructing and applying a High-Level Language</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>LEAN node pairings with changes to model evaluation</td>
<td>26</td>
</tr>
<tr>
<td></td>
<td>(1) The “produces” pairing exists only for the Agent → Resource relationship</td>
<td></td>
</tr>
<tr>
<td>3.3</td>
<td>Evaluation ontology using LEAN</td>
<td>27</td>
</tr>
<tr>
<td>3.4</td>
<td>a) Standard ISA building approach b) approach proposed in [Vas07]</td>
<td>28</td>
</tr>
<tr>
<td>3.5</td>
<td>Proposal for pipeline with Information Systems evaluation</td>
<td>28</td>
</tr>
<tr>
<td>3.6</td>
<td>Model Processing Framework</td>
<td>29</td>
</tr>
<tr>
<td>3.7</td>
<td>Model Converter</td>
<td>29</td>
</tr>
<tr>
<td>3.8</td>
<td>Model Analyser</td>
<td>30</td>
</tr>
<tr>
<td>3.9</td>
<td>Expanded view of the Model Analyser</td>
<td>31</td>
</tr>
<tr>
<td>3.10</td>
<td>Results Converter</td>
<td>31</td>
</tr>
<tr>
<td>3.11</td>
<td>ODE's Double Loop Meta-Process</td>
<td>32</td>
</tr>
<tr>
<td>3.12</td>
<td>UML Profile definition in openArchitectureWare</td>
<td>36</td>
</tr>
<tr>
<td>4.1</td>
<td>Relationship between Client, Operator and IT Infrastructure</td>
<td>41</td>
</tr>
<tr>
<td>4.2</td>
<td>ISA for the Call Center</td>
<td>42</td>
</tr>
<tr>
<td>4.3</td>
<td>ISA for the Call Center (on oAW)</td>
<td>43</td>
</tr>
<tr>
<td>4.4</td>
<td>UML Object diagram for the deployed ISA</td>
<td>44</td>
</tr>
<tr>
<td>4.5</td>
<td>UML Class diagram for the ISA with added Monitoring Points and a Metric</td>
<td>46</td>
</tr>
<tr>
<td>4.6</td>
<td>Results after a run of the MPF cycle</td>
<td>47</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Comparison of MDSD tools ................................................. 17
2.2 Technology for data processing ........................................... 19
### Acronyms

<table>
<thead>
<tr>
<th>Acronym</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ADM</td>
<td>Architecture Development Method</td>
</tr>
<tr>
<td>CEP</td>
<td>Complex Event Processing</td>
</tr>
<tr>
<td>DSL</td>
<td>Domain Specific Language</td>
</tr>
<tr>
<td>KPI</td>
<td>Key Performance Indicators</td>
</tr>
<tr>
<td>MPF</td>
<td>Model Processing Framework</td>
</tr>
<tr>
<td>LEAN</td>
<td>Lightweight Enterprise Architecture Notation</td>
</tr>
<tr>
<td>OLAP</td>
<td>On Line Analytical Processing</td>
</tr>
<tr>
<td>PT-C</td>
<td>PT - Comunicações</td>
</tr>
<tr>
<td>IS</td>
<td>Information Systems</td>
</tr>
<tr>
<td>IT</td>
<td>Information Technology</td>
</tr>
<tr>
<td>WFMS</td>
<td>Workflow Management System</td>
</tr>
</tbody>
</table>
1 Introduction

"My conception of an information system, which reflects the classic work of many MIS scholars, is that it consists of not just the technology (hardware, software, data, networks) or the social setting (people, business processes, politics, economics, psychology, culture, organization, and management), but also the rich phenomena that emerge from the interactions between the two."

Allen S. Lee [Lee99]

1.1 The world we live in and the need for Organizational Self-Awareness

Laudon describes in [LL06] the importance of knowledge and knowledge management in the today's enterprises. Even though humans have the innate trait of self-awareness through consciousness, organizations — becoming more and more living complex social organisms — fail to automatically develop similar capabilities of sensing and learning. This task then falls upon the individuals, who must work together, interact and share knowledge and, therefore, engage the task of sense-making as to improve the collective self-awareness.

Nowadays, most companies suffer from IT Blindness [Luc04]: the inability to understand which events (or patterns of events) originated by IT are relevant from a business perspective. This failure has already been proven disastrous on certain occasions [Luc04].

1.1.1 Architectural Representation of Organisations

To transfer knowledge, communication must be done through known shared boundary objects, that are described using a set of concepts and with a semantics which cannot be ambiguous.

The need for Organizational Knowledge is already a widely accepted fact, and in order to improve the holistic view of the Organizational Structure, Enterprise Architectures have been proposed as valuable artifacts due to their communicative nature [MST08].

With mechanisms to represent organization blueprints, easy-to-use high level notations to describe high-level requirements and real-time processing, firms will become increasingly aware of themselves and their ecosystem, and will be able to make quicker, smarter and better informed decisions.

1.2 Monitoring Information Technology behaviour: Next steps

To increase the collective organizational knowledge, information has to be applied, which in turn can only be produced by processing relevant data in the first place [Ack89]. The ability to collect more data from more organizational sources is there unavoidably valuable. We will focus on the use of data produced by operation systems, middleware, applications and other technological elements to better understand how the technology and information layers of a deployed enterprise architecture behave in real-time.

Regarding architectural representations as boundary objects, we intend to take advantage of their creation and maintenance to encourage stakeholders in an enterprise to discuss, define and perform evaluation through the definition of quality attributes.
This thesis therefore addresses today’s inability to evaluate systems in real-time through the “eyes” of Enterprise Architectures. To accomplish this, we established four hypothesis that were subject to validation through the development of our solution:

1. It is possible to extend an existing modelling Information System Architecture framework to express what can be observed in deployed Information Systems, how to measure and which measurements influence (which) qualities;
2. The tasks of modelling an architecture and its metrics and analysing a deployed model’s behaviour can be combined into a framework;
3. Each of these framework’s steps can be connected to form an automated circular flow that, once configured, requires no human interaction;
4. Organizations can have temporal views of their deployed Information System Architecture regarding previously established metrics.

1.3 Main contributions

This thesis focuses on the problem of providing an unified, architecture-centric evaluation framework of the real-time behaviour of information systems. In order to accomplish this objective, several tasks were executed, from which the following contributions emerged:

1. An analogy between the sociological and philosophical domain that studies phenomena, or Phenomenology, and enterprise manifestations and evaluations;
2. A conceptual Model Processing Framework (MPF) that comprises a cycle of Modelling, Instantiation, Data Collection and Evaluation;
3. An implementation of the MPF using openArchitectureWare (oAW) in which:
   (a) We extended the CEO Framework to support our ontology;
   (b) The primitives of the framework were used to populate an oAW’s profile diagram;
   (c) We created a template project on oAW that implements each step of the MPF, along with an automation workflow that executes the full cycle of the MPF.

1.4 Following chapters

In chapter 2, we cover the state of the art regarding Information System research, Enterprise Architecture modelling, definition of qualities and metrics in several domains and finally existing instrumentation for dealing with modelling and business monitoring.

Next, we present the solution in chapter 3, which is divided into three main sub-chapters.

We begin by providing a conceptual setting for the more practical, engineering work. We focus on the social aspect of our objective, and attempt to gather an abstract, unified view on the dynamics of the enterprise architecture. After that, we design a conceptual framework that will empower modellers and analysts with a graphical tool for creating Enterprise Architectures and define its manifestations of real-time behaviour.
In chapter 4 we apply the implementation of our solution to an existing scenario within the context of PT-Comunicações (PT-C). By successfully modelling that evaluation scenario, we were able to partially validate the implementation and provide an example so that others can understand how to continue this work, or simply attempt to replicate the scenario.

We finish by summing up the results of our work, presenting our limitations and intentions for future work in chapter 5.
2 Related Work

After having established the motivation, we skim the academic research areas that construe the domain of this thesis by presenting key concepts, describing common problems and solutions (and the recurring tools and frameworks used in them).

This chapter begins with an overview of the concepts revolving around Enterprise Architectures and provides a description for the Center for Organisational Engineering Framework (the academic setting of this thesis), Archimate (for its rapid increased use\(^1\)) and finally the Lightweight Enterprise Architecture Notation (or LEAN) for its different approach on constructing a meta-model. We then investigate, in section 2.2, the existing means to identify qualities in Software Engineering, Computer Networking and Enterprise Engineering, as well as the metrics used for measuring said qualities. Since our work is not exclusively conceptual and also planned and developed an implementation, section 2.3 gives an overview of instrumentation trends and tools used to support the theoretical works presented throughout this chapter, specially applications that provide graphic modelling environments. The last section provides some insight on the driving force behind projects such as the one this thesis leans on. We describe the evolution of tendencies in the Workflow Management Systems, through Business Process Management and Business Activity Monitoring.

2.1 Enterprise Architectures

As stated in chapter 1, architectural representation is already as a benefit to enterprises. These enterprise architecture meta models are usually constructed in layers (fig. 2.1) that separate, for example, hardware and software from human actors and even information.

A thorough overview of Modelling Frameworks - either for Enterprise, Information System or even Software Architectures - is documented in [Vas07]. Next, we present three Enterprise Architecture Models: LEAN [Kho07], CEOF [Vas07] and Archimate [Lan05].

2.1.1 Unified Enterprise Architecture Modelling Language and LEAN

Gerald Khoury produced, in his doctoral thesis [Kho07], the Lightweight Enterprise Architecture Notation (LEAN) as a "unified, human-centered language". By having these unified and human-centered qualities, the LEAN is able to both represent any Enterprise Architecture (unified) and be easily used in real-world situations by avoiding too much complexity.

Khoury's findings on the use of metaphors have shown that EA languages and frameworks possess underlying metaphors. Problems arise when an author of a framework uses a metaphor unconsciously and therefore fail to adopt a proper one. To address this problem, Khoury studied metaphor type hierarchies, and was able to identify that a carefully chosen metaphor augments an Enterprise Architecture' descriptive power to encompass different granularities and support a greater heterogeneity of realities.

\(^1\)www.opengroup.org/archimate
2.1.1.1 The metaphor

The identification of a metaphor has been shown to be an advantageous approach to produce a valid ontology for a given domain [Kho07]. So too has the link between such metaphors and models not only been shown to exist but also has its strength been demonstrated by philosophers and language experts.

