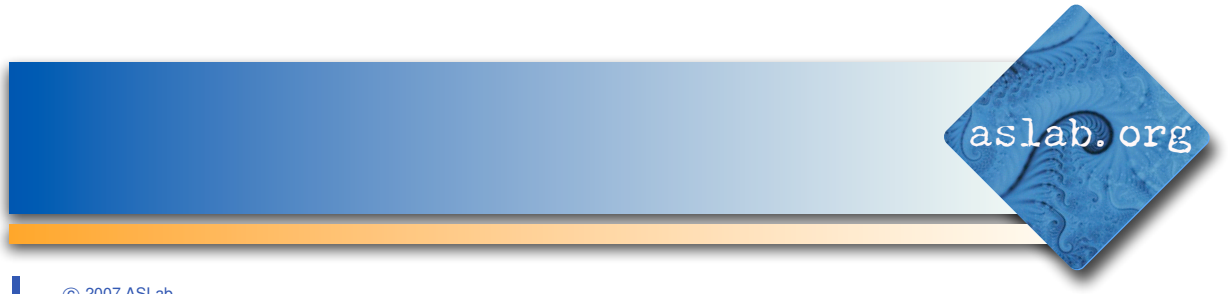

Title

The ASys Vision

Engineering Any-X autonomous systems

Author Ricardo Sanz, Manuel Rodríguez

Reference R-2007-001
Release 0.3 Draft
Date February 11, 2008

Address
Autonomous Systems Laboratory !
UPM - ETS Ingenieros Industriales
José Gutierrez Abascal 2
28006 Madrid
SPAIN

The ASys Vision

ASLab R-2007-001 v 0.3 Draft of February 11, 2008

Abstract

This report describes the vision behind the **ASys Project**.

This is a long-term research project focused in the development of technology for the construction of autonomous systems. What makes **ASys** different from other projects in this field is the extreme ambitious objective of addressing *all the domain of autonomy*. We capture this purpose in the motto *"engineering any-x autonomous systems"*.

The report presents the context for the technology sought and the different aspects under consideration in its development.

It also contains the strategic approach taken — *i.e.* the vision itself— including summaries of the knowledge of relevance in the different areas involved.

One of the central topics in the **ASys** Project is the pervasive model-based approach. An **ASys** will be using models to perform its activity. An **ASys** will be built using models of it. An **ASys** can exploit its own very models in driving its behavior. Model-based engineering and model-based behavior then merge into a single phenomenon: *model-based autonomy*.

Keywords

Autonomy, product-line engineering, control architectures, complex control systems, software-intensive real-time systems.

Acknowledgements

We acknowledge the support get from the *Spanish Ministry of Education and Science*, the *European Comission* and the *Goverment of the Madrid Autonomy* that have provided support for this project through several grants (MEC M3, MEC C3, IST ICEA).

Table of Contents

Chapter 1

A Rationale for ASys

1.1 Introduction

After many years of research and many projects with lost of deliverables there is no cosolidated base of tangible results that can sustain with adequate support future endeavours of the ASLab research group.

But, if analysed from a certain distance, this shouldn't be impossible because the central topic is clear and precise: *develop science and technology for building custom autonomy in any technical system*.

Obviously the task is daunting and thousands of researchers during decades have not reached the level of competence and technoogy that we are envisioning. No matter how dreamy this may sound, the path is clear and it starts with a decided, somewhat sttuborn, step in the right direction.

And this direction comes from a simple observation we did many years ago: there certainly exists a class of competence that may maximise system autonomy; we observe when technical systems overcome the unexpected beyon wat was technically planned and set into them; it is the competence of *resident engineers*. The maximal efficacy controller is an engineer —human or cybernetic.

When we exploit technical systems they can be operating in many types of situations:

1. The **operational situation**: when all is working as planned; *e.g.* the plant is producing the methyl-isocyanide with all plant units working as planned[1] or the spacecraft is approaching Mars with all flight systems working as planned[2]. The system is working in design operational

[1] On December 3, 1984, the Union Carbide's plant at Bhopal (India) filled the air with the deadly poison methyl isocyanide (MIC), leading to one of the world's biggest industrial disasters. According to the posterior investigation, the problem was the triggering of a release valve due to excess temperature caused by a limited refrigeration caopability caused by a program of operation cost reduction.

[2] The Mars Polar Lander entered Mars atmosphere entry on December 3, 1999. No further signals were received from the lander. According to the investigation that later followed, the

setpoints.

2. The **abnormal situation**: when something is not working as planned.
   (a) The **abnormal expected situation**: something is not as it should be but in any case it was previewed and then it can be understood and dealt with using predefined strategies; *e.g.* a refrigeration pump breaks and a secondary pump is activated to provide the necessara refrigerant pressure.
   (b) The **abnormal unexpected situation**: something is not as it should be and it was not previewed and then it cannot be dealt with predefined strategies; *e.g.* a truck doing some maneouvers gets rid of a "failure free" stock pipe.
      i. The **abnormal unexpected manageable situation**; the pipe broke releases gasoline in the soil that is evetually detected by general fugue detectors and the punping is shutdown as a protection meassure.
      ii. The **abnormal unexpected unmanageable situation**; the fugue pass unnoticed because there's no sensing around to trigger any effective countermeasure.
         A. The **abnormal unexpected unmanageable engineerable situation**: the operator detects the fugue by realising some abnormal slowing in the filling of the storage tanks and a field worker is sent to identify the problem. Upon identification an emergency task force is sent to the stock piping to repare it.
         B. The **abnormal unexpected unmanageable unengineerable situation**: the spilt fuel ignites and eventually produces a BLEVE[3] in the storage tanks. No system enough remains to be re-engineered and, worse enough, no engineer remains to re-enegineer the system.

What is the morale of this story? There are degrees of anticipation and varieties of strategies to be used in the plethora of states that a technical system may have.

The first step to reach universality in autonomous behavior technology is to clarify the issues and decide if we want to go for 1., 2.a, or, being driven by total hubrys, go for 2.b.ii.B if some remainings still keep some intelligence into them.

This is **ASys** : let's go for downsizing engineer's capabilities to the level of atomic, resilient subsystems in all kinds of operational conditions in technical systems.

---

most likely cause of the failure of the mission was a software error that mistakenly identified the vibration caused by the deployment of the lander's legs as being caused by the vehicle touching down on the Martian surface, resulting in the vehicle's descent engines being cut off while it was still 40 meters above the surface, rather than on touchdown as planned.

[3]One of the horrors of the chemical plant catastrophe movies: Boiling Liquid Expansion and Vapor Explosion. A 30 meters diameter bomb.

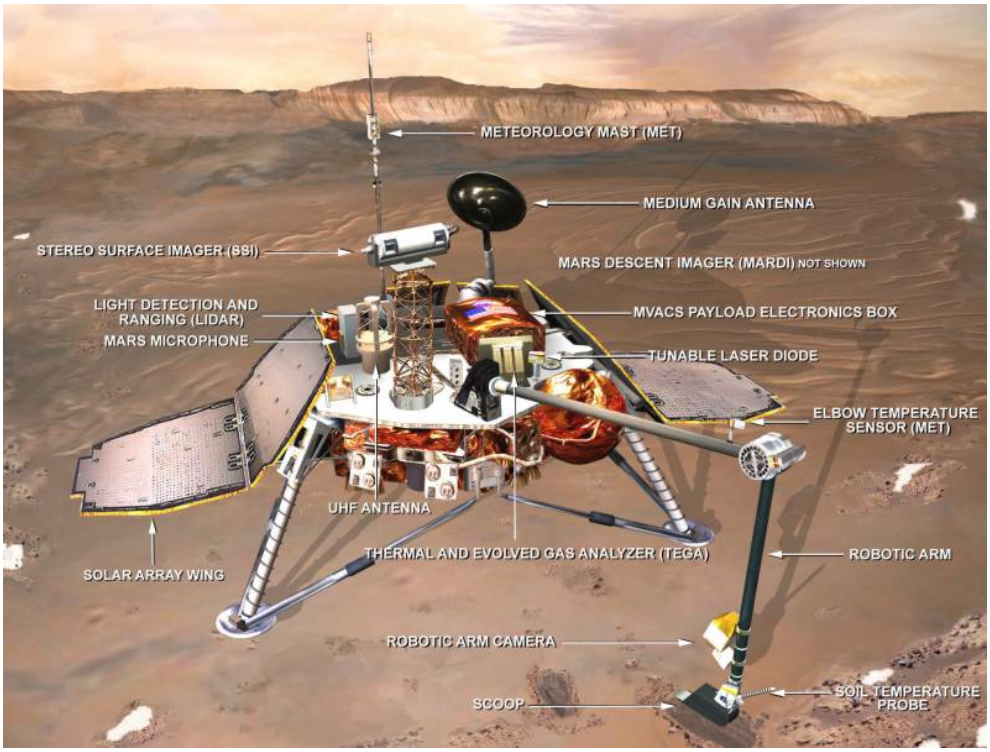

Figure 1.1: Frontal representation of the Mars Polar Lander probe as it should be in the surface of Mars if everything had worked as expected.

1.2 Automated systems desideratum

Obviously, there are still many open issues in the varios fields of competence involved in the different technical processes that subserve complex system engineering. This is a field plenty of opportunities for a control systems research program.

Some of these issues fall into the category of being able to solve concrete problems regarding particular domain aspects of certain systems, *e.g.* programming automata effectively or improving substratal plant resilience. Other issues shall be considered transversal as they potentially affect many of the systems of tomorrow. They constitute the fundamental desiderata of general research programs. Two of these issues are specially relevant for us:

- The maximal desideratum of production engineers is both simple and unrealizable: *let the plant work alone*.
- The maximal desideratum of automation engineers is both simple and unrealizable: *make the plant work alone*.

Working alone — i.e. being *autonomous*— seems to be at the very central objectives of most engineers. Autonomy is one of such transversal issues that may potentialy affect most future systems.

The search for autonomy has many reasons and implications but the concrete research target of this field is not clear at all as demonstrates the fact that even the very term *autonomy* has too many interpretations. But the search for autonomoy is a major thrust in systems innovation. This is generally true for two main reasons: economical and technical.

