Praxeme Institute

Guide

PxM-02en **“Modus: the methodology Praxeme”**

General Guide

Objective This document lays down the foundations of Praxeme, an enterprise methodology. It addresses those who are interested in, who appraise and develop enterprises and their IT assets. It presents the basic principles and concepts that structure this open method.

Contents

- The foundation: “The structuring principles”, “The notion of “service””
- The products
- The processes
- Modeling guidelines

Author Dominique VAUQUIER

Translators Carolin CHAI, Nigel STRANG, Dominique VAUQUIER, Joanne TOWARD

Version 1.99.1, 27 November 2010

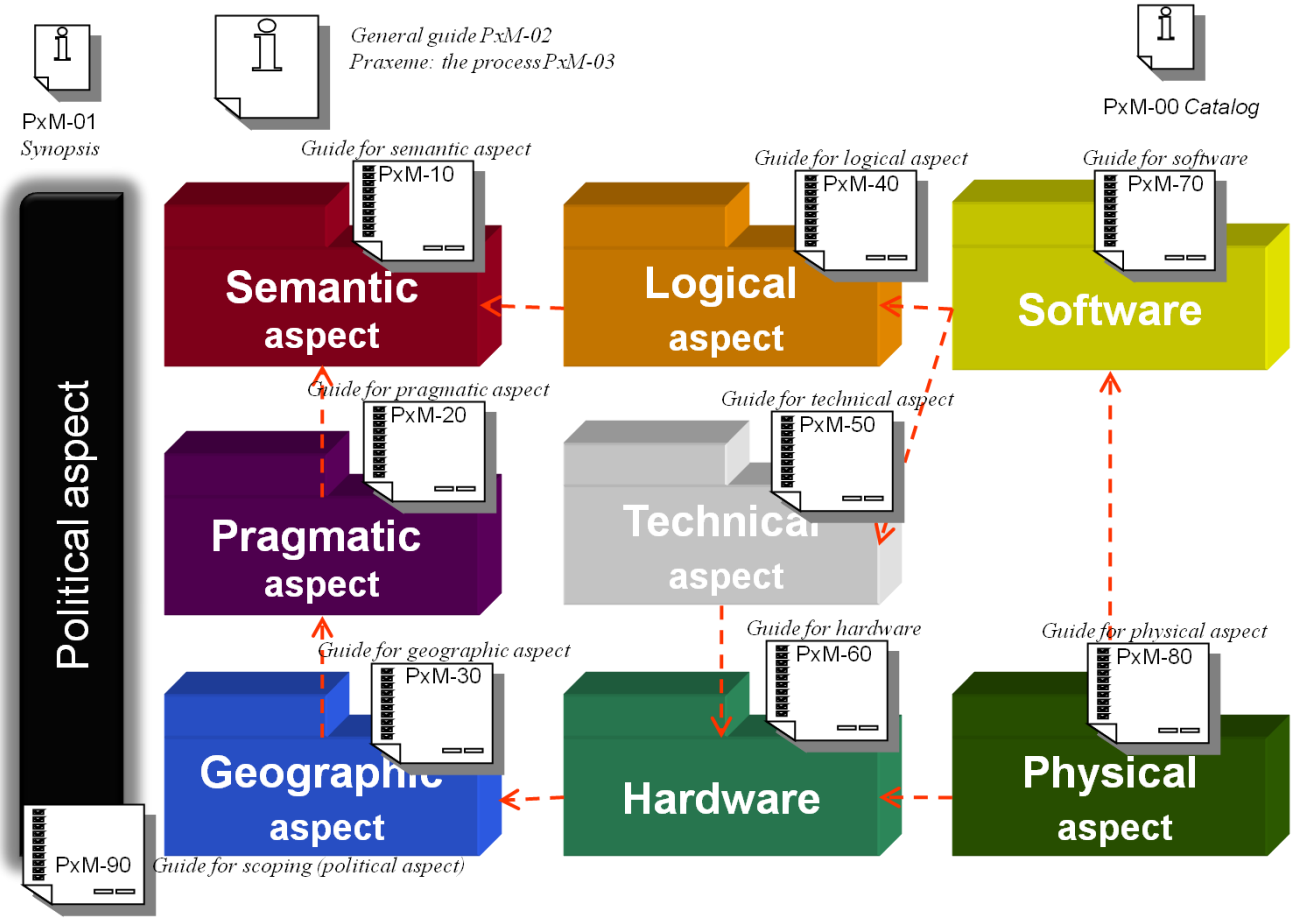
Configuration Elements

The position of this module in the methodology

Situation in the documentation

The methodology Praxeme is based on and structured by the "aspects" and the Enterprise System Topology. The general guide (PxM-02) explains this approach. PxM-41 is an addition to the guide for the logical aspect (PxM-40).

Figure PxM-02en_1. Structure of the Praxeme corpus in the "Product" dimension



Owner

The Praxeme methodology results from the initiative for an open method. The main participants are the enterprises SAGEM and SMABTP, and the French army[1]. They combined their forces to found a public 'open' method. The Praxeme Institute maintains and develops this joint asset.

Any suggestions or change requests are welcome (please address them to the author).

Availability

This document is available on the Praxeme website and can be used if the conditions defined on the next page are respected. The sources (documents and figures) are available on demand.

[1] See the website, www.praxeme.org, for the entire list of contributors.

Configuration Elements

Revision History

Version	Date	Author	Comment
	March 2004	DVAU	First Writing (Dromos: method Sagem for enterprise architecture of the UAV[2] system)
	November 2005	DVAU	Extended version (Amos: the SMABTP method; SOA approach)
1.0	27 April 2006	DVAU	Generalized for submission to the first "Circle of experts", launching the Praxeme Institute
1.1	3 July 06		Reviewed by the Circle of Experts Praxeme (see list below)
1.2		Carolin CHAI Nigel STRANG	Translation
1.99.#	October 2009 November 2010	DVAU Joanne TOWARD	Review of the translation
1.99.1			Current version of the document

The French version of this document has been reviewed by: Guy BOISSARD (Conix Consulting), Pierre BONNET (Orchestra Networks), Antoine CLAVE (Fidelis), Philippe DESFRAY (Softeam), Fabien VILLARD (Praxeme Institute).

[2] Unmanned Air Vehicle.

License

Conditions for using and distributing this material

Rights and responsibilities

This document is protected by a "Creative Commons" license, as described below. The term "creation" is applied to the document itself. The original author is:

- Dominique VAUQUIER, for the document;
- The association *Praxeme Institute*, for the entire methodology Praxeme.

We ask you to name one or the other, when you use a direct quotation or when you refer to the general principles of the methodology Praxeme.

Content

Epigraph

"Theory without practice is mere intellectual play but practice without theory is blind."

Immanuel Kant

Introduction

A new method for new concerns

The situation

General situation Obliged to adapt constantly, enterprises must make the most of new technologies, while unleashing their potential for innovation in terms of expertise and organization. The principal obstacle they encounter is the promotion of synergy between experts who find it hard to interact and to acknowledge each other. Sharing a common framework that represents and interconnects their skills is a necessary condition to surmount this obstacle

An enterprise methodology Praxeme is an enterprise methodology that aims to provide such a common framework. The system topology, described here, is its theoretical foundation. Praxeme also contains the processes that guide how to approach specific aspects of a system, without losing sight of the global coherent view.

To control information systems The most developed axis in Praxeme currently concerns information systems. These systems, known to be complex, are at the heart of the economic battle. However, this complexity, coupled with the disorder found within specialties, considerably decreases the possibilities of optimization and increases costs and uncertainties. Praxeme has been built, principally, to help CIOs meet their new challenges. It builds on currents such as the object-orientated approach, Service Orientated Architecture (SOA) and the standard, "Model Driven Architecture" (MDA). Praxeme uses and provides a guide for the use of the standard notation UML (The *Unified Modeling Language*).

The objectives

This document provides an overview of the framework. It explains the foundations of the approach and justifies the different methods and products.

The "General Guide" defines the principles that structure the framework. It considers needs and circumstances. It does not cover the entire methodology, but provides an introduction to it. Other documents focus on specific procedures and types of product (the forms, notably).

The content

The first part of the guide defines the principles. These are then developed through the three dimensions of the methodology:

1. Product (what needs to be produced, the deliverables),
2. Process (how to be organized to produce),
3. Procedure (how to work concretely).

In the third part, we focus on modeling procedures.

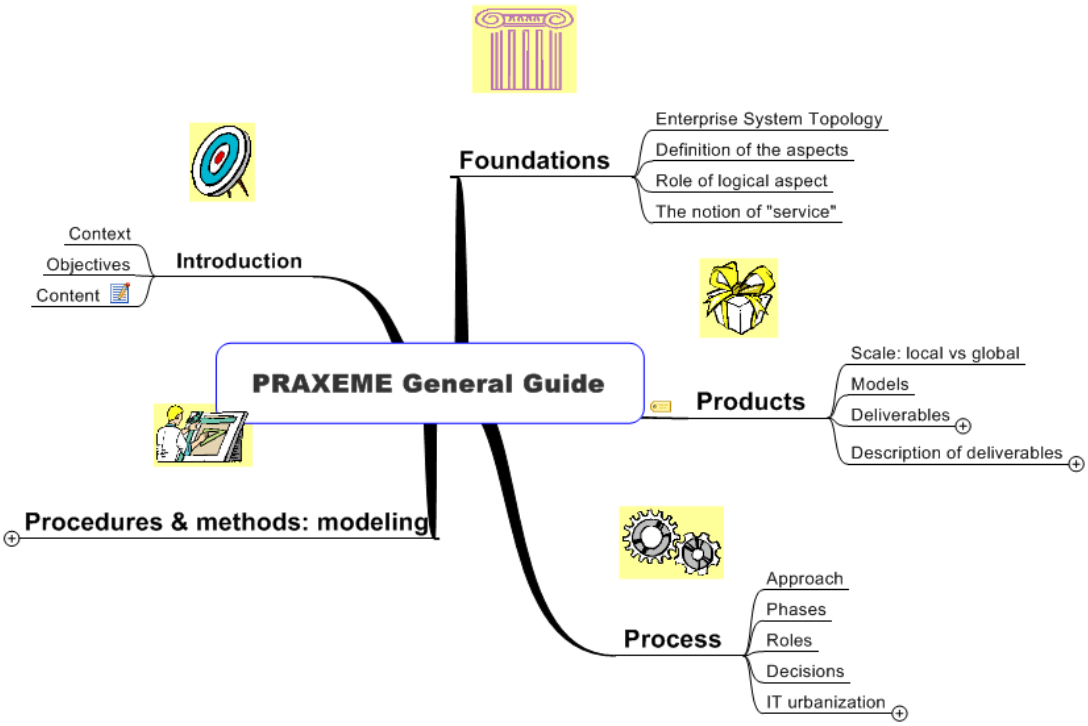
Related Documents

Apart from the guide to the different "Aspects" (illustrated by figure p. iii), a complete set of commented examples completes the general guide. Its reference is "PxM-02x".

Introduction

The three dimensions of the methodology: Product, Process, Procedures

Figure PxM-02en_2. Structure of the document



Information on the procedures

The document presents a selection of procedures in the order of the methodology:

- Semantic modeling, from the preliminary to the identification of high-value services
- System specification based on use cases.
- Logical architecture.
- The design of services.

Foundations

Construct practice on a solid base

Motivation

Enterprises and information systems are complex objects calling on many forms of expertise. It will only become possible to guide intervention on these objects, once a theoretical framework has been defined that fixes the relative contribution of the various forms of expertise.

The structuring principles

- An initial question springs up: "What needs to be represented, for us to be able to intervene on the system?". Praxeme answers this question with the **Enterprise System Topology** , an inventory of the aspects of the system.
- The principle of the **separation of concerns** structures the approach to the system: it separates the different models and conditions the transformation process.
- Among the identified aspects, **the logical aspect** is particularly important from the perspective of service orientated and enterprise architecture.
- Praxeme gives particular attention to the **notion of service**, the elementary constituent of the information system and vector of exchange with related systems.

The following pages explore these principles.

"What needs to be represented?"

The Enterprise System Topology continues the tradition of software engineering:

- It inherits the notion of abstraction levels from Merise and methods in the 80s.
- It is inspired by enterprise architecture approaches such as the Zachman's framework.
- It updates these legacies in accordance with the standard "MDA" (*Model Driven Architecture*, standard of the OMG[3]). MDA proposes the modeling principle that some models are independent of technology, and some are dependent on technology (PIM: *Platform Independent Model* and PSM: *Platform Specific Model*).

[3] The Object Management Group (OMG) gathers many of the actors in the field of software industry and engineering. It publishes standards among which the most famous are CORBA (Common Object Request Broker Architecture) and UML (Unified Modeling Language).

Foundations (cont.)

The Enterprise System Topology

The problem

Once it has been understood that modeling is an indispensable preliminary to the design of complex systems and that models are one of the factors enabling the intellectual control of enterprises and information systems, the question is: what exactly needs to be modeled? UML as a notation standard does not answer this question of methodology.

Failure to address the question "What needs to be modeled?" exposes projects and transformations to major risk, and in all cases leads to a significant waste of resources.