As already mentioned, Khoury set out to find a proper metaphor for Enterprise Architectures.

Metaphors are constructed by linking a certain concept (the source) to another (the target) so that some of source's essential characteristics can be identified when describing the target. To exemplify (using the example in [Kho07]), the desktop metaphor was applied to “desktop pc” by defining the source as the “desktop area of an office” and the “personal computer that appeared a few decades ago” as the target.

The author analysed model hierarchies and extracted the criteria that the source of the metaphor should be of a higher level of abstraction than the source, in such a way that the metaphor would not fail to portray every possible manifestation of its target.

His next step was, then, to identify an unifying metaphor that would encase all possible granularities of an enterprise, including its systems, its actors and its processes. From the several metaphors widely used in the present day - such as machines, learning organisations and nervous systems - to describe organisations, society was adopted as the best suited metaphor source since, among other reasons [Kho07], it complied to the previous described criteria of source-target abstraction level.
2.1.1.2 LEAN

Armed with a metaphor, the author produced an ontology that, once formalised, was encoded into a language and as a result, LEAN was created.

Khoury extracted a set of four high-level concepts from Giddens’ Theory of Structuration: Agents, Resources, Actions and Rules (fig. 2.2) and defined their relations, both homogenous and heterogenous ones. For example, if one wishes to express a specialization, the relationship “is a type of” should be used (an example of an exclusively homogenous pairing of LEAN nodes).

![Figure 2.2: Graphical representation of LEAN nodes](image)

Using these nodes and pairing relationships, the author successfully defined the Unified Enterprise Architecture Modelling Language with positive feedback from case study users.

2.1.1.3 Shortcomings

Having been developed from scratch and being positioned at a higher level of abstraction, the LEAN does not possess a strong tool support (apart from customized Visio stencils), does not integrate with other frameworks and “warrants refinement” of the LEAN set and relationships (all problems are stated in [Kho07]).

2.1.2 CEO Framework

The CEO framework was developed within the Center for Organisational Design and Engineering (CODE) group at INESC-ID with the objective of defining the set of concepts needed to represent an Information System Architecture. This framework goes further and provides the means to represent a coherent and comprehensible picture of the enterprise by supplying the modeler with a high level package of Enterprise Architecture that includes three sub packages: Business, Organisational and Information System architectures (fig. 2.3).

The Business Architecture represents strategic points of view and business processes. The Organisational Architecture provides the concept of resource as its central element. Resources are essential to organisational function and can be materialised in internal policies, roles, competencies and human resources.

André Vasconcelos [Vas07] greatly improved the 2001 version of this framework by detailing the Information System Architecture, formalising the primitives in UML and OCL validation expressions. The ISA’s purpose is to represent the structure of information system components that support business goals and is further subdivided into Application Architecture, Information Architecture and Technology Architecture:
• **Application Architecture** — defines the essential applications necessary to manage data and support the business layer;

• **Information Architecture** — identifies the critical information needed by the business layer and provides the data structures (independent of implementation);

• **Technology Architecture** — support for the application layer is described in this architecture, focusing on the technology used such as servers, networking, communication infrastructures.

2.1.2.1 Architecture Comparison

Another of Vasconcelos’ contributions was the addition of a series of measurements for architectural qualities. One of the benefits of constructing TO-BE Information System Architectures would be the ability to clearly define if the new design is better than the old, and quantify the gain. Therefore, to compare and evaluate structural qualities between the old “AS-IS” and “TO-BE” Enterprise Architectures (or even different “TO-BE” ones), the author created a set of metrics at the meta level, so that they could be applied to all models.

2.1.2.2 Shortcomings

The CEO framework possesses an implicit problem by creating a rigid structure of concepts, and that is the inability to guarantee that its 37 primitives are sufficient to describe the multitude of existing systems. Concepts such as clusters, network links, and even TCP/IP and UDP ports cannot be expressed without extending the CEO framework which in turn can change the accuracy of metric results.
2.1.3 ArchiMate

The ArchiMate enterprise modelling language was developed by a group of public and private Dutch entities with the common desire to improve communication of enterprise architectures, avoiding informal diagrams and ambiguous vocabulary. The language describes a taxonomy for mapping architectural components, along with different viewpoints aimed at different stakeholders.

One of Archimate’s goals is to ease the integration between the business, application and technology layers (fig. 2.4):

- **Business layer** — relates to the products and services the organisation provides to its (outside) environment. They are supported by business processes that are executed by actors that play certain roles;

- **Application layer** — this layer supports the aforementioned processes through application services that are, themselves, realized by the technology layer;

- **Technology layer** — software, hardware infrastructure and networking services and other infrastructural services support the realization of the upper layers.
2.1.3.1 Evaluation

Another of ArchiMate’s goals is to aid Change management. Change has been a concern of the developers of Archimate and so Lankhorst’s “Enterprise Architecture at Work” [Lan05] presents a “number of techniques that help architects and stakeholders to compare alternative designs [...] and to be able to study the impact of a change to the design”. The evaluation’s purpose is then to study and compare the performance, quality and cost of architectures prior to their implementation.

For performance measurements, five views (described in fig. 2.5) have been identified:

![Figure 2.5: Archimate Performance Views](image)

2.1.4 Summary of Enterprise Architectures

We have described three architectural frameworks with different backgrounds.

The Unified Enterprise Architecture Modelling Language (and LEAN), is a highly conceptual metamodel for enterprise modelling based on a societal metaphor. It can be understood as a meta-metamodel for Enterprise description, from where concepts of CEOF and ArchiMate (for example) can be specialized.

The Organisation Engineering Center (CEO) Framework stems from the academic domain of Organisational Engineering and defines a clear hierarchy of concepts for Enterprise Architectures, views for different desires, and provides structural metrics for TO-BE architecture comparison.

The ArchiMate language stems from a mix of academic and organisational environments, and defines a taxonomy, along with views that target different stakeholders. Regarding evaluation, ArchiMate also focuses on evaluation of TO-BE architectures, but instead provides analysis of their dynamic behaviour.

What all have yet to provide is the ability to, once the architecture is deployed, perform evaluation of behaviour of this architecture, using real-time data and events. To be able to execute this action we can provide benefits to the architecture life cycle:

- By executing real-time evaluation, we assert the deviation from the before(model) and the imple-
mentation, for events specified using model elements and so we perform a “reality-check”;

- Perform observations on how the architecture actually behaves after implementation, so that the evaluations such as ArchiMate’s can be corroborated (along the temporal axis).
- Understand changes in behaviour, either from infrastructure degradation or unknown change in practices, habits or processes.

### 2.2 Evaluating Information Technology: metrics and measuring qualities

In [Fil99], Robert E. Filman describes the steps towards achieving “ilities” - typical attributes of systems such as reliability, performance, availability and maintainability - in compositional architectures. These concepts exist in a myriad of domains ranging from Networking to Software Development - and recently, Enterprise Architectures. Even within a given domain, these definitions are not clear. For example, in multimedia domains, the notion of quality of service is subjective since there isn’t a widely accepted QoS Framework [ACH96] and so QoS must be always understood in the context of the domain or framework used to measure it.

Filman then states that there is more than one definition for concepts such as reliability; to define reliability is to specify the requirements established in a certain environment and by certain people to attain reliability. The consequence of this looseness is therefore an impossibility to uniquely and globally define ilities. Nevertheless, for one to be able to measure the behaviour of an information system according to a set of qualities, the definition of the set of functional, aesthetic, systematic and combinatoric requirements must be accomplished by stakeholders.

#### 2.2.1 Quality Meta-Model

A quality meta-model (fig. 2.6) is described in [Alb03] to help identify common characteristics in quality models. This meta-model identifies “quality attribute” as something that is measured through metrics and, as a source for variables are consumed. These variables relate to external characteristics of the target entity and can be either observable during execution or not. The latter refers to variables that belong to static aspects of the monitored entity.

#### 2.2.2 Quality of Service and Quality of Protection

In the domain of networking, intra and internet routing, the notion of Quality of Service represents an end-to-end property described as “a set of service requirements to be met by the network while transporting a flow” [CNRS98]. The multimedia systems domain gave birth to several architectures for assessing Quality of Service (QoS) through low-level mechanisms such as flow shaping and static policing of hardware and operating system resources. The constraints, expectancies and tolerances regarding this set are usually described in Service Level Agreement (SLA).

The QoS/SLA terminology has been transported into other domains (e.g. at the Web Service level for supporting e-Business transactions in Service Oriented Architectures [DLP03]). “A survey on QoS Architectures” [ACH96] presents a series of such frameworks that follow the QoS concepts introduced by
Andrew Campbell [CCH94].

Quality of Protection (QoP) surfaced as QoS counterpart regarding security and risk management. Its measurements and metrics are discussed in the annual QoP Workshop. The emergence of quickly developed interconnected systems using unstable new technology makes risk [Hol04] assessment a critical asset.

2.2.3 Key Performance Indicators

Key Performance Indicators represent a “set of measures focusing on those aspects of organisational performance that are most critical for the current and future success of the organisation” [Par07] and therefore measure the critical success factors of the organization [Reh]. KPI are often tuned to each organisation and reflect tacit knowledge and informal metrics for difficult to measure qualities. The KPI Library\(^2\) contains a list of indicators for many domains, from Business to Vertical Industries.

2.2.4 Software Metrics

Software Engineering scholars have always been concerned with the quality of software and so, in order to measure code quality, several metrics were created and are now widely used:

- Lines of Code and Cyclomatic Complexity measurements were created to access software complexity and maintainability;
- Weighted Methods per Class, Number of Children, Coupling Between Object Classes (and others) were found to be useful when measuring Object Oriented code quality;

\(^2\)KPI Library - http://kpilibray.com
ISO 15408 and IEEE 1061 define vocabulary, criteria and frameworks for evaluating quality and security in software engineering and IT.

2.2.5 Conclusion

We were able to observe the multiplicity of meanings attributed to qualities and measurements. Regarding Enterprise Architectures, as we presented in the previous section, CEOF provides structural metrics, while ArchiMate focuses on predicting behaviour. Archimate's evaluation, however, isn't built on a well defined structured of metrics, which is a problem that we will discuss in the next chapter.