Economical motivation is a major force because automated plants are less costly from an operational point of view (human personnel cost reduction, improved operating conditions implying less failures, *etc.* But technical reasons are, in some cases, no less important because automated plants can be more productive, can operate fast processes beyond human control capabilities, can be made safer, more available, *etc.*

1.3 Bounding autonomy

In a sense, *full autonomy* seems to be the central objective of any automatic control systems research program. But, for different reasons, it is not (remember the story in section 1.1). Let's analise some of the reasons for this apparent paradox.

First, nobody in charge of real production systems want the plants to be fully autonomous because of trust[4]. Not just due to the perceivable higher robustness of human behavior but because in general full autonomy would mean that the systems were not complying with our own objectives but with theirs (*i.e.* of the plant, not of the owner). This is, in fact, a big difference between research programs that focus on pure biomimicry, *e.g.* animatronics or artificial life, and research programs that focus on pure economical value, *e.g.* process control or vetronics. This last —economic operation— is our case; so, in a sense, what we want as engineers is not full autonomy but *bounded autonomy*, *i.e.* we want to be able to make the system being autonomous up to the level where this autonomoy may impinge on the system ceasing to comply to some constraints imposed by us, their masters.

Engineers want bounded autonomy.

Second, no automation engineer tries to build a fully autonomous plant because in the general case of having plant-environmental uncertainty, that is a daunting, not-yet-clear-how-to-do task. There are indeed too many issues to take into account and less than perfect knowledge on how to achieve them. This iplies that some levels of autonomy may be not easy to reach not only technically but economically (*i.e.* the effort needed to achieve them is not justified by its value).

Engineers want made-to-fit autonomy.

[4] And old joke in control systems engineering: Some people runs afar a machine one of them screaming "It's out of control! It's automatic!".

So what we really want as automation engineers is just *bounded* and *made-to-fit* autonomy, *i.e.* we strieve to having the (control) knowledge, the (software) technology and the (engineering) process necessary for making the system autonomous up to the level where i) this autonomoy may make the system violate some constraints imposed by design; and ii) this level of autonomy does not cost too much to achieve.

Real considerations notwithstanding, generally speaking, and from a systems researcher point of view, there is still no clear way on how to achieve full autonomy or how to progres in this made-to-fit autonomy way. Humans seem to be the most autonomous entities we know about (thanks to their reasoning, learning and creativity capabilities) and that is the reason for bioinspired approaches. No matter how poetic or sensible this may sound, we have no definitive clues on how to leverage our limited knowledge of these humanly capabilities.

1.4 The meaning of "autonomy"

Obvioulsy there is another, for sure marginal, aspect that increases difficulties. Research implies comunnication with peers and problems may arise related to the hermeneutics of the very term "*autonomy*".

Etimologically it means *"giving itself its own laws"*[5]. That meaning, translated to the language of technical systems, can be read as *"the system decides about the control strategies to follow"*. From this perspective, the main difference between natural autonomy and artificial autonomy seems to be related with the the origin of the laws to attain-to and perhaps the objectives to pursue by following them[6].

The main argument against the possibility of autonomous artificiality coming from the biological systems world is related to this aspect of self-goaling (*e.g.* the highly mentioned causal closure of autopoietic systems). Natural systems seem to pursue their own objectives while technical systems pursue externally specified objectives. This analysis is indeed debatable, because artificial systems[7] can establish by themselves intermediate goals and even biological systems cannot escape the external fixation of goals — *e.g.* by evolutionary pressure (33)— then banning any real possibility of freedom in top-level goal setting.

However, this analysis will not be continued here because it deserves more detail and, perhaps, a different setting.

[5] Greek $\alpha\upsilon\tau o\varsigma$-self and $\nu o\mu o\varsigma$-law.

[6] Strictly speaking autonomous means just self-laws not self-objectives as some may interpret. But in the context of autonomous systems research there is not much clarity concerning the otherwise necessary separation of goals and laws.

[7] Care must be taken around the use of the word *"artificial"*. We use it here in Simon's sense (40) of *"built for a purpose"* and not in the sense of *"built by humans"*. In this last case, top level goals to be pursued by the system may not be explicitly set by the constructor as is the case of art or artificial life.

Another question of relevance is that given a certain interpretation of *autonomy*, we need a metric of it if we want to assess the perfromance of the different engineering methods for autonmous systems construction. Bertschinger (6) proposes a measure of autonomy based on information theory. Barandiaran (4, p. 18) gives a formal definition of autonomy based on agent and environment interaction models. A formal but more encompassing model of autonmoy will be necessary to support this task of evaluation.

1.5 Problems and strategies for increased autonomy

1.5.1 Two ways for increasing autonomy

The strategies for building autonomous systems fall into two main categories:

- build systems that are intrinsically autonomous and
- improve autonomy by means of a control system.

In purely theoretical sense, the first strategy seems to be the best —*intrinsically* sounds good for a property—, however, this implies that the system enginering process may get very complicated as it must take into account cybernetic aspects into the very design of the core physical process. This implies for example, that the physical design of a chemical reactor takes into account questions regarding weather or strategic decissions concering product quality. In some cases that will be possible, in many others the strategy to follow will be to try to encapsulate behavior to achieve modularity that renders proper operation no matter the environmental changes. So, in general, the second strategy is what is usually followed: improve autonomy adding control capability. However we should remember that these are just approaches for the *engineering* of autotonomy and not intrisically different types of autonomy.

1.5.2 Sources of need

We may consider what are the reasons why the systems do not operate properly. There are three main classes considering the standard core process of control systems engineering —specification, construction, operation:

External perturbations: when the system suffers an alternation in the operational conditions that suround it and induce a departure form the desired operational state.

Internal departure from specification: when the system changes and starts behaving in a different way that does not fulfill the original specifications and design.

Unmodelled dynamics: when the system operates as specified/built and in the proper operational conditions but it reaches an undesirable operational state that we failed to predict.

All the three cases can be considered part of two general cases of imperfect model of the system+environment in operational conditions and lack of capacity to have a perfect implementation of the system model.

1.5.3 Strategies for autonomy

Improving implementations —the mapping from specifications to realisations— is an obvious way to improve system autonomy. The model-driven approach, a genralisation of formal development methods, is the current engineering trend trying to tackle this problem. Later, section 3.6 will address this topic in more detail.

Improving models is the task of science and improving the model selection process is the task of design. Being the two sides of the engineering coin analysis and synthesis cannot be fully separated. Perfectioning our theories about the workings of reality will certainly improve the quality of our system+environment models. However, from the synthesis point of view, it will be necessary to reach an economical trade-off between model complexity and cost, tecnological limitation notwithstanding. But this limitations and trade-offs can also be modelled and embedded into the system inner knolwedge, hence making it more aware of its own limitations, something fundamentally missing in the control systems of the past[8].

In spite of modeling limitations —or perhaps because of them— one of the central topics in **ASys** is the pervasive model-based approach. This is based on the matching of two facts: i) the methods for engineering tend to become model-based (see section 3.6), and ii) autonomous control shall be based on model exploitation. Bacon's insight that *knowledge is power* can be mapped to the *model is control* motto for **ASys** .

Obviously, the models used by the agent in driving its behavior must to be in constant revision and tuning to be of any practical use in a changing, uncertain world. (42) for example claim for *"making models self-evolving, such that they continuously evaluate their accuracy and adjust their predictions accordingly"*. For some, this learning capability lie at the very core of intelligence.

An **ASys** will be using models —of its environment, of itself— to perform its activity. An **ASys** will be built using models —of it, of its environment. Model-based engineering and model-based behavior then merge in a single realisational phenomenon: *model-based autonomy*.

1.6 The ASys vision

In this context, stating the **ASys Vision** is simple:

[8] In the field of expert systems, a technology widely used in the implementation of intelligen controllers, the phenomenon of the system going beyond its cognitive capabilities is known as the *cliff effect* due to the enormous lost of perfromace when operating beyond the limist of knowledge.

It is possible to engineer any-level autonomy systems using cognitive control loops.

A cognitive control lop is a control loop based on knowledge, *i.e.* an internalised information structure that is isomorphic from a certain, useful perspective with some portion of reality. This knowledge —a model— will address the environment of the **ASys** , the **ASys** itself and the task that the **ASys** must fulfil in that context.

This vision does not intrinsically bound autonomy as it was argued for before, possibly going even beyond the engineering criteria limit for autonomy. We need an aproach to this that is both economically effective and boundable. We intend an *engineering process*, a *toolset* and an *asset base* that subserve this vision.

What follows in this document describes the **ASys** vision in some more detail; in paticular it focus on three core aspects:

- The Architecture-centric design approach (chapter 2): focusing on the architecture of the systems as the core aspect for guaranteeing a certain level of perfromance.
- The Seamless Process for Any-X (chapter 3): being able to reach any level of autonomy by means of a clear methodology.
- The Scaled workplan for asset construction (chapter 4): building modular elements to fill the roles specified in the architectures.

The report ends with some bits and pieces to be properly addressed in future releases and some bibliographic references.

Chapter 2

The Architectural Approach

The architectural approach seems the best alternative for tackling the inherent complexity of any technology fulfilling the **ASys Vision**.

Focusing on systems architecture is focusing on the structural properties of systems that constitute the more pervasive and stable properties of them (39). Architectural aspects are what critically determine the final ppossibilities of any information processing technology. Obviously this is true if any aspect of relevance is considered architectural, but the consideration that we do here is different: architecture —as the core set of organisational aspects— is what critically determine the capabilities. The form drives the function.

Architecture is key to function.

Many of the system capabilities sought in systems engineering are classified as functional and non functional capabilities (the first ones being critical for the provision of the service, the others considered ancillary aspects): this distinction becomes blurred in a detailed analisys; *e.g.* the functional/non-functional distinction dissapears when consider that some purely non-funcional aspect like *portability* can effectively hamper the provision of a certain service in a context of continuous platform change; the dream of the system built, frozen and exploited for years is nothing but a dream in real-world cybernetic technologies.

In the following section we will analise system traits highly affected by the architecture. This will be seen as a set of requirements to be met by the **ASys** architectures.

Later we will analise the architectural approach followed so far in previous projects to try to offer a path into these traits.

2.1 Engineering Desiderata

There are many system desiderata for fielded systems: *Functional*, the system perform the function it is intented for; *Flexible*, the system can be tweaked to

meet departing requirements; *ready, precise, integrated, autonomous, evolvable, dependable, cost effective, open, etc.*

These desiderata can be mapped into many system traits that critically depend on systems architecture:

Availability: The degree to which a system is operable and in a committable state at the start of a mission, when the mission is called for at an unknown, *i.e.* , a random, time. Simply put, availability is a measure of the time a system is in a functioning condition.