The solution

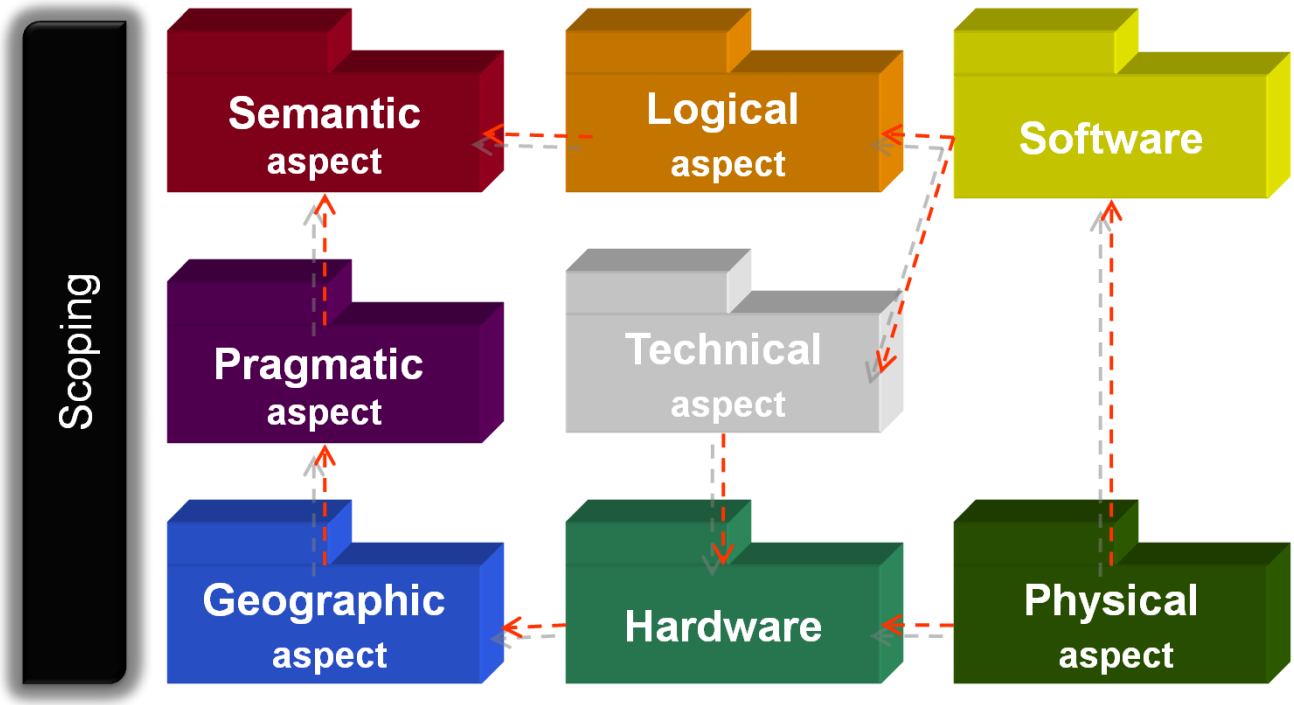
The proposed response consists in establishing the inventory of the *aspects* of the targeted reality (the enterprise in this case) that need to be examined to describe it, pertinently and exhaustively. These aspects are rigorously articulated in the "Enterprise System Topology" (EST) which forms the theoretical basis of Praxeme.

The essence of the targeted reality must be correctly grasped in order to be able to design the new organizations, processes or IT solutions that will provide the tools needed to support it.

The term "topology" has been chosen to evoke the idea of "place" (in ancient Greek: *topos*). The Topology defines the places where we manage and stock information and decisions acting upon the system.

The illustration below summarizes the Enterprise System Topology. A justification of this structure can be found in the appendix (Please see the appendix).

Figure PxM-02en_3. Framework of the Enterprise System Topology



Foundations (cont.)

The definition of aspects

Isolating the aspects

Enterprise architects intervene on systems dealing with complex realities. These systems involve a large amount of information and issues of interest to diverse lines of work. It is simpler to master the complexity if the issues are divided by discipline or specialty into separate homogeneous groups. These homogeneous groups form the "aspects" of Praxeme.

Isolating aspects helps to master the description of and facilitate the evolution of the system.

Aspect An aspect is a view of the system. The system is considered from the perspective of a particular preoccupation. So, although an aspect is a component of the system, it is relative not absolute: it is linked to a point of view, to a particular preoccupation, to a specialization. Some aspects have nothing to do with IT (Information Technology).

The table below defines the eight aspects retained for the Topology. (Please see the appendix for the justification.)

Figure PxM-02en_4. The definition of the aspects of an Enterprise System

Aspect	Equivalent Terms	Definitions
Semantic	Conceptual, essential, "The Key Business "	The semantic aspect describes the objects at the heart of the business. It describes the fundamental core independently of how the business is done.
Pragmatic	Organizational	The pragmatic aspect regroups the different choices as to how business is done: the actors, the responsibilities, the actions on objects, the processes and the work situations.
Geographic	"Communication", "Situation"	The geographic aspect records the physical location of objects and actions. Here the notions of sites, locations, and communication needs appear.
Logical	"Functional"	This intermediate aspect allows the expression of important decisions that structure the information system, with relative independence from technical issues.
Technical	Technological	The technical aspect describes the choice of technologies and their implementation.
Hardware	Logistics	The hardware aspect describes the physical machines that make up the system, and their characteristics (capacity …).
Software	Application, IT	The software aspect describes the software components which automate some of the actions of the Information System.
Physical	Deployment	The physical aspect describes the hosting of software components (databases included) on the IT infrastructure.

Foundations (cont.)

The order of the examination of the aspects

Motivation

Collecting information following a rigorous set of ordered steps during the design phase is good practice with the following advantages:

- **Productivity** due to a strict sequencing of the work and the issues to consider.
- Better **use of models** and documentation: each model has its own life cycle (the semantic model is very stable and will serve as a long-term reference; the logical model is independent of the technical architecture and does not change when technology changes, etc.).

Examples

The following table illustrates the aspects proposed by the topology.

Figure PxM-02en_5. Illustration of aspects

Aspects	Examples	Principal categories of representation	Comments
Semantic	Products, Contracts, loss, Articles (Insured article or assets)	Classes, state machines	The semantic model captures and formalizes the business foundation. Very stable.
Pragmatic	Actor, partner, organizational rules , User profiles, “Submit a claim”, “Order a product”	Actors, Use Cases, Process	The practices and organization rules are isolated. They can be changed more easily.
Geographic	Headquarters, regional branches, agencies, offices, overseas, nomadic access	Type of sites, networks (non IT)	The geographic model describes the underlying assumptions and constraints of physical sites.
Logical	“domains”, common resources, “Structures”, “blocks, districts, areas…”	Logical machines, logical services	This facilitates decisions about the structure of the system.
Technical	Data Support, *middleware*, technical components, languages…	Technical choices, *frameworks*	The technical architecture explains how to derive software from the logical description for a given target system.
Hardware	Machines, processors, connectivity, networks	“Nodes” and connectivity	
Software		Software components, applications	Software components are obtained by combining logical units and technical choices.
Physical		Software components and equipment	The software components are located on the IT infrastructure of the physical IT architecture.

Foundations (cont.)

The relations between aspects of the system

The associations

The links between aspects in the Enterprise System Topology indicate the transitions between aspects. These links indicate dependencies between the aspects.

Models must be articulated in order to be able to sequence work. The following paragraphs justify the links represented by the dotted arrows in the figure on page 4.

Link from the pragmatic aspect to the semantic aspect

Actions affect fundamental (real or conceptual) objects. In consequence, the pragmatic model (describing actors and their actions) refers to the semantic model which describes these fundamental core objects.

Link from the geographic to the pragmatic

Different types of location are typed according to the kinds of actors and their responsibilities. If a regional branch exists it is because certain types of responsibility are located there.

Link from the logical to the semantic and pragmatic

The terms of the logical model come from the transposition, guided by structural decisions and rules, of the semantic and pragmatic descriptions. This point will be described in detail later on.

From hardware to geographic

This association allows the description of the physical location of hardware. This description is completed with quantitative and qualitative elements.

Link from technical to hardware

Either can be chosen in function of the other. The topology chooses to give precedence to the hardware architecture as it has more constraints. Even if the reasoning is done the other way around, it is the dependencies determined by the topology that will be documented.

Link from software to logical and technical

All software components, whatever their dimensions, are the expression of the implementation, in a given technical architecture, of an identified logical component.

Link from physical to hardware and software

The software production chain culminates in the physical aspect. This last step consists of implementing software components on the appropriate hardware.

The importance of the relations

This articulation of aspects shows how the information, representations and decisions form a continuous sequential chain from one end of the software production cycle to the other.

The relations between the aspects summarize the dependencies on and references to the most finely detailed model elements.

It is possible to define derivation rules for these relations, which identify how to pass, sometimes automatically, from one aspect to another. For example, an operation defined in a class of the semantic model becomes a "logical service", after a few changes to its signature (its interface) in the logical aspect.

Foundations (cont.)

The Topology in intaglio: explanation of omissions

In search of efficiency

The Topology provides a schema of principles that organize the information and decisions concerning the Enterprise System. It has been elaborated over numerous experimentations where we have sought the most efficient organization with:

- Neither too many aspects, to avoid complications and a proliferation of models;
- Nor too few, leading to overlapping responsibilities and a confusion of roles.

This also holds for relations between aspects. Adding relations increases coupling and the likelihood of confusion and complication.

The Topology framework should therefore be read taking into account its omissions that is to say: the things that are not shown are just as important as those that are.

Logical / technical

So, for example, the Topology does not indicate a relation between the logical aspect and the technical aspect. This absence – consciously willed and affirmed – expresses the principle of logical independence, which states that the operational description of an IT system is independent of technology. This independence guarantees the sustainability of the logical model and so it becomes economically interesting to invest in the development and administration of extensive logical descriptions. This leads to the procedures and dispositions that we implement for the logical aspect. The logical aspect is, equally, unaffected by the geographical configuration of the system..

The technical aspect

The place of the technical aspect does not raise any difficulties: software is understood to be the expression of logical specifications in the chosen technology. However, the status of the technical aspect as a whole is questioned. Technical architecture addresses two types of element: specific software components and development rules. The first of these, specific software components, are included in the logical aspect. The second, development rules (potentially automated) lie on the path that leads from the logical aspect to the software aspect. As a consequence it could be theoretically acceptable to include the technical “pseudo-aspect” in the software aspect and as rules of transition between the logical and software aspects.

However, the technical aspect has been conserved in the Topology due to both the symbolic and political weight of technical architects in IT departments. In addition, it facilitates the management of multiple architectural scenarios on a practical level.

The adaptation of the Topology

It is very tempting to adapt the Topology, either to seemingly simplify it or to enrich it. However its internal coherence must be respected. One way of respecting the theoretical structure, while adapting it for use, is to regroup several aspects within a single deliverable (see p. 22).

Foundations (cont.)

The role of the logical aspect

An intermediate reality

Due to its role as an intermediary and the arbitrary nature of its representation, it is very difficult to define the position of the "logical" aspect.

The logical aspect lies between :

- The "external view": the business world of domain objects and actors of the business system.
- The IT system (technical choices, software components, deployment).

This intermediate level is inserted to facilitate structural decisions about the IT system.

The library metaphor (explained in detail on the following page) illustrates the objective of logical architecture and the type of questions treated by this aspect.

The place of the logical aspect

The logical aspect does not exist in isolation. It only has value as an intermediary between the external view (aspects: business core, organization and geography) and the internal view (IT system). It bridges the IT system with the reality. It is often expressed in terms of metaphors, for example:

- City planning or "Urbanization", where the IT system is compared to a city in need of organization.
- "Service", where the system is seen as a set of elementary responses to requests.

The reach

The logical aspect can lead to two types of intervention depending on whether the scope is system wide (global) or limited to the application (local).

Logical architecture The logical architecture is the primary description of the IT system. It is represented by an architectural graph that prefigures the future system and directs the evolution. The logical architecture is the referential description of all the information of the logical layer and should be provided to the developers.

Logical design Logical design also applies to applications. On the one hand, designers explore the logical architecture to identify services provided and used by the application. On the other hand, the developer, faced with new requirements, gives feedback that consolidates and enhances the logical architecture.

What is logical design? – The library metaphor

Let us consider a library. Optimizing the organization of its contents and facilitating its use would begin by a study of the library domain: the classification of subjects and themes, the analysis of relationships between subject areas, authors, books and so forth. This semantic analysis would render a very cluttered model in which everything would be related to everything else. The perception of this reality is essential even though it cannot be directly implemented either in the library structure or in the information system. The model provides a point of departure and an ideal target for the design process. It is produced by "**Semantic**" modeling.

The behavior of the library's clients would then be considered. What kinds of people visit the library? What are they looking for, what do they need? How do they act? Some come with precise goals, others come to browse. Some come with a very practical approach – to find everything concerning a precise topic – irrespective of the subject area or the media. Others come with a more playful approach – for the love of books, the pleasure of classified collections, etc. These issues are studied and modeled in the "**Pragmatic**" aspect.

The designer then needs to take into account the limits and possibilities imposed by the layout of the library: the division between public and administrative areas, the number of floors, the communications systems and so forth. These issues are addressed in the "**Geographic**" aspect.

The library also has "**technical**" characteristics. The library's equipment will have specifications that need to be respected. The choice of styles and materials determines the lengths and widths of shelves. There will be mechanical transport facilities and storage bays of book silos with optimal exploitation of space and so forth. These issues are addressed in the "**Technical**" aspect.

