



A comprehensive Introduction to

# PRAXEME

A Methodology for the Design of Sustainable  
Enterprise Systems

This version has been prepared and is presented by :  
Bernard Hauzeur, independent ICT Architect  
([bh@artofe.biz](mailto:bh@artofe.biz))

under license:  **creative  
commons**

1



## Front matter

- Acknowledgements
- License (legal)
- Pointers
- Objectives

under license:  creative commons

2

### □ Version History:

- July 2009 – compiled from sources cited further + original slides
- Sept 2009 – updated to reflect suggested amendments to English terminology, enhanced intro to modelling activities, and more dynamic presentation

### □ Etymology:

**Praxis** (Greek)

Action, activities that change the surrounding context

**Séméion** (Greek)

Sense, meaning, signification

Hence the subtitles:

- “Le sens de l'action” [fr]
- Meaning in Action [en]



## Acknowledgements

- This version has been compiled from
    - materials from the Praxeme institute ([www.praxeme.org](http://www.praxeme.org))
    - the presentation of the methodology available from <http://www.sustainablearchitecture.com/materials> and <http://www.orchestranetworks.com/fr/soa/>
    - a compilation by Fabien Villard @inno.com (<http://www.praxeme.org/DocumentsEnAnglais/IntroducingPraxeme.Inno.2009.04.17-01.pdf>)
- ... plus original contributions

The methodology has been originally designed by:

- Dominique vauquier, methodologist, [www.praxeme.org](http://www.praxeme.org)
- Pierre Bonnet ~ Orchestra Networks

under license:  creative commons

3





This document is published under a Creative Commons license, with the following attributes

**Praxeme and Praxeme Institute** are registered trademarks

( <http://creativecommons.org/licenses/by-sa/2.0/fr/deed.en> )



## License

### You are free:

-  **to Share** — to copy, distribute and transmit the work
-  **to Remix** — to adapt the work



### Under the following conditions:

-  **Attribution** — You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
-  **Share Alike** — If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.

under license:  **creative commons**

4

### NOTE on the template:

In compliance with the clause of the license requiring to avoid any suggestion that the PRAXEME Institute endorses works by third parties, a personal logo has been added in the top left corner. If you decide to reuse any part of these works, you can easily edit the logo via the main slides master (view > Slides Master). You will also edit the very first slide. There's no other place elsewhere with a specific indication of name or company.



## Pointers

### ■ Sites

- <http://www.praxeme.org>
- <http://www.orchestranetworks.com/fr/soa/>
- <http://www.sustainableitarchitecture.com/>
- <http://www.mdmalliancegroup.com/>

### ■ Books

- "Le système d'information durable: la refonte progressive du SI avec SOA " by P Bonnet, JM Detavernier, D Vauquier
- "Sustainable Architecture: the progressive way of overhauling information systems with SOA" *idem*

under license:  creative commons

5



## Objectives

- Understand what makes PRAXEME different, and why + how this methodology does obtain outstanding results

## Audience

- IT specialists with some methodological background, exposed to modelling, and UML in particular
- *... eager to discover at least one method that can help design Service Oriented systems other than by the empirical grouping of data accessors, activities, and bits of functionality into services tied together with BPM glue...*



## Positioning PRAXEME



## The Context

- You know the mess...

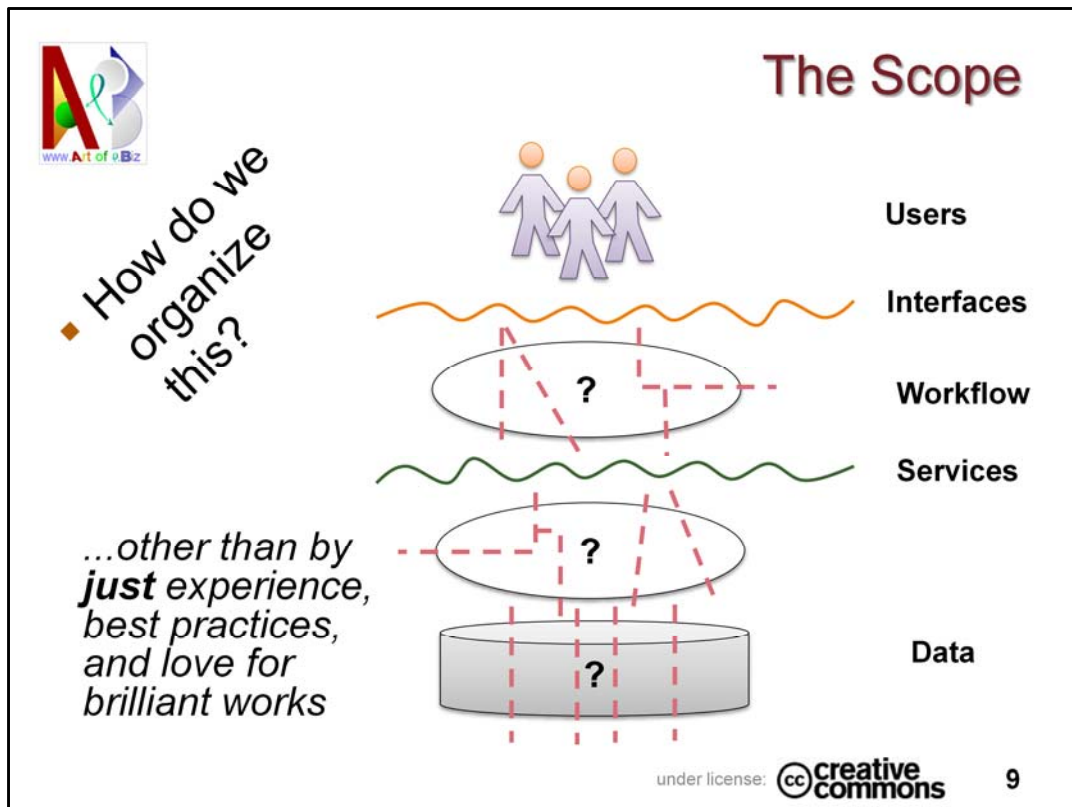
at minimum, if we keep trying...  
why not with **Praxeme**?

under license:  creative commons

8

- Lack of reference methods
  - Merise, SDM/S, Axial, SA/SD, SSADM, Jackson, ... are not used anymore
  - UP, RUP, XP, Scrum are delivery methods
  - EA Frameworks, TOGAF, Zachman, insist on processes and forget models (this is changing)
  - Best practices are not methods: ITIL, CMMI, CoBit, SOMA, ...
- Recurrent Issues
  - Architecture in silos
  - Redundancy at all levels
  - Communication between actors
  - Find a good path to target
  - Knowledge management and documentation
  - Organizational dysfunctions
- The current low-level of modelling activities do:
  - Reduce sharing of ideas
  - Increase the silos syndrome
  - Increase the duplication of work
  - Shrink opportunities to innovate
- **The current abuse of Business Process Modelling as the central modelling technique can only cast into stone the existing splits and conflicts between functional domains**
  - PRAXEME will demonstrate that Pragmatic models can't possibly yield Semantic artefacts. Consequence: with only Business Process Modelling you are doomed to miss something
- Other disciplines provide tools and good bricks but no plans
  - Object-oriented Patterns
  - Business Process Management (BPM), object oriented design, SOA (Service Oriented Architecture)...
  - MDA (Model Driven Architecture) specified by OMG

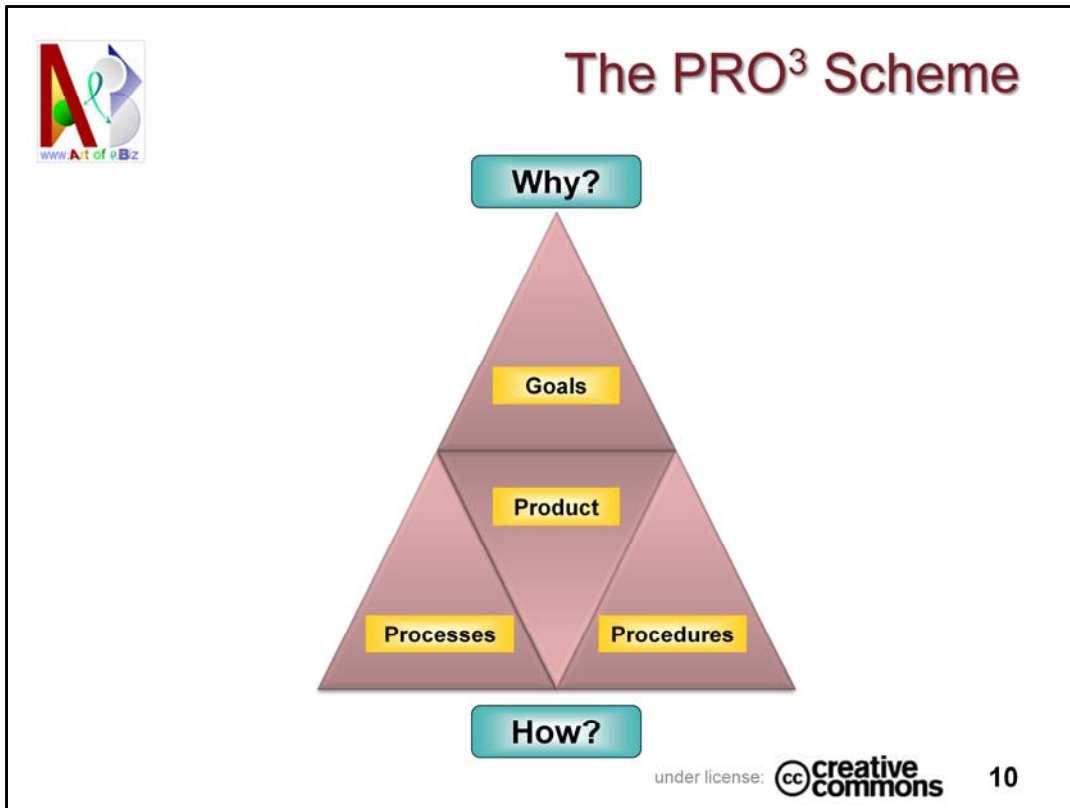




Architecture is the art of splitting. Splitting a problem into bits organized in a structure, so that implementation can proceed smoothly and fulfil requirements. Many viewpoints can be used: functions, data, time sequence, processes, states, services, ...