Accessibility: The degree to which a system can be accessed to its parts and workings for the different purposes of long term operation.

Constructable: The degree to which it requires exceptional effort to build the system.

Evolvability: The degree to which the system can be adapted to changing environmental and operational conditions.

Integrability: The dregree to which a system can be part os a bigger system (a systems of systems, SoS).

Integrity: The degree to which a system maintains its functional structure no matter what are the perturbations its suffering.

Maintainability: The degree to which a system can be kept into operation —by solving new appearing or old manifesting problems.

Performance: The amount of valuable result the system is producing.

Reconfigurable: The amount of form change tolerance to keep function.

Regulatory: The degree of compliance to commmon policies and regulations.

Reliable: As measured by its mean time between failures (MTBF).

Resilience: The capability of keeping function in the presence of perturbation.

Reusability: The possibility of resuing the system or parts of it in the contruction of a new system-

Scalability: The possibility of increasing the size of the task that the sytem is handling without negative effect on other traits.

Security: The protection against intrussion.

Upgradability: The availabity of a series of levels of performance with seamless transitions between them

See for example (31) for a mapping desiderata-traits in the context of military field applications.

2.2 Implications of Architectural Traits

The software engineering community, and in particular those related with the complex systems community, is becoming progressively aware of the importance of software architecture. Architecture based development is seen as the "good way" to achieve high levels of software quality. Especially those referred to non-functional requirements for the software systems.

As (10) say, architecture based development offers promising perspectives that we are trying to achieve for autonomous control systems:

- Systems can be built in a rapid, cost-effective manner by importing (or generating) large externally developed components.
- It is possible to predict qualities analyzing the architecture
- Developing product lines sharing architectural design
- Separation of interface and implementation at the component level
- Advantages of design variability restrictions

Based on domain analysis, generic architectures are proposed to address a whole bunch of applications in a specific domain. See in Figure 1 to the architecture based process proposed by the ARPA STARS project. Development is divided in two separate phases:

Domain engineering: One effort shared by a complete collection of instances of a product line. Its final product is threefold: a domain model, a generic architecture and some reusable components.

Application engineering: The process of getting the real applications (the products in the product line).

A great level of effort has been put in the last years in addressing the architectural problem for all types of software systems. In the case of autonomous systems, many generic architectures have been proposed and components developed. Stunningly, most of them use a layered approach based on three levels of control.

2.3 Core Architectural Aspects

As commonly defined in many contexts, a *system architecture* describes the topology of system elements, specifies the element interfaces, and identifies functional models associated with those element arrangings.

From the experience gained in the past —in our own research and also in others'— we have reached some design principles of major importance in **ASys** :

Multi-level abstraction: Organise the system in layer according to the abstraction level of the informational entities used.

Functional modularity: Modularise the system in terms of functions —and not platforms, or technologies, or domains, etc.

Standardisation: Follow/establish standards and standardised practices.

Composability: Build complexity by decomposition/composition.

Late design: Postpone any design decission that is not necessary to do at any point in th life-cycle.

...

A core objective of **ASys** should be to address these and other design principles witht breadth and precision.

2.4 Domain Specific Architectures

Domain Specific Software Architectures (DSSA) was a trendy approach in the nineties[1] that still has a lot of hold in the different technological domains of interest to **ASys** .

DSSA-based technology is grounded on the concept of an accepted and practiced generic software architecture for the target domain: *e.g.* process control, avionics, manufacturing execution, command and control, *etc.* To be a domain architecture the architecture must apply to a wide range of systems in the domain; then being general and adaptable.

Over an established architecture is is possible to build the system by populating the empty architecture with the right components in the right places. The components must necessarily conform to the architecture design specifications, *i.e.* they must implement their functionality and offer it —or request from other components— within the proper interaction framework. See for example the architectural guidelinges of our former COMPARE project (13).

Reusable components to fulfil architecture-ladden roles may be acquired by identifying, gathering, tyiding and adapting existing components or by specifically creating them. See for example the Object Management Group OMA Approach (26). One of the ways they may be created is through automated component generation; *e.g.* DARPA has sponsored work in this area at USC Information Sciences Institute —the AP5 application generator project (7).

The existence of a domain-specific architecture and conformant component base was promising to enable a significantly different approach to software-intensive system development. This promise is still not fulfilled as demonstrate for example the GENESYS project recetly started (see ASLab website for more info).

[1] http://www.umcs.maine.edu/ larry/latour/WISR/ contains proceedings of workshops of that time.

In theory a developer using a DSSA and a component base will not wait until detailed design or implementation to search for reuse opportunities; instead, he will be driven by the architecture throughout. Design will use the architecture as a starting point. Contruction will use the architecture as scaffolding. Maintenance will use the architecture as the way into the understanding of the legacy. The domain architecture and the asset base reify the bunch of requirements and allow design and construction of rapid prototypes or final applications.

The **ASys** project has as domain the domain of autonomous systems; ill defined and too broad. However, there are technologies trying to address the general issues and are open to embrace new requirements from particular domains. IN the past, starting from the DIXT Project[2] we focused ourselves on the technology around OMG's Common Object Request Broker Architecture. A DSSA for the domain of distributed, object-based systems extended with some basic capabilities for real-time embedded systems.

As a result of this and following work one of our core assets —the ICA Broker— was produced.

2.5 Architectural Patterns

Patterns are the way for capturing the concrete organisations that offer the system-level functionality.

2.6 Dimensions in Any-X

We intend an architecture, an engineering process, a toolset and an asset base that let us build systems that show the Any-X property.

Any-X means that whatever the positioning in the design space the architecture/process/toolset/assets are stil valid. Pure hubrys.

Engineers like to use a sigle technology for everything (hammer-nailing).

A description of some of the possible dimensions for engineering design follow.

2.6.1 Domain

Universal techlogies are domain-neutral. In ASys we intend at least the following domains:

[2] http.//www.aslab.org/public/projects/DIXIT

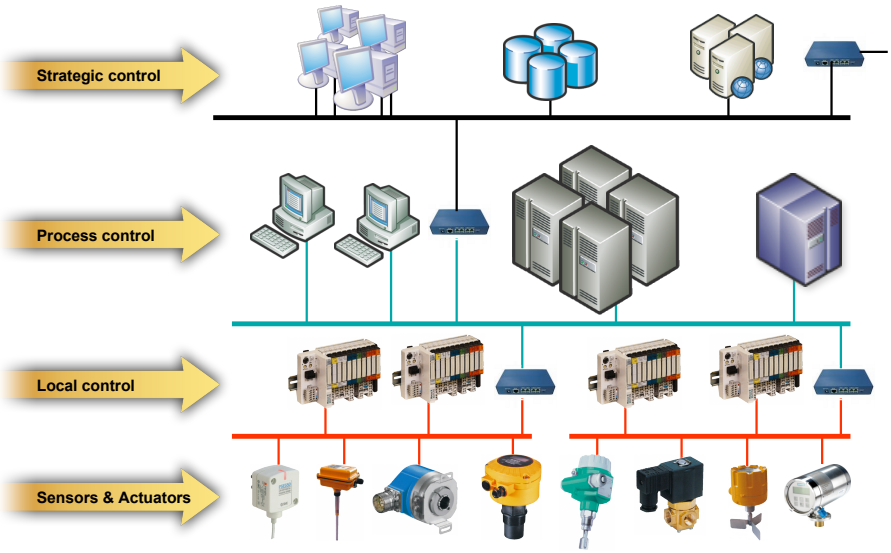

Figure 2.1: A hierarchical distributed control system (DCS) of an industrial plant is structured in many different control layers. Control objectives gain in abstraction level when going upwards. Temporal criticality and precision grow when going downwards toward the plant.

Process control : The plant is a continuous process plant: a refinery, a power plant, etc.

Robotics : The plant is a mobile robot.

Vetronics : The plant is a transportation vehicle.

Networked apps : The plant is a networked computing infrastructure.

For some of them we will synthetise tangible systems. For ohters we may keep ourselves in the world of analysis.

2.6.2 Size

From the process plant control (see Figure 2.1) to the embedded devices of modern cars (see Figure 2.2).

A hierarchical distributed control system (DCS) of an industrial plant is of enormous technical complexity. The number of elements involved —its size— is so high the thay must be structured in clusters organised in many different control layers. (32) describes size as the primary form of *complexity*.

In these hierarchies, ontrol objectives gain in abstraction level when going upwards and temporal criticality and precision grow when going downwards toward the plant. This was described by (36) as the principle of *increasing precission with decreasing intelligence*.

The size problem that is so obvious in the large plant controller is also becoming a major issue in the field of deepley embedded devices. As a paradigmatical example, modern vetronics (see Figure 2.2). Vehicle electronics has been playing a vital role since its introduction in the late 70's for the engine control but during the last decade most of the automobile innovations have been made possible by means of a massive use of electronics and embedded systems —nowadays it is largely agreed that around 90% of the vehicle innovation is enabled by this.

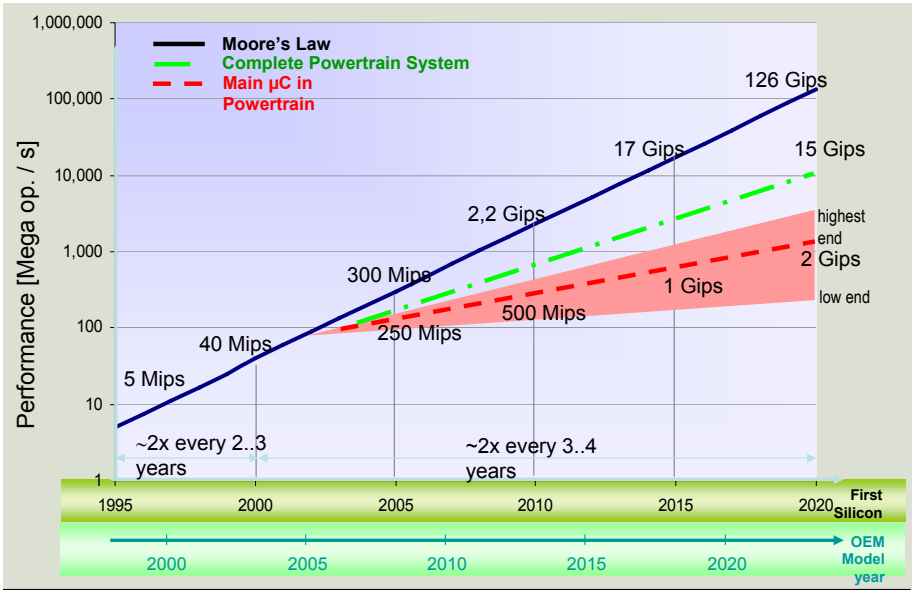

Figure 2.2: Estiated evolution of computing performance requirements for embedded powertrain control. This is just a piece in integrated vetronics.