Taking into account:

- the semantic model as the ideal: real for the reader but unattainable as a reality
- the behavior and expectations of the users
- the technical possibilities and the imposed choices

The architect will sketch out a plan to organize the library's contents. As much of the semantic proliferation and pragmatic fluidity as possible will be expressed within the imposed technical limits. The consequences of design decisions that arise from these limits will be clearly identified. The plan takes its origin in the major principles and is then refined until it allows a librarian to know, without ambiguity, where any given book goes on the shelves.

That is logical architecture.

Designers constantly have to decide amongst options. For each decision they clearly identify what is lost and may try to lessen the consequences through additional options[4].

[4] For example, imagine that the classification of a book is ambiguous: either multiple copies of the book could be purchased and cataloged multiple times or place markers could be placed on the shelves directing the reader to the appropriate and alternative location.

Foundations (cont.)

The notion of "service"

The service-oriented architecture

The logical architect has the choice among several different styles and approaches to logical architecture. For example: functional architecture breaks down the system according to function; many architectural styles use systemic approaches; the vocabulary sometimes draws on city planning terminology.

The architecture of an information system is said to be "A Service Architecture" or "Service Oriented" if the system is structured around the elementary unit of logical services.

In a service-oriented architecture, no other visible components are smaller than the logical service: every information need, each action or transformation is met by a service.

The definition of a logical service

The importance of the notion of service is such that it needs to be defined from three angles: what it is, what it does, what it will become.

What is a service?

The service is the lowest level constituent of the logical architecture.

Hence "Service Architecture".

The service is the atom or elementary constituent of the logical construction of the system.

What does a service do?

A service is the elementary response of the system to a request for information, for an action or for a transformation.

This prohibits both direct data access and remote operations.

Where does a service come from? What does it become?

Logical services are the design units of logical architecture and are derived from the preceding models (semantic and pragmatic).

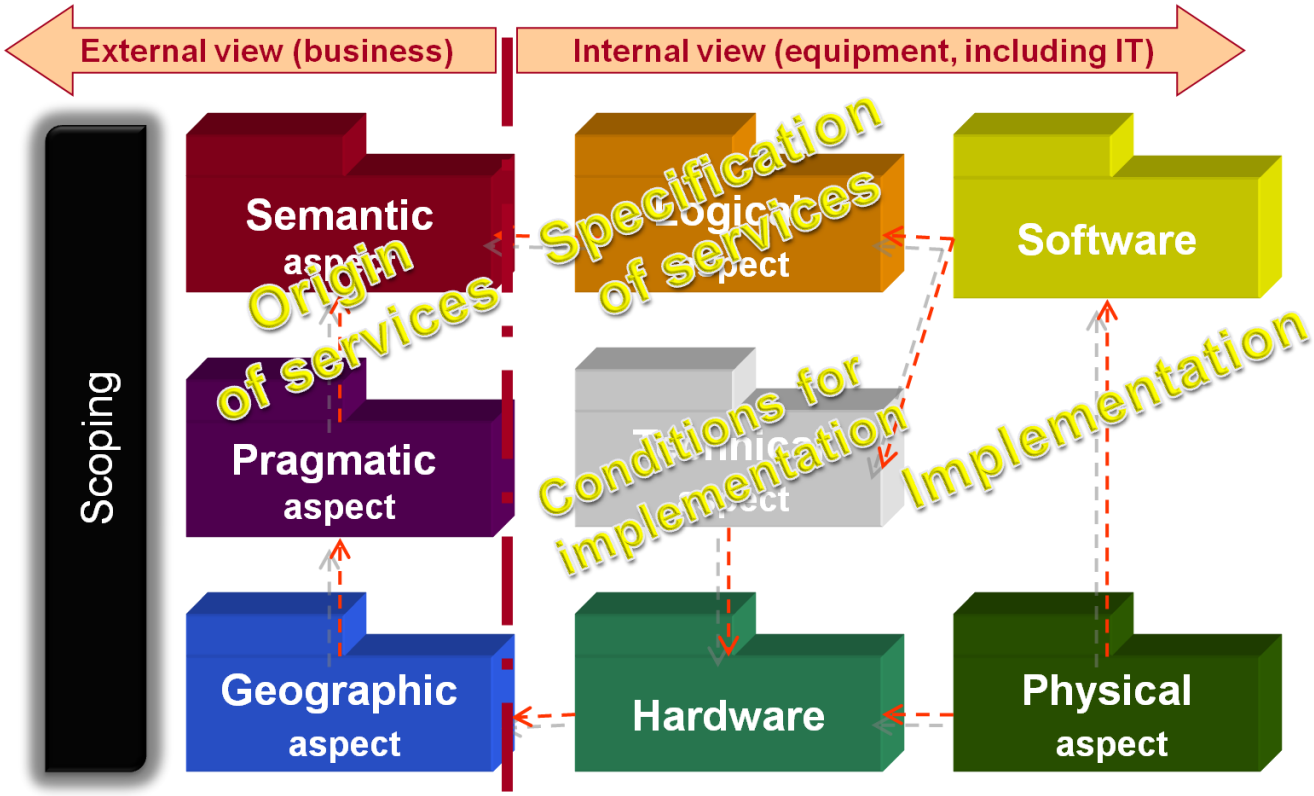
They are implemented as software components in the chosen technology. The software components are located on one or several machines and can be activated at execution time.

This life cycle of the logical service directs the transformation chain, as shown schematically on the following page.

Foundations (cont.)

The transformation chain

Figure PxM-02en_6. The lifecycle of a service, as regards to the topology aspects of the Enterprise System.



Related notions

The logical services (they are counted in thousands) are combined into logical groups with three levels of aggregation:

- "Logical machines" (represented by UML classes);
- "Logical workshops" (packages regrouping closely associated machines);
- "Logical factories" (packages corresponding to domains).

The nature of a logical service depends on its location (see following figure):

- Either in a "business logical machine" (BLM), at the core of the system (a BLM is derived from a semantic class);
- Or in an "organization logical machine" (OLM), in an intermediate layer.

The products section explains these notions in more detail.

The rationale

Service-oriented architecture permits increases in the quality of the system structure, even in the absence of object oriented technologies. It leads to the publication of software components as services, in a direct continuation of w*eb services* technologies. As such it promotes open systems.

The products

General terms

The reach

Enterprise Architecture distinguishes between:

- **Local** objectives: the specific solution addresses a specific business issue of an internal client (application, business process, business domain…).
- **Global** objectives**,** which involve the whole enterprise in the long-term.

Local objectives are characterized by the clear identification of requirements, a specific development and a short time frame. Exactly the opposite applies to the global objectives. Whether local or global, the aspects of the reality to be analyzed remain the same and the topology applies to both.

The model

A model is a relevant representation of reality.

A software engineering model is made up of:

- Graphical representations (models in UML);
- Information about, and details of, model elements (definitions, descriptions, quantitative information);
- Eventual justifications of the chosen models.

Every development requires a model, prior to development.

The model and the aspects

There are many options, situated between two extremes, when modeling aspects:

- Each model develops a single aspect;
- A model develops all the aspects necessary to reply to a requirement.

Certain connected aspects could be mixed in the same deliverable.

The choice depends on the scope of the study. As far as possible, it should be coherent across related projects.

The recommended solution is to use a standard set of models structured in packages with one package for each of the eight aspects of the topology.

The repository

A repository is a set of objects and information, shared among a community of actors.

Sharing a common repository is essential to Enterprise Architecture. Promoting and enabling reuse is part of the same current as the business component approach.

It is preferable to build one repository per aspect. "Business Knowledge Repository" for the semantic model encompassing the entire company, "Organization" for the pragmatic model, the architectures…

The products (cont.)

The semantic model: concentrate on fundamentals to reveal the stable core

The attitude

Modelers aiming to reveal the semantics of reality, approach it without any preconceived ideas. This attitude is not spontaneous or natural. It requires conscious and renewed efforts to set aside organizational and technical determinations or habits. The quality of the resulting semantic model comes from this ability to stand back from current practices and existing solutions.

Modelers must also strive to capture the essence of the fundamental core of the domain in abstraction of its apparent complexity.

The simplicity of the model then must be preserved against the general tendency to complicate things. One response is to demonstrate in what way the fundamental model reflects reality and how it can "unfold" into multiple forms taking into account the diversity of real situations.

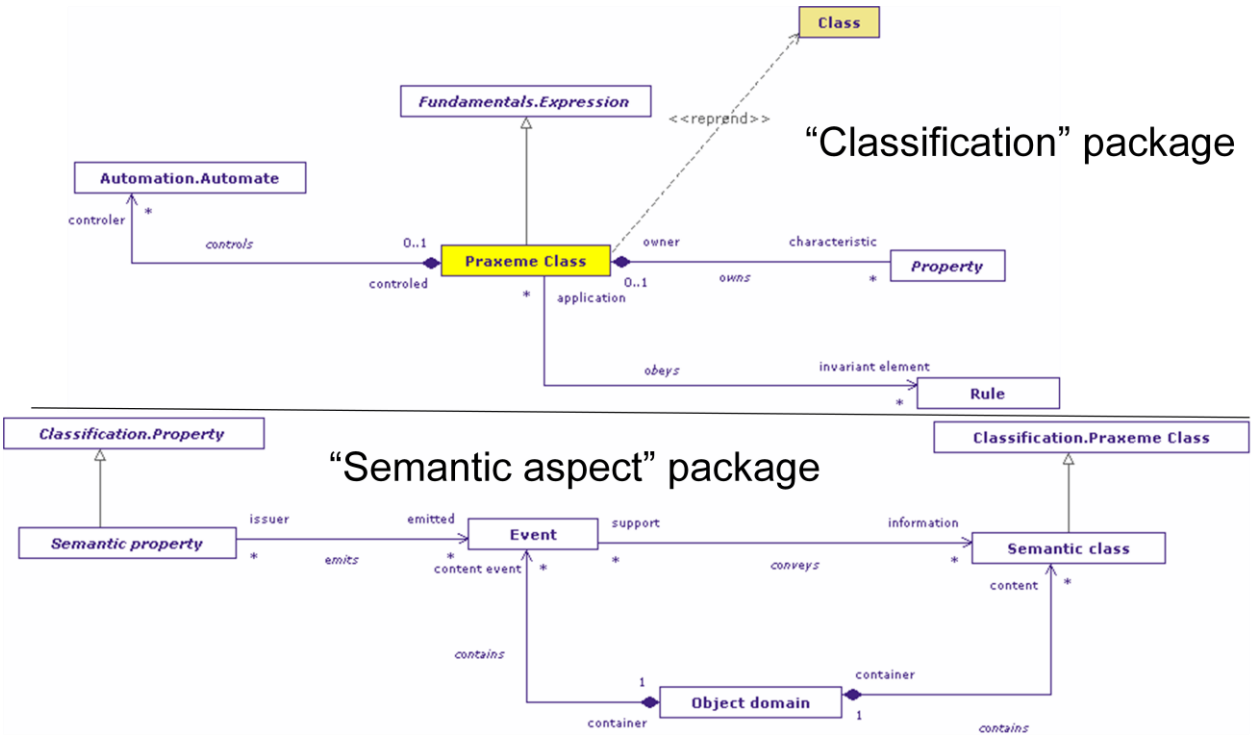
The terms

The language used by Praxeme for semantic modeling comes from the object orientated approach. The modeler aims to represent the mental universe of the actors, and hence looks for the most natural means of representation. Object notation, standardized as UML, is an adequate representation, as long as the models are sufficiently expressive.

UML terms such as classes and class properties will be used (information: attributes; activities: operations; linking: associations). The structural properties are implemented by relationships: inheritance or associations. The model is structured in object domains, represented by packages (see the chapter "The procedures" about the breakdown of object domains).

Since these are standard representation categories, the model below is just a simplified view of the UML metamodel.

Figure PxM-02en_7. Semantic modeling terms (a fragment of the Praxeme metamodel)



The products (cont.)

The pragmatic model: express the –local– need and guarantee –global– consistency

The attitude

The modeler's attitude depends on whether he/she is elaborating a Utilization view or an Organization view.

Utilization view In the first case, the functional approach will try to express the needs and preoccupations of the "user" and is guided by empathy for the user. This may lead to immobility and the perpetuation of current practices and user habits.

Organization view In the second case, the information flows and the evolution of business objects across the whole system are taken into account. The barriers of the first approach are removed and an overview of the whole system is sought. The design of processes may encourage further simplification.