In the field of SOA, notably, the most known methods are good collections of best practices and heuristic rules. They can probably lead to the definition of sane systems if not strong. However, they always leave the sentiment that services (and the possible web service specifications that ensue) are casted out of 'thin air'. In other words, it's very hard—but by self conviction—to make any demonstration that this set of services is the one that derives from the business itself, and any other definition would just be good guesses.

That is were PRAXEME makes a significant contribution. Numerous services in a service oriented architecture will be derived—like calculated—by an application of PRAXEME procedures, instead of being best guesses! Moreover, the distinction becomes very clear between the stable core, and the versatile periphery that will follow (or even precede) any reorganisation of the company operations.



PRAHEME is a methodology. What does a methodology contain?

The **Pro<sup>3</sup> scheme** defines 3 dimensions which an Enterprise Methodology must describe:

- **Products**: this is the result of the method application. This dimension is completely dedicated to what we want to realize and why, absolutely not by how we will manage to realize it.
- **Processes**: this is part of how we will realize the thing. The Process dimension how the works are distributed to different people, what are their respective roles, their interactions, the work steps and activities to carry on, and in what sequence. This is a collective view on the way to perform the works.
- **Procedures** (or techniques): this part of the "how", describing the techniques to be used to conduct the various activities by each person. For instance, how to derive meta-data for the database from the UML class diagrams. This is an individual view into the techniques beside each activity inside the processes.


Side example:

process: invoicing = collect order details + draft invoice + approve + account + send,

procedures: query database, add & multiply by quantities, calculate tax, exchange a document electronically



## The main contribution of PRAXEME is about PROcedures

PRODUCTS	PROCESSS	PROCEDURES
	ITIL RUP ISO9000 TOGAF Zachman 6σ CMMI ...	<b>PRAXEME</b>

*comparison based on main focus*

under license:  creative commons

11

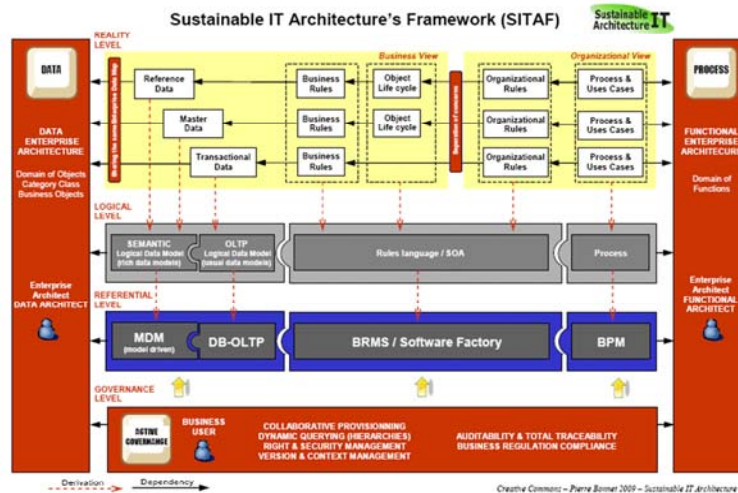
All methodologies share the purpose of producing artefacts (products in the form of solutions, documentation, plans, means of productivity, software, component assemblies) with a higher quality and faster. PRAXEME achieves this with an emphasis on procedures, and therefore may complement all others that focus on the process dimension.



# Is there an architecture matching PRAXEME ?

- yes: SITAF

Sustainable IT Architecture Framework



under license: creative commons 12

SITAF is quite original, and we just want to cite this architecture at this stage. Indeed, understanding is impossible without first understanding the PRAXEME methodology. Yet, it is important to assess that PRAXEME is not an architecture, neither an architecture style, but a methodology. This means too that other architecture frameworks can be used along with PRAXEME. The architecture of choice is actually injected into PRAXEME at the point of *Logical Modelling*, and onwards.

**SITAF** relies on four levels of IS restructuring (Reality, Logical, Referential and Governance) and two transversal Enterprise Architecture (Data and Process).

**Reality level:** This level describes the reality of the company; its data and its processes without IT concerns. The usual separation of concerns is used to clearly distinguish the business view from the organizational view.

**Logical level:** Once the reality is described, the logical representation must be managed. The objective is to attain more IT representations but not yet reliant on any specific software tools.

**Referential level:** With help from MDM, BRMS and BPM, hard-coded implementations are no longer required. Indeed, a set of derivation rules allows for setting up business referentials from logical models directly.

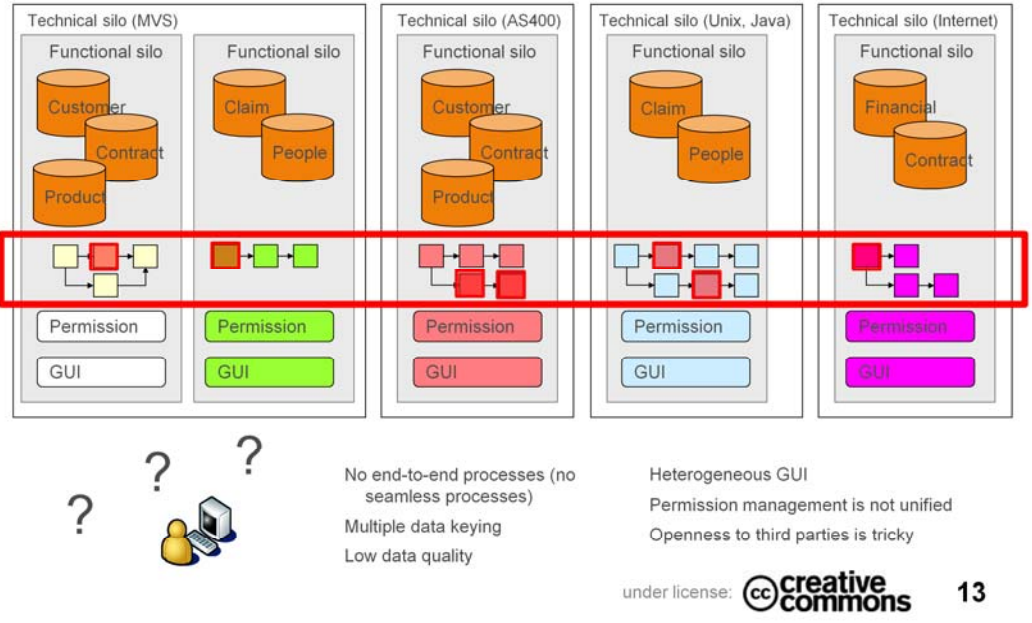
**Governance level:** Obviously, this level is not modelling nor IT oriented. The governance provides business users and IT specialists with all business governance features such as: collaborative provisioning, dynamic querying (data hierarchies), right and security management, version and context management, auditability and total traceability, business regulation compliance, etc.

This governance is feasible with help from business referentials only: MDM, BRMS and BPM. Without setting up these referentials, a large part of the Information System would be hard-coded and opaque for users. Conversely, a right utilization of MDM, BRMS and BPM (in this natural path) allows for freeing IS agility and traceability. IS Architects must understand that agility and traceability are feasible only if reference and master data are tackled first. It is not relevant and actually un-sustainable to set up a BRMS first when data management and quality are approximate or bad.

The **Agility Chain Management System (ACMS)** is formed by three types of Business Referentials corresponding to Master Data Management (MDM), Business Rules Management (BRMS) and finally Business Process Management (BPM). It is not feasible to restructure Information Systems in a sustainable manner without fixing reference and master data first. In a second step it becomes possible to add rules management. Only in the last step, processes can be implemented in a right way.



## ...which you may compare with: the usual Functional and Technical silos



Process Modelling in such a context does attempt identifying and extracting logical *bits* out of each silo, along the idea to cut-out duplicates and increase sharing, make some bits re-usable, and re-link those bits in new chains of activities that ought to be *more flexible*...

It's like trying to hold a wall from collapsing while at the same time people change bricks in many places. In the end, the wall will roughly look the same, with different materials at some places...



## Functional silos generate redundancies



- numerous techniques and products attempt to identify and reduce these redundancies, yet they do not solve the problem at the core.

Using Business Objects and SOA, urbanization is enhanced and redundancies can be removed

under license: creative commons

14

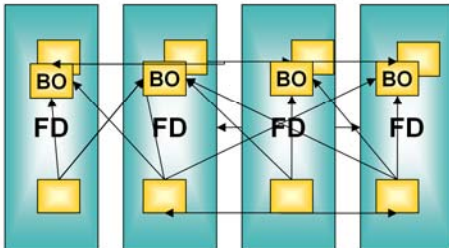
With or without PRAXEME, an objective of SOA is fostering re-use, here in the form of services. In other words, a recurrent claim of SOA is the benefits that can be earned from increasing the sharing of resources.

To simplify, these resources can both be data resources and functional resources. The 'service' paradigm encompasses both data-access-type-of services and calculation-type-of services, plus all intermediate mixes of forms.