The typical low-mid end modern vehicles are equipped with 25-30 electronic control units (ECUs) while the high-end cars could reach up to 70-100 control units, each one hosting circa 180-600 components, according to the different level of complexity (not only IP cores but discrete electronics).

The most demanding applications, mainly dealing with powertarin control, body control and safety issues, are currently using software-intensive microcontroller and a more complicated architectures are in the process of being introduced[3]. More information on these topics can be found in the IST GENESYS project we are partners of (www.genesys-platform.eu).

2.6.3 Time

The time scales for autonomous operation will range from the lifelong cycles of long-term strategies to the minimal reaction times of safety measures in

[3]Examples of extant systems are diesel Common Rail injection systems, innovative diesel technologies for the NOxreduction, valve electronic control systems, injection and after treatment systems for CNG engines, dual dry clutch transmissions, hybrid and alternative propulsion systems, etc. not to talk about onboard infotaiment.

the pressence of fast dynamics (see Figure 2.3).

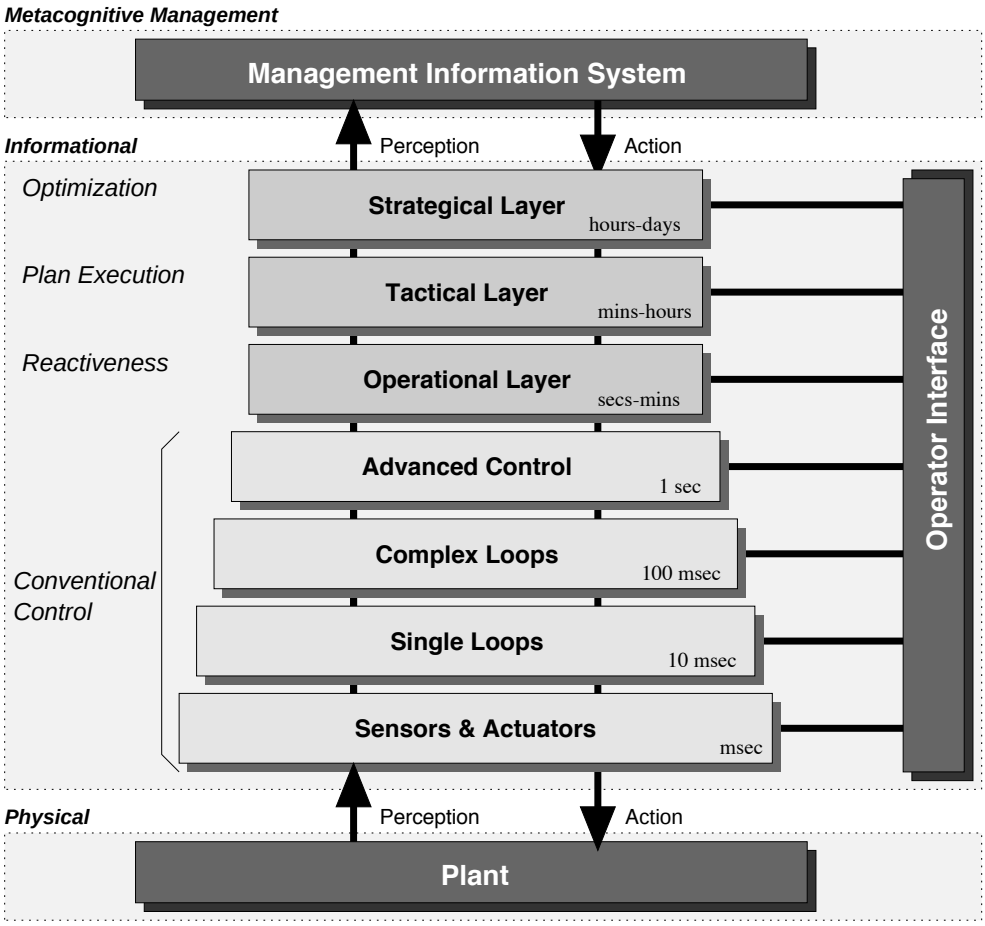

Figure 2.3: The temporal landscape in a whole-plant integrated hierarchical industrial control system ranges from the homeostatic and fast responses to local goal settings and conditiosn in the temporal range of milliseconds to the days-weeks-months strategic control horizon of the upper layers.

2.6.4 Intelligence

The general principle mentioned before of *increasing precission with decreasing intelligence* (36) mentioned before can be read also in the intelligence dimension.

Here we enter cloudy waters as the issue of *what is intelligence* is still unresolved. However, in the naivest sense of it we understand that the information and knowledge processing capabilitites needed for performing different tasks will range enormously.

From the elementary regulatory loops dealing with a single, 12 bit-discretised, analogical magnitude measure to the multiobjective —production, quality, safety, maintainnability, *etc.* — strategic control of process plants (see Figure 2.3).

In a sense, the very nature of the knowledge and knowledge processing capabilitites needed to perform these function sits at the core of ASys; for if we are able to cleary set the requirements and technology for this, most of the general objectives would in some sense derive from it. This vision is what led the knowledge-based control of the not so recent past (35).

2.6.5 Robustness

Something that makes a big difference between a laboratory experiment and a real-wold deployed system is it robustness. Strongly related with issues of reliability of systems, the massive incorporation of high doses of software into the technical systems has rendered systems somewhat more fragile.

In other order of things, the assumptions and necessary abstractions done when appliying an infromation system to the real world —e.g. when assuming that we can represent an acceleration with 8 bit for firing an airbag— are sometimes worse than misguided. Extreme examples can be found in all extremes of the abstraction specturm; from space missions that fail due to a units mismatch to the well known *cliff effect* of expert systems —a phenomenon of highly non-linear unfitness of domain knowledge.

At the end of the day, achieving robustness is not only a question of just doing things properly, but a continuous engineering trade-off between what we know how to do, what we can do on time and technological capabilities and what we want to do in terms of economics and policy.

2.6.6 Cost

This takes us to the question of costs. Robust systems are expensive as quality is. The trade-offs mentioned before may imply that we can trade autonomy or robustness —or whatever— for money; but this must be done in a sensible way.

2.6.7 Platform

Last —many other dimensions will ever remain— is the question of platform. The technology for autonomoy must be deployable on any platform that fulfils the requirements. This is the dream of MDA and we wan this here mainly due to the platform impossitions from different domains.

Chapter 3

The Seamless Life-Cycle

3.1 The Process-Product Gap

In all the text before, we have been talking from two viewpoints: engineering —mainly design— and runtime. We want to highlight a simple fact: Due to the gap between engineenring and operationa phases, technical systems operate at *runtime* with their traits and capabilities set and frozen at *design* time.

A major characteristic of the engineering life-cycle is that we cannot but recognise the impossibility in separating the system as it is from the engineering process used to make it.

Why all this makes a problem? Because as was said before, systems will have frozen traits that do not respond well to change and uncertainty.

Break the design/runtime dichotomy.

The problem is not just a problem of frozen capabilities but indeed of lost capabilities. Capabilities that available during the implementation process are no longer available in the final product, in particular, regarding reconfiguration.

Consider the classical phasing of a well-engineered-project:

$$inception \rightarrow analysis \rightarrow design \rightarrow construction \rightarrow$$

$$\rightarrow deployment \rightarrow maintenance \rightarrow decomission$$

The product, tangible or not, is usually not fully coupled with the (production) process. The whole process, from the user needs to the delivered product should be treated as a unique, tightly coupled (or at least related), process to maximise efficacy and fitness.

What we need is an intrinsic, runtime capability, for continuously updating the system by the system. We need the system to be a product of its very own

process.

Break the process/product gap.

This is obviously the case of *autopoiesis* (22) but with a teleological limitation: system's autopoietic processes must converge into required function.

When talking about system we usually mean the physical system (set of equipment) that we are interested in. But, in a sense, what we are really interested in is not the system bu the product that the system produces. For example the electrical network would be the system (and the electricity the product), the chemical plant (just the unit operations equipment) would be the system and the produce chemical, the product, the car (without any control system) would be the system and its features (the car as well) would be the product, *etc.*

3.2 A Seamless Engineering Life-cycle for Systems

There are two main scenarios for system engineering: i) the system does not exist and has to be designed from scratch and ii) the system already exists (but does not comply with the user requirements).

The first engineering scenario is stated as *systems engineering*: given the user product-level needs (electricity available....) design the overall system (physical system plus control) that produces the product.

The second engineering scenario is stated as *control engineering*: given an already designed/extant system (the electric network) and some user product needs (electricity available at any time without interrumptions,....) design the control system that make the system fulfill the needs.

The engineering scenario we are interested in ASys is stated as *autonomy engineering* and can be stated in crosscutting terms: given the user product-level needs (electricity availability ...) design the overall system (physical system plus control) that *keeps producing* the product embracing unexpected change.

What this necessarily imply is that what is kept as target setpoint for the autonomy loop is not the system *as structure* but the system as *set of processes* that produce the product. So the objective of autonomy is not self-sustainment of systems but self-sustainment if functions.

The vision of a *seamless engineering process for autonomous systems* captures this capability by embedding into the system the very engineering mechanisms used in its construction. There are no seams between design and runtime.

Artificial autonomous systems are self-engineered systems.

Self-engineered systems are a subclass of autopoietic systems that realise a teleological process towards a specified function (so, in a sense, function is the

setpoint injected in an autonomous controller). These systems will embed the capabilities of design-time engineers for their runtime operation. Obviosuly the engineering competence requires high-level cognitive performance but there are lots of efforts done into the automation of systems analysys and design tasks[1]

The question of the seamless system life-cycle and the breaking of the gap between engneering and exploitation requires the construction of technological assets to implement the automated self-engineering processes. These assets can be divided in three broad classes:

Models: That capture the information regarding the system (*i.e.* are systems representations)

Engines: That exploit the models to produce new information.