The terms

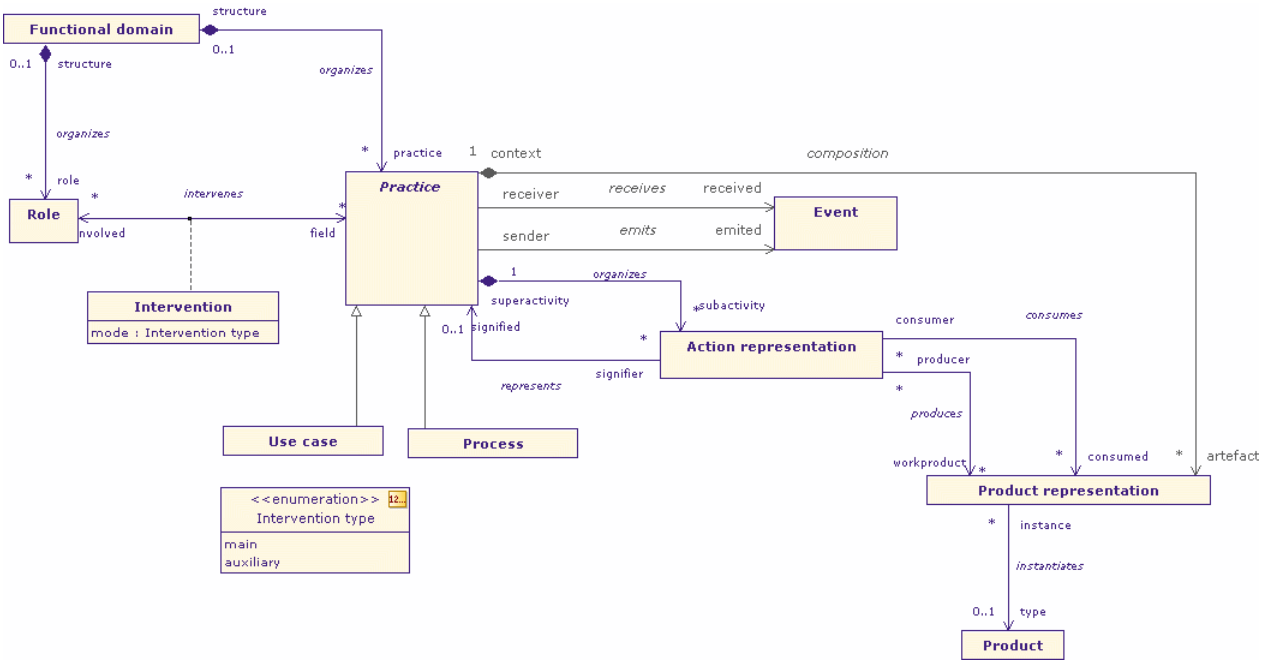
Pragmatic aspect models use the classical UML categories:

- Actors (types of actor).
- Activities: a notion with variable granularity covering high-level business processes down to the details of operations (or even lower), by way of use cases..
- Use cases: the elementary interaction between an actor and the system (assimilated to a "functional transaction": the execution of a use case involves only one actor and is indivisible unless it is aborted.

This definition of a use case restricts the use of UML. Use cases are identified according to the actor's objectives.

It may also be necessary to model organizational objects (dossiers, structures…). In this case, we use the same categories as the semantic model.

Figure PxM-02en_8. The terms for pragmatic modeling (extract of the Praxeme metamodel)



The products (cont.)

The pragmatic model: how to innovate processes

The dependence of pragmatics on semantics

According to the Topology the pragmatic aspect is dependent on the semantic aspect. This dependence may seem unorthodox and certainly contradicts most current Business Process Modeling practices which make use of specific tools and are completely autonomous. The process model is also almost always considered to be the first model to establish.

It is not surprising then that this subordination of pragmatics to semantics, gives rise to debate with BPM[5] analysts and needs to be justified. The justification is that this specificity has important consequences as it permits a truly innovative approach during the design of business process models.

Objects and Actions

The semantic aspect focuses on "business objects". The semantic model is refined and lean and all references to organization are expelled. There are no references to actors, their activities or organizational rules, which are described in the pragmatic aspect.

In a nutshell: The semantic model describes objects and the pragmatic model describes actions.

Our tenet is that to design actions well, the objects the actions effect must be clearly identified and well understood.

This common sense principle is represented in the System Topology precisely by the dependence of the pragmatic aspect on the semantic aspect. This allows the description of processes to refer to the "business objects", described in the semantic model. The business process design process is therefore inverted.

[5] BPM: *business process management*.

The products (cont.)

The pragmatic model: the terms

The role of the metamodel

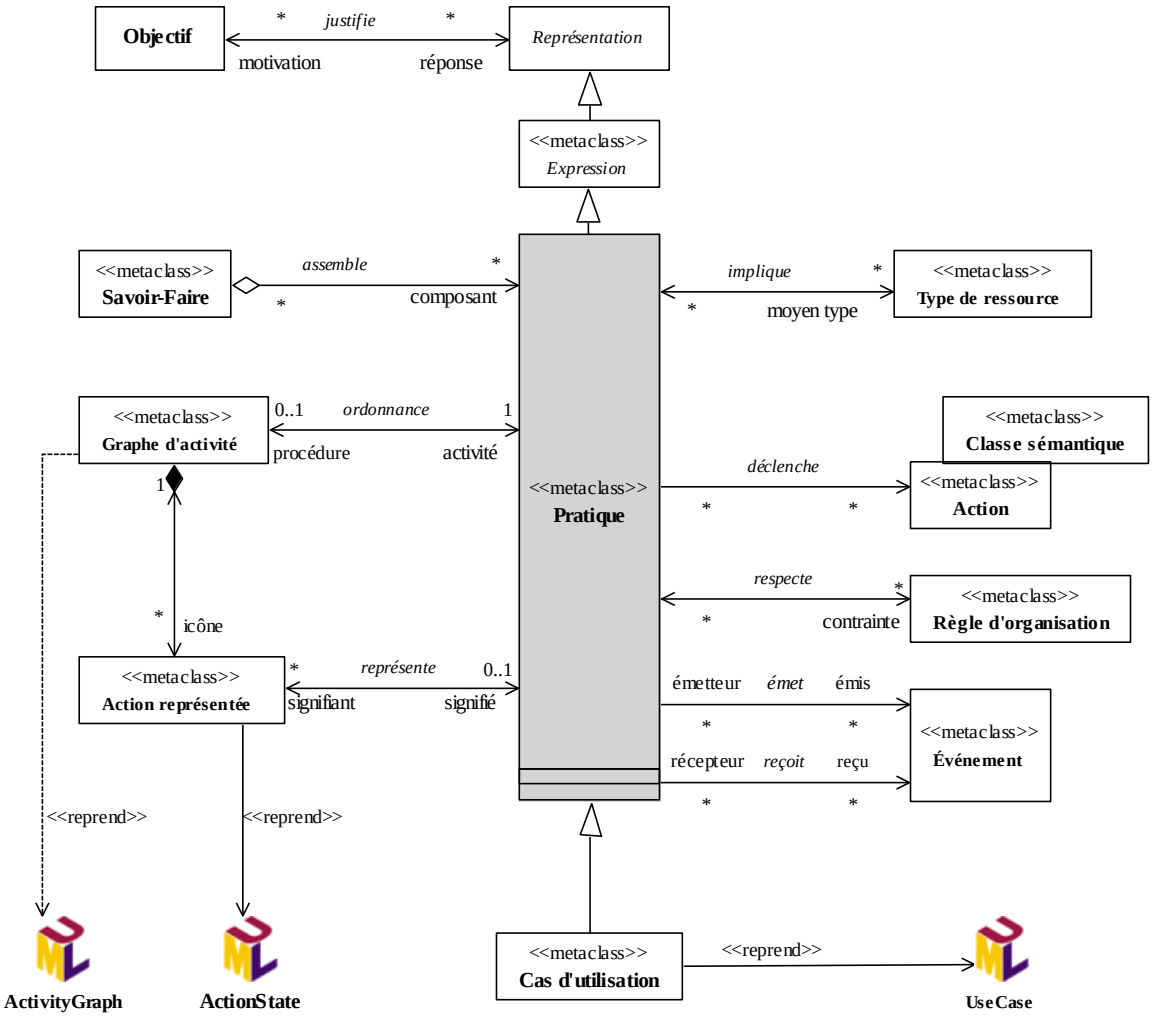
The metamodel, which is being gradually presented, aims to found the method on a precise terminology. This cannot be achieved by a simple glossary, which is always somewhat vague, but must resort to a detailed and argued model.

This endeavor is particularly edifying for the pragmatic aspect. It brings to light fundamental differences of perception and approach. Their consequences and the distance we take from usual process modeling practices are detailed in a specific guide (reference "PxM-20").

Activity and practice

The term "activity" is, on examination, very ambiguous: it applies both on a descriptive level (model) and on an operational level (reality). UML uses the term for all levels of processing and, what is more, uses it to evoke human intervention. This is why the Praxeme metamodel has discarded the term "Activity", adopting in its place the term "Practice" on the descriptive level and the term "Mission" for the implementation of a practice (see the "Production" model in Praxeme metamodel).

Figure PxM-02en_9. The terms of the pragmatic aspect (extract of the Praxeme metamodel)



The products (cont.)

The logical model: building sustainable structures

The Perspective

The goal of logical architecture and design is to structure the software in a way which will:

- Take into account future potential changes.
- Bear in mind strategic objectives.

Among the motivations are:

- The drive to 'urbanize' the information system in order to reduce redundancy and allow component reuse.
- To open up the system and allow easy integration with partner systems[6].
- IT agility or, at least, to easily adapt to reconfigured organizations.
- The possibility to multiply, implement or change the kinds of interfaces proposed at minimal cost…

The logical aspect is considered to be totally independent of the technology. This is detailed below.

What is at stake

Logical architecture gives a unified vision that goes beyond the abundant and diverse technical solutions.

There is, and there always will be, a gap between the representation of reality (concepts, objects, processes…), on the one hand, and software, on the other. It is possible, however, to take certain structural decisions about the software at the logical level. These decisions are not, or are only partially, subject to technical constraints and changes.

The logical architecture echoes the generic choices that structure the IT system as defined by software policy and enterprise strategy.

The objective of logical design

The logical architecture aims to elaborate the optimum structure for software, independent of final technical decisions.

Logical design provides sufficient detail for the definition of logical components.

[6] That is about interoperability and federations of systems.

The products (cont.)

The logical model: a language of its own

The terminology

Because of its intermediate position between the external view of the enterprise system (real life) and the IT system, the logical model has its own vocabulary. This vocabulary has to be able to render reality (semantic and pragmatic) in the system, whilst enabling decisions on how to structure it.

In Praxeme the terminology of logical modeling is derived from the notion of a logical service (see definition p. 11). It uses the metaphor of a factory, with its machines, workshops, etc. The choice of terms is not fundamental and other metaphors have been used for this aspect (*business component*, urbanization, city planning, blocks …). What are important are the topological constraints that these metaphors impose.

The terms

The logical service is the elementary constituent of the system on the logical level.

Thousands of services are assembled in different levels of logical aggregates.

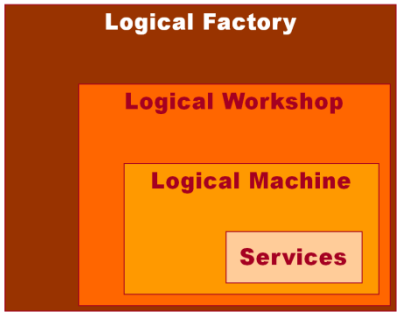


Figure PxM-02en_10. The interlocking of logical aggregates

The table below provides, along with definitions, some examples of rules.

The aggregates	Logical Machine	Logical Workshop	Logical Factory
Their definition	A coherent set of logical services.	A set of logical machines.	A set of logical workshops.
Demarcation criteria	The services apply to the same class (the same notion). "Unit" services are distinguished from "set" oriented services.	The roles of the logical machines regrouped within the same workshop are closely related.	Factories correspond clearly to object domains enhanced by cross-sectional mechanisms.
Their relationships	The machines in a workshop can work together through delegation.	Machines addressing the same sets of data (tables) are gathered within the same workshop.	Workshops using the same database (or collection of databases) are placed in the same factory.

The logical data model

The logical model not only contains the definition and structure of services, but also the logical data model. This data architecture can be very complicated, and may even be totally disconnected from the service architecture. This is the case when the service architecture is implemented on an existing database. Whatever the situation, whenever SOA is adopted, the service level will always be found to overlay and mask the data level.

The products (cont.)

The logical model: the stratification of the system

The categories of logical machines

Depending on their origin, we distinguish between different categories of logical machines:

- Machines that implement a semantic class are called "**Business Logical Machines**" (BLM) or "distributors".
- The machines expressing organizational decisions are called "Organization Logical Machines" (OLM) or "orchestrators". They are delimited using criteria which could be the use case, the type of actor, the process, etc.
- Other machines arise from logical or technical considerations. They are called "**Transversal Machines"** or "**utilities**". They provide general facilities such as event management or encoding.

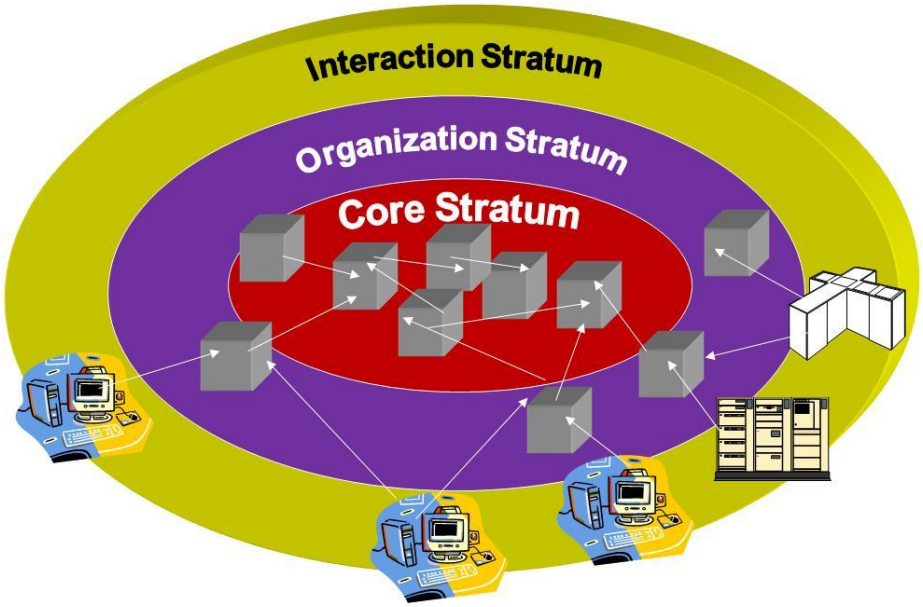
The logical architecture fixes the rules of identification, structure and design which apply to all the elements of the logical aspect. Concerns of loose coupling and system governance motivate these rules.