## The face of change...

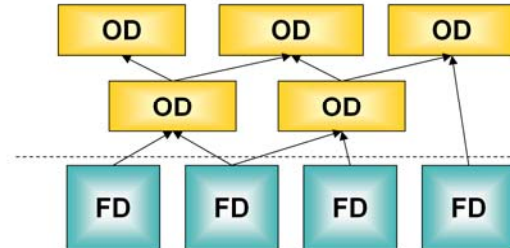
### Sample architecture based on functional approach



Logical blocs based on Functional Domains from a pragmatic and historical model  
Strong interdependencies and redundancies:

- Business Objects are found multiple times
- All components may be linked to others

### Logical architecture diagram following SOA



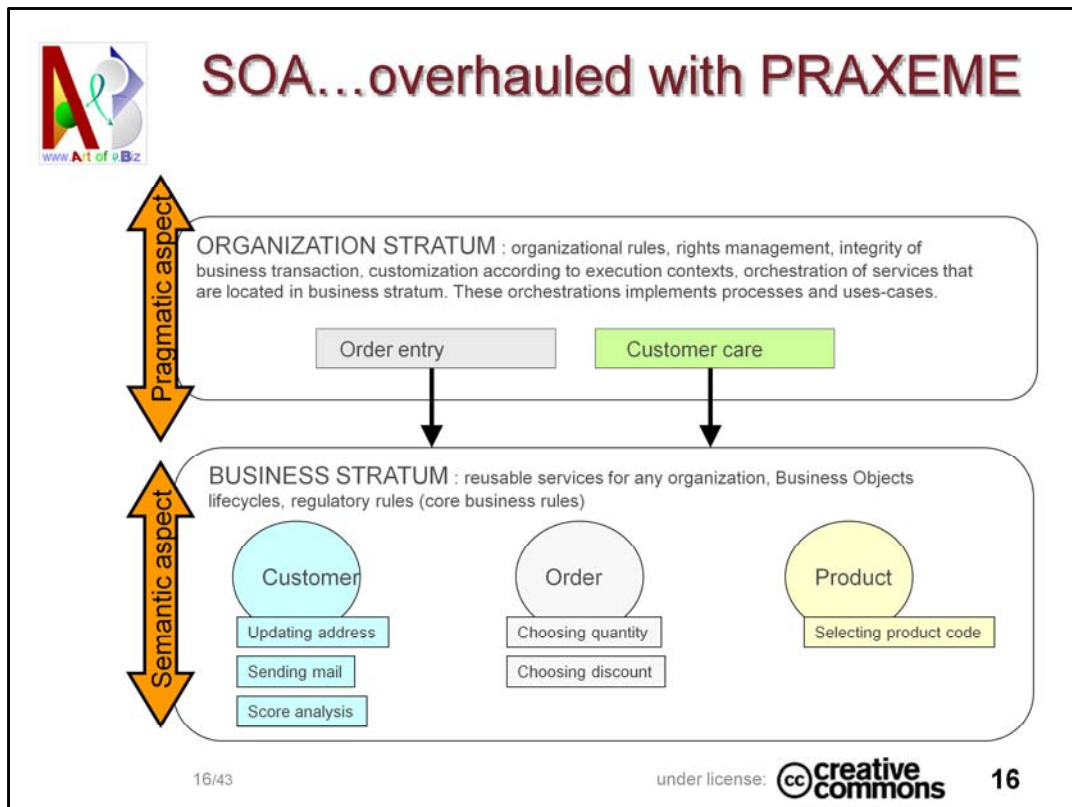
Logical blocs based on Objects Domains structuring the semantic model  
Dependencies linked only with topological constraints

- From organization stratum to business stratum
- Mutual relations are forbidden
- No dependencies between OD blocs.

under license:  15

With or without PRAXEME, the use of **SOA design procedures changes dramatically the aspect of IT systems.**

The main changes are bound to a simple decision: **isolate the business objects** in well identified fields of the system. The core of the system must now be structured with Objects Domains and not with functional domains. The part isolated this way is largely reusable.



A key in designing **SOA with Praxeme** is the distinction between **Semantic** and **Pragmatic** aspects. Once such distinction is understood, and the proper modelling works are conducted in each case, most of the other models—and the services notably—can be derived almost mechanically. Of course, designers are capable of taking many different orientations, but the core semantic models will never be compromised.

**Pragmatic** aspects reflect the **organisation**: how the business actually takes place, what the company wants to control, how it tracks the flow of goods and money, the division of labor, who can do what and at what time, and so forth.

**Semantic** aspects model the objects of the world, whether in "reality" ( a car, the damage to a house, cash, color, measurements, a financial instrument, a portfolio, a date, a company, ...) or virtual (the assignment of an account manager to a customer, a role, an indemnity, depreciation, preferences, ...) , and the associations between these objects.

The distinction between semantic and pragmatic, and the systematic derivations that ensue will altogether drive the definition of services in the architecture.





## PRAXEME does not re-invent the wheel!

- Good sense at the base: decoupling, encapsulation, separation of concerns, abstraction, contractualization, testing, ...
- PRAXEME reuses all the knowledge inherited from **object oriented** design
- no other formalism than available in UML is required

*but the way to use all of that is quite original!*

under license:  creative commons

17

PRAXEME makes an extensive use of existing modelling standards, tools and techniques. To that extent, nothing is original in PRAXEME. It is the way these tools are used that is original.

PRAXEME has anyhow an original contribution of its own: a notation for pseudo-code used while producing logical models. Pseudo-code does prevent the ambiguities possible with plain natural language. There's no standardized pseudo-code notation so PRAXEME proposes one, but any other formalized notation would do the job.



What makes PRAXEME different?



## PRAXEME reverses the way you design systems!

- The usual way:  
top-down hierarchical-functional decomposition



- PRAXEME:  
by DERIVATION, following *the curve of the sun...*



under license: creative commons

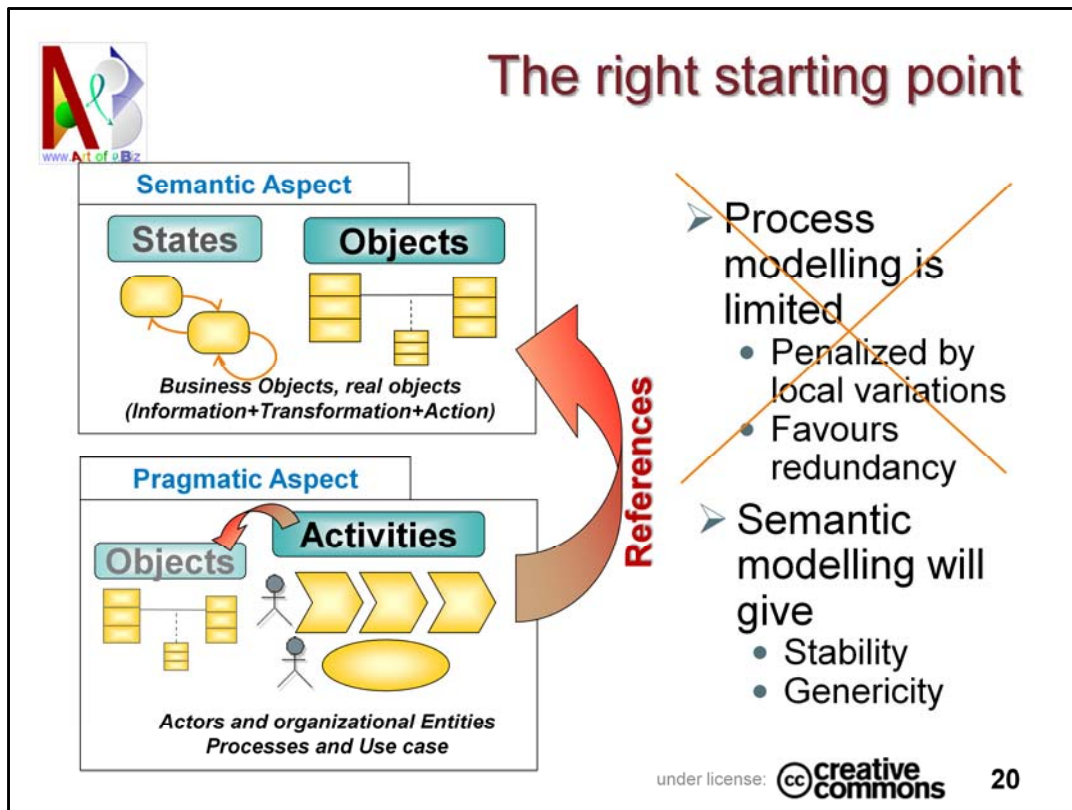
19

**The usual top-down hierarchical and functional decomposition can only replicate internal divorces.**

PRAXEME models the semantic and pragmatic aspects (objects, state machines, constraints, use cases) and then **derives** the processes (except for pure organisational aspects, they are **NOT** modelled from interviews and the existing flow of works), and then can propose a new organisation, or support the existing one with much more coherence and agility.

*Yes indeed, the above diagrams are much simplified and one can notably argue that in object-oriented modelling, logic and data are tightly associated by principle; so, why do we have two boxes above? we have two boxes, because we roughly illustrate the order of building the variant representations of these two aspects of the same 'objects'.*

The '**curve of the sun**' is explained later on...



Starting or preparing a SOA initiative with an exhaustive study of all the enterprise processes is a mistake.

The Process/Activity approach is obviously (and intuitively) incomplete if it is not accompanied by a precise analysis of the objects of the business. Indeed, while modelling activities, references are obviously made to objects from the business domain and therefore some modelling of the data used by activities will ensue, somehow following the references from Activities to Objects.