Infrastructures: The provide the adequate scaffolding for models and engines. This, in principle, should be the very cognitive architecture.

Self-engineered systems leverage models and engines.

Another interesting source of information regarding cognitive tasks in systems engineering comes from the so-called field of *cognitive engineering*. Cognitive engineering is an interdisciplinary approach to designing computerized systems intended to support human performance of complex, systems-oriented tasks (30). Figure 3.1 show a potential life-cycle for engineering problem solving depicting various phases of the system engineering life-cycle where cognitive engineering methods may have a substantial impact.

3.3 Product, system and process characterisations

In order to implement this engineering activity several characterizations have to be made in advance or as a result of the activity: product characterization, system characterization and process characterization. Characterisation implies the precise bounding of operational quantities of the system concerning both sturcutural and behavioral aspects (incluing the I/O aspects that determine the system utility).

Product characterization: This is the set of goals to be pursued concerning the material, energy or information output flows of the system. This is always imposed externally by the users or designers in an artificial system (40).

System characterization: A proper system characterization will guide the production process design. If the system is hardly observable or fully

[1] Klir and Elias (18) identifies three big classes of systems problems: systems analysis, systems inference and systems design.

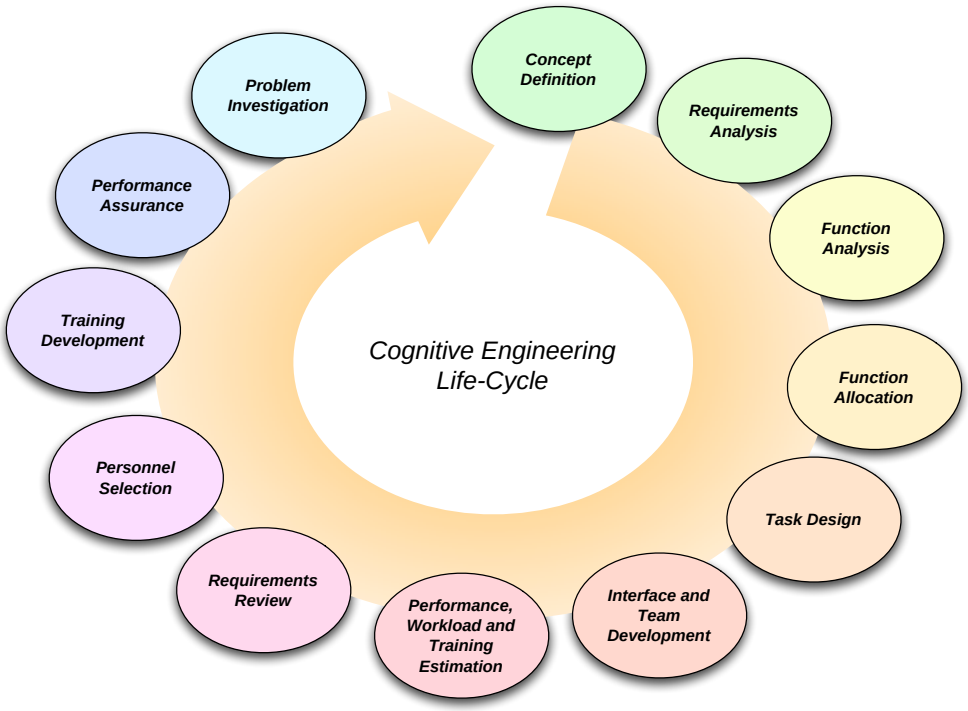

Figure 3.1: A potential life-cycle for systems engineering problem solving depicting various phases of the system engineering life-cycle where cognitive engineering methods may have a substantial impact.

observable will mean different processes and control The architecture (cognitive, RCS-like, SOUL or other) will be different (although the architecture basic elements will be always the same, what changes is the node interconnection and the nodes internal elements implementation). If there is a lot of knowledge of the system and it can be solved (it is not so large or not so non linear that conventional numerical algorithms cannot solve it) will mean one type of element in the node, if the knowledge is scarce will mean another implementation in the node (neural networks, FSM,...) And so on with all the properties of the system.

Process characterization: The properties of the production process (control system or system plus control) will come partly determined by the system characterization and partly by the product characterization. It is clearly different the security of a nuclear plant that the security of a chupa-chups factory.

The conclusion is that the reification of the design of a well characterized system (including product and process) can be the guide for engineering autonomy into the system.

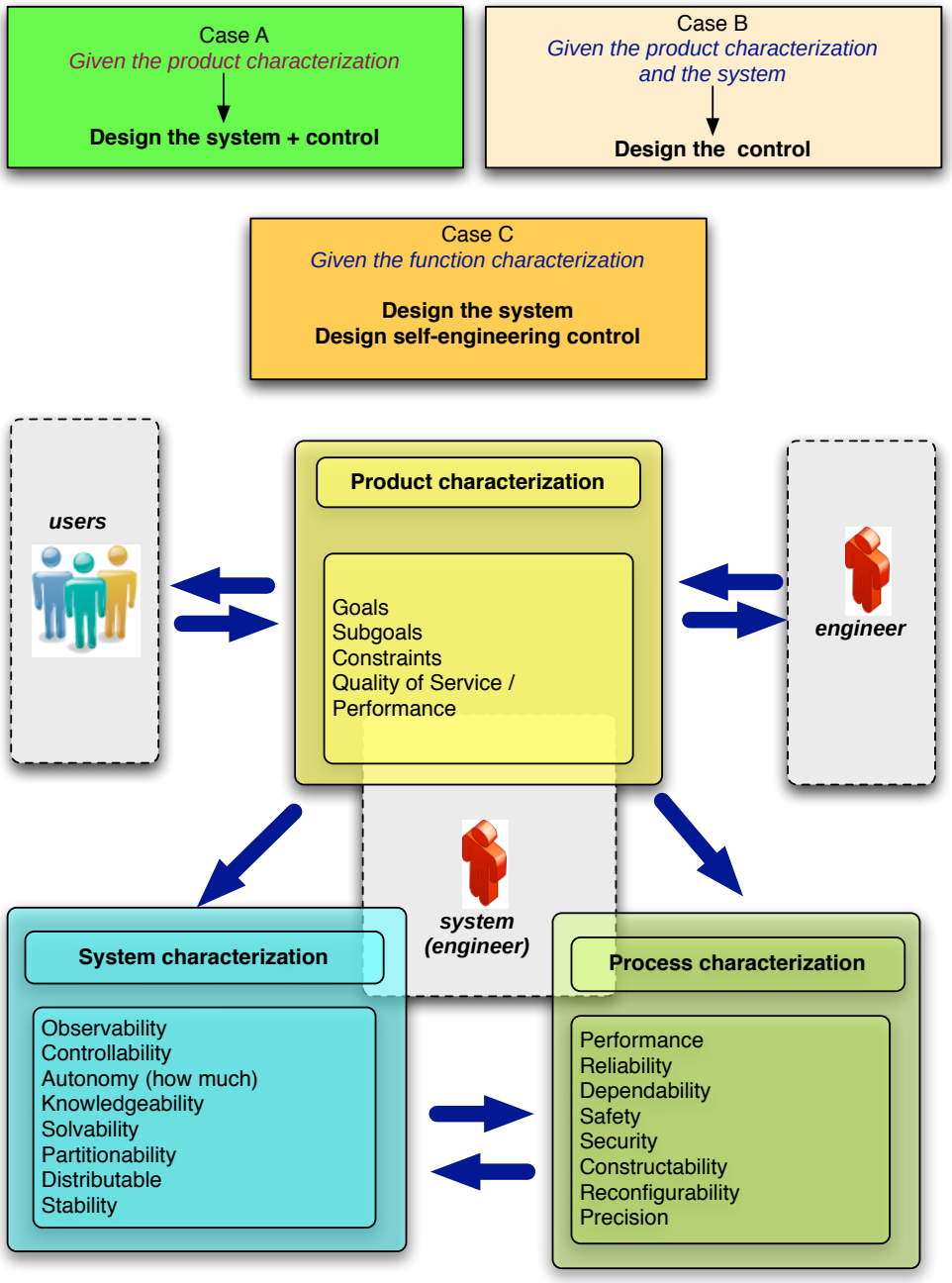

Figure 3.2: Some of the aspects of product-system-process characterization.

3.4 What's first, the process or the product?

Notice that the central problem now is how to design a system as a set of coupled processes in such a way that it produces two things: the system itself and the product.

Now we are close to the realm of biology. In a sense, the focus of life studies is the first one —the system— and the focus of engineering is the second —the product. In fact the world of bioengineering is the other side of the coin from us. We start from technical systems that have productive capabilities trying to add self-sustaining capabilities. They start from biological systems trying to add productive capabilities.

If we observe with detail, the bioengineering movement is coming from life studies and the dependability movement is coming from engineerig studies into the common grounds of robust behavior[2]. What is crucially important then is the maintenance of the organisation.

And we need more *science* for this that shall go beyond the wishy-washy talk of present day post-modern robotics and cognitive science and produces realible theories to base our engineering upon. We can quote Schrödinger here:

> *How would we express in terms of the statistical theory the marvellous faculty of a living organism, by which it delays the decay into thermodynamical equilibrium (death)?*
>
> Schrödinger (37)

Obviously we are in the realm of *artificial life*, but with a tint. The ASys approach to autonomy somewhat unifies engineering and theoretical biology, not just gives clues on life by building life simulations. The *product* requirement *shall be always met* in engineering.

3.5 Progressive Domain Focalisation

As was said before, the seamless system life-cycle requires the construction of technological assets to implement the automated self-engineering processes (models, engines, infrastructures). These assets will constitute the asset base that constitutes the central resource for atacking the problem os autonomous systems construction using a product line approach.

Product line engineering is the most sensible approach for the development of a collection of systems that fit on a particular technological niche (driven by user requirements). The produc line approach scales into what is called domain engineering when the technoogical assets are built to cover the wide spectrum of a domain (as the any-x vision claims).