2.3 Instrumentation

Throughout the years, assemblers, compilers, linkers and interpreters were created to help developers handle higher levels of abstraction and, therefore, help to create more powerful programs. In the same way, with UML's broadness and flexibility, Model Driven Software Development (MDSD) applications emerged, ranging Open-Source projects, to low-budget solutions, to full-fledged Enterprise suites and even a development platform who created its own metamodel to escape UML's flaws (The Eclipse Modeling Project).

2.3.1 Unified Modelling Language

The Unified Modelling Language is a general-purpose, layered (fig. 2.7) structure of concepts that facilitate activity of analysis, design and implementation of systems that are, for example, software-based or business-based [Gro07]. UML is defined and maintained by the Object Management Group3.

Starting from the first major revision of UML, extension mechanisms have been improved by means of using profiles and stereotypes. The OMG hosts several UML profile specifications that extend UML in order to model concepts such as Testing, Systems Engineering (SysML), CORBA and CORBA CCM4 and Enterprise Application Integration.

2.3.1.1 Time Specification

The "UML Profile for Schedulability, Performance, and Time Specification" provides UML the notation needed to model concepts used in quantitative analysis of software [Fom02]. The Model Processing Framework (MPF), described in [Fom02], contemplates five interconnected processing blocks between whom flow models, configuration data and results of analysis. The blocks, named "Model Editor", "Model Configurer", "Model Converter", "Model Analyzer" and "Results Converter", execute the model processing tasks so that conclusions from one iteration of the processing stream can be fed into a new iteration, and achieve better results.

3OMG - www.omg.org
4CORBA Component Model
2.3.1.2 UML usage in Enterprise Architecture frameworks

In order to reify abstract concepts, Architecture Model developers have been taking advantage of the Unified Modelling Language for its flexible, heavily scrutinized and broad-scoped structure, although complexity and cohesion are topics of criticism [Kob04] regarding the UML structure:

- CEO Framework [SCV+06], TOGAF [Pub07] and DoDAF [Gro05] support UML notation.
- ArchiMate framework has a different notation, but studies have been made to estimate the portability of ArchiMate’s concepts and semantics to UML [WBB+04].
- Due to their high-level abstract purpose and concepts, frameworks such as the Zachman Framework, EUP⁵ and EAP⁶ are UML-independent.

2.3.2 Eclipse Project

The Eclipse Project evolves around a software development platform that started from a Java Integrated Development Environment (IDE). This project includes several extensions such as Java Development Tools⁷ (the original codebase), C/C++ Development Tools⁸, Graphical Editing Framework⁹ and the Eclipse Modeling Framework¹⁰.

---

⁵Enterprise Unified Process
⁶Enterprise Architecture Planning
⁷http://www.eclipse.org/jdt/
⁸http://www.eclipse.org/cdt/
⁹http://www.eclipse.org/gef/
¹⁰http://www.eclipse.org/emf/
The latter, EMF, started as a MOF implementation of a meta-modelling structure [BSM+03], now called EMF Core or “Ecore”, but later became a full modeling framework and code generating tool that relies heavily on OMG’s XMI specification for model and metamodel serialization. Taking advantage of EMF, the Modeling Development Tools project\(^\text{11}\) was then created to provide several metamodel implementations for UML2, OCL, BPMN and XSD.

2.3.3 openArchitectureWare

openArchitectureWare is a model driven software development (MDSD) tool that draws from several components of the Eclipse Project and implements a workflow engine at its core (fig. 2.8). The workflow components (oAW has several built-in) can read, instantiate and transform models, perform code generation and execute validation checks (using OCL). oAW relies on the EMF core meta-model but also supports UML2, XML and JavaBeans based models.

![openArchitectureWare structure](image)

Figure 2.8: openArchitectureWare structure

A workflow is created as a linearly invoked set of components that perform some kind of activity, and use slots (that can be named and assigned to), so that the results of one component can be transferred to another. Some typical components are the Reader and Writer, Xpand, Xtend and Xtext.

The Reader components serve as entry points for the workflow: they receive a UML, Ecore or other supported model and create a named slot that will be available on the global scope of the workflow, while writers have the function of writing a model slot back into a file.

The Xpand component is responsible for code generation in the Eclipse M2T (model-to-text) project and works a template engine. The engine accepts macro definitions and the places (model elements) where these macros should be expanded. These macros can have simple control flow structures such as loop and if statements. Once the engine is parametrized with a metamodel and loaded with a model, it will navigate the model and apply the macro expansions accordingly, generating code. On a final note,

\(^{11}\)http://www.eclipse.org/mdt/
oAW also provides a “beautifier” for some languages so that the code is properly indented after generation.

The Xtend component’s purpose is to execute queries on a model and invoke statically-typed functions to perform Model-to-Model (M2M) transformations. Aside from the language provided by oAW, the Xtend component can be further extended with user-defined Java functions, giving it additional expressiveness.

2.3.4 Enterprise Architect

Enterprise Architect is a commercial CASE tool developed by Sparx Systems that aims to be a complete tool for Enterprise Architecture design. As of version 7.5 Professional (available to students at Instituto Superior Técnico), Enterprise Architect implements:

- UML2.1 (along with all the 13 diagrams and the ability to define custom ones)
- UML profiles
- XMI2.1 for importing and exporting models
- A model validation mechanism that can be invoked to ensure conformity to the metamodel standard;

Being a CASE tool, EA supports Model Driven Development [MM03] practices of constructing high-level models, performing transformations for specific platforms, and finally generating artifacts such as code. Both reverse-engineering (creating models from code) and code generation are available for many programming languages, such as Java, C, C++, C# and Python.

2.3.5 Comparison and Conclusion

Considering that our solution manipulates models and we wish to provide users with an executable representation of the enterprise architecture, we identified several necessities for these tools. Support for metamodels and metamodel extensions is essential since Enterprise Architecture Frameworks are metamodel level entities. To augment portability and mobility of this solution, import and export of models is desirable. One way of performing validations is through the verification of constrains on the model; for UML, the OCL performs this function and therefore it is required for the tool not only syntax validation of the constraint language but also execution. The last but equally important feature is the ability to chain model transformation actions, since we’re going to perform model edition, validation and execution of custom tasks.

From this comparison of the two tools we presented above, we can conclude that openArchitectureWare provides the necessary feature for task at hand. Also, by being Open Source, we have the possibility of extending the tool to better suit our needs.

12 Computer-Aided Software Engineering
13 www.sparxsystems.com.au
### 2.4 Business Process Management

A new-found way of thinking in Organizations, that stemmed from three trends in Information Systems (described in [HW03]), propelled the extension of classic Workflow Management:

1. From programming to assembling (of Information Systems);
2. From data-oriented to process-oriented applications;
3. From design to redesign and organic growth.

These trends translate the shift from the generic, static and all-purpose solutions to custom-made applications, capable of evolving along with the enterprise. Traditional Workflow Management Systems - even though process-aware and capable of capturing real-time data from its execution - focused on the lower section of the BPM lifecycle (shown in fig. 2.9).

![Business Process Management cycle](image)

In the classic view of Workflow Management, processes were (re)designed, followed by the implementation (usually done by configuring a generic WFMS) followed by the actual IT-supported execution. Once organizational change occurred, it was back to the drawing board, leaving no room for diagnosis and adaptation. Business Process Management brought a bigger emphasis on monitoring making use of collected data from application logs and database activity.

As enterprises grow aware of their processes, their (re)design and diagnosis have become the main concerns of the BPM life-cycle. Hence, BPA\(^{14}\) (one key aspect of BPM) has evolved to encompass activities such as simulation, verification and validation, giving birth to Business Activity Monitoring and BAM tools.

---

\(^{14}\)Business Process Analysis
2.4.1 Business Activity Monitoring

Business Activity Monitoring was (first) introduced in 2002 by Gartner Inc. [McC02] and, stimulated
by the BPM growth, made several organizations adopt or design solutions to address IT blindness. The
cradle of such tools was the enterprise world (outside academia) and, sprouting from common sense
[Ada02], aimed at providing real-time access to critical business performance indicators that improve
the speed and effectiveness of business operations [McC02].

![Diagram](image)

**Figure 2.10:** Positioning of BAM systems in relation to the stakeholder and delivery latency

Modern enterprise systems (fig. 2.10) are capable of generating large volumes of logs and sources
of measurement, ranging from CPU usage to application transactions. To collect all this data, a BAM
system must monitor orthogonally all the enterprise, and work on more than one time frame (integration
and multiple timescales principles [CCH94]). Regarding the construction of a Business Activity Monitor
system, Joseph DeFee and Paul Harmon provide an overview on the tasks it must accomplish [DH04]:

1. convert data about actual events into digital information;
2. provide context for the digital data being accumulated;
3. apply logic to data to identify problems, diagnose them, and to recommend managerial actions.

A 4th additional step is displaying the near-real-time information produced in the previous step on an
easy to consult location like an online dashboard.

There are several approaches to step 3 (seen in the table below), being the most straight-forward the
design of a rules-based system: data is matched against a set of rules that, when triggered, executes the
appropriate actions.
From the domain of Business Intelligence, reliance on Data Warehousing and applying a collection of powerful tools such as On Line Analytical Processing (OLAP) gives the ability to extract meaningful patterns from historical data. Knowledge from the AI domain and Rule-based techniques are also used to Business Intelligence’s benefit. Simulation-Based Systems use patterns extracted from historical flows (and sometimes from current data) to perform trend analysis. The results can be used to both eliminate potential threats and simulate formulated hypothesis. Finally, all these approaches to Decision Making can be combined into a mixed system, adapting to the needs of the organization.

<table>
<thead>
<tr>
<th>Technology</th>
<th>Advantages</th>
<th>Disadvantages</th>
</tr>
</thead>
</table>
| Rule Based Systems    | • Can be provided for every specific task and can operate independent of other systems  
• Can be easily define and tested  
• Is a well understood approach | • Can become complex if the range of the variables are extensive  
• Can become complex to test and maintain over time as the business changes  
• Difficult to graphically validate relationships to process |
| BI Systems            | • Have powerful algorithms for analyzing trends and patterns  
• Can pull together data from EAI and ERP systems and from “best-of-breath” applications | • Work best when used in conjunction with large amounts of data  
• Must create Data Warehouse as precondition to using BAM capabilities  
• Aren’t designed to use a process context for reporting  
• Have not been designed to operate in real-time. |
| Simulation Systems    | • Provide capability to model highly complex and dynamic processes  
• Provides better insight into the future predicted state if the business, based on the validated process flows  
• Notices gradual changes not so readily identified by other techniques  
• Can take advantage if both BI and Rule-Based approaches to provide even better simulation results at run-time, | • Simulations technology is not so well understood at the user level  
• Development of complex simulation requires specialized knowledge  
• Requires an initial analysis and specification of a business process model |

Table 2.2: Technology for data processing
2.5 On providing quality views over Information System Architectures

During this chapter we provided an up-to-date overview on Enterprise Architectures, Instrumentation and also the driving forces that created the need for this thesis.