Stratification

According to the rules of architecture, the information system is made up of three circles:

1. The core is made up of "business logical machines". It is called the "Foundation" or "Core" stratum.
2. The intermediate circle isolates the organizational decisions in "organization logical machines". That is the "Organization" (or "Operation" or "Activity") stratum.
3. An outer circle is the "Interaction" stratum, which provides the system entry points for workstations or other interfaces (possibly to other systems).

Figure PxM-02en_11. The stratification of the information system, according to the rules of logical architecture



The products (cont.)

The elaboration of service architecture

The terms

As well as the terms exposed so far, logical design distinguishes between:

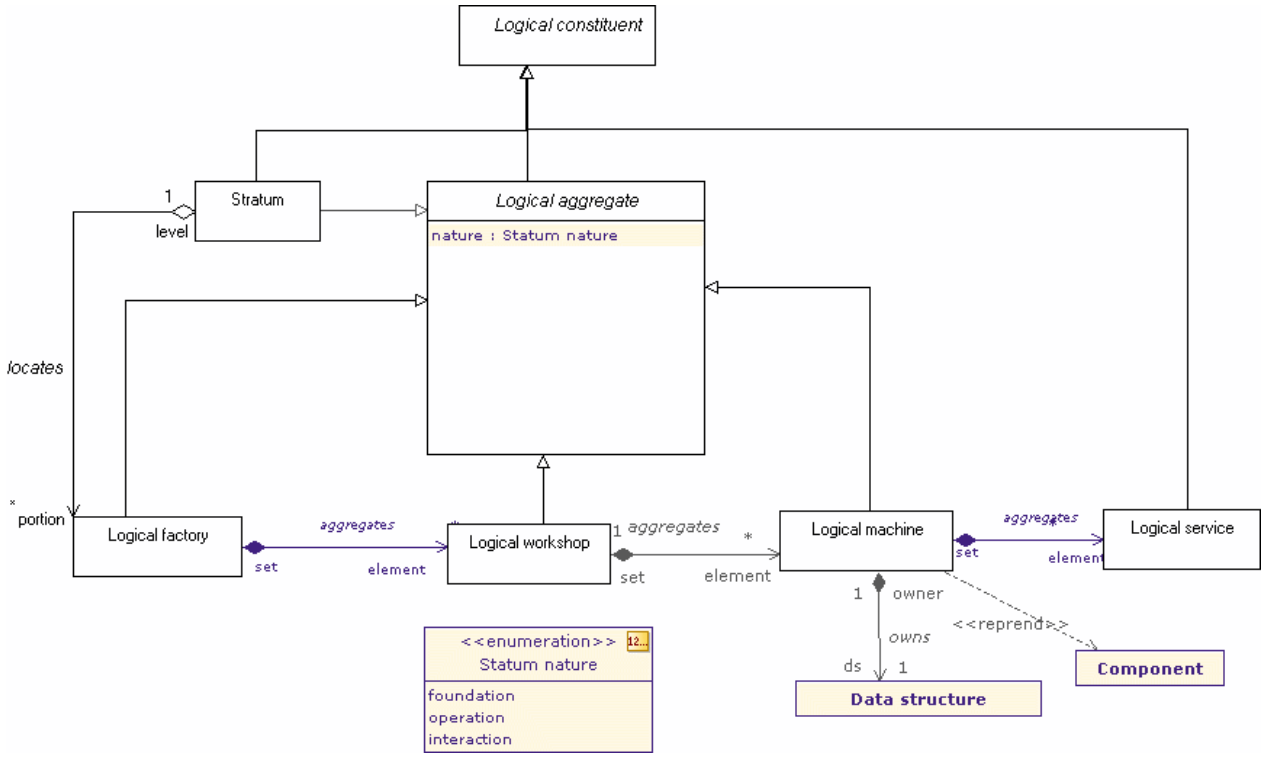
- The **individual (unitary) logical machines**, whose services process only one instance (only one object or occurrence).
- The **set orientated logical machines**, regrouping services such as instantiation, queries and statistics.

Logical architecture translates semantic class behavior as two types of logical machines. The properties of the class (including the associations) are distributed on these machines depending on their reach.

The equivalence

The metamodel below maps higher level modeling elements onto the logical elements.

Figure PxM-02en_12. Synopsis of the Praxeme metamodel for the logical aspect



Comments

The class diagram above only presents part of the metamodel. It shows the main terms used to describe the logical architecture, with a "service" style. Most of these terms are linked to the metaclasses of higher-level aspects, making derivation possible.

The details of the metamodel express topological constraints that oblige the logical architect and the designer to loosen the coupling of the system.

The metamodel presents other considerations for the logical aspect, regarding in particular the data level.

The products (cont.)

The system covered by the models and viewpoints

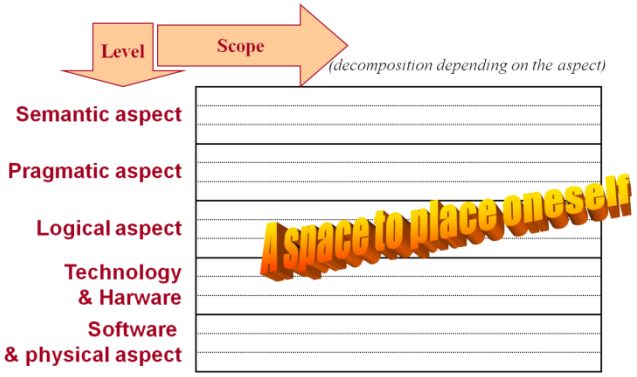
The principle

The Enterprise System Topology provides the structural principles which Praxeme refers to when defining models and deliverables. It offers the advantage of defining models as exact and complete representations, limited to a clearly identified aspect. It also presents the dependencies between models. These dependencies indicate how information is linked from one model to the others. In this way, responsibilities at every level of modeling are clearly identified.

The deliverables defined by the process

The definition of deliverables lies at the junction between the dimension "Product" and the dimension "Process".

Figure PxM-02en_13. The workspace of definition of deliverables



The deliverables are defined in function of the process. They are defined for the ends of phases which have the advantage of being clearly identifiable. They meet the needs of communication and decision. Situated in the "Product" dimension, their definition conforms to a rule: at the end of the process, the added content of all the deliverables must cover the entire scope of the study. Praxeme divides up the field by projecting the list of aspects over scope[7].

Aspect *versus* view

The notion of "view" has been used for several years now and by several methods. A view is not the same thing as an aspect. Praxeme uses both notions.

With the term "aspect", we designate a characteristic of the system object (the scope of the study), characteristic which is independent of the point of view and linked to the nature itself of the observed reality.

However starting from a typology of actors (strategists, process owners, organizers, IT people, etc.), we define views, which can refer to elements drawn from one or more aspects.

For example, the "Organization View" and the "Utilization View" are both extracted from the complete pragmatic model. The first provides a global vision of the organization and processes. It interests the organization designer. The second provides insights into the perspective of different types of actors in the organization. The "functional view" provides another example, examined on the next page.

In conclusion the notion of aspect cannot be mixed with the notion of view.

[7] The document with reference "PxM-03" deals with the process and related topics.

The products (cont.)

The deliverables

The functional view

The notion of functional specifications is coherent because it is used to communicate with a certain category of actors. Functional specifications can contain, pell-mell, a small fragment of semantics, operational requirements, some screen (UI) specifications (software elements) and all this sprinkled with some organization rules.

The functional specifications

By definition, functional specifications express requirements in terms of *functions*. Going straight to essentials the use case dossier is perfectly adapted to the expression of requirements concerning user activity. It does not describe "screens" (UIs) which, being subject to design decisions, are left to the mock-up and the related dossier.

The opposition between generalities and details (general specifications *versus* detailed specifications) is not a structural element of Praxeme. Every aspect requires a formal and complete model. The level of detail and completeness of each artifact can be adjusted to the context of the project and immediate needs. A model is not considered to be definitive until it describes the complete scope of the study, for the designated perimeter and an aspect of the topology.

On the other hand the notion of reach allows the introduction of artifacts which tend to make the 'urbanization' and system-level decisions sustainable.

The mock-ups

To ensure good communications in projects, deliverables can be accompanied from the initial phases with a mock-up. Mock-ups, rather than text and models, provide a directly appreciable and realistic image of a solution.

If it is an IT solution, the mock-up is a software component, to be placed in the software aspect. Nothing prevents it, however, from being included in an artifact like the functional specifications, before development or even technical decisions.

Dossiers and folios

The artifacts include:

- **The project dossiers**: they record decisions as a project progresses and they are only of value during the project's lifetime.
- **The global dossiers**: their reach is the entire system; they are a reference for the whole enterprise. They can change but they are under the control of a central authority.
- **The folios**: they may potentially be produced within a project, but their format is designed to make them common and sharable elements. As such, they acquire a visibility and a lifespan that surpass the project. The folio life-cycle therefore takes into account its use by multiple actors on multiple projects.

The description of deliverables

The method describes the artifacts by commented forms. They can also be the object of UML profiles, installed in modeling tools.

The process

Implementing the topology

Content of this section

This section "Process" does not prescribe a complete process and does not give all of Praxeme's indications for this dimension. Here, only major guidelines are presented, those which help organize the work and processes as far as the transformation of the Enterprise System is concerned. The document "PxM-03" presents this approach more fully. It describes the conditions under which the method Praxeme can be coupled with another method or with a reference process. This is easy with Praxeme, because Praxeme highlights the "Product" and "Procedures" dimensions, whereas other methods deal mainly with the "Process".

Consequences on the work phases

The separation of aspects leads to the isolation of different elements of a solution, which evolve at different speeds.

This leads to following advantages:

- Facilitate the definition of work steps (see p. 25).
- Give more visibility to decisions, by serializing.
- Impose a strict organization of information in the documentary base of the Product (the Enterprise System or a domain or an application).
- Isolate the stable core (semantic and logical), preserving it against organizational changes and technological innovations.
- Manage skills thanks to specialized disciplines (see p. 30).

Separating aspects leads to the introduction of long-term governance and quality requirements, such as adaptability (or agility), robustness and technology independence, into the structure of a solution. This provides a solid basis for 'urbanizing' the IT system.

The gains

The topology is a first reply to transformation needs and it aligns the IT System with the strategy of the enterprise:

- It helps design the information system in a modular way, so that it can adapt to technology changes, and it can be opened up to partners and to multiple interfaces.
- The topology separates design elements which evolve at varying timescales and with different logics. It encourages, hence, structuring the software in a way that allows it to absorb change more easily.

The "services" orientation

The "service-oriented architecture" (SOA) enforces the benefits of separating the aspects. The building or acquisition of services (more than simple classes) imposes supplementary design rules. The design targets a middleware integrating the components, of which some can be published, found on the market or be exchanged with partners.

The process (cont.)

Modeling: between analysis and design

Analysis *versus* design

The dichotomy "analysis/design" is a classical issue of software engineering (and in general of engineering science).

The tendencies of the past years (especially the generalization of iterative cycles) have blurred this simple opposition.

Classic software development methodologies define the activities (of analysis or design) and the phases (time segment, breakdown of work unit). They follow the waterfall lifecycle. The names of the phases indicate the activities. Now we must separate these two notions, since a given phase can unite both types of activities. In an iterative development process, one iteration often contains analysis, as well as design activities.

Even though the same tools can be used for analysis and design, analysis and design need two radically different attitudes.

Analysis **To analyze is to observe.** The term evokes the breakdown into the smallest of elements; an attention to detail. When analyzing, the posture of the modeler is characterized by:

- being passive (he does not take any initiative; he is satisfied by depicting his observations);
- minuteness and thoroughness;
- potentially tracking pain points in order to feed critical reflection (analysis of the current situation, audit).

Design **To design is to invent.** The designer's posture is very different. He is expected to imagine one or more solutions to a problem or for a stated need. The characteristics of this activity are:

- taking into account the requirements (which have been formulated and analyzed beforehand) or covering the need;
- inventiveness, based on technical know-how of possible solutions;
- economic evaluation of proposed solutions…

The process (cont.)

Pre-modeling: facilitates the transition

From natural language to the first model

The real function of processes is to organize collaboration between people with different expectations, different cultures and different incentives. We insist a lot on modeling because this 'representation' technique is the only instrument of a reliable and trustworthy communication, from one end to the other of the transformation chain.. The Enterprise System Topology segments this chain, but one has to start somewhere: at the place where the model does not exist.

The true starting point is the knowledge the actors have of the system. This knowledge is seldom formal or formalized. At best it is written down, either in pre-existing notes, or by the notes the modeler writes in his search of knowledge.

There is an abyss between the first formulation and the first model! The pre-modeling stakes out a first bridge between user language and the first model.

Pre-modeling

Pre-modeling is a step which should not be neglected. It has an essential role in the communication, especially with senior management and "business" management.

The products of pre-modeling are not distributed – at least, not directly – on the aspects of the Enterprise System Topology. This is not a reason to exclude them. They have their place in the general framework, before the modeling. The collection must be very flexible and allow different perceptions of the system. One observes that a same notion can have different names in the same organization or that the same term can have variable meanings, depending on the roles or people's backgrounds.