[2]Note that biological metaorganisations — *e.g.* biofilms— are much more resilient than

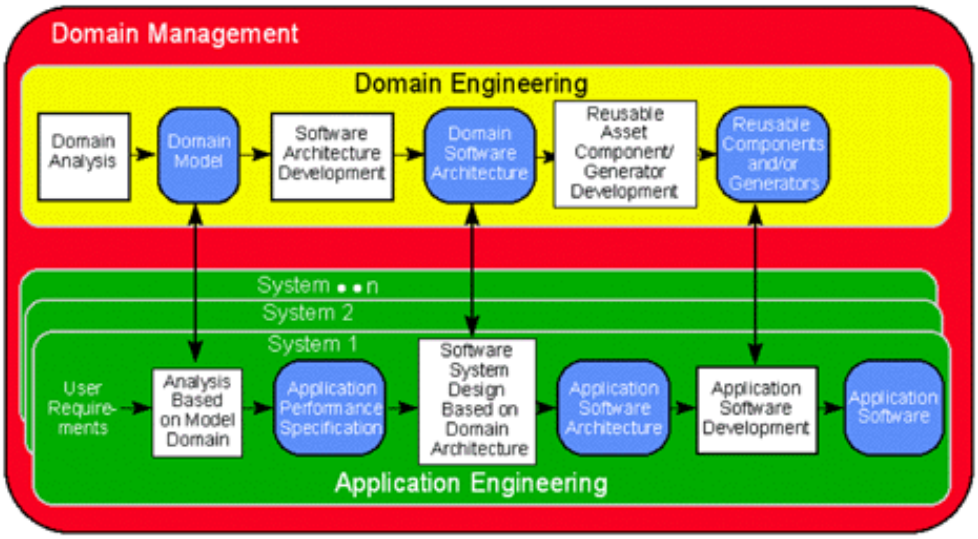

Figure 3.3: The process and products of the domain engineering activity organisation (from (15)).

Figure 3.3 shows the process and products of the overall domain engineering activity. The figure shows the products, tasks, relationships and interfaces of domain engineering to the conventional system engineering process (for a concrete system). This approach is the result of the DARPA-funded STARS program on systematic reuse and is known as the double life-cycle model.

In (34) we proposed a variant of the double life-cycle to accomodate the possibility of cross-domain engineering processes. In this approach the reusable assets can be of different levels of abstraction being of applicability to different domains. Domains are seen as progressively focused and assets can be built for any one of them. Obviously the more abstract the placement of a concrete component the more domain-wide but also less efficacious and usually the more difficult to build. This approach is depicted in Figure 3.4.

Target assets to their maximum potential level of abstraction.

3.6 Research focused on models

Models permeate **ASys** and are the cornersone of development and system operation.

Models are the engineering way to capture systemic visions —requirements, designs, *etc.* . Models are used in all branches of engineering. Even in the world of software-intensive systems, models are becoming the central drive of the engineering process (3; 9).

isolated cell soups for the production of a ceratain service: synthesis, catalysis, bioletricity, *etc.*

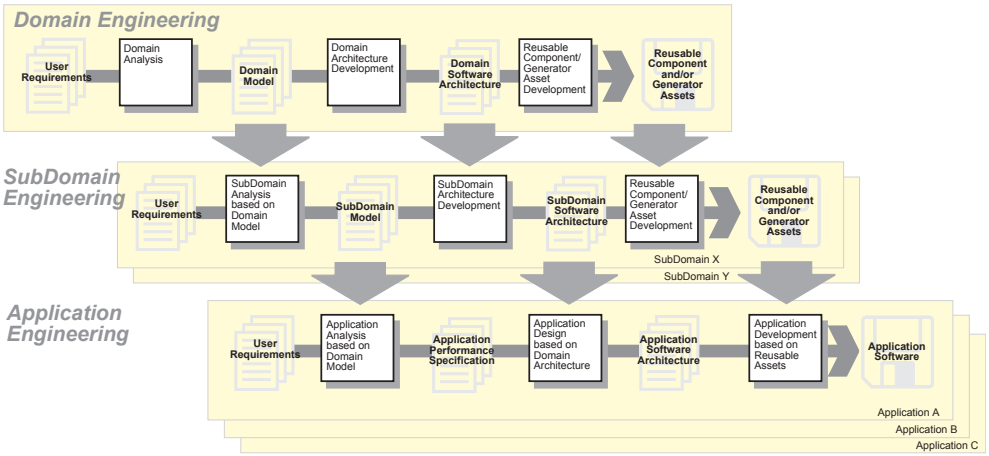

Figure 3.4: The progressive domain focalisation approach to product line engineering.

The ASys vision makes an step further using models as the very runtime representations used by the systems themselves during their operation. The age old issue of knowledge representation —frames, semantic nets, rules, *etc.* — coalesces into an unifying view on *models*.

This is in line with a perspective in cognitive science that holds that mental content has the form of models. Initiatied by Craik in the fourties (14), the theory that minds are based on the construction and exploitation of mental models — *i.e.* the *mental model theory of mind*— has had a long history that somewhat surprinsignly hasn't got enough hold. While it is clear that mental *representation* of entities has had a long journey in philosophy and psychology, the question of more complete, more ecompassing, more integrated models has not attracted big attention. Representation and modelling seem to imply similar cognitive structuring but, however, they have very different implications. We are not going to enter in detail the theories regarding representation and will focus specifcally on the issue if mental models.

The question of mental models is not restricted to theories of mind in but is also a relevant topic in many other disciplines, *e.g.* human computer interaction or in management. Let's see a definition targeted to corporate management:

> *deeply ingrained assumptions, generalizations, or even pictures or images that influence how we understand the world and how we take action. Very often, we are not consciously aware of our mental models or the effects they have on our behavior.*
>
> Senge (38)

Chapter 4

The Scaled Workplan

4.1 The Work to Do

ASys 's main purpose is oriented towards the development of autonomous systems (**ASys**), taking into account all the phases of the life-cycle. This is a very ambitious goal and needs a lot of work regarding many different areas. From conception and design to the final deployment and maintenance, all phases are to be taken into account; but, obviously, design, construction and operation will possibly preseny the mosta challenging aspects.

The whole ASys engineering process goes from the raw material (specifications and knowledge) to the final product (the physical autonomous system). There are many things to do and many assets to build to achieve an effective domain engineering process.

The scaled workplan identifies the set of core assets and establishes intial criteria, scheduling and milestones that should be accomplished in the process of developing the technology for ASys. A description of each of the items/tasks follows. Finally, these tasks are sequenced in a tentative roadmap, shown in figure 4.2.

4.2 The Too Many Threads of ASys

In the **ASys** long term project there are many aspects that are intertwined: systems, engineering, control, intelligence, etc. The following ongoing topics give a partial vision about some of those aspects:

ICa : An Integrated Control Architecture. The reference framework for modular construction of complex software-intensive controllers.

ICb : A broker for the Integrated Control Architecture. The core software infrastructure-

OASys : An ontology that underlies all **ASys** efforts. To have in the mind of the engineer; to have in the mind of the ASys; to be used in the synthesis

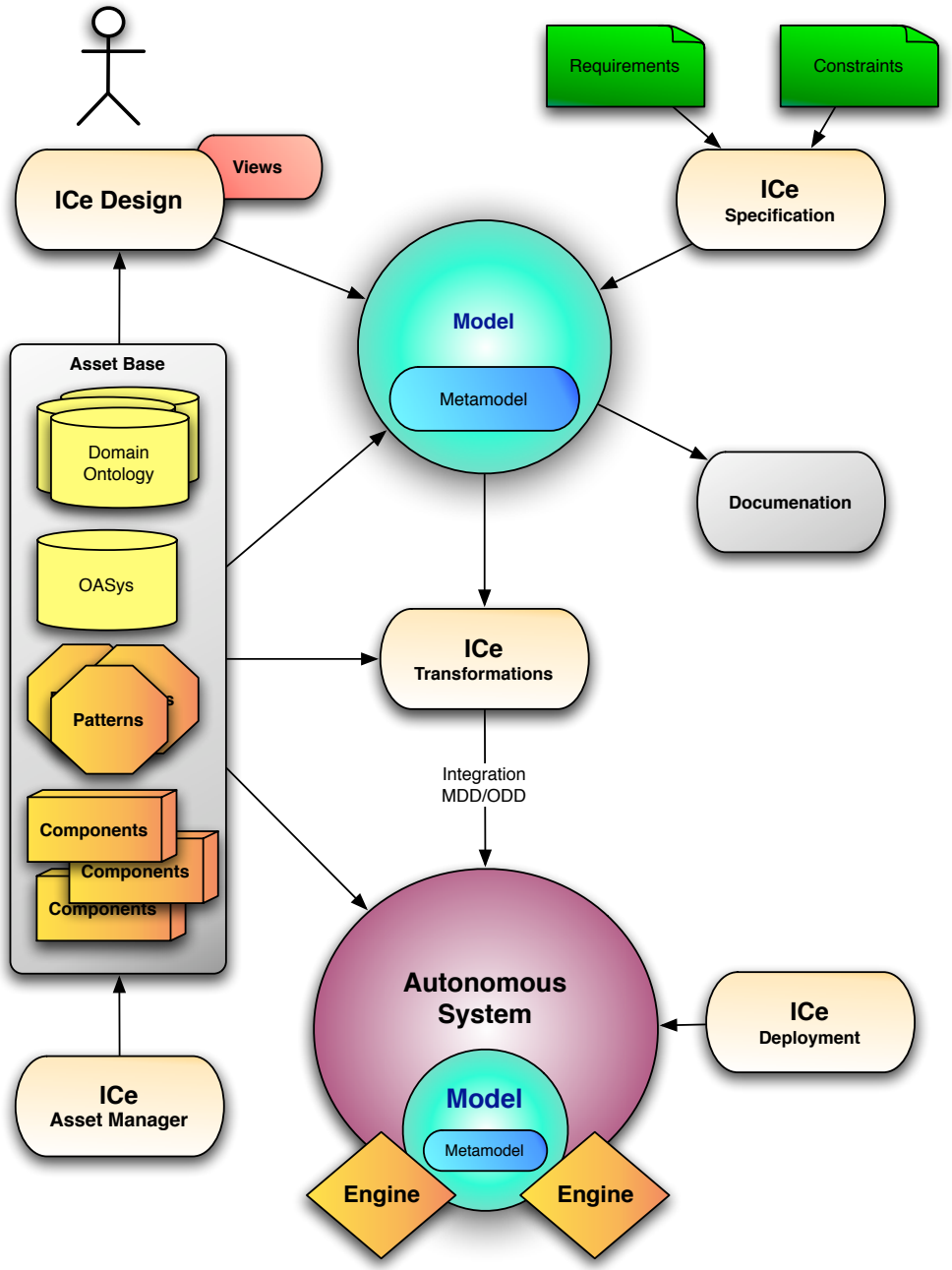

*Figure 4.1: Some of the core tasks and products in the realisation of the **ASys** Vision.*

of the ASys modules.