We were able to establish that, of the Enterprise Architecture Metamodels, none performs real-time evaluation of deployed architectures. As said before, this feature has potential benefits such as helping to reduce the gap between reality and the modelled architecture, provide constant feedback on its behaviour and finally reduce the time to assert the necessity of change.

So too exists a lack of proper instrumentation for performing real-time evaluation but there are, however, tools that can (and will) be used to accomplish this task.

In the next chapter, we will take Gerald Khoury’s work on the Lightweight Enterprise Architecture Notation and André Vasconcelos’ improved CEOF Architecture Metamodel and construct a conceptual framework for processing models to perform real-time evaluation. Next, we will take openArchitecture-Ware and provide an implementation of this framework so that we can attest its feasibility.
3 Solution

In the previous chapter we concluded that evaluation of information systems and information technology has several problems such as the lack of proper ontologic support, the inability to define what can be observed in real-time and finally the inexistence of a generic, unified, meta-model independent framework of evaluating information system behaviour.

The solution we developed aims to tackle these problems and therefore provide means to describe architecture run-time manifestations and to perform evaluation modelling based on them.

The first problem we had to address was how to find the necessary and sufficient abstraction for this activity so that we would be able to contemplate every possible real-world scenario. For this, we turned back into Khoury’s societal metaphor [Kho07] with the intent of constructing (or extending) an unified ontology. As we describe next in section 3.1, we approached sociological domains that study experience, knowledge and phenomena to help us capture essential aspects of enterprise behaviour. By understanding action and perception through philosophy and (particularly) phenomenology, we sought ways to better understand what it means to evaluate information systems. The purpose of this thesis was not, however, to delve deeply into difficult and complex philosophies and so we limited ourselves to the very basic concepts and understandings that we deemed necessary and sufficient to accomplish our task.

An unified description of evaluation from an abstract point of view has well-defined purposes. For one, it enables the solution designer to translate his ideas into a tangible visual representation. Second, and most important, it creates a level of indirection between these ideas and their use on an existing Domain Specific Language or Enterprise Architecture metamodel. Section 3.2 describes how this conceptualization can be used to drive the design of the evaluation aspect into architectural metamodels, while lessening the constraints needed to apply our objective to other environments, aside from the one we describe in section 3.5.

In section 3.3 we modify the ISA building pipeline described in [Vas07] to encompass the evaluation activity. To accomplish this, we added an evaluation cycle at its end that takes as input the ‘AS-IS’ ISA and the Information Systems (of the existing pipeline), add observable variables and measurements and, through a processing block, determine behaviour issues.

To model the necessary steps that will be occurring during evaluation, we took advantage of the work done on the UML Profile of Time Specification [Fom02] concerning the Model Processing Framework, which we adapted for our purpose. This is described in section 3.4, along with a proof-of-concept implementation using a UML tool.

3.1 Locating a Proper Metaphor

"[…] a good part of the answer to the question ‘why philosophy?’ is that the alternative to philosophy is not no philosophy but bad philosophy. The ‘amphilosophical’ person has an unconscious philosophy, which they apply in their practice - whether of science or politics or daily life."

Andrew Collier [Col94]
3.1.1 Establishing the Phenomenological setting

In [RN08], Reck er and Nieha ves explained the relevance of philosophy in Information System research by linking several philosophical disciplines such as Ontology, Methodology and Epistemology to IS research paradigms (e.g. positivism, interpretivism). By understanding how we perceive others, what we observe and how we judge, we were able to come up with unified, global, overview of the concepts that are involved when performing evaluation of Information Systems and, therefore, be able to model this domain.

Phenomenology is the “study of structures of consciousness as experienced from the first-person point of view.” [Uni08]. Its key concepts are: intentionality (experience always is directed at something or is about something), qualia (sensory data), bracketing (putting aside the question of the existence of a real world) and consciousness itself. We found this philosophical discipline and these concepts to have a profound link to the Information Systems and Information System research area [RN08]. In the following subsections of this introduction we describe the mentioned concepts, and then move on to determine in what way the phenomenological dialectic fits the problem at hand.

3.1.1.1 Phenomenon, Noumenon, Thing-in-itself

Regarding Phenomenology as the study of phenomena as it appears to us in consciousness, it is important to distinguish the concepts of phenomenon and what Immanuel Kant called noumenon or “thing in-itself” [KM34]. Contrary to phenomena, noumena are objects that exist independent of the senses, while the former refers to appearances and objects produced by senses. This accentuates the subjective aspect of this philosophy, as to say that a thing of the world can be experienced in different ways, from the phenomenological point of view. As we describe next, the bracketing phenomenon is related to this duality. As stated before, Information System research paradigms are tightly linked to these philosophical disciplines. For example, the positivist point of view establishes that the only source of knowledge is that which stems from phenomena, rejecting noumena altogether.

3.1.1.2 Facticity

Facticity is a concept that grew in significance during the 20th century within the phenomenology domain, through philosophers such as Edmund Husserl, Martin Heidegger [Hei96] and Jean-Paul Sartre [Sar55]. Its meaning suffered variations throughout history, but as said in the beginning, we did not concern ourselves with such details and simply focused on how a concept like facticity helps our metaphor, and will therefore use the Sartrian notion of facticity. As intended by Sartre, facticity “includes all those properties that third-person investigation can establish about me” [Uni08]. Such properties can be of historical and psychological nature, include physical traits and even things known to be true in the future (e.g. the inevitability of death).

3.1.1.3 The bracketsing, noema and qualia

Edmund Husserl strongly advocated that the objects of consciousness do not exist only inside it, but transcend it, therefore being able to exist independently of the mental activities executed upon them,
such as thinking and judgment [So100]. This decoupling does not imply that perceptual experiences are then void of content, but instead generate what he called \textit{noema}, the perceptual content.

Further still, Husserl established the necessity of describing an object from the first person standpoint, in such a way that the nature (e.g. reality, dreams, hallucinations) of the object itself is irrelevant, leaving only the item as it was experienced by the subject. To the act of “suspending judgment of the world”, Husserl called \textit{bracketing} [So100].

Both concepts of \textit{noema} and \textit{bracketing} come together in phenomenology since former is that which results from latter, when experiencing an object while putting aside its nature (e.g. an awakened experience, a dream, an hallucination).

Qualia stands for the sensory data captured by an entity.

\subsection*{3.1.1.4 Intentionality}

\textit{Intentionality} (not to be confused with \textit{intensionality}) refers to the directedness of consciousness or mental states [Uni08]. In other words, it reflects the notion of phenomena always being about objects, that exist within consciousness (once again, independent of its nature, through \textit{bracketing}).

This concept was used in many schools of thought such as Existentialism, where both Martin Heidegger and Jean-Paul Sartre used \textit{intentionality} to proclaim that things such as moods were always intentional (about something) and, therefore, rejected the notion of being sad, happy, angry without a motive. This was also used by Sartre to his central theme of responsibility, freedom and “bad faith” [Sar57].

\subsection*{3.1.2 Connecting Phenomenology with modelling and evaluating}

As said in the beginning of this chapter, society has been established as a proper metaphor for enterprises. In order to accomplish our goal, we set out to find a parallel between experiencing, judging, perceiving-capable agents since, inherently, these activities occur inside society and consequently within the metaphor.

Since \textit{Phenomenology} imposes \textit{subjectivity} upon the world, we can place its rationale within the \textit{interpretivist} trend [LG03, Min01]. Considering that the phenomenologic concepts and their relations apply to society, we then propose that they can also be mapped into the society metaphor and therefore its target (enterprise systems).

The act of perceiving (\textit{noesis}) the enterprise structure allows the detection of its meaning, therefore producing the boundary object (\textit{noema}) referred in the Organisational Engineering domain as Enterprise Architecture. And by determining what manifestations are observable on this architecture, we perform the \textit{phenomenological bracketing}, for the assumption develops that objects will always be perceived as modelled, regardless of the particularities relative to their nature.

This construction of an enterprise architecture allows the specification of enterprise elements as simply ‘being there’ - or \textit{noumena} - while by describing their manifestations we define how they appear before the senses - \textit{phenomena}.

Modelling evaluation as an activity that is inherently dependent on observations - \textit{qualia} - we establish a link to the \textit{intentionality} aspect of experience, according to phenomenology. And, finally, by having
agents record manifestations as observed (from the first person standpoint) we construct the observed objects’ facticity.

3.1.2.1 Subjectivity of Qualities

One of the problems raised in chapter 2 was the fact that different domains establish different meanings the concept quality they wish to monitor. This phenomenological take on evaluation provides evidence on why it hasn’t been possible to pin down notions such as Performance and Availability and also makes it difficult to determine the metrics used to measure the behavioural attributes of a given system, as presented in [LCPH00]. The compromise needed to transfer these concepts into the meta plane has the unavoidable effect of generalizing to a point where the concepts lose their meaning. So too, in phenomenology, the dialectic does not encompass attributes of perception and experience, but exclusively establishes the abstract concepts of the acts and objects involved (i.e. the essential structures of experience). As a consequence, we did not attempt to globally define quality concepts, nor methods of calculation, for evaluating behavioural characteristics of existing systems. Instead we will consider that evaluation criteria are phenomena that emerge from consciousness, either through one’s preconceptions or transfer of knowledge from a third party.

Another observation, that we were able to extract, was the importance of accepting the inability to evaluate something in its essence and, instead, placing our judgement upon what is observed of that thing. This thinking should be applied to both human-machine and machine-machine levels. For a computer application A to use another application B, the latter will display a set of manifestations observable by the former (creating B’s facticity) which then can be used by A to evaluate the B’s actions.