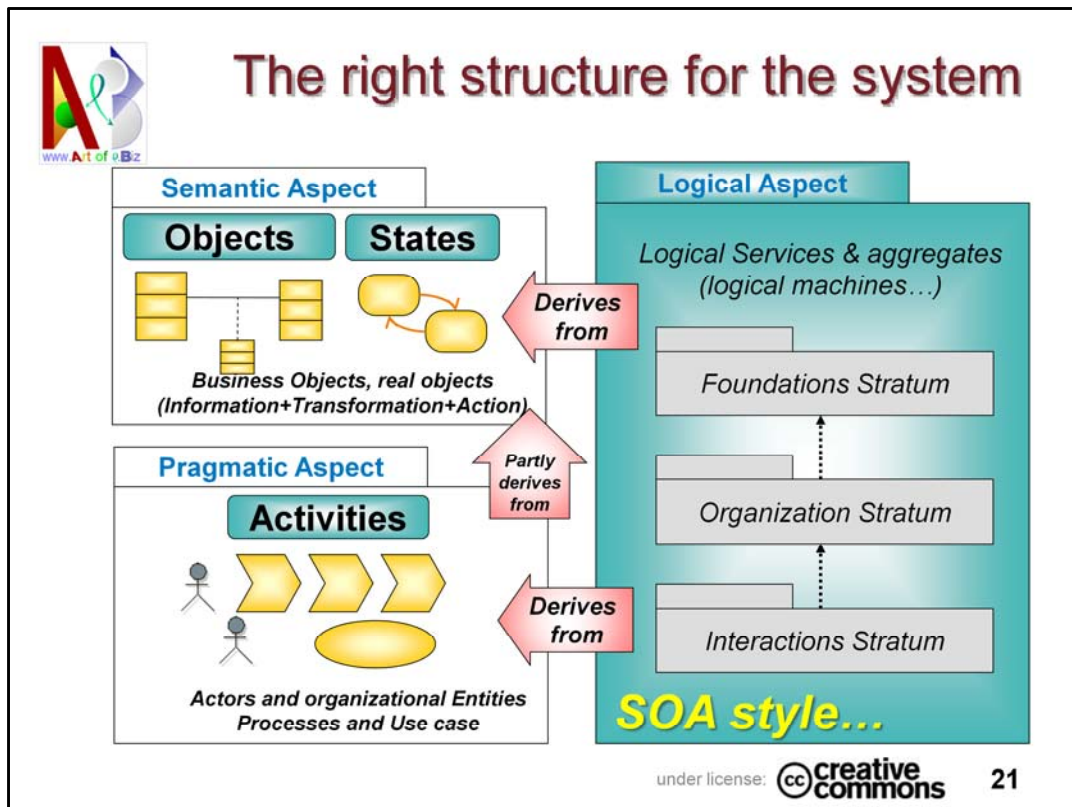
This way of discovering objects mixes objects of all kinds: some that support operations, some that model the domain; and the notion of state is generally completely overlooked, being degraded into combinations of object's attribute values.

Moreover, activities are penalized by local variations, cultural differences between organizational entities, bias dependent from the actors, redundancy, and a rapid pace of change. Architects will run after it: trying to keep their models in line with what's really happening.

Activities are not the right candidates to serve as foundations. Their instability raises a big risk when it's time to change things at any level.

Whereas it is obviously true that Activities are supported by data objects, the belief that References from these Activities to data could provide the path to data models is actually wrong. The idea of PRAXEME is to dissociate first two brands of Objects: **Pragmatic** ones (that directly support activities for the lifetime of the activity) and **Semantic** ones (that persist the state of the business). PRAXEME further demonstrates that a significant number of activities can be derived from semantic objects; precisely, from the state machines of semantic objects. It does mean that we reverse the design path from Objects towards Activities for all semantic objects, whereas the pragmatic objects are still discovered from the Activities themselves.

Semantic objects and the associated state machines are hooked to the 'universe of discourse'. Consequently, they provide a very strong and stable starting point.



Shall we then believe that, with the above semantic and pragmatic categories, we have captured everything into models (processes and data) and can start making implementation specifications by mapping those artefacts onto an architecture?

Actually no!

Another major aspect of PRAXEME is the introduction of a third model: the Logical Model. This model is still independent from implementation, and definitely not a luxury. It is actually the point where Services are casted, not by rules of thumb, but by derivation. And who would dare pretending that Services shall be dependent from the implementation?

The Logical Model is the Corner stone between abstraction and implementation. It is an essential transition point between analysts and developers. It is the cradle of Services, which could become Web Services in relevant technical infrastructures, but not only.

Pragmatic Models partly **derive** from Semantic Models, and the Logical Model **derives** from both the Pragmatic and Semantic Models. All these **derivations** are ruled by procedures, which are the essence of PRAXEME.

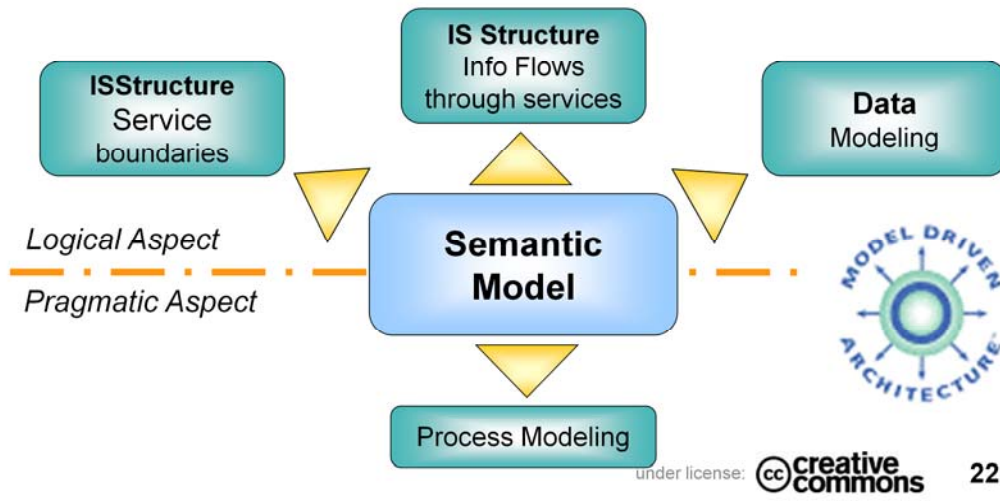
The logical model is itself structured in order to trace the derivations between models while respecting a "separation of concerns". Hence the structural elements called *stratums* to avoid the too often used terms like "layer" or "block".

Although independent from any implementation, and although derived from other models, the logical model offers numerous degrees (or we could say dimensions) of flexibility. It is a playfield for architects who could adopt various styles for organizing the bits of the model. **That is the point where an important decision is made: SOA is one architectural style that can be used for the logical aspect.** It is only a style and therefore is not implicit even if this is obviously the style of choice at this time. Other architecture styles that could be used to build the logical models include: EDA (Event Driven Architecture), Functional approach (the classical one), Agent or Multiple Agents approaches, etc... But it is true that the only style that has been fully explored and documented in PRAXEME is actually the SOA style.



## The magic in PRAXEME is its use of DERIVATION rules

- Illustration: the 4 derivation paths originating from the semantic model



Being a precise documentation of the business domain, a semantic model has **a lot of added value**. But there is more: by applying derivation rules it is transformed into those other models that help building up the system.

There are four derivation paths:

- **Derivation of the Pragmatic Model** (see later) Significant parts are derived by inverting the state machines of semantic objects. The result is the base for process models, enriched with actors and organizational constraints derived from pre-modelling activities.

- **Derivation of the Logical Aspect** (see later) which is actually derived from both the pragmatic and semantic models.

- o **Service Models** derive from the semantic aspect (according to an architectural style that, as a strategic decision, can be the SOA style). Services are not discovered by an intuitive process based on personal skills and knowledge but rather analytically deduced from previous modelling works. This reduces the risk of creating services which evolutions will reveal as non-reusable. The procedure also highlights governance principles about versions, variants, contexts and so forth.

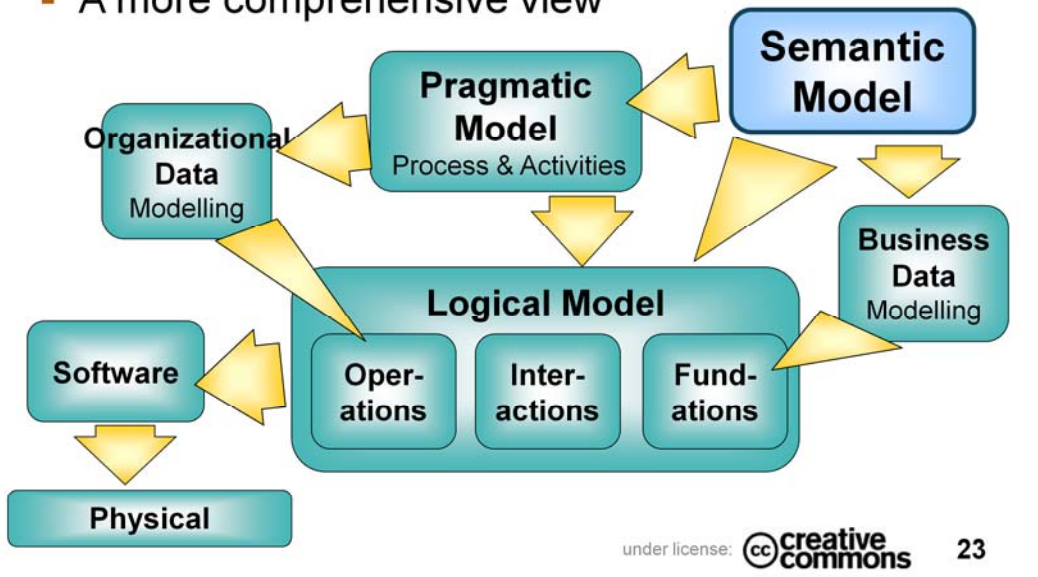
- o **Flow Models** derive from semantic models, in particular about state machine transitions.

- o **Data Models** derive from semantic models, quite obviously. Logical Data Models (something close to the classical Conceptual Models) in a relational form (a style) simply derive from an application of normal form rules.



## The magic in PRAXEME is its use of DERIVATION rules

- A more comprehensive view



Obviously, DERIVATION replaces all the guessing and approximate work behind the application of empirical design rules, by logical (almost calculated) designs that reflect the fundamentals of the business and separate clearly those elements that derive from the division of labour in a particular enterprise (the Organization).