SOUL : An Universal Cognitive Architecture with Self-Awareness. A Metacognitive real-time architecture that realises the model-based

ICm : An Integrated Control Methodology. The specficiation of an engineering process to build ASys.

RCT : A Robot Control Testbed. An application case-study in robot control.

PCT : A Process Control Testbed. An application case study in process control.

4.3 Ontologies

Ontologies are very important as they are intended to be a conceptualization to describe the knowledge of an specific domain. This knowledge (ideally) should be reusable and understandable by different people and communities. Regarding the use of ontologies for**ASys** we can distinguish between (mostly existing) domain ontologies we can profit from and the (in development) ontology for **ASys** (or **OASys** or metaontology or transversal ontology).

4.3.1 Domain Ontologies

In this part, a survey of existing ontologies related with the area where the applications are going to be developed should be performed. Initially, mainly in the area of software, process control systems and robotics. Generic ontologies (as physys,...) should be explored as well. This work should also address the issue of ontological formalisms.

→ Task: *Domain Ontologies* / TBD

→ Product: *Domain Ontologies Technology Survey* / Report

4.3.2 Autonomous Systems Ontology

The ASys ontology (OASys) captures the concepts of the ASys research programme and reify them to provide software support for the process of ASys engineering. Tt is necessary to have this ontology not only to describe the ASys but to help and guide in their development as a provision of conceptual and software assets. The need of this ontology has been discussed and justified by the arguments of the preceding sections.

→ Task: *Ontology for autonomous systems (OASys)* / TBD

→ Product: *OASYS Concepts and methodology* / Report

→ Product: *OASys* / Software

4.3.3 From ontologies to models (O2M)

Ontologies are to be used. An ontology has many uses (just to describe a system or to have a common understanding). In this part the transition from ontologies to models is studied. While ontologies can be considered as models, they appear as static captures of conceptual structuring. The differences between ontologies and models will be clarified and the missing parts identified.

One of the central goals of this task is to produce a submethodology for ICm to exploit the knowledge gathered in the ontologies and generate models based on that knowledge. Models will have permanent links with the corresponding ontologies so they can be updated (or they can modify the ontology).

In a first impression ontologies are just partial models and the ontology-model relation will be an instance of a more general *model-uses-model* relation.

→ Task: *Ontology language selection* / TBD

→ Task: *How to link models and ontologies* / TBD

→ Task: *O2M Translation tools* / TBD

→ Product: *Specification of Ontology Languages* / Report

→ Product: *Ontology Toolset* / Software

→ Product: *O2M Methodology* / Report

→ Product: *O2M Translation tools* / Software

4.4 Models

Models are critical for this research approach. Models are the core, around them everything is developed. Models are important as they are the way that everybody (human and systems) knows how to interact with the real world. In this section three items related with models that deserve further development are considered.

4.4.1 Model characterization

Model is a very generic word. In every domain everybody uses models. Everybody defines what a model is. There are many types of models, and even different classifications of models. Thus, it is important to characterize them. What is a model for us? How to define a model? What are the best models for our applications? What is the best way to implement, code, these models?

Many modeling languages have been used in the different domain of ASys and we strive to find a common unifying framework for getting the best from

them. A central issue is related with how the model is going to be used because this defines a big part of the modeling strategy.

The use of SysML to build this general framework for models has to be thoroughly explored. Model requirements is another point to take into account. User requirements, designer requirements and requirements imposed by the actual system should be clearly identified in order to guide the development of the best (which means the simpler models that complies with all the requirements) model.

→ Task: *Meaning and models* / TBD

→ Task: *Model definition* / TBD

→ Task: *Model requirements definition* / TBD

→ Task: *Modeling language selection* / and requirements

TBD → Product: *Modeling for ASys: models and model uses* / Report

→ Product: *Specification of ASys Modeling Framework* / Report

→ Product: *Modeling Framework* / Software

4.4.2 Model Development Methodology

A methodology to develop the models have to be established(19). Existing software development methodologies like MDD (Model Driven Development) or the coming ODD (Ontology Driven Development) may be good approaches to adapt to the development of more general models (not only software). This can (is) related to the O2M issue.

→ Task: *Advantages and limitations of MDD, ODD and Other Approaches.* / TBD

→ Task: *Define ASys Modeling Methodolgy* / TBD

→ Product: *Model Development Methodologies Assessment* / Report

→ Product: *ASys Model Development Methodology* / Report

4.4.3 Model Integration

The most important aspect —from a practical point of view— is how to use the built models. The best possible approach is to build models in such a way as to allow the many possibilities they have. This is what guided the work in HOMME Rodríguez and Sanz (29).

Obviously this is in strong relation with wat we called *engines*. Models that can be easily embedded in commercial applications that provide the adequate engines (generic like Matlab or domain specific like Aspen) or can be used by their own by generating model+engine executable modules.

→ Task: *Model uses analysis* / TBD

→ Task: *Model compilation* / TBD

→ Task: *Commercial environments integration* / TBD

→ Product: *Model and Engines Analysis* / Report

→ Product: *Model Compilers and Engines* / Software

→ Product: *Commercial environments integration tooling* / Software

4.5 Beyond models

A model is a simplified representation of the reality. A model may have as many views as reality has. The question is to define the more interesting (useful) views to be included in a model. It seems that two phylosophically fundamental views —functional and structural— are mostly important and have to be considered.

4.5.1 Functional and structural views

The functional view ir related to the performance of the system modeled. It may contain all (part of) the behaviour of the considered system, and distinguish between the intended behaviour (the one that is designed for) and the not intended behaviour (collateral behaviour, many times denoting malfunctioning). This view is very useful to perform different analysis on system, like the risk assesment analysis. A follow up of the functions that fail as result of an initial failure as well as the possible causes of a function failure can be obtained using this view.

The structural view is related with what the system is. The components that compose the system. The hierarchical structure of components. Each component may have one or more functions associated. A relationship between views exist but a good methodology to pass from one to the other have to be developed.

A model contains all the views considered in its design (or imposed by its requirements). The problem now is how to extract from the model a particular view (perhaps it is not interesting to extract it as separate entity but at least to use it that way). Is it interesting to derive views in the format needed by the algorithms that are going to exploit it.

→ Task: *Functional View Definitions* / TBD

→ Task: *Structural View Definitions* / TBD

→ Task: *Extracting functional and structural views from a generic SysML model* / TBD

→ Product: *SysML for functional view* / Software

→ Product: *Structural Hierarchy with SysML* / Software

→ Product: *Integration of other functional techniques* / Software

4.5.2 Metamodels

A metamodel is a model of the model[1]. This is really a crucial part in the ASys research program.

If the research is focused to the development of ASys that are able to implement the seamles process vision, it seems unavoidable to have metamodels to enable the agent to think about its very own processes. It seems that this is the way to provide the required level of self-awareness to the system. The system has to know its own mental state and its current and actual possibilities (to know it something is impossible for it the agent has to estimate it using a model of itself, the metamodel).

A characterization of these models similar to the one commented for the models is needed. But, in a sense, the very core OASys and the associated modeling methodology must necessarily address some of the issues of relevance here. The procedure is alike and many things will be directly applicable (as metamodels are nothing but models) but new, extra things may have to be considered.

The integration between models and metamodels is an open issue due to the different asbtaction leveles. This must be resolved at the level of modeling techniocloy as this masut provide support for the progressive domain focalisation concept that will raise similar issues of interaction between different levels of abstraction. There are may issues —coherence, scale, common representation, *etc.* — to be studied. The same for rutime execution and engine implementations that must address also a span of differen abtraction levels. Here, the methods of general systems theory (17) may be of some utility.

→ Task: *Metamodeling analysis and definition* / TBD

→ Task: *Metamodel - Model Integration* / TDB

→ Product: *Metamodeling Methodology* / Report

→ Product: *Metamodeling Engines* / Software

→ Product: *Reusable Consciousness Models* / Software

4.6 Architecture

The ASys focus on architecture seem to imply that it is necessary to have a concrete architecture to implement all the items described. Obviously domain architectures will be useful for some kind of applications and in any case, each application will have an architecture

Many different and types of architectures exist. The Integrated Control Architecture (**ICa**) provides a model for application engineering much in the

[1] *"Anything you can do, I can do meta!"* was a pun that Samuel Hahn did in an humorous response to some rather philosophical argument about programming higher-order functions (functions that build other functions) in LISP or Scheme.

line with distributed agents architectures. The RCS architecture offers a hierarchical strategy to scale the control to levels of high intelligence.

4.6.1 Architecture suitability and requirements

The RCS architecture is a hierarchical and heterarchical architecture that decomposes the system under study in small pieces as the best way to know how it behaves and to establish and action plan. In this architecture every node has all the elements (knowledge-ontology, world model, *etc.*). A thorough study of this and other architectures, their suitability, lacks and extensions are to be studied.

→ Task: *Extant Architecture Study. Components and performance* / TBD

→ Task: *Matching architectural elements with ASys: models, algorithms, views, ontologies* / TBD

→ Task: *Reflexity Architecture Study. Architectures of self-awareness* / TBD

4.6.2 Architecture. Putting all together

Two aspects shall be managed here. First is the desire to have a single universal architecture and second the degree to which such an abstract view can render truly effective systems. The idea to explore is that of architectural patterns that capture ways of organising components in functional subsystems. This gives a way to immediate realisation without losing the abstraction capability. Patterns may go from low level, low scope patterns (*e.g.* a filtered sensor pattern) to top level general system architecture.

Once the architecture is chosen, all the elements have to be implemented. Patterns provide the scaffolding and components are used to fill the roles. The quation of how to implement or integrate coded models is solved by providing pattern-based transformations into code.

→ Task: *Architectural Patterns Capture* / TBD

→ Product: *Implementation of Patterns and Components* / Software

Although it may be considered when characterizing models, two aspects have quite importance by themselves:

Distributed models The advantages of distributed (federated) models. Its possibilities. Its limitations...