Finally we established intentionality as a criteria for traceability: measurement methods should never rely on implicit or tacit knowledge but instead have clear representations of the sensory data used to calculate them.

3.1.3 Conclusion

In this section we provided insight on how the atoms of society experience themselves and others as a form of metaphor for evaluating Information Systems. In societies, human beings are constantly perceiving, evaluating their environment, their peers, and surrounding objects. These agents possess senses that allow them to perceive things in their environment through action, i.e. by interacting with that environment.

By considering this “phenomenological ontology” in a level of abstraction greater than the target of our desired metaphor, we were able to assert its validity and also construct a parallel between evaluation of Information Systems and the philosophical discipline of phenomenology. Many of the central concepts of this domain can be understood under the scope of agent interaction, should they be systems, humans, communities and organisations. We asserted subjectivity as an essential aspect of this problem and, as a direct consequence, qualities are subjective and should always be understood from the perspective of the one measuring it.
### 3.2 Producing a High-Level Language Extension

Given an enterprise architectural model, accomplishing the task of modelling its information systems' evaluation and execution requires the following actions [Kho07]:

![Diagram of steps in constructing and applying a High-Level Language](image)

**Figure 3.1: Steps in constructing and applying a High-Level Language**

Identifying a metaphor has been done in the previous section, where we establish the relationship between evaluating Information Systems and society's model of behaviour and experience.

According to Khoury's works [Kho07], the sufficient lexicon to model an enterprise was drawn from Giddens' Theory of Structuration and consists of four primitive concepts: Agent, Resource, Action and Rule, along with their homo and heterogeneous relationships. Using the Lightweight Enterprise Architecture Notation, we described a high-level model for this problem. We began by performing the following changes on the LEAN node pairings (fig. 3.2):

- added an "observes" relationship between an **Agent** and a **Resource**;
- add the **Agent → Resource** and **Agent → Rule** to the "produces" pairing.

These two changes reflect the necessity on bringing Agents and Resources closer. This necessity arose from our closer inspection of the philosophical notions of experience. The causality relation between Action and Rule is in fact important, but it is also essential to identify the Agent as the producer of Resources, even if Actions were the means. This close binding also supports the approach of modelling what can be observed on an entity and how to measure that entity's actions.

Having extended a high level notation to encompass the necessary relationships, we modelled evaluation according to the society metaphor (fig. 3.3). Once again, it is a behaviour that exists within it, and therefore can be expressed using these nodes and pairings. We created three specialisations: **Manifestation** (from Resource), **Expectancy** (from Rule) and **Evaluation** (from Action). We used the primary
Figure 3.2: LEAN node pairings with changes to model evaluation

(1) The "produces" pairing exists only for the Agent → Resource relationship.

<table>
<thead>
<tr>
<th>RELATIONSHIP SET</th>
<th>HOMOGENEOUS PAIRINGS</th>
<th>HETEROGENEOUS PAIRINGS</th>
</tr>
</thead>
<tbody>
<tr>
<td>is a type of</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>supports</td>
<td>x</td>
<td>x</td>
</tr>
<tr>
<td>interfaces with</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>is a part of</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>precedes</td>
<td>✓</td>
<td>x</td>
</tr>
<tr>
<td>reports to</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>performed by</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>uses</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>produces</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>complies with</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>apply to</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>supports goal</td>
<td>x</td>
<td>✓</td>
</tr>
<tr>
<td>observes</td>
<td>x</td>
<td>✓</td>
</tr>
</tbody>
</table>
Agent node since we didn’t have a necessity to constrain the subtypes of Agents that fit this situation. *Manifestations* represent the phenomena that are produced by Agents and are, as such, observable and knowable. They are the essential elements that are consumed when performing evaluation. *Expectancies* reflect the desired expectations, based on the observed phenomena, regarding an Agent’s behaviour. To this act, we called *Evaluation*.

![Evaluation ontology using LEAN](image)

**Figure 3.3:** Evaluation ontology using LEAN

With a proper metaphor for the act of evaluation, we proceeded to construct our Model Processing Framework. The next (brief) section defines the setting on which the MPF operates.

### 3.3 Revisiting the Information System Architecture building pipeline

In [Vas07], the traditional approach to building Information System Architectures was extended to encompass iterations of redefinition and readjustment of the **TO-BE ISA** (fig. 3.4). Along with the creation of architectural metrics, this new pipeline created the ability to evaluate Enterprise Architectures regarding pre-established qualities as means of comparing different **(TO-BE) ISA**.

The solution we created further extends this pipeline so that systems can be monitored from the perspective of the final **(AS-IS) ISA**. The revisited pipeline (fig. 3.5) includes an extra cycle located at its end that feeds the ISA artifact, the manifestations and measurements into a processing framework. This framework is able to evaluate the state of the deployed ISA regarding defined metrics, as well as the ability to initiate this evaluation process at will.

In the next section we describe a framework that aims to support this extension of the pipeline.
3.4 Defining a Model Processing Framework

One of the objectives of this thesis was to create the means to automate the evaluation of Information System and Information Technology behaviour. By automation, in this context, we mean the ability to perform all the actions necessary and sufficient to execute our extension of the building pipeline without human intervention. That sequence of actions excludes, however, the initial bootstrapping process of describing what is the reality, its elements, their actions and how they should be measured.

In [Fom02], the concept of Model Processing (fig. 3.6) is described as a set of tasks, namely creating a model and performing analysis techniques, that are put together in the form of a Model Processing Framework. In the following subsections we describe each of the MPF’s components we used as a base for our solution. We explain their tasks and input/outputs and how they’re connected but, in order to
separate the conceptualization of our framework and the tool support (described in section 3.5), we have kept out all the implementation aspects.

![Figure 3.6: Model Processing Framework](image)

### 3.4.1 Model Editor

The Model Editor supports the creation and edition of models, with proper lexicon and syntactic validation (according to an Enterprise Architecture meta-model). These meta-models must encompass the four concepts of the Lightweight Enterprise Architecture Notation: agent, action, resource and rule. The creation and edition of models must be fully supported by the tool, as to not force the user to understand lower-level programming concepts.

As we explained in section 3.1.2, evaluation occurs by observing agents while they’re performing actions. Therefore, object representation is necessary so that individual characteristics can be modeled. Objects as “things” must be reified within the tool, so that Object diagrams or similar representation of object-level schemas are possible.

### 3.4.2 Model Converter

![Figure 3.7: Model Converter](image)

The Model Converter’s purpose is to act like a pre-processor, transforming the modelled architecture into a format that can be consumed by the Model Analyser. The converter, loaded with the Enterprise Architecture meta-model, has the capacity to navigate a model and generate the code needed to capture the values of monitoring points defined by the modeller.
3.4.3 Model Analyser

The core of this MPF is the Model Analyser. It is responsible for performing the evaluation of the Information Systems based on the enterprise architecture model (along with manifestations and measurement specifications) that were defined in the Model Editor. To execute this function, the Analyser block must accomplish the following steps:

1. model – two activities are performed in this step: validation and evaluation of the architectural model:
   - validation - the model is first verified for inconsistencies. Semantic constraints such as cardinality are verified according to predefined rules of the meta-model;
   - evaluation - structural qualities are calculated in this step as well. For examples of this kind of metrics, refer to [Vas07];

2. instantiate the model – after validation, and in order to accomplish object-level evaluation, the object diagram must be transformed into an executable set of instructions, whose purpose is to support the next step. One example of this is the generation of object instances whose types, names and attributes map into tables in a Database;

3. populate diagrams with manifestation data – ranging from CPU usage, web service invocation timestamps, queue sizes. This step collects all manifestations from a data source and sets the values on each object of the internal representation of the model;

4. calculate object-level metrics – again, a set of constraints - that target the object level - concludes how the current model instantiation is faring regarding the metrics defined during the model edition phase (section 3.4.1);

5. generate warnings regarding unmatched acceptance thresholds – finally, to help communicate the results of this iteration, it is desirable to generate notifications of which metrics failed to reach desirable levels of acceptance.

To sum up, this block is responsible for the evaluation of metrics, both at the Model and the Instance level (fig. 3.9). The Model-Level Analyser acts upon metrics that are defined within the meta-model and don’t require user intervention; on the other hand, the Object Level Analyser uses continuously stored data and user-defined metrics to judge instance behaviour.

In both situations it is desirable to create not only the metrics - the method of calculating a value that means something for a given domain - but, instead, define the expectancy along with metric. Since the purpose of this evaluation is not comparing results with some other architecture in real-time, the outcome of calculating (for example) a performance index is only useful if there are pre-assumptions on how low or how high said index should be. Modeling the expectancy can be as simple as changing an
index formula into an inequality (so that acceptable/unacceptable correlates to true or false) or, in more complex domains, include degrees of membership and other fuzzy logic theory [Uni08, Zad65].

3.4.4 Results Converter

The final stage in the MPF cycle is the transportation of information produced by the Model Analyser back into the visual representation of the model as in the Model Editor. The values (of data collected) are displayed in each of the objects' manifestation attributes and the generated warnings by the Model Analyser are displayed to the user.

3.5 Implementing the Model Processing Framework

An abstract framework for evaluating enterprise models was constructed and, to attest its viability, we describe in this section a possible implementation for the MPF, using an existing architecture meta-model and an UML tool.

To sum up our objective, the aim was to construct a model driven evaluation framework for the real-time manifestations of deployed enterprise architectures.

As already stated, this thesis was developed in both an academic and an organisational environment. As such, this implementation draws from Center of Organisational Engineering meta-model, and from the
PT-Comunicações monitoring and data collecting infrastructure. These two domains create constraints on the abstraction we have already presented, but also helps us implement it.

To better position this implementation, we considered the ODE loop detailed in [Mat07]. According to this loop, the MPF targets the Domain Monitoring and the Domain Analysis phases of the ODE Meta-Process (shown in fig 3.11).

![Figure 3.11: ODE’s Double Loop Meta-Process](image)

In the following subsections we describe the set of assumptions on which this implementation relies, the extension to the CEOF meta-model and the MPF implementation using openArchitectureWare.

### 3.5.1 Pre Assumptions

The gap between the conceptual Modelling Processing Framework and the reality on which it operates manifests itself in a requirement we have taken for granted so far: the infrastructure (attached to the deployed ISA) that collects the values of monitoring points along a temporal axis. This assumption is due to the fact that such task is outside of the scope of this thesis and was already addressed issue within the Organisational environment in which this thesis was conceived (further detail on the organisational structure is presented on the section 4.1 of the case study).