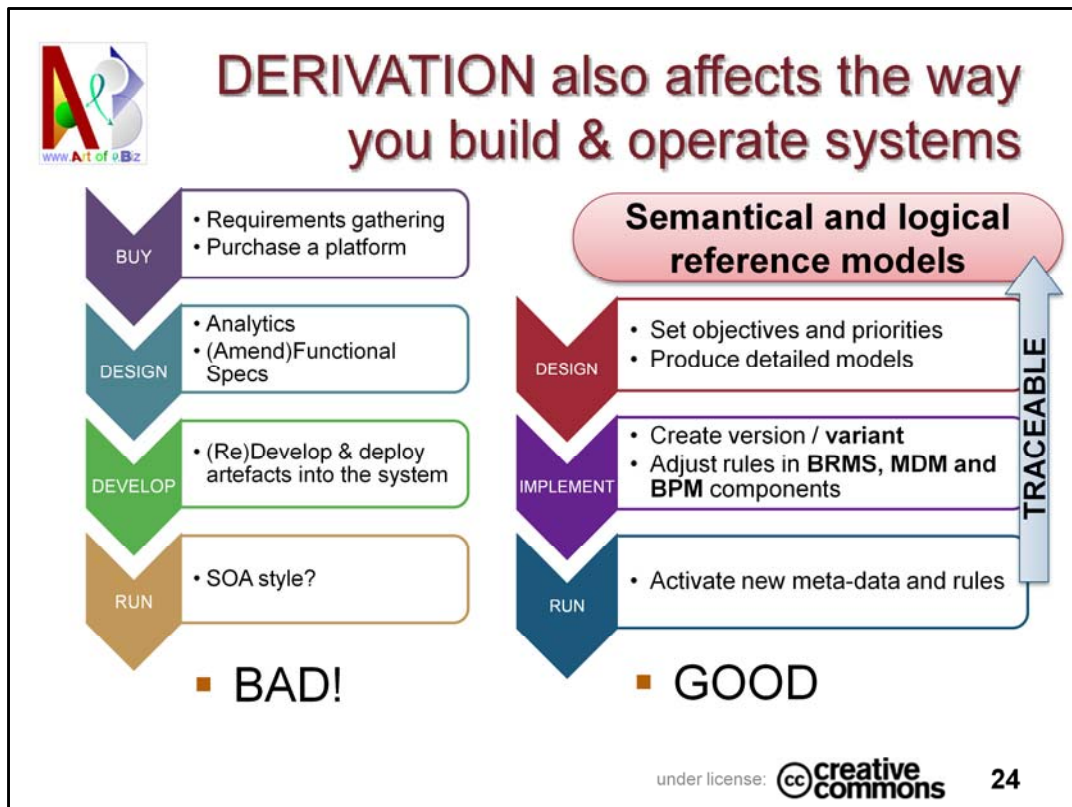
Web services are, by the same token, derived from higher level models. There's no guessing about which (Web) Services would maximize future reuse and agility! With PRAXEME, they just have to be 'like that' and then agility and reuse will naturally follow!

Of course, there's still a lot of room for different groupings or artefacts, different styles, slightly variant domain boundaries, coarser or thinner sets, and many optimization decisions to take through the creation of 'variants'. The role of designers is far from being extinct!

Of course too there is a lot of existing systems with which to interact and that may not possibly offer the Services derived by PRAXEME; and that is too well managed through additional implementation 'variants'.

By the same token, the Organizational aspect is a field of great freedom in the way to organize the business, but such flexibility is very clearly framed inside semantical fences that, at the same time, limit the freedom and show the improvement paths.

The schema that is represented illustrates additional derivation paths with regard to the previous figure. But it is yet unstructured. PRAXEME will organize all these blocks into a framework that is named the **Enterprise System Topology (E.S.T.)**. The EST defines the PROcess-side of PRAXEME with regard to the PRO<sup>3</sup> reference model for an enterprise methodology.



The implementation of systems designed emphasises the importance of meta-data. Actually, PRAXEME demonstrates that the flexibility and re-use objectives expected from a service-oriented approach are hardly achieved without suitable tools for managing parameters and rules that enable the creation of variants and versions.

#### PRAXEME and TOGAF:

The production of reference models in PRAXEME (that capture the semantic, pragmatic, geographic and logical aspects) is definitely an element of the Enterprise Continuum of TOGAF (covering several architecture domains of TOGAF, with products mostly in the Solutions Continuum), whereas PRAXEME itself like pre-built models available from the industry (MDA, SCOR, Telecom FORUM, REA, ...) shall be seen as members of the Resource Base. However, PRAXEME doesn't strictly follow the TOGAF ADM (Architecture Development Method) and phasing. Most of the PRAXEME upstream aspects (Semantic, Pragmatic, Geographic) and Logical Modelling altogether fit Phase C (Info Systems Architecture), whereas selected downstream aspects (Technical, Hardware, Physical) fit Phase D. The PRAXEME Software models being much concerned with implementation mostly escape to the scope of TOGAF.

Referential data (actually meta-data) from the Semantic, Pragmatic and Logical aspects of PRAXEME do conform with the general TOGAF philosophy for maintaining architecture reference models that are re-used and maintained. However, PRAXEME is more precise in producing also detailed parameter sets, rules and configuration objects that will drive the flexibility of any future implementation. Yet, such implementation aspects escape by essence to the scope of TOGAF.





## ...and this is the Path to Agility

- With help from composition of existing services, new processes can be developed quickly and easily
- Existing services can be modified easily so as to create variants of services
  - Product customization
  - Reference data filtering
  - Pricing table configuration
  - Mail customization (polite phrase, logo, etc.)
  - Rules customization
  - Etc.
- With help from rules and parameterization, services can be configured without modifying the software and without systematic involvement of IT specialists



under license:  creative commons

25

This is the point where one can look back and SITAF, the Sustainable IT Architecture Framework, and possibly begin to understand the dominance of **meta** layers in SITAF.

Basically, the key to flexibility is clearly to drive operations with meta-data telling what to do at what point in time (rules and processes), what is valid or invalid (rules), and how things actually take place (system/services interactions are managed, and versioned, and declined in variants, with the help of parameters, themselves stored in a MDM system). All such parameters and meta-data actually constitute a **referential** that drives the business; hence the opportunity—and actual need—for **governance** about who decides how the system shall behave; in other words, who can modify the referential. Understanding what parameter, what rule, and what piece of meta-data drives what element from the system is supplied by the **logical** model, itself hooked to the ‘**reality**’ that describes the business itself, so that the link between referential data and the business reality is clear.

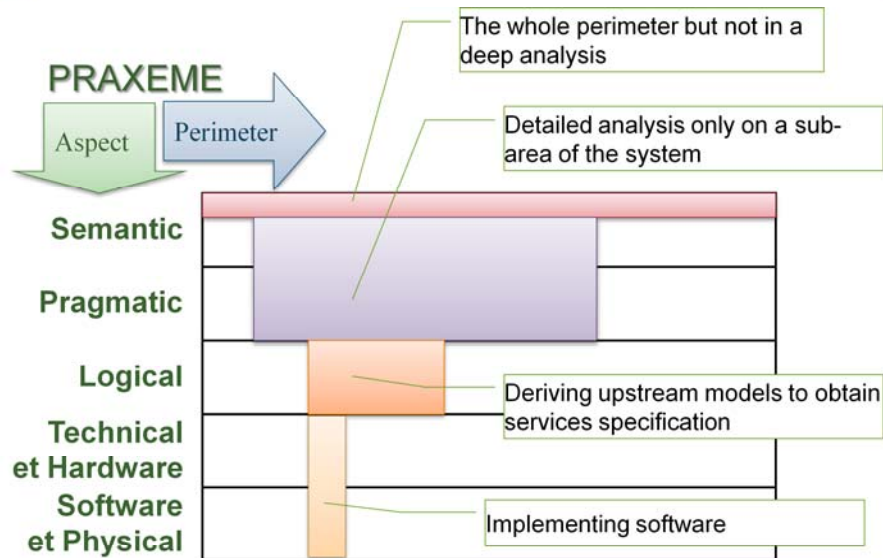
That is roughly the idea behind SITAF.

A consequence of possibly adopting SITAF is the requirement for original modelling and development environments capable of industrializing the execution of the method, and a new class of **Unified Platform for the Development of Services (UPDS)** that emphasize the role of **BRMS** (Business Rule Management System), **BPM** (Business Process Management), and **MDM** (Master Data Management), the later being at least dedicated to manage referential data in the first place and would be properly called Referential Data Management.

A Virtual Engine for PRAXEME (VEP) has been created that provides a possible implementation base for logical components. The VEP provides a framework inside which are plugged the BRMS, BPM, and MBM facilities. The purpose of the VEP is about forming a code-base so that the generation of code from the logical models is facilitated. No ‘PRAXEME’ framework is available off-the-shelf at this date from any vendor but all software components are available in the market and a UPDS can be assembled for a limited integration effort.



## ... and the path to Incremental Delivery



under license: creative commons 26

PRAXEME is indeed a software development method. In contexts dominated by the deployment and configuration of packages, the value of PRAXEME shifts to the assessment of packages and modules with regard to reference models (themselves traced back to business requirements, and help in driving configuration towards the most stable set of services that will best serve integration requirements).

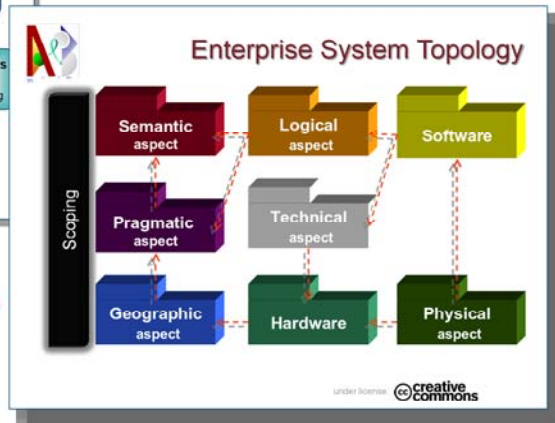
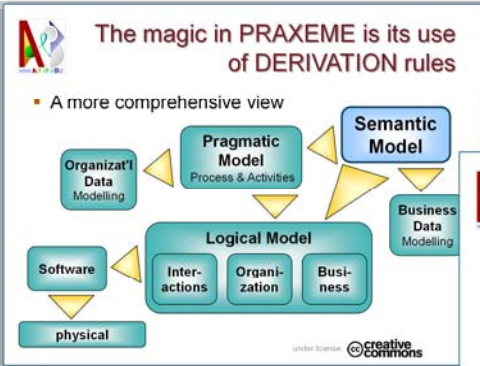