The terms and tools

The pre-modeling uses simple tools, allowing the manipulation of self-evident notions:

- First of all, the glossary (dictionary) is a means of collecting terms and spontaneous user definitions. To be more efficient, the models show how these intuitive terms are reused and restored. This instrument of communication should become a more elaborated form of thesaurus, showing semantic relationships.
- The management of requirements is also part of the pre-modeling. Its tools answer the same needs of traceability towards the models. The development process should schedule a review phase of requirements, before the design.
- The strategic and operational goals are another category to be taken into account in the pre-modeling. One should ensure that each investment, translated at one moment or another in the model, is attached to the objectives tree.

These simple notions of pre-modeling – terms, requirements, objectives – can give rise to quite a few concerns in practice…

The process (cont.)

The target levels

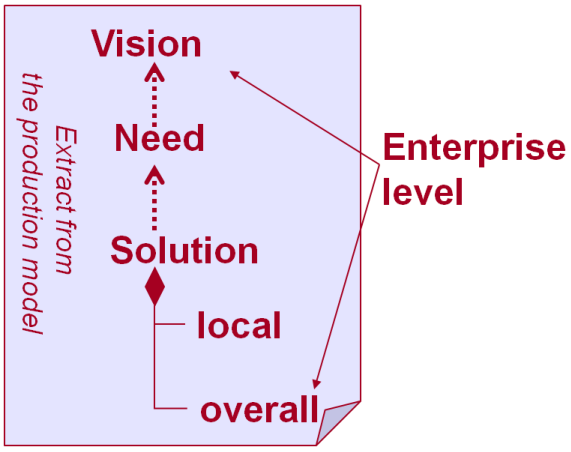
The elements which polarize the activities

In addition to the need and the local solution (specific response to a need), enterprise architecture adds two considerable categories of objectives:

- the "**System**" itself, that is the global solution, with a managed structure, change and content management;
- the "**vision**" or analysis and anticipated needs (watching outside the organization: strategy of the organization, the market, possibilities and future preoccupations).

In other terms, the same movement of Enterprise Architecture reconciles the overall design of the system on the one hand and prospective analysis of the market on the other.

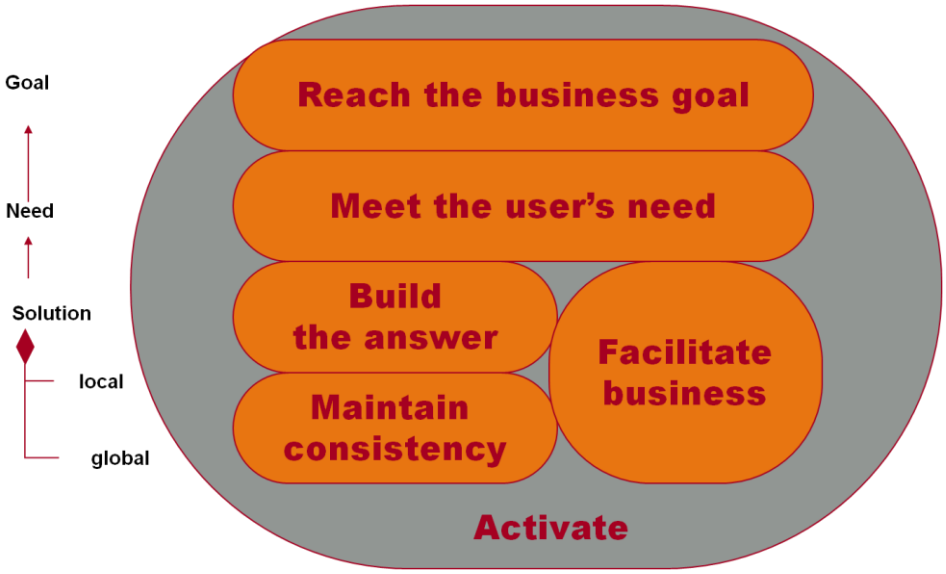
Figure PxM-02en_14. The target levels



The consequences

The transformation processes include the necessary activities to serve the organization on all these different target levels and to contribute to operational, organizational and strategic adjustments. The figure below illustrates this (see "PxM-03" for details).

Figure PxM-02en_15. The macro-activities depending on target levels



The process (cont.)

The approach: possible actions in parallel

The basics of the approach

The diagram on page 22 shows the articulation of the elements describing the system. The transformation process respects the logical dependence between the products. One can deduct from this diagram:

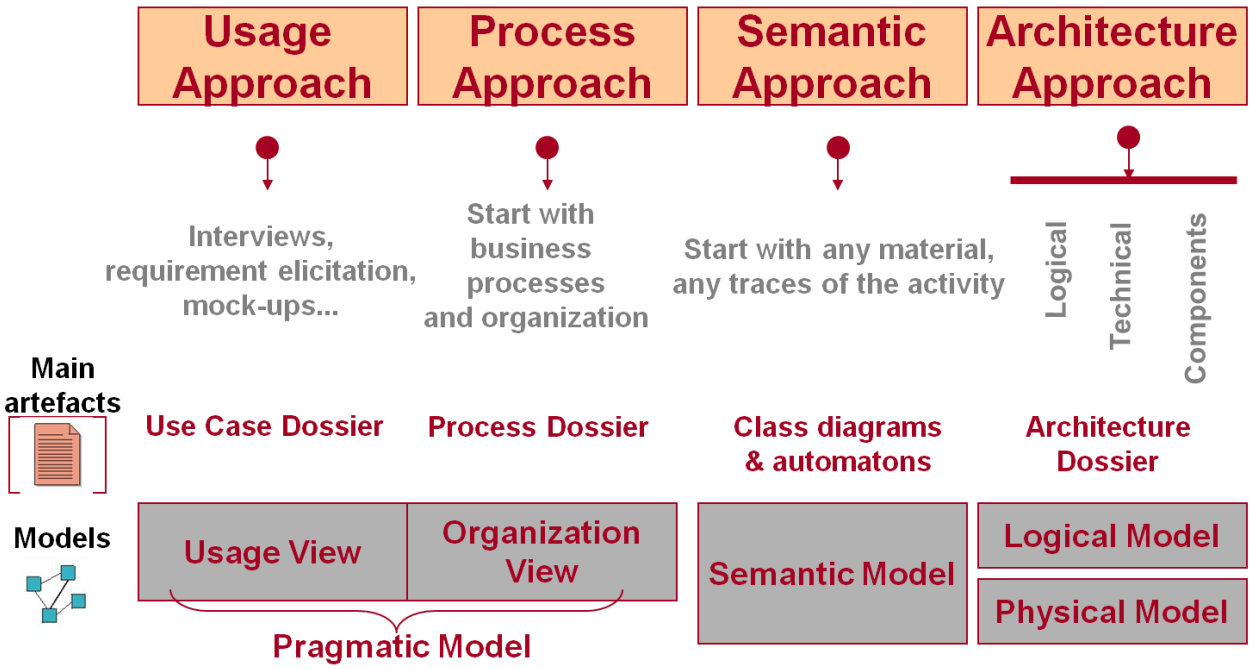
- The possibility of drawing a parallel between the modeling of the semantic and the pragmatic aspects.
- The source of service design (the logical design is inspired from the semantic and pragmatic models).
- The constraints and objectives are taken into account in the logical architecture.
- The need to have synchronization points between the aspects.

Parallelism

Without forbidding parallelism, the dependency relationships between models require some precautions. The diagram on page 22 shows a dependence between the usage view (the use case folders) and the semantic model. This dependence is precisely the result of the value given to the semantics in Praxeme and our preoccupation with basing the documentation on the substrate of the Business Reference Model.

This leads to a practical consequence: work on the semantic and pragmatic models can be led in parallel providing that regular synchronization checkpoints are held. These checkpoints give the opportunity to verify the convergence of resulting products of the two aspects. The success of such an approach relies on **coordination meetings**.

Figure PxM-02en_16. The design approaches



The process (cont.)

The approach: work on architecture

The constraints and objectives

The first step of this approach is the architectural analysis, where constraints and goals, especially high level ones, are collected.

This work allows the alignment of the information system with business interests and future anticipated changes. This subject will also enter into the almost certain arbitration between different design scenarios or between divergent local interests. Furthermore, architectural analysis will prepare long-term budget discussions.

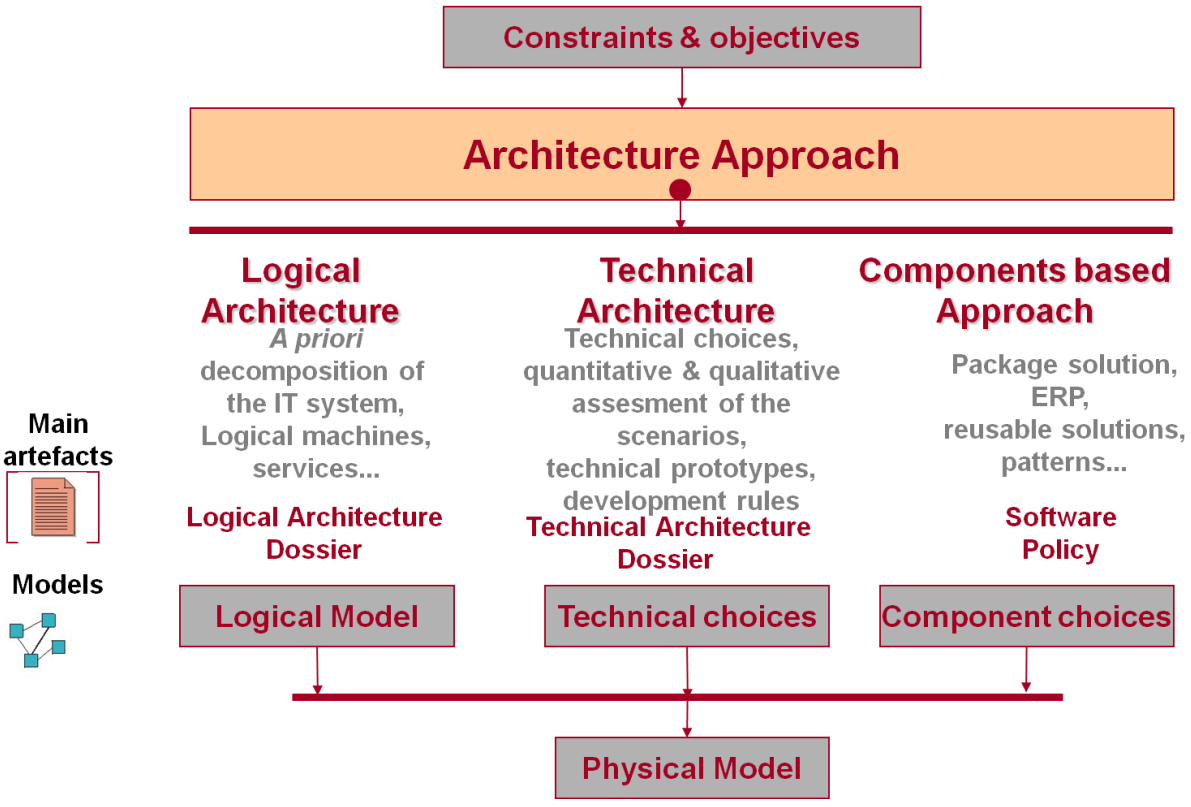
Three sub-approaches

Architecture is the design discipline which fixes overall choices, in other words it addresses the entire system. It can be broken down into three sub-approaches:

- The logical architecture which will be supported by enterprise architecture arguments (see pp. 31 et seq.).
- The technical architecture (cf. the folder type FRM-50).
- The component approach for those parts of the system which can be covered by off-the-shelf software.

These sub-approaches can be led in parallel with some precautions, especially for the negotiation between logical and technical aspects (see p. 41).

Figure PxM-02en_17. Architecture approach



The process (cont.)

The activities of the overall scope

The characteristics

The overall dynamics (p. 27) show the necessary activities from the target levels onwards. The will to structure the System introduces new activities, needed to ensure the objectives and requirements of the overall scope. These particular activities are different from usual development activities by their goal, scope and delivery date.

The goal The objectives of these activities are part of the organization's strategy.

The scope These activities take into account the comprehensive approach to the Enterprise System (as opposed to the classical development process and to the project mode).

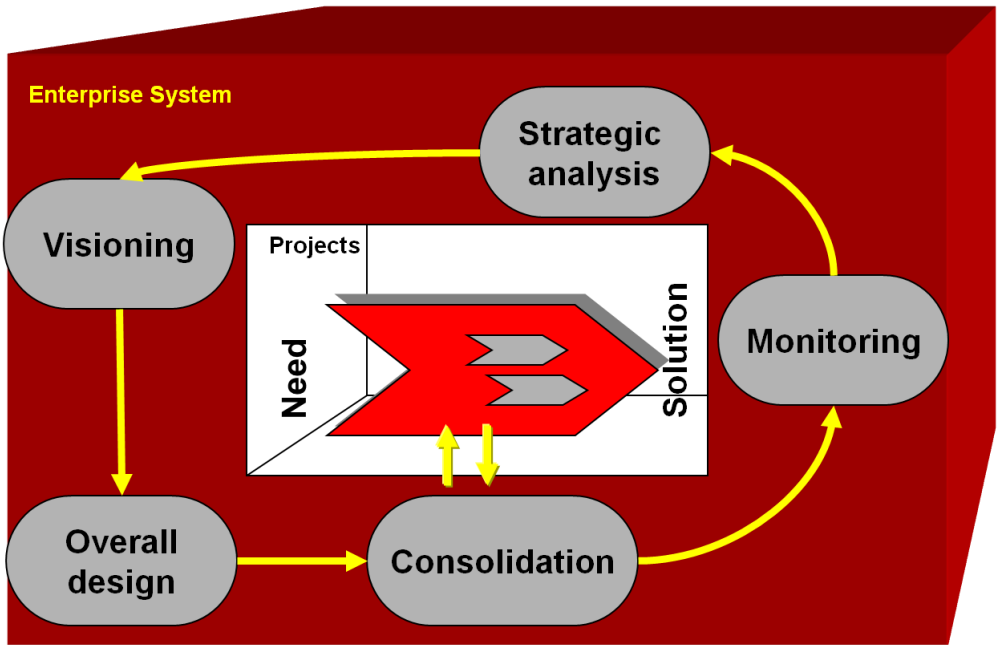
The timeline These activities are deployed on a long-term basis.