The infrastructure itself should include a central repository and agents that capture (through its senses) certain dynamic attributes of another agent. By agents, we mean both people (that populate a table with what they observe) and software (that register state variables from its environment).

Creation and maintenance of this infrastructure may not always be pacific and can generate adverse reactions simply by means of the Observer Effect alone (e.g. a monitoring application consumes the host’s resources). As such, among other things, agents (by the Heisenberg uncertainty principle) should cast the least interference possible when performing their observations. For this we suggest other literature and studies regarding change management [ZMT08].

### 3.5.2 Extending the Meta-Model

The framework chosen was the Center of Organisational Design and Engineering Framework (CEOF) since it draws a clear line between the several layers of the Enterprise Architecture, and describes the
entities that belong to each.

As already presented in section 2.1, an Enterprise Architecture modelled using CEOF has:

- Business Architecture
- Organisational Architecture
- Information System Architecture
  - Application Architecture
  - Information Architecture
  - Technological Architecture

We have extended this framework at the Information System Architecture level, through its Block and Service elements since, in this architecture, the concept of agent as action-performer entity does not exist.

3.5.2.1 Monitoring Point

Vasconcelos [Vas07] defines Block as “the representation of a modular component of information systems that aggregate a set of operations which describe a system or other elements”. Looking at the previous section, we’ve identified that entities are measurable through their manifestations. Thinking back to our LEAN extension, Blocks represent Agents and therefore we must define how (or through what) they manifest themselves in reality. For this meta-model, we changed the manifestation semantics slightly in order to better align out concepts with CEO’s. The ODE loop we presented in the previous section shows the concepts “Variables” and “Monitoring Points” as the meta-level inputs for Domain Monitoring. To better suite ODE’s syntax we used the latter concept “Monitoring Points” to refer to the observation standpoint of a certain variable of an Agent’s behaviour. Monitoring points are therefore the equivalent of the Manifestations, but add a perspective semantic to it. This characteristic is useful due to the nature of the Case Study monitoring platform, that relies on small footprint agents located near systems that observe their manifestations.

**Definition**

A Monitoring Point is a dynamic property of a Block that can be observed over time.

**Architectural Attributes**

No additional attributes.

**Relations**

This primitive has the following associations:

- Block - the Monitoring Point exist within Block.

**Semantics**

Monitoring Point primitives represent observable variables of information system components.

**Specializations**

There are no specializations.

**UML definition**
3.5.2.2 Metric

Looking back into the ODE loop, Domain Analysis is performed adding models (structure) and heuristics (evaluation) to the results of Domain Monitoring (observable data). To the heuristics we will call Metrics. Essentially the Metric concept represents an expectancy that is based on collected data from Monitoring Points.

In CEO Framework, the Service is a collection of operations made available by architectural blocks. We therefore apply the metric concept to the Service element, so that we can evaluate actions, using manifestations from the actor, according to LEAN and our LEAN extension.

Definition
A Metric represents a behavioural expectancy of a Service.

Architectural Attributes
No additional attributes.

Relations
This primitive has the following associations:

- Service - Metric primitives exist within Service.

Semantics

Specializations
There are no specializations.

UML definition
Well Formed-ness Rules

- A Metric can only:
  - a) Exist within the context of a given Service

  ```
  context Metric inv Metric_Relations:
  self.class.oclISKindOf(Service)
  ```

Notation

Metric follows the standard UML representation for Operations

Representation

There are no alternatives.

Options

```
<metaclass>
  Operation
  `\n
<stereotype>
  Metric
```

UML Profile diagram

(partial)

3.5.3 Instrumenting the MPF cycle

As already reviewed in section 2.3, there are several applications designed for modelling information systems, which is what occurs in this step: using a well defined meta-model to construct a model of a given architecture. Once described in a structured form, this description can continue the flow of the Model Processing Framework.

For the purpose of this thesis, the EMF is useful for it provides the modeller the ability to specify what to observe and how to measure.

The tool selected was openArchitectureWare since, of all those reviewed in chapter 2, it presents the better support for the needed characteristics: graphical interface, UML2, UML Profiles, Code Generation, XMI import and export, OCL checking and for being Open Source.

3.5.3.1 Model Editor

As referred in section 3.4.1, the model editor requires easy-to-use, UML2 class and object diagram support as well as support for UML profiles.

openArchitectureWare includes the Graphical Modelling Framework\(^1\) (from the Eclipse Modelling Project) that provides this tool with a graphical environment for creating diagrams. Since UML2 is supported, it is possible to create Class, Deployment, Component, Activity, Composite Structure and other standard diagrams and therefore it is suitable for implementing this MPF.

As seen in figure 3.12, we’ve recreated the Information System Architecture of the CEO framework on a UML Profile (using a Profile diagram in the graphical interface). We have also inserted our meta-model

\(^1\)http://www.eclipse.org/modeling/gmf/
extension in this diagram, by creating two Stereotypes named Monitoring Point and Metric as described in section 3.5.2.

![Figure 3.12: UML Profile definition in openArchitectureWare](image)

Having the CEOF profile and its extension defined, it is now possible to create an UML model (using the Class Diagram), apply the profile, and model an architecture with the correct taxonomy.

To model object instances in oAW, the eclipse UML2 implementation supplies the InstanceSpecification element, which is a M1-level representation of the M0 object.

3.5.3.2 Model Converter

The next step in the MPF workflow is to convert the UML model into a format that can be used by the Model Analyzer. In order to manipulate the UML model as an input for other components, a XMIReader component was used. This component receives a model, a meta-model and allocates a slot so that it can be read by other components. Then, the Generator component was added to the workflow to perform code generation. The component receives a model (plus the corresponding meta-model) and a Xpand file, and uses the latter to translate into a programming language each of the Meta-Model, Model and Object level elements.

The target language chosen for the code generation was Ruby for its simplicity, meta-programming support, time required to implement and because some of the infrastructure, described later in section 4.1, is already built upon Ruby. The CEOF meta-model was recreated in Ruby using an Object-Oriented approach so that, when the Generator is invoked, it creates two ruby source files: one for the model, and another for the instances, and each level is supported by the upper.

3.5.3.3 Model Analyzer

openArchitectureWare supports OCL expressions and validations, that are used to calculate Service metrics by navigation InstanceSpecifications, accessing Monitoring Point values in Blocks, performing calculations and finally comparing the output with a threshold which warns the user.

2 www.ruby-lang.org
By bringing objects (M0) into the model (M1) level using the uml::InstanceSpecification, oAW allows writing metrics as constraints. The object diagram can be navigated from instances of services to the blocks that support them by using the navigable associations created in said diagram. Basic arithmetic operations are built-in in the language and, in the necessity of more complex ones (e.g. filters, statistic functions), it can be extended using Java and the already existing set of libraries for such purposes.

3.5.3.4 Results Converter

Finally, the Results Converter transforms the manipulated model (now possessing values for the monitoring points) back into UML. This processing block also has the responsibility to guarantee that the end of the processing cycle leaves the openArchitectureWare in a state where the process can be run again. In order to accomplish the necessary idempotency, the only oAW project objects that are altered during the workflow cycle are run-time model slots (which disappear after execution) and the Value elements in Slots of InstanceSpecifications where the monitoring point values are stored. To be able to show the user the values used for calculating metrics, the model UML(.uml) file is rewritten, and special care is given so that in subsequent executions of the workflow only changes these Values.

3.5.3.5 Tying it all together

The openArchitectureWare tool has, as already referred in section 2.3.3, an workflow engine in its core. This enables the creation of the MPF workflow, by executing several tasks and automating the flow between each of the MPF’s steps (the code for this workflow is listed in appendices 6.1):

1. Load UML model into a slot (XMLReader component)
2. Verify model and architectural qualities (Check component)
3. Generate Ruby code for model and instances (Generator component)
4. Transform the slotted model by inserting value nodes into InstanceSpecifications (Xtend component)
5. Execute checks on the transformed model (Check component)
6. Overwrite the UML model with the slotted model (UML2Writer component)

3.6 Summary

As a base for our implementation, we came up with an abstract ‘picture’ of how the dynamics of agent interaction occur and how these should be modelled.

We constructed a conceptual framework for modeling manifestations and evaluations on an Information System Architecture. The Model Processing Framework is comprised of a graphical Model Editor to ease the modelling process, a Model Converter for serialization, a Model Analyser that performs both model and object level evaluations and finally a Results Converter the presents results to the modeller and We were able to create a fully automated flow between each of the MPF’s steps so that, once configured, it can be triggered periodically to refresh the current “view” of the dynamic status.
3.6.1 Limitations

The implementation has only the necessary and sufficient constructions to perform very basic navigation on the model. Nevertheless, support for more complex interactions is possible for the implementation was built upon Eclipse’s UML2 and therefore higher level functions can be constructed, to ease the user’s experience of the rule writing process.
4 Case Study and Validation

In this chapter we explain how the solution was applied to a real-world situation. This thesis was developed in an enterprise context, therefore our testbed consisted of the enterprise PT-Comunicações and its technological infrastructure.

We begin by describing the monitoring framework available to us in this enterprise environment, which we used and attempt to improve, by using its collected data and providing a better evaluation experience.

Then, describe a “proof-of-concept” implementation of an existing scenario using a Call Center. Our purpose is to demonstrate the steps needed to construct a cycle of the MPF using the implementation we proposed, using the CEOF metamodel and openArchitectureWare. For this, in section 4.2, we describe a fraction the PT’s Call Center system (sufficient for an example) and the creation of an evaluation scenario.

4.1 Pulso Architecture

PT-C has been concerned over the years with measuring IT performance, availability and errors. For that purpose, the Pulso platform was developed [JA05] to sustain mechanisms of near-realtime measuring, monitoring and storing basic performance indicators for major systems that support key business processes.

However, the evaluation procedures are still ad-hoc since there's no technical or conceptual framework that allows - in a simple, declarative and semantically robust way - to define what entities are being targetted and how to specify and calculate the relevant indicators for either Quality of Service, Quality of Protection or even Quality of Maintenance.