## The PRAXEME Framework

### The Enterprise System Topology (E.S.T.)

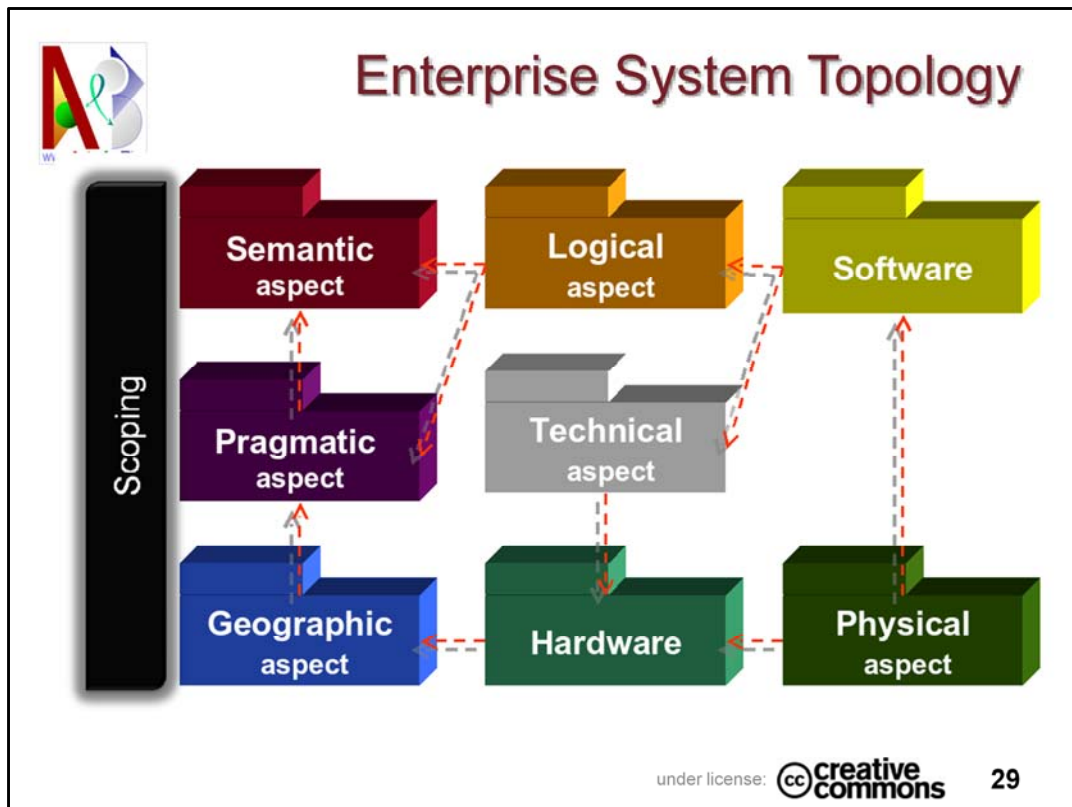


## The framework is mostly about organizing derivation activities



under license: creative commons 28

The EST (Enterprise System Topology) illustrates the PRAXEME methodological process. It covers 9 aspects (think modelling views, but the term 'aspect' fits better) plus a pre-modelling/scoping activity. The diagram shows the dependencies between all these aspects.



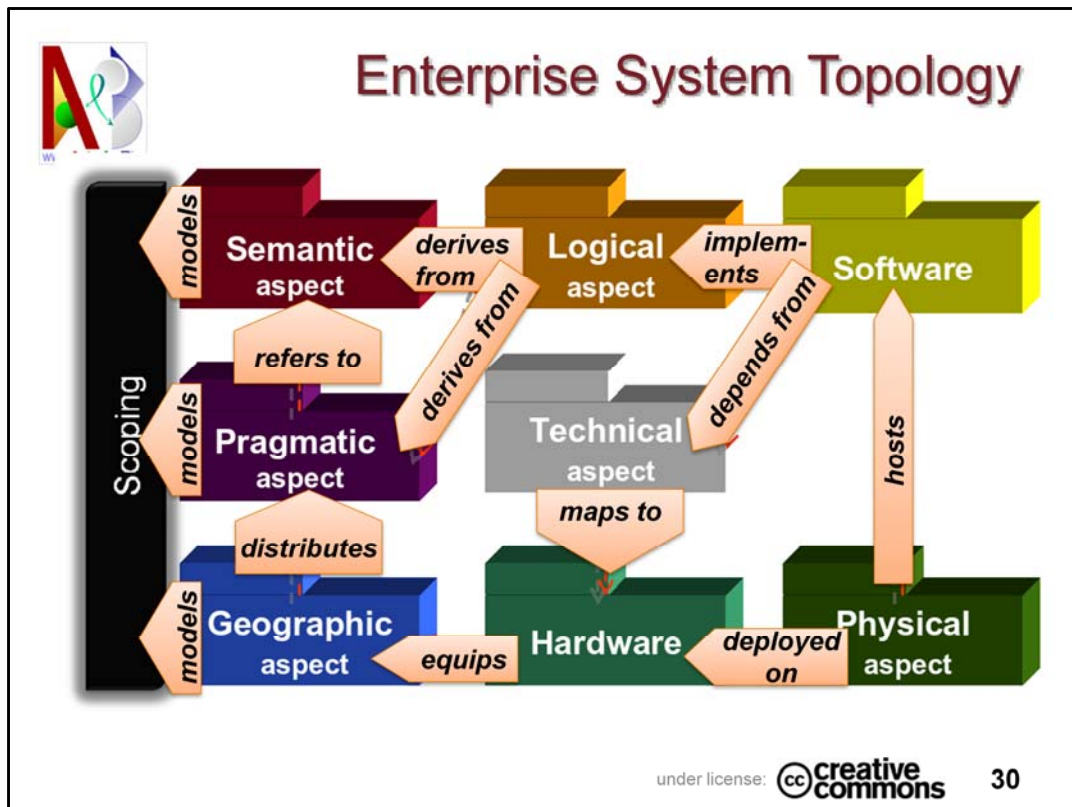
**The diagram** is like an UML package diagram, except for the Scoping (or Pre-Modelling) box which floats in front of the EST.

The EST formalizes the structure of the enterprise representation. It defines the boxes – i.e. the types of representations – where to record the information we need to capture, and the decisions we have to make regarding every aspects of the enterprise.

**Colors** convey 2 ideas. Upstream aspects, Logical aspect and Downstream aspects adopt related tints to show this high level grouping. The Technical aspect adopt the gray to show its different position and the special links with other aspects. Another reading exists: the colors cover the complete spectrum, meaning that with the 8 aspects, the structure covers the whole enterprise.

**Arrows** are dependency relations. They mean « Depends upon » or « Refers to » or « Uses ». In some cases they also have a more precise sense of « Is derived from ».

Relations are oriented (they are UML dependency relations) which means that **reverse relations do not exist**. This is important because those non existing relations are therefore **forbidden**. This preserves the separation of concerns and avoid dependency loops that would be impossible to manage in a design process.



This is a representation of what each 'arrow' means. It is critical to note the arrows that are not represented. They are not missing but clearly claim an absence of dependencies between the relevant aspects.

People often note that the Technical aspect itself depends from nothing, just like pre-modelling/scoping. True, the Technical aspect will capture technical constraints that do not depend from any of the illustrated aspects. Yet, it does depend from constraints like 'non-stop operation' requirements, asset protection requirements (security), performance and storage requirements themselves possibly dependent from archiving policies, targeted response time, number of connected users, the distribution or centralization of computing resources, facility management opportunities, expertise of the operations team, re-use of existing equipment, and the like.

It does not depend from the geographic aspect either, because geography, and the means to equip each site, will translate into technical requirements for supporting multiple sites or only a central one, making sites autonomous or not in case of network failures, and so forth.



## Reading the EST

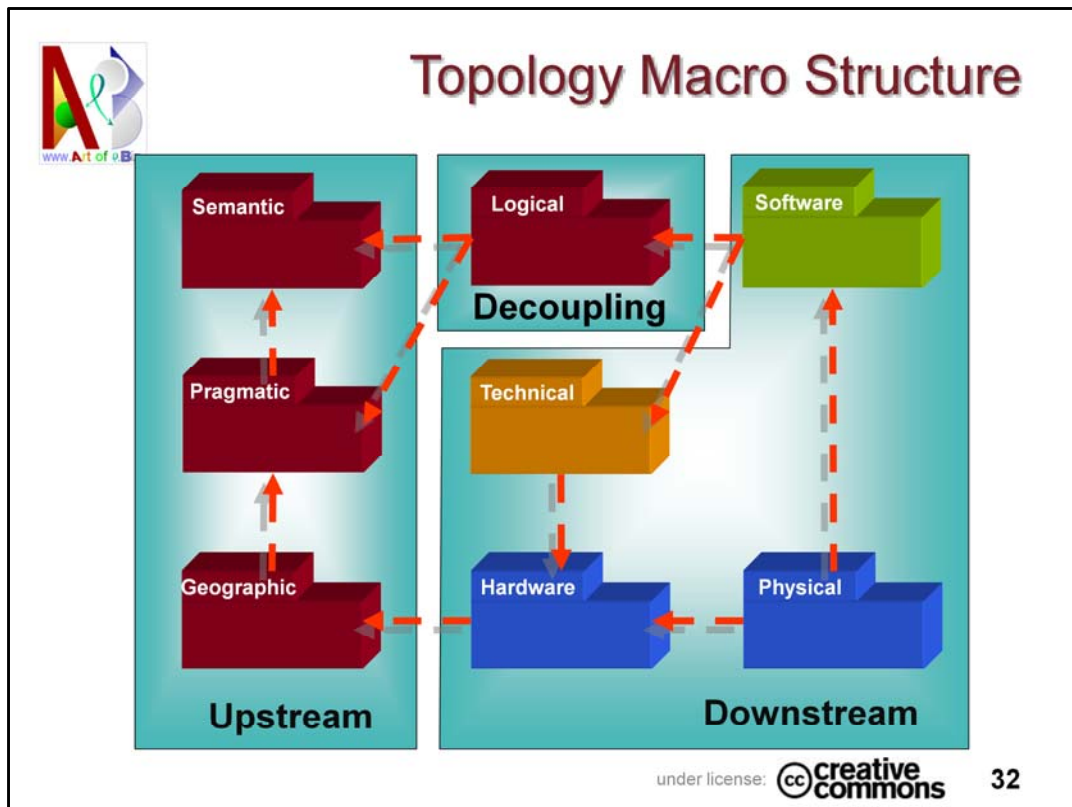
- Arrows represent dependencies between aspects
  - A package references content from another oneOR
  - A package is derived from another
- One notable exception: Tech depends from Soft? No!
  - Technical Aspect gives
    - Technical choices
    - Recipes to use the technologies
- Arrows are oriented
- Missing links
  - May carry more meaning than existing links
  - == Forbidden dependencies
- The EST supports traceability

**Warning!**

under license:  creative commons

31

**Traceability map:** models have relationships between each other. These relationships are rationalized with the semantics of «Is Derivated From» which means that when looking to some parts of a model we know precisely from what part of the source model it comes from. Obviously the derivation procedures guarantee that every part of a source model is mapped onto a precise part of the target model. Hence the traceability.