Their organization

We enter into a new dynamic coordinating project activities and system activities.

The diagram below shows 'system' activities and their position as related to the normal 'project' development activities.

Figure PxM-02en_18. The activities of the overall scope interacting with the projects



The process (cont.)

Enterprise Architecture and IS urbanization

Definitions

The Enterprise Transformation Manifesto approaches the notion of Enterprise Architecture from the standpoint of the enterprise, as a whole[8]. The manifesto positions this discipline as the necessary link between the multifarious specialties that contribute to analyze and transform the enterprise, starting with the strategy and ending with deployment and monitoring. Therefore, Enterprise Architecture is an important factor that comes into play when transforming the enterprise. Its mission is to put together every specialized approach, so as to embrace all aspects of the Enterprise System, to reconcile the various points of view and to stimulate synergy and creativity. The definition below is an extract of the Enterprise Transformation Manifesto:

> Enterprise Architecture is the discipline that analyzes the strategy of the enterprise and determines the main decisions for transforming the Enterprise System.

The basic principle of Enterprise architecture is to adopt an attitude which gives priority to the overall and long-term requirements rather than local short-term ones.

IT urbanization

Inside Enterprise Architecture, urbanization of IT system[9] is the design discipline which focuses on the information content of the Enterprise System.

Over the years, software development covering specific local requirements, added on top of existing applications, has built an increasingly complex IT system. Various functional application units, built with different technologies, rapidly become obsolete when new technologies emerge and may be subject to all kinds of change due to the speed of business evolution and change.

To master this complexity and to take advantage of new technological possibilities, it becomes necessary to reorganize the system with a more economic architecture. This architecture must take into account new technologies. First of all, it must embrace a comprehensive vision of the enterprise, its constraints and objectives as mentioned beforehand.

> IT urbanization (or IT city planning) is a design discipline which aims
> to structure the information system and
> align it with the enterprise's strategy and business.

As for Enterprise Architecture, the horizon and scope of IT urbanization cover the whole system and the long-term vision and roadmap.

The terms of IT urbanization

Essentially, the procedures and methods used for IT urbanization are those of mapping applications (for the 'as is' situation analysis) and defining the *logical* architecture for the design. Today, UML supports this discipline and transforms it from the inside by giving it a precise notation. As a result, there remains no excuse for the *ivory tower* syndrome that all too often characterizes this craft. The modeling technique assures **continuous coherence between**

[8] See http://www.enterprisetransformationmanifesto.org.

[9] Also known as "IT city planning".

overall models of architecture and detailed design models. This coherence leads to a revolutionary change of roles and responsibilities.

Logical architecture

The definition of logical architecture results from the combination of both notions:

- architecture, which implies that the scope is the entire system – whatever aspect is considered;
- the logical aspect, as defined through the Enterprise System Topology.

> Logical Architecture is a design discipline that develops the optimum description of the software system. This description is relatively independent of technical choices and takes into account general constraints from business, as well as IT legacy and orientations.

A logical architecture is also the by-product of this activity. It is the description of a given system, at a logical level.

The target architecture

The founding act of IT urbanization consists in publishing a target architecture for the enterprise ('to be'). It simply presents a logical architecture graph showing the "ideal city map", an overall description of the IT system one would like to build. This target is even better when the architect can free himself from the current situation, simplify the system and redesign it with criteria ensuring its stability. With such an aspirational and daring view, the architect will face skepticism and fatalism. It is, therefore, important to explain the intended uses of this "ideal" description. Very rare are the situations where it is possible to implement the system in strict accordance with the target. Other uses of the logical description include:

- specification of the flows and services to be developed;
- integration policy;
- interoperability policy;
- simplification of the existing systems or compensation of identified flaws;
- overhaul of the system or its improvement.

It is not the place, here, to elaborate on these topics.

The roadmap

The target architecture can only be achieved after several long years, on the condition that it is supported by the senior management, the only authority capable of driving it with continuous (super-) vision and necessary effort. The success of enterprise architecture also relies on operational skill and leadership.

The roadmap for transformation contains a succession of stable and predefined states of the system, which will lead, year after year, from the 'as-is' situation to the 'to-be' target. Each state is described by an intermediate architecture graph, which allocates the annual investments as closely as possible to the target. In this way, the investment reaps more benefits.

More on processes and organization is to be found in "PxM-04".

Modeling Procedures & Methods

Portraying before doing

Definition

Procedures and methods are instructions for the work to do. Contrary to processes, they are situated at an individual level: they are the guidelines for a precise work step. A procedure can be related to a discipline (for example, internal design) but it can also be used by several disciplines (for example, a documentation procedure). Sometimes, it is supported by a tool.

The major procedures

The following major procedures can be incorporated into Praxeme.

Figure PxM-02en_19. The list of procedures in Praxeme

Procedure	Aspect concerned	Definition (objective of the procedure)
Semantic Modeling	Semantic aspect	Representation of the analyzed reality, true to itself, that is abstracted from organizational and technical contingencies.
Requirement-gathering	Pragmatic aspect (Use-case View)	Formulating functional and operational requirements by using the technique of Use Cases .
Proof of exhaustiveness of Use Cases	Pragmatic aspect (Use-case Views)	Verifying the exhaustiveness when identifying Use Cases.
Process Design	Pragmatic (Organization View)	Design of activity flows, not from existing processes, but from the life cycles of business entities.
Logical architecture	Logical aspect	Design of the overall structure of the software system, with a relative independence of technological choices.
IS Urbanization (IS city planning)	Logical aspect and Software	Elaboration of a target to channel long-term developments pushing them towards an ideal structure. Elaboration of a roadmap leading to this target with regards to investments.
Technical Architecture	Hardware and Technologies	Design of the technological infrastructure: technical choices, technical components, development rules and tools associated with technical choices.
Elaboration of scenarios	All aspects	Determining scenarios for the design of a solution. (Several solutions can cover a requirement. The solutions should be assessed so as to retain the "best" one.)
Internal Design	Software	Design of a software element from a specification, using the object approach and design rules imposed by general objectives (enterprise architecture, reuse, etc.).
Test Design	All aspects	Identification of test cases.
Simulation of the models	All aspects	Verification of the quality of a model by implementing it and running it in an ad hoc technical environment.

Modeling Procedures & Methods (cont.)

Semantic modeling: going straight to essentials in order to isolate the stable core

The definition

In the semantic aspect, the representation of the System aims to depict:

- the notions, concepts and business entities of the scope,
- their relationships,
- the attached rules.

Organizational and technical contingences are removed from this representation. Its value lies in the abstraction and associated simplicity. The semantic model contains the essentials only; this makes it simple and stable.

The Stakes

The simplicity of this description frees the imagination and allows the designer, later on, to have a larger range of choices relating to the organization, the logistics and the technology.

The effort of abstraction helps us get back to basics and, in so doing, frees us from various existing practices and paves the way for a simplification of the System.

The construction of the semantic reference model (the core business knowledge) equally allows us to capitalize on the intellectual knowledge available, the ways of action and even on the solution itself. It is a knowledge management tool.

The Attitude

The modeler, who has the task of constructing the semantics, approaches the business reality, without prejudice. Despite appearances, this attitude is not spontaneous. It demands a particular effort on the part of the modeler, an effort which needs to be constantly renewed, in order to set aside organizational and technical circumstances. The quality of the semantic model resides in its ability to break away from current practices and the existing solution.

Furthermore, instead of troubling itself with all the apparent complexity of the domain, the model needs to capture the vital things and to isolate the fundamental knowledge.

The semantic modeling may reveal its inventiveness.

The modeler has to defend the simplicity of his/her model against the general tendency to complicate things. One argument for this is to show how the essential model restores the reality and how it can “unfold” and multiply itself to take into account the diversity of concrete situations.

Modeling Procedures & Methods (cont.)

Semantic modeling: some precepts

Identifying the main classes

The principal classes play an essential role in the transformation of the Enterprise System. Indeed, they provide the starting point to build the stable core around which the system will crystallize.

Criteria The strongest criterion is this one: at least one Use Case uses it as the central business entity. This criterion is also part of the procedure of verifying the completeness of the Use Case inventory.

The first justification of a central business entity class is its importance in the user's language. This is then a pure semantic criterion, difficult to formulate.

Here are some other formal criteria to designate a principal class:

- Cardinalities of associations from the class (it is linked to several satellite classes with multiple cardinalities; maximum cardinality = 1, on its side).
- Aggregation (the class assembles several other classes).
- The presence of a state machine for the class (with significant states from the user's point of view).

Encapsulating the rules

The semantic model, even at the highest level of abstraction, has to obey all the requirements of a true model: it is complete, detailed and valid. It describes in full the notions of the scope of analysis. Most importantly, the model must contain the description of the information and operations attached to the semantics of the notions and business entities. The rules related to the organization are excluded from the model (they belong to the pragmatic aspect); but the business rules must imperatively be encapsulated. The modeler designs the semantic classes like micro-machines, they are responsible for their internal state and capable of providing all the services underlying their semantics.

Respecting the change of objects

Business entities evolve, transform themselves and have a different behavior depending on their state, at least the principal ones. Their life-cycles are an important part of the model. A solution or IS (Information System), based on state machines, reveals itself to be simpler and much more robust.

Documenting the operations

The encapsulation of business rules and the design of life-cycles of business entities make operations emerge with semantic value. The model describes these operations completely: signature (name, input/output parameters with their types), contract (pre- and post-conditions), internal description (algorithms if necessary).

Decomposing the system

The System has to be broken down on the semantic level. This breakdown uses the business entity criterion (as opposed to the function); it breaks the IS down into business object domains.

Modeling Procedures & Methods (cont.)

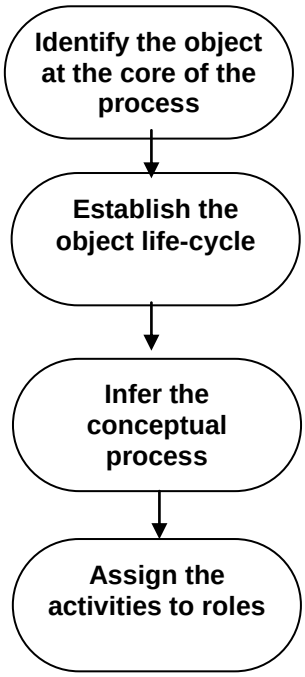
Business Process Modeling (BPM)

How to innovate

We will not talk here about the classical functional method of BPM. This method is too close to existing practices and stops us from taking enough distance to really innovate. Here we rapidly outline Praxeme's innovative approach to Process Modeling [10].

We start from the Enterprise System Topology with the link from the pragmatic aspect toward the semantic aspect. We consider that every important process is defined by a goal which produces or transforms an essential business entity. Thus, the best process is the one which accompanies the life-cycle of the business entity, the most truly and with the least complications possible. So, to model the business process, we do not start from the activity – which is most probably already wrought with organizational rules – but we observe the business entity's "life" and inner logic. In doing so, the approach is turned around and gives us another point of view. This is why this method helps us to thoroughly rethink the activities, the business processes and even the organization.

The procedure



The business entity is present in the semantic model. The designer of the business process must eliminate all secondary artificial objects and only keep those business central entities (to give an example: the act of ordering rather than the invoice).

The semantic modeler should have associated a state machine to the semantic class representing the business objects. If not, it is the moment to complete the semantic model!

The state machine is represented by a diagram showing the authorized transitions between states. This is then turned inside out, like a glove, to become an activity diagram. This activity diagram, without swim lanes, expresses the designed business process. It helps to isolate the necessary activities.

Only at the last step do we add the notion of the actor and his responsibilities .This is why the modeler has more freedom to redistribute the activities. He can, if possible, propose a new distribution of roles, activities and perhaps a new organization around the process.

Further reading

Article "The Six Fallacies of Business Process Improvement" (available on the website) .

[10] For further details, please refer to the Guide of the pragmatic aspect (reference: "PxM-20").

Modeling Procedures & Methods (cont.)

Analyzing requirements via the Use Cases

What is a Use Case?

Praxeme's definition: **A Use Case is an elementary working situation**.