The EDS¹ division of PT-Comunicações is responsible for this monitoring platform and, as such, provided an environment that was both resourceful and accessible for us to experiment with our solution since we were given access to an infrastructure of monitoring, data collector agents attached to every relevant server, database and network link. By “relevant set” we mean the entities that are, at the time of this writing, critical to the business.

4.1.1 Agents and Probes

The Pulso monitoring framework is comprised of system monitoring agents (e.g. Linux, Windows, HP-UX, Oracle) and network probes (client-server and server-server).

Agents are system-specific software whose job is to periodically capture basic behaviour readings of their hosts. For instance, Linux agents capture observable variables such as CPU usage, CPU load (for 1, 5 and 15 minutes) and memory usage. For databases, agents designed for Oracle SGBDs capture waiting times for query processing queues. Depending on the business demands, the agents are further developed to produce more observable variables.

Probes passively monitor network traffic between either servers and clients or between servers. They capture data regarding to bandwidth consumption and protocol-specific traffic (e.g. SQL transaction

¹Eficiência, Disponibilidade e Segurança de TI/SI
requests and errors, IIS communication).

4.1.2 Measuring IT

As previously said, one of the desired features of Pulso is the evaluation of IT and the ability to verify compliance with Service Level Agreements.

In the Pulso platform the definition of Enterprise entities (ranging from disk partitions to business processes) is constructed using a graph of "things" and the links between them. Therefore, there are no classes, no type hierarchy and no architectural layers. This approach provided some benefits due to the lack of constraints on the evaluation process, making it is faster to implement. However, by allowing any entity to be related to another and by providing full flexibility and no constraints, this solution has also been proven to be hard to maintain as systems grow in complexity. Since there are no a priori validations that can be performed on the Technology Infrastructure model, the structures evolve into large graphs, inherently difficult to debug and complex to change in the long term.

4.1.3 Storing observations

As stated before, the Pulso platform possesses agents that capture sensory data from their environment. To make use of such data, a centralized repository was created in the form of a database, to where all events produced are dispatched after being sent to a central logging server. All events have a common header format that includes an identification of the monitored entity, what is observed attribute and the timestamp of the event.

```
<Pri><Timestamp><Host>PULSO:<Version><Agent><Metric><Counter><Target><Ts_end>
<Duration><value1><value2>...<valueN>
```

Listing 4.1: Pulso Event message format

- `<Timestamp>` - Instant the event was sent to the Pulso Syslog server (format RFC 3164);
- `<Host>` - Host name of the machine that generated the event;
- `<Agent>` - Identification of the event generator agent;
- `<Version>` - This header's format version;
- `<Metric>` - ID of the metric this event relates to;
- `<Counter>` - Auto increment counter;
- `<Target>` - Name of the entity being measured;
- `<Ts_end>` - Timestamp the event was generated (format ISO 8601:2004 3.2);
- `<Duration>` - Interval in seconds during which data was collected to generate this event (e.g. 5 min average of cpu usage => 300)

The header helps a dispatcher determine where to insert the event in a database where the timeseries for each event type are stored, for future analysis.
4.2 Call Center

As a case study, we chose to analyse a fraction of the Helpdesk Call Center that provides assistance to PT’s internet, television and phone clients. The infrastructure that supports consists of:

- Interactive Voice Response - is an interactive technology that allows a computer to detect voice and keypad inputs. It is used to channel the client to the correct pool of call center agents by a telephone key driven menu;
- Automatic Call Distributor - is a device or system that distributes incoming calls to a specific group of terminals which are use by agents;
- Computer Telephony Integration - allows interactions on a telephone and a computer to be integrated or co-ordinated;
- Customer Relationship Management - manages client data such as personal information and account data. It is supported by Siebel software.

In this proof of concept we modeled two metrics based on three observable manifestations of the ISA’s behaviour, that we explain further below.

4.2.1 Process View

The typical flow of a client's call starts when the client dials the number and is picked up by the IVR. The client is asked to specify one of the pre-selected classes of problems and, after the IVR has the necessary data, it transfers the call to the ACD. The distributor selects an available agent, from the correct pool one agents, and the conversation can now begin. Below (fig. 4.1) we can see only the interaction between IT components and human actors:

![Figure 4.1: Relationship between Client, Operator and IT Infrastructure](image)

4.2.2 Setting up openArchitectureWare

Our first step of the implementation was to create an openArchitectureWare project and properly configure it, according to the instructions in the appendix 6.1. This involves:
1. creating an empty openArchitectureWare project and configuring it for UML2;
2. loading the CEOF profile;
3. creating an empty UML model and a UML Class diagram.

Having set up the base project, we started modelling the Call Center into the oAW application.

4.2.3 Information System Architecture

The creation of the Information System Architecture is essential to our modelling of the phenomena produced by these systems and so our next step was to describe it. Since oAW has limitations regarding the display of custom shapes, we first constructed the model using CEOF's customized Visio stencils (shown in fig 4.2). This can also help the communication of the model for those already familiar with CEOF's graphical notation. While modelling this fraction of the ISA we only included the blocks and services we were able to identify from the existing documentation and therefore elements such as databases or even software components were possibly left out. Nevertheless, the business drive for this evaluation focuses on the architectural constructs shown below are are, therefore, sufficient to model the evaluation scenario for this use case.

![Figure 4.2: ISA for the Call Center](image)

Transporting the ISA to oAW's UML Class diagram is a direct conversion (fig. 4.3), since the UML
The metamodel and CEOF profile is fully supported in the application. To stereotype either classes (Blocks) or interfaces (Services), the context menu of that element provides an “Apply Stereotype” entry where only the appropriate stereotypes can be applied (e.g. Block stereotypes can’t be applied to interfaces).

Figure 4.3: ISA for the Call Center (on oAW)

### 4.2.4 Object diagram

Since what we desire is to capture the dynamic behaviour of this architecture, we also need to construct the corresponding Object diagram. This diagram contains the instances of the Class diagram that exist and will be the target of our evaluation.

To add the Object diagram onto our solution, we used the existing UML Class diagram where we added the representation of the objects through UML’s InstanceSpecification primitive and “Usage” relationships.

### 4.2.5 Adding Evaluation

The final step of the configuration is to establish what we can observe on each of the Blocks and how do we wish to evaluate the services on which they are implemented. As observable variables of the existing
blocks we have:

- “callAlerting” on the Computer Telephony Interaction IT Block
- “callConnected” on the Siebel Platform Block
- “cpuUsage” on the UnixServer

For the data collection activity to function the agents on these systems must be previously configured to capture the variables and store them on the DB.

4.2.5.1 Defining the Metrics

We chose to model the measurements as expectancies (that evaluate to true or false) for this fits better with the OCL paradigm and provides a way to warn the user of this MPF if an expectancy isn’t met.

The responsiveness metric was created to calculate the difference between the two moments of the call alerting and the moment the operator picks up the call. For the purpose of this proof-of-concept, we assumed the last timestamp of each moment and the metric simply determines if the difference is greater than a certain number. Since timestamps’ measurements don’t have milliseconds and most of the systems are on automatic answering, we do not want them to be greater than 1 second.

The performance metric is a simple threshold checker to see if the unix server’s cpu is not being overused, degrading the system’s performance.
The following code listing shows how the performance and responsiveness metrics were defined, using oA\textsuperscript{W}'s implementation of OCL. The metrics are defined using the model-level elements and also the thresholds that define if the metric passes or fails according to the expectations for the instances.

\begin{verbatim}
context InstanceSpecification if hasMetric(this, "responsiveness")
    WARNING "Last call took too long to pick up ":
    (getValue(this, "Computer_Telephony_Integration", "callConnected") -
    getValue(this, "Siebel", "callAlerting")) < 1;

context InstanceSpecification if hasMetric(this, "performance")
    WARNING "Last call took too long to pick up ":
    getValue(this, "UnixServer", "cpuUsage") < 95;
\end{verbatim}

Listing 4.2: Responsiveness and Performance Metrics

4.2.5.2 Resulting ISA

Adding monitoring points to Blocks is done by adding an attribute to a Block stereotyped Class and applying the CEOF::Monitoring Point. Similarly, adding operations to Services defines metrics, after applying the stereotype (of the same name). We applied three Monitoring Point and the two Metrics to the existing ISA (fig. 4.5).

4.2.6 In Execution

Having our class and object diagrams, manifestations and metrics modelled, we were now able to execute the Model Processing Framework's workflow and observe its output. As seen in figure 4.6, at the end of the cycle, the instances that possess monitoring points now have values associated, and the Console View at the bottom of the oA\textsuperscript{W} screen has launched a warning since the last registered call took 1 second to be answered (the difference of timestamps between the two moments).

4.3 Conclusion

We managed to perform a full MPF cycle using the implementation presented on section 3.5, therefore showing its feasibility. Nevertheless, its implementation depends on a monitoring infrastructure such as PT-Comunicações and so we limit the ability of easily replicating this scenario on another environment. Being a proof of concept, this example is very simple even though it explores all the elements of our solution. As we state in the Future Work section, we hope to instill curiosity and further exploration of this tool so that larger problems can take advantage of the MPF, both at PT-Comunicações, any willing enterprise and with other enterprise architectural models. Regarding PT-Comunicações, the MPF has not been implemented in a full scale information system architecture, but given a growing concern regarding the necessity of structuring the enterprise and its systems, the author of this thesis hopes to able to aid that process by using some, if not all, of this solution.
Figure 4.5: UML Class diagram for the ISA with added Monitoring Points and a Metric
Figure 4.6: Results after a run of the MPF cycle.
5 Conclusion

The Information System Research is still in a very immature state [Kho07]. Several contributions have started to emerge during the last 20 years and the interest on the topics such as Information System modelling, Enterprise Architectures and Organisational Engineering and Design are growing fast.

In this thesis we attempted to further develop the works of Gerald Khoury and André Vasconcelos by providing some insight on how to perform architecture-centric, ontologically supported, evaluation of Information System behaviour. We began by finding the appropriate mindset to approach this problem through a higher level abstract analogy to our problem. We turned into the metaphorical view of societies as a superset of enterprise systems, and searched within this metaphor for guidelines on how to model a solution.

We encountered a parallel between the field of Phenomenology (that studies human and group experience and phenomena) and the acts of judging the behaviour of architectural components by transforming their observable variables.

To help modelers, analysts and every participant in the enterprise community, we proposed an automated cycle of evaluation using the Enterprise Architecture as the standpoint and communication medium between all parties.