This representation gives a good view of what the logical aspect is, between upstream aspects describing the business and downstream aspects describing the solution designed.

**The logical aspect is a decoupling apparatus between upstream and downstream aspects.**

- Upstream Aspects
  - Represent the **core business** knowledge and the **way to do it**
  - Do not include IT considerations
  - Keep strict **links with Scoping**
  - Is **stable** compared to IT representations
- Logical Aspect
  - Represents a **decoupling structure** between Business and IT systems
  - Regardless of technical choices
  - Is modelled with an **architecture style**
    - SOA is such a style
- Downstream Aspects
  - Represent the **IT solutions** to be delivered
  - Includes all means to deliver the solutions
  - **IT operations** are located here
  - **Technical agility** lives here

When comparing PRAXEME aspects with the classical conceptual, logical and physical modelling views, the Semantic and Pragmatic aspects belong to the Conceptual view, the Logical aspect conforms with the Logical view that is still implementation-independent by nature, and the Software aspect fills the physical view.





## Upstream Aspects

### ➤ Semantic Aspect

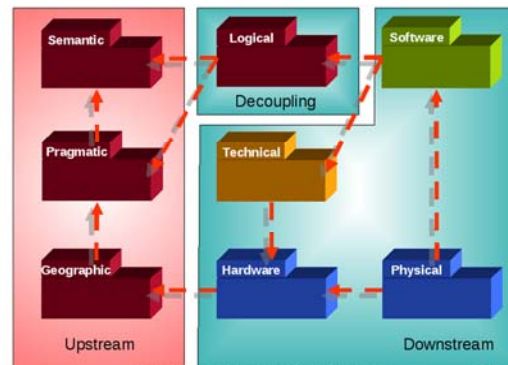
- The core business knowledge (data, logic, rules, states, relations...)
- Organized in Business Domains

### ➤ Pragmatic Aspect

- How an enterprise acts in its business domain (use cases, activities, processes, admin data, life-cycles...)
- Organized in Functional Domains

### ➤ Geographic Aspect

- Geographical constraints, localization of activities and systems



under license: commons

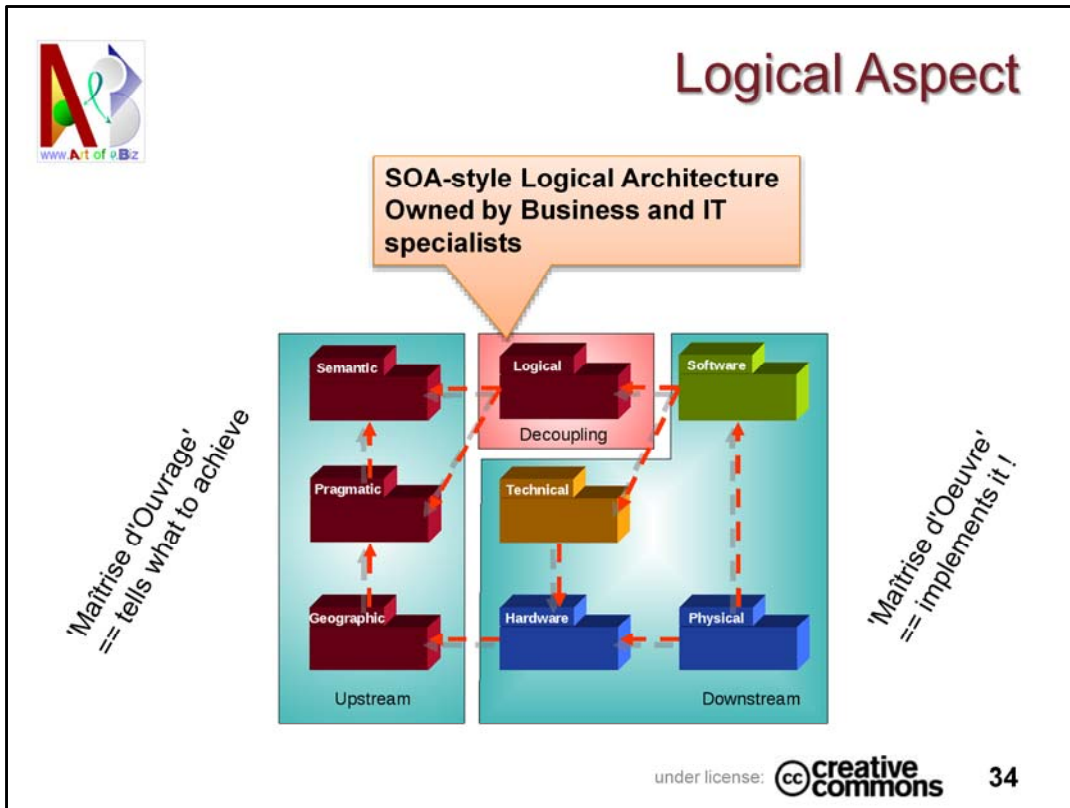
33

The **Upstream Aspects** represent **fundamental business knowledge**. Together they document what the business is, how it is done, what are the rules, constraints, goals and wishes, and all events that matter to the business.

They do not include technical details, not even architectures of a particular system which would be a mapping of business descriptions. **Upstream aspects clearly describe an enterprise information system**. The Upstream Aspects relate to the **Computation Independent Model in MDA (CIM)**.

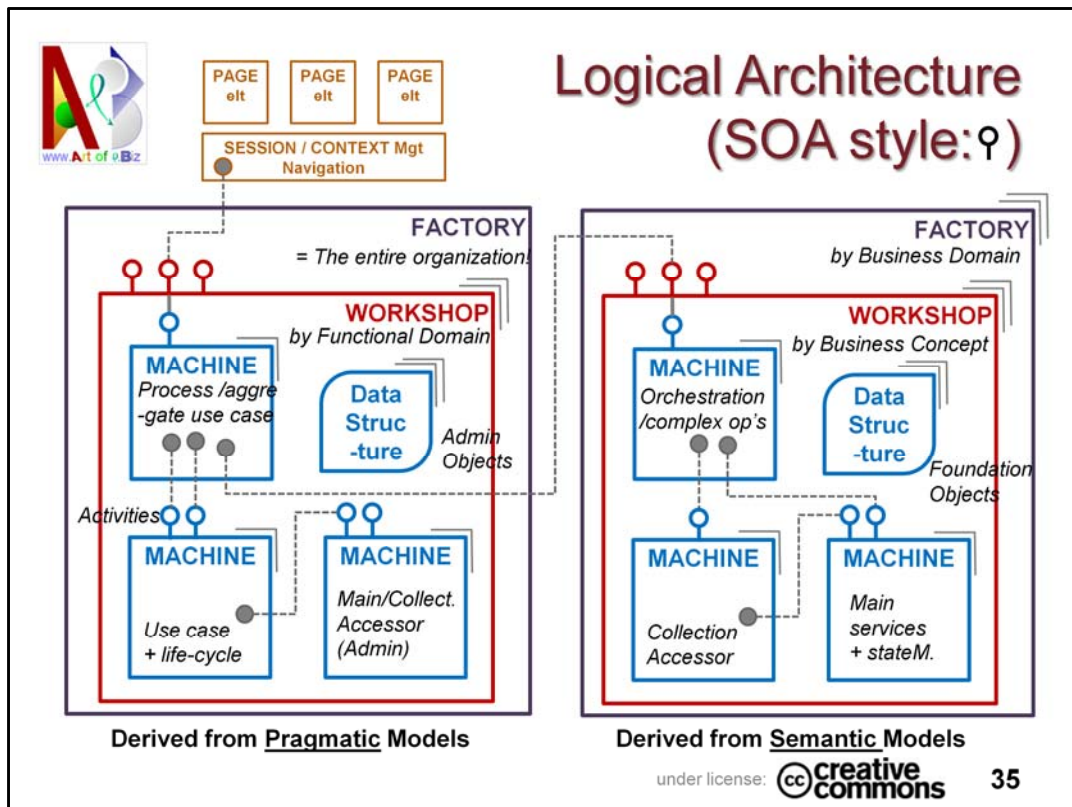
The Semantic Aspect:

- Captures the being of reality
  - In terms of objects and applying an **Object Oriented Approach**
  - All the being without technical or organizational details
- Does not describe
  - **Actors** and organizational details
  - **Actions** on objects (Processes and use cases)
  - Know-how and recipes on how to deal with business
- Does describe
  - **Real life objects** with their information and relations
  - **Object Statuses** representing the object transformations
  - Pure **Business Rules** (*règles de gestion*)
    - From the business domain reality
    - From external constraints like regulations, standards
    - Enforced rules that can not be avoided



The LOGICAL aspect is still completely independent from any implementation choice and technology.

Upstream Aspects relate to business domain. Downstream Aspects relate to technologies. This easy to understand. In between, the Logical Aspect is probably more difficult to grasp because it is not fitting either side. But it is fundamental as the decoupling layer between the two. It's role is precisely to reconcile the very different life cycles on both sides. Not designed by pure business concerns, and neither a collection of technologies, it establishes the structural rules and architectural style used to transform upstream models into downstream models, or, in other words, to get from the business strategy (the information system) to the implementation strategy (the IT system).



According to the Service Oriented approach, the Pragmatic and Semantic models are mapped into the Logical aspect comprising three levels: Machines, Workshops, and Factories. These three levels appear both within the Foundation (Business) stratum and the Organizational Stratum.