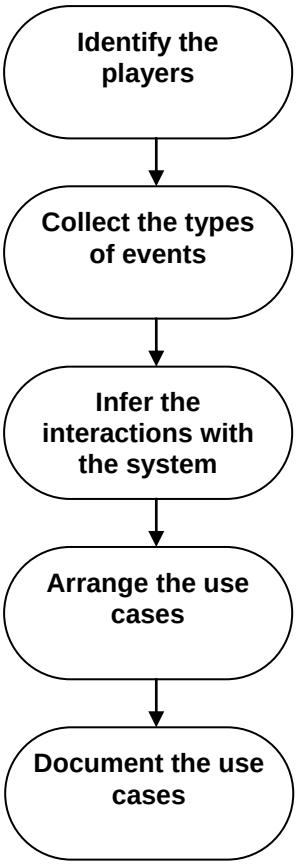
It implies an interaction between an actor and the system.

The definition given by the standard UML allows different interpretations. On the ground, modelers find it difficult to agree on the level of granularity of this notion. Praxeme's definition helps to overcome this difficulty. It also provides the modelers with an identification criterion with the following practical consequences:

- Only a single user participates in the run of a use case (one occurrence)[11]. This run-through of a use-case is uninterruptible (excepting run-exceptions, error cases).
- The use case covers a precise user intention towards the system.

The procedure

NB: this procedure is creative and aims to go beyond the simple gathering of requirements. It focuses especially on the actor's motivations.



In this approach, the modeler proceeds from the exterior to the interior. Outside the system, he first identifies the external actors and then the internal ones. For the first he can only state things. For the second, he can possibly propose a new distribution of tasks via the roles (design decision).

The best criterion for identifying use cases is the actors' motivation. "What do they expect from the system?" "What events bring them to interact with the system?"

Knowing the circumstances of the interaction, one can better analyze the actor's expectations, taking care to focus attention on the actor's motivation or intention (instead of on how the system reacts). "What result does the actor expect from the system?"

The Use Cases represent and summarize the elementary interactions between actors and the system. By "elementary", we mean the running of a use case with the intervention of only one actor.

By rapidly analyzing the content of each use case, the modeler can factorizes those which could be satellite use cases. For this task, he draws the diagram of use cases. He retakes and verifies the analysis of user access rights.

Further reading

See the worksheet FRM-25 and its annotations FRM-25c.

[11] When an activity requires the action of several players or when it has to proceed through several steps, then it is not a use case but a process.

Modeling Procedures & Methods (cont.)

Logical architecture

Summary of the method

The terminology is defined p. 19. It defines and conditions the approach of the logical architecture.

Logical Machines The business logical machines (BLM) are derived from the classes of the semantic model. The modeling effort is concentrated on the semantics (the core of the business, exempt from organizational and technical choices). This has several advantages:

- It formalizes the core business knowledge.
- It simplifies the model, making it more compact and more "natural".
- It helps discover the **services with high business value**.

The Organization logical machines (OLM) are derived from the pragmatic model (organizational), mainly from the use cases. They serve as relays of the core system, comprised of the Business Logical Machines**.**

The Stratification The logical aspect is structured into three strata[12] and obeys topological constraints:

- **"Core" stratum** (or Foundation): it isolates the services that translate the semantics, the core business knowledge. It is stable, highly reusable and protected by the other strata. The services of this stratum are called "internal".
- **"Organization" stratum** (or "Operation" or "Activity"): The OLM isolate organizational choices, which include access rights and visibility conventions with partners. This stratum offers "external" services. One can have several of these services as relays of the same internal service: for example, one for internal use by the organization's staff, another one for external use by partners or clients to control or restrict access rights.
- **Interaction stratum** (or "Periphery"): the work-stations (depending on technical choices for the GUI) and other interfaces giving access to the system. It includes all sorts of communication channels and interactions with other systems.

Logical Services The machines provide services. The services can call each other and conform to the cooperation principle of the object approach.

This position of the term **service** is conform to the software engineering tradition (metaphor for the client-server, TACT method, service-orientated architecture). It is not incompatible with technical categories (such as, notably, the Web Services solution).

Derivation rules Near mechanical derivation rules allow the passage from the upper models (semantic and pragmatic models) to the service-oriented architecture (SOA). They are described in the Guide of the logical aspect (reference: "PxM-40"). The logical architecture hosts the business core and preserves, as far as possible, its structure. The transformation of the activity model (use case, business process, organizational structure…) depends on several decisions, of which some are conditioned by the logical-technical negotiation (see below).

[12] The term "stratum" is favored rather than "layer", since the latter connotes technical architecture.

Modeling Procedures & Methods (cont.)

Identifying logical services

Introduction

In a service-oriented architecture, the atom – the smallest grain of the system – is the logical service. The logical architecture provides the framework arranging thousands of services. This begs the question: "How can we identify a service?". The table below proposes some ideas for identifying the services.

One can use two complementary approaches:

1. **Top down**: the logical components are derived from the semantic and pragmatic models.
2. **Bottom up**: the logical model is completed by rereading the functional details expected (use cases or activities of business processes).

Identification rules

Each service has a logical identifier, unique in the system (it should be meaningful).

Recommendations

Figure PxM-02en_20. Deriving logical services from external models

Origin	Starting terms	Deriving	Comments
Semantics	Operations of a semantic class	Basic or internal service	Usually 1 to 1.
	Attributes of a semantic class	Basic services (type accessor); "read" service of a set machine	Depends on the technical architecture: either an attribute (if object technology), or a programmed service. Take into consideration: write protection and principle of uniform referent.
	Semantics Class	Control service of the invariant class Set services	Most of the semantic classes lead to set services. These propose creation, storage, and administration (counting, statistics) services Exception: the singleton.
	Association	Navigation services	Should be placed according to the multiplicities and orientation of the usage relationships between the machines.
	Object domain	Logical Factory	Important tool to reduce the coupling and for Enterprise Architecture management.
Pragmatic	Pragmatic Class (actors, organizational objects)	"Business" type Logical Machines "Organization" Machine	Two possible options: ▪ Same treatment as above. ▪ Or some form of parameters in an "Organization Machine" (tables of rights, for example).
	Use Case	Can be a starting point for designing OLM	Analyze the Use Case scenarios to detect user requirements for the system (these can be interrogations or action requests). They are covered by the services .

Modeling Procedures & Methods (cont.)

The documentation of logical services

Information to collect

The logical services are documented by:

- a 'request of service' form (FRM-44), standardizing communication between different IT departments;
- a description note, more complete (FRM-45).

These forms can be integrated in the modeling tool, using a UML profile.

Definition of a service

The definition of a logical service should let an external requester understand the objective of the service and its behavior. It contains the following information:

- The name of the service (see preceding page);
- The signature of a service with the parameters, their nature and their input/output function;
- The objective of the service (a sentence);
- The run conditions of the service;
- The signals, which the service can emit.

The run conditions cover the prerequisites and guarantees of the service (pre- and post-conditions), as well as the temporal mode: synchronous or asynchronous.

Description of a service

It may be necessary for complex services to provide an algorithm. In all cases, the specification of the service must be clear and one should be able to list the services called by the documented service, as well as the manipulated business entities. Indeed, this information allows the verification of the architecture and assesses the coupling value of the service.

Other information for architecture is necessary: the usage frequency of the service, the exchange volume (logical flow) generated by the service.

Representation of a service

The logical service is represented in Praxeme as an operation on the class stereotyped "Machine". These operations can themselves be stereotyped "Service".

About the temporal mode

The behavior of the service – synchronous or asynchronous – is essential architecture information. It is sometimes preferable to regroup, in the logical workshops, the services of the same nature (rule of homogeneity).

In order to quantify the logical architecture, one can try to measure the runtime of each service. At this stage, this may not appear realistic. The designer can, however, evaluate the volumes (number of instructions at the logical level, number of other services called, different weighting by distance…). This information can be aggregated and can provide a starting point to reflect on the choice of physical (technical) architecture.

Modeling Procedures & Methods (cont.)

Technical architecture

Logical/technical negotiation

The methodology recommends a step before the elaboration of both the logical and technical architectures. This step, the logical/technical negotiation, is a precaution which does not go against the principle of logical independence – which states that the system or solution should be first designed on a logical plan, independent of technical choices.

The negotiation helps to:

1. fix the terms of definition of the logical architecture (for us: the services) and ensures that these can be easily transposed into software components;
2. share the transversal issues between the logical design and the technical design;
3. determine the units to be handled by subsequent phases (delivery, deployment…).

Transversal Issues Examples: communication inside the system; error handling; transaction management; business rule engine; translation of state machines; event management; etc.

The principle of sharing transversal issues is as follows: when the technical architect can propose an off-the-shelf solution which covers the requirements, one opts for this solution (this is always cheaper); in the other case, the logical architect is responsible for the design.

Units The logical architecture organizes the services in different aggregation levels : logical machines, logical workshops, logical factories.

Which level of aggregation should be chosen for the deployment or the maintenance?

- **The deployment unit**: this is the unit which cannot be broken down into different pieces to be hosted on different machines. The usual candidate is the logical workshop (linked to a data structure).
- **The delivery unit**: work unit for maintenance intervention. It can be thought of as a service. On the other hand, wider non-regression tests, at least at the machine level, must be put into place.

Covered aspects

The technical architecture, is a skilful design discipline covering the entire IT system for the following aspects:

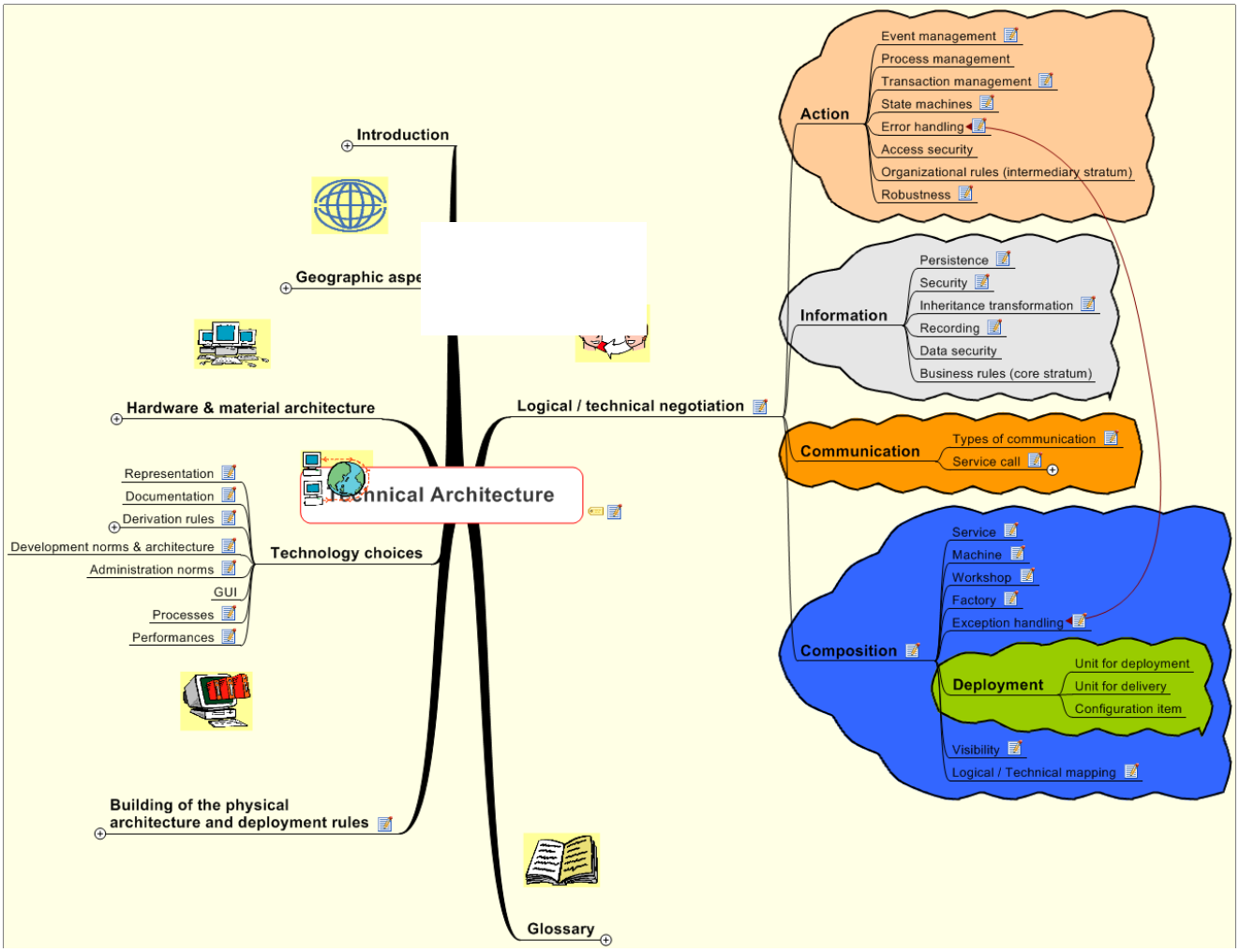
1. **The material aspect or hardware**: set of all machines and other electronic components, infrastructure components (with their capacity, cost, behavior…).
2. **The technical aspect**, strictly speaking or the technology itself: basic software, technical components, as well as the development rules that are implied.
3. **The physical aspect**: the technical architect can only establish the localization rules for the software components to be hosted on the hardware (he cannot describe the complete physical architecture, because for this, he would need to know all the software components).

The attitude

Technical architecture happens in two phases:

1. analysis, which draws up a balance-sheet of the existing IS, collects the constraints and the objectives (architectural analysis).
2. design, which explores technological possibilities, examines the options and combines them into architectural scenarios.

Figure PxM-02en_21. Picture showing the themes of the technical architecture



========== The appendices are missing in this partial translation ==========

To be informed of further publications, please register via the page “Stay informed” on the website.