The Model Processing Framework was successfully implemented using the openArchitectureWare tool, and finally we constructed an example taken from a real-world enterprise situation and we were able to measure a simple metric from observed manifestations.

To recapitulate, our initial hypothesis were:

1. It is possible to extend an existing modelling Information System Architecture framework to express what can be observed in deployed Information Systems, how to measure and which measurements influence (which) qualities;
2. The tasks of modelling an architecture and its metrics and analysing a deployed model’s behaviour can be combined into a framework;
3. Each of these framework’s steps can be connected to form an automated circular flow that, once configured, requires no human interaction;
4. Organizations can have temporal views of their deployed Information System Architecture regarding previously established metrics.

We accomplished hypothesis 1, 2 and 3 through the creation of the MPF and the implementation using the CEOF enterprise meta-model and openArchitectureWare. For the last hypothesis, only partial compliance was attained, since the proof of concept implementation of the MPF was done for just real-time evaluation. Another use for this activity is the ability to extract, for example, monthly views of the constructed metrics. Even though this functionality is not present in our solution, we have studied the required work to accomplish it:

- It is necessary to make the notion of time explicit in the enterprise architecture. For CEOF, this can be done through the Time Model UML package [Fom02] that possess classes such as Clock and Time Period;
- The code used to extract values from the facticity database can be improved to access time intervals;
The calculation of metrics using openArchitectureWare's OCL implementation can be extended to perform operations using arrays of data instead of scalar operations.

5.1 Shortcomings and Future Work

Regarding the phenomenological aspect that we provided in this domain, we think that much more can be done to improve the holistic view on Enterprises by further understanding the key ideas of this philosophical domain. This requires, however, a less traditional engineering approach and willingness to nearly “decipher” some of the cornerstone books of this discipline.

Regarding the ontological extension we provided, we believe it can be further extended to support more complex interdependencies between metrics and observable variables. At the same type, we limited the Monitoring Point to an observable variable, when it could possible be understood as an one-to-many relationship.

The Model Processing Framework implementation has some limitations that can be overcome with a harder focus on software development:

- The cycle cannot be repeated automatically (the user has to press the “Run” button again). Investigation has already been done (also the community forum is very active) regarding this issue, and it can be solved with a Java Ant task that executes the workflow as a normal Java application;
- The helper functions that abstract the metric design from the Eclipse/Ecore model navigation can be further developed to provide a friendlier experience;
- The presentation of negatively evaluated metrics can be improved by, for example, marking the diagram elements with color codes;
- Not all meta-model checks of the CEO framework were implemented.

Finally, being “unified” an aspect of our ontology extension and conceptual MPF, we did not attempt to apply our solution to another architecture meta-model such as, for example, Archimate. We do, however, consider the solution to be perfectly compatible with Archimate, for it also has a well-defined notation, even though not based upon UML. openArchitectureWare is also prepared to handle such flexible notations as JavaBeans-based models and XML-based models, so this should not present a major challenge.
Bibliography


[JA05] Ricardo Ramalho José Alegria, Tiago Carvalho. Uma experiência open source para “tomar o pulso” e “ter pulso” sobre a função sistemas e tecnologias de informação. (CAPSI), 2005.


6 Appendices

6.1 Appendix 1 - MPF Project Set up

6.1.1 oAW Configuration

Below are the steps necessary to create a new oAW project and correctly setup the MPF, using the CEOF meta-model and Ruby code generation. For the files used in this project and more in-depth description of the implementation, configuration and use go to https://fenix.ist.utl.pt/homepage/ist153893

1. create empty openArchitectureWare project

2. in META-INF/MANIFEST.MF, go to the "Dependencies" tab and add "org.openArchitectureWare.uml2.adapter" to the required plug-ins;

3. in Project -> Properties -> openArchitectureWare activate "UML2 Profiles" and "EMF Metamodels"
   - Note: the "UML2 Profiles" must be on top since order determines method overriding

4. Copy files in solution.zip to root of the project

5. Create UML model in src/ directory, along with the Class Diagram

6. Edit workflow.oaw and set the name of the project accordingly

7. Open the Class Diagram:
   (a) in the context menu Load Resource -> Browse Workspace select "src/metamodel/code.profile.uml"
   (b) again in the context menu Apply Profile -> CEO

8. Define using the Class Diagram:
   (a) the model
   (b) the object instances
   (c) the monitoring points in Blocks
   (d) the metrics in Services

9. In the src/metamodel/ObjectChecks.chk file define how each metric is calculated using OCL

10. Finally, execute the workflow using Project -> Run and you’ll see:
   (a) If the model is semantically incorrect, an error will occur and the workflow will be interrupted
   (b) Otherwise, slot values in InstanceSpecifications will be populated (by the last observation of the monitoring point)
   (c) Object metrics will be calculated and for those metrics above desired levels, a Warning notification will be displayed

11. The workflow will end and it can be restarted for a fresh picture

---

1If reading the PDF version click here to detach a “tar.gz” archive with the configuration files
6.1.2 Configuration Files

6.1.2.1 Workflow File

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<workflow>
  <bean class="oaw.uml2.Setup" standardUML2Setup="true" />
  <component class="oaw.emf.XmlReader">
    <modelFile value="src/Inbound.uml" />
    <outputSlot value="model" />
  </component>

  <component class="org.openarchitectureware.check.CheckComponent">
    <metaModel class="oaw.uml2.UML2MetaModel" />
    <metaModel id="profile"
      class="oaw.uml2.profile.ProfileMetaModel">
      <profile value="src/metamodel/ceo.profile.uml" />
    </metaModel>
    <checkFile value="metamodel::ModelChecks" />
    <emfAllChildrenSlot value="model" />
    <abortOnError value="true" />
  </component>

  <component id="generator" class="oaw.xpand2-generator" skipOnErrors="true">
    <metaModel class="oaw.uml2.UML2MetaModel" />
    <metaModel id="profile"
      class="oaw.uml2.profile.ProfileMetaModel">
      <profile value="src/metamodel/ceo.profile.uml" />
    </metaModel>
    <expand value="templates::Ruby::Root FOR model" />
    <outlet path="src-gen">
      <postprocessor class="oaw.xpand2-output.JavaBeautifier" />
    </outlet>
  </component>

  <component class="oaw.xtend.XtendComponent">
    <metaModel class="oaw.uml2.UML2MetaModel" />
    <metaModel id="profile"
      class="oaw.uml2.profile.ProfileMetaModel">
      <profile value="src/metamodel/ceo.profile.uml" />
  ```
Listing 6.1: openArchitectureWare workflow file

6.1.2.2 Java Extensions

```java
import uml;
import ecore;

extension org::opengeometry::uml2::profile::profiling;
extension org::openarchitectureware::util::IO;
extension org::openarchitectureware::util::stdlib::io;

/*@Starting point for the model modifications*/
Void transform(Package m) :
    findMonitoringPoints(m) ->
    m;

Void findMonitoringPoints(Package m) :
    m.allOwnedElements().typeSelect(uml::InstanceSpecification).Test();

Void Test(InstanceSpecification i) :
    i.slot.select(e | e.definingFeature.getAppliedStereotypes().exists(f | f.getQualifiedName().matches("CEO::Monitoring_Point"))).collectValue();
```
```java
package mystuff;

import java.util.Random;

import org.eclipse.uml2.uml.InstanceSpecification;
import org.eclipse.uml2.uml.umlFactory;
import org.eclipse.uml2.uml.umlLiteralInteger;

public class Extensions {

    public static void setIntegerValue(org.eclipse.uml2.uml.umlSlot slotInstance) {
        String slotStr = null;
        LiteralInteger value = null;

        slotStr = ((InstanceSpecification) slotInstance.getOwner()).getName();
        slotStr += ";" + slotInstance.getDefiningFeature().getName();

        if(slotInstance.getValue("collectedValue", null) == null) {
            LiteralInteger literalInteger = UMLFactory.dINSTANCE.createLiteralInteger();
            slotInstance.createValue("collectedValue", null, literalInteger.eClass());
        }
        value = ((LiteralInteger) slotInstance.getValue("collectedValue", null));
        value.setValue(retrieveValue(slotStr));
    }

    public static int retrieveValue(String a) {
        System.out.println("Collecting value for: " + a);
        //System.sleep(1);
        return (int)(System.currentTimeMillis()/1000) + (new Random()).nextInt(2);
    }
}
```

Listing 6.3: Java Extension

6.1.2.3 Ruby code generation

The Xpand configuration file describes how the model nodes should be expanded to Ruby code.

```xml
<DEFINE Root FOR uml::Package>
<FILE "model.rb">
    require 'src/metamodel/ceo'

    <EXPAND BlockRoot FOREACH ownedType.typeSelect (CE0::Block)>
    <EXPAND InterfaceRoot FOREACH ownedType.typeSelect (CE0::Service)>
```

58
FILE "run.rb"

```ruby
require 'src-gen/model'

EXPAND instanceSpec FOREACH ownedElement.typeSelect(uml:InstanceSpecification)

DEFINE ClassRoot FOR uml:Property
    attr #:name#
ENDDEFINE

DEFINE ClassRoot FOR uml:Operation
    def #:name#
        nil
    end
ENDDEFINE

DEFINE NodeRoot FOR uml:Node
    class #:name#
        puts "undef node"
    end
ENDDEFINE

DEFINE ClassRoot FOR uml:Class
    class #:name#
        puts "undef class"
    end
ENDDEFINE

DEFINE InterfaceRoot FOR uml:Interface
    class #:name#
        puts "undef interface"
    end
ENDDEFINE

DEFINE BlockRoot FOR CEO:Block
    class #:name# < getAppliedStereotypes().first().getQualifiedName()>
    EXPAND AttrRoot FOREACH ownedElement.typeSelect(CEO:Monitoring_Point)
end
ENDDEFINE

DEFINE BlockRoot FOR CEO:IT_Infrastructure_Block
    class #:name# < CEO:ITInfrastructureBlock
```
6.1.2.4 Ruby code for CEO's ISA metamodel

```ruby
# require 'eva.rb'
```

Listing 6.4: Xpand ruby template
Listing 6.5: Ruby FCEO Metamodel

Listing 6.6: Ruby Block source code

Listing 6.7: Ruby Service source code