The derivation of each Semantic object yields a Main machine, a Collection-Accessor machine, and a Data Structure. Additional machines can implement complex or composite operations, else orchestrate diverse services such as to improve the management of inter-dependencies. Machines are grouped in Workshops by 'affinity' to the same business concept, and such as to minimize interactions between workshops.

Workshops on the foundation (business) side are then grouped into business domains at the 'factory' level.

The derivation of Pragmatic models yields one organizational machine per use case, along with a pair of Collection-Accessor machine and Main machine for each administrative object. There are also additional machines derived from parent use-cases and processes. Machines are then grouped by functional domain (reflecting the entire organization and governance) into Workshops. All the organizational workshops are then found inside one single Factory that is the entire organization itself.

Each machine follows an imposed structure with pre-conditions, logic and post-conditions also derived from the models (conditions in state diagrams, organizational and business rules, etc.)

Services are elsewhere inside machines (private services), at the interface to machines and at the interface to workshops. Most of the services are actually derived from the models.

While the Logical model is built, existing application packages and software components are injected at the machine, workshop, or even foundation-factory level.

The Logical model that ensues shall then be mapped into a Software model. This will be a main source of variants (not to mix with versions!) allowing various service optimizations (like reduction of data structures, compensation, composition, or splitting needed to interface existing systems, etc.)



## Downstream Aspects

### ➤ Technical Aspect

- Execution environment (OS, persistence, transactions, containers, security...)
- DB, ESB/middleware, BPM, MDM, BRMS, IAM, ...
- Human Interface technologies

### ➤ Software Aspect

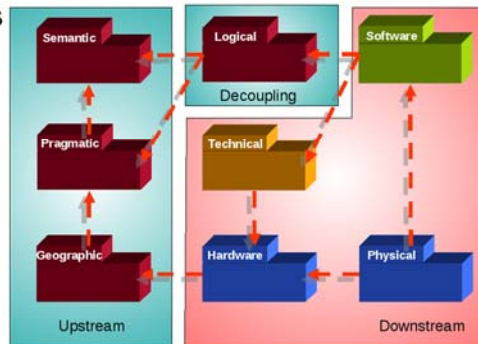
- Code & structure, packages, rule sets, storedprocs, meta-data, data bases, ... all software artefacts
- Assemblies / components
- **Referential**

### ➤ Hardware Aspect

- Machines, networking

### ➤ Physical Aspect

- Mapping software onto servers / processes / threads, onto HW (deploy)
- Failover, Load-Balancing



under license: creative commons 36

Downstream aspects are turned toward the IT system. We will finally have to deal with technologies and technical constraints to build the final system.

The MDA (Model Driven Architecture, OMG) approach is much relevant this stage.

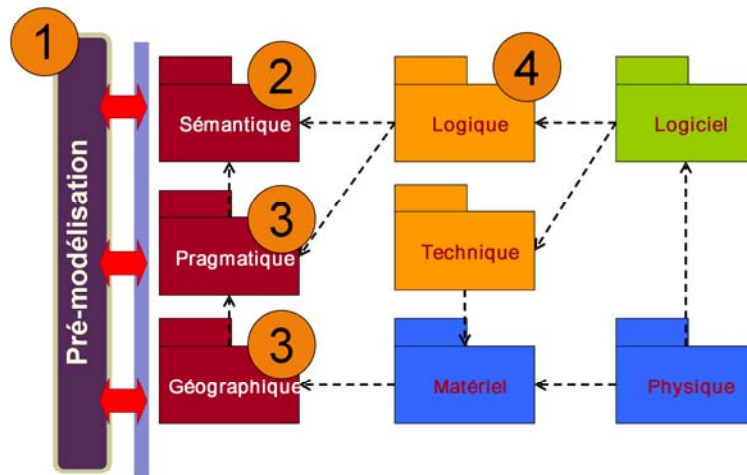
The life cycle of the infrastructure (with all execution artefacts) on the one hand, and of business objects on the other hand, do exhibit quite different timescales and requirements. Serviceability, continuity, support, contracts, and maintenance do not bear the same meaning on either side.

It is therefore compulsory to isolate the upstream and downstream concerns such as to prevent conflicts in life cycle management, hence the role of Logical Modelling. To that respect, it is important to understand that Logical Modelling (according to its dependency from the Technical aspect!) must be preceded by a **Technical-Logical negotiation** step, whereby important decisions about the future architectural style and software artefacts are taken and will influence the derivation of the Logical models from upstream aspects in order to facilitate the next derivation from Logical Models to Software.



## Life-cycle

- Example of a usual life-cycle



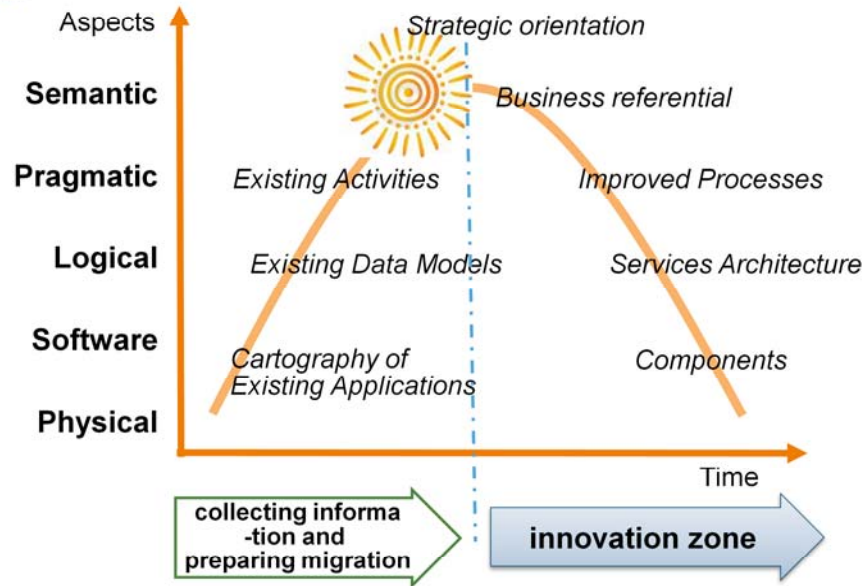
under license:  creative commons 37

It shall now be clear the dependencies illustrated in the Enterprise System Topology do yield a fairly logical Process (cfr PRO3 scheme, the sequence of analysis and design activities) for the execution of PRAXEME.

However, when considering the application of PRAXEME to one or several functional domains with varying scope and constraints from existing systems, one can easily create a few parallel threads of progress through the methodology, providing much room to organize the works in different sequences, or even iterations.



# The "curve of the SUN"



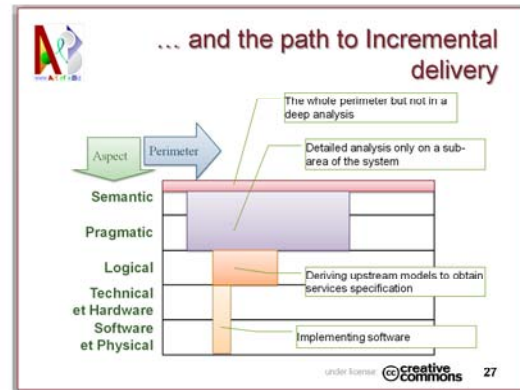
under license: creative commons

38



## Key concern

- **How to deliver a part of the future IS while ensuring**
  1. A high level of abstraction
  2. The ability to integrate this part in the future big picture of the new IS



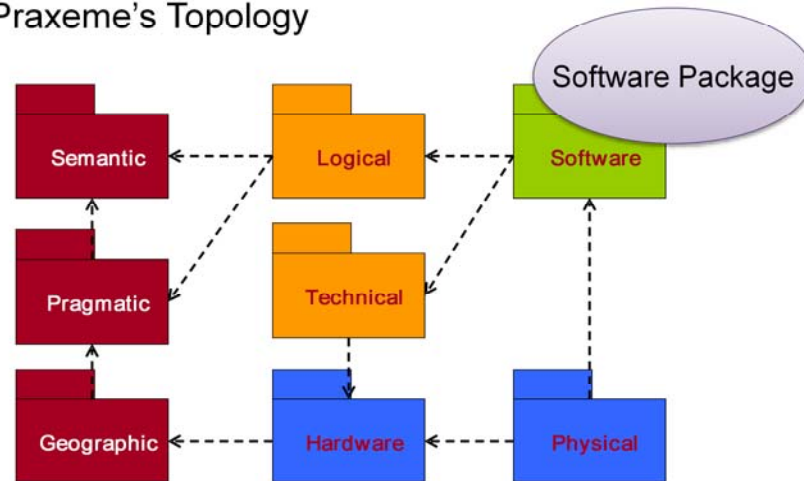
under license: creative commons 39

The previous figure can suggest that PRAXEME could only scope the whole Enterprise and all systems at the same time. This would be a miss-conception. PRAXEME does allow incremental delivery; PRAXEME does allow integrating existing systems. PRAXEME actually does much better on these two fronts notably because of the separation of semantic and pragmatic aspects that provide much visibility to actual compromises and thus allow driving better, informed, decisions at a level of vision higher than the usual 'improvement guess-works'.



## Software package and SOA

- The software package is located in the Software Aspect of the Praxeme's Topology



under license: creative commons

40

Existing software applications are modeled in PRAXEME as workshops, or even factories. Semantic and Pragmatic modeling must encompass (to some degree) the domains covered by the existing components/applications, in order to :

- keep control of the business (semantic) and organizational/operational (pragmatic) knowledge;
- assess the constraints imposed by existing packages

### Questions relating to the integration of existing packages:

#### At the level of Logical Architecture

- Business: Does the software package cover one or several domains of business objects?
- Organizational: Does the software package cover one or several functional domains?
- Can we use only useful domains of the software package or not?

#### How agility chain is taken account by the existing package with BRMS, MDM and BPM components or the like?

- Do these IT components exist? Are they reusable beyond software package's boundaries?
- How can we manage variants and versions onto existing systems?





## It's time for Questions & Answers

Thank you!

Bernard Hauzeur (bh@artofe.biz)

under license:  creative  
commons

41