Real time models Mathematical problems. Accurate vs. not so accurate models. Execution time. Applications. ...

4.7 Documentation

Documentation is usually not taken into account when talking about models, but documentation is very important. It increases the usability of the models, the interchangeability of the models, the study of the modelled system, eases the update of the models, *etc.* A good, open format, documentation has to be generated as requested for any model (or view) in any application.

Translating models into OpenDoc format could be an interesting thing to do. Generation of documentation (reports) from questions to the model is also of importance (and strongly related witl model exploitation engines).

→ Task: *Documentation format. Standards.* / TBD

→ Task: *M2D: from model to documentation* / TBD

→ Task: *Model Querying* / TBD

→ Product: *Documentation Toolset* / Software

4.8 Testbed Applications

A theoretical work is not useful if it is not applicable. It is important to have final applications (of different nature) where the developed system (model, architecture) can be used.

4.8.1 Test applications

Some benchmark problems have to be defined to test if the ongoing development is correct. To show unidentified problems or requirements and as a logical and interactive way to develop the final system.

Two systems are in this category: The Process Control Testbed (**PCT**) and the Robot Control Testbed (**RCT**). A potential third system is the Network Control Testbed (NCT).

→ Task: *Process Control Testbed (PCT)* / TBD

→ Task: *Robot Control Testbed (RCT)* / TBD

→ Task: *Network Control Testbed (NCT)* / TBD

→ Product: *Process Control Testbed (PCT) Implementation* / Software

→ Product: *Robot Control Testbed (RCT) Implementation* / Software

4.8.2 Final applications

Final, actual applications where the system can be wholly tested. Some of these applications may be: control reconfiguration, model based control, risk assesment, autodiagnostic (model based thinking), . . .

4.9 Roadmap

Next figure shows the roadmap of the activities to be performed.

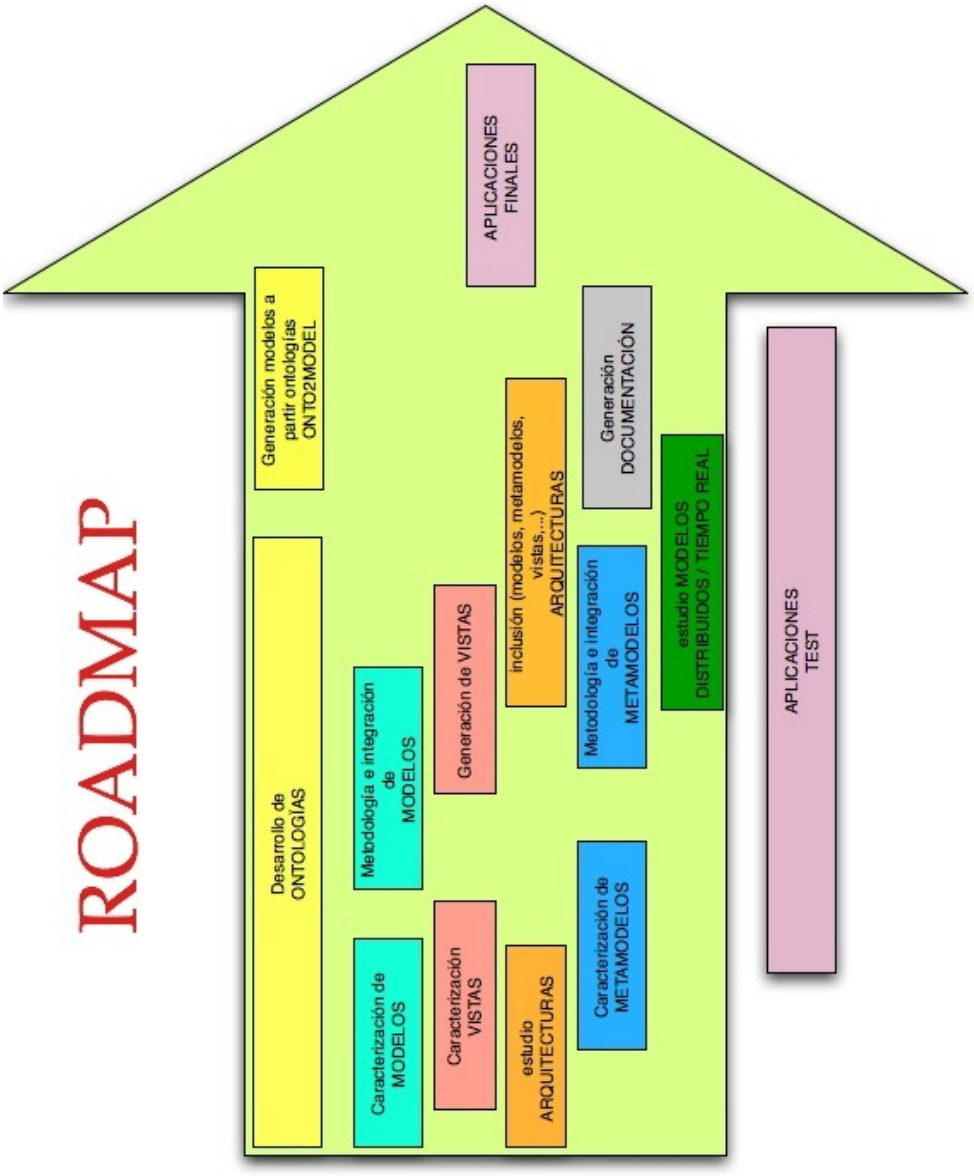

Figure 4.2: ASys tentative roadmap

Chapter 5

The Thing Reconsidered

5.1 Reconsidering ASys

The ASys Vision describes an enormously ambitious research program that at the same time is perceived as:

Impossible: As demonstrated by the (apparent) failures of AI and model-based reasoning in dealing with high complexity problems.

Unavoidable: As inferred from the (apparent) identity between models and knowledge.

Obviously the very execution of the activity will render clearer ideas on it and a Reformulation of ASys may appear on demand in the vision becomes obsolete.

The following section captures many of the questions to answer during this work.

5.2 Questions for a research line

5.2.1 Architecture requirements

1. What aspects of a system does the architecture capture?
2. Has the architecture to be universal, *i.e.* fulfillng a wide set of requirements although depending on the system not all of the are used?
3. Is it better to have a kind of universal architecture template (an abstract architecture) and to generate (derive) a specific architecture with just the needed requirements for the system?
4. Is this architecture intended to be used in all the lifecycle of the system (be it a plant or other system)?

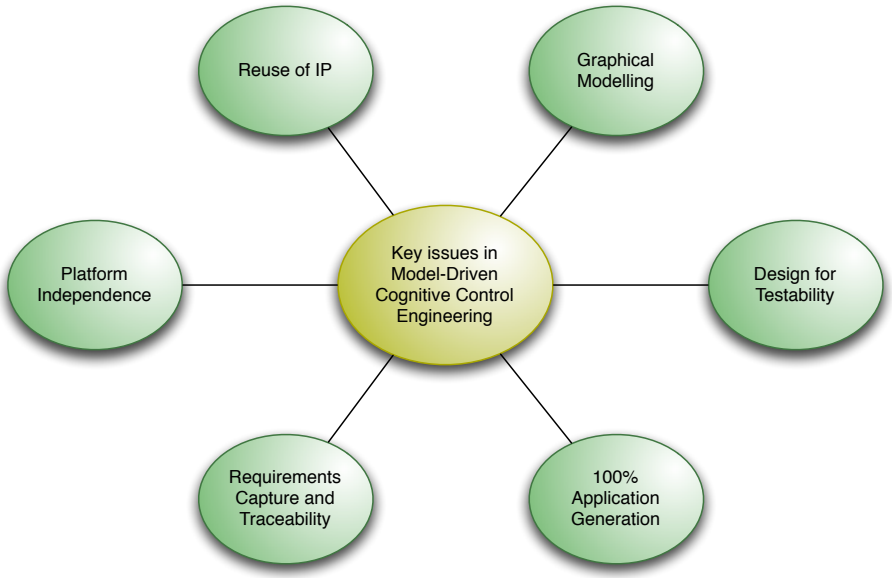

*Figure 5.1: Some of the key issues tasks involved in the development of the model-based engineering proposed in the **ASys** Vision.*

5. The architecture should be embeddable and open, what's the best way to get this (adhere to the standards)?

6. Should the architecture provide open modules (being a module a component of the basic node) besides the open node? This means that the node itself can be used elsewhere but also some components as the model, or the controlling algorithm or the perception system.

5.2.2 Epistemology

1. What is the best way to store the knowledge: ontologies?

2. How to use (take advantage of) existing knowledge (standards, ontologies...)

3. Should models be stored as world knowledge? What's the best "format" to store it?

5.2.3 Engineering Knowledge

1. Can we establish a methodology (or guide) to build the Best Available Model (mathematical, empirical, deterministic, continuous, ...) for our system?

2. Is there a way to automatically go from the knowledge repository to the model?

3. Real time models. Are they possible? Do real-timeliness depend on the type of model or does it depend on the model use? Can this be bounded to have a WCET (Worst case execution time)?

4. Can continuous models be federated (distributed)? What are the limitations? What are the advantages and the drawbacks? Does again distributability depend on the type (and/or use) of model?

5. Should we have a complete model and derive views from it or is it better to build the model views?

6. Awareness scope. The metamodel should be aware of the real world it represents but also of the real implementation on which it is being used (i.e. the model itself and the code that explicits it)

7. Is SysML (or UML) the best way to code (represent, show) the models? Or again it depends on the model?

5.2.4 Operation

1. How many inputs and outputs to the architecture? (perhaps if the system already exists one per sensor and one output per actuator,,,)

2. How must be selected the inputs and outputs to each node? Control structure configuration, variables matching.

3. What are the criteria (if any) to interconnect nodes?

4. How can we make connections "weak" and change them dynamically?

5. How many nodes do we need? How do we partition our system? Can it be done (semi or) automatically or do we need experts knowledge?

6. How information should flow between the different nodes?

7. Can (should) nodes be distributed?

5.2.5 Components

1. The controlling algorithm. What algorithm to employ? or have we a pool of algorithms (from PID to Expert Systems) and a procedure to choose?

2. Is the controlling module also responsible of generating actuation plans? or only the next action?

3. What's the best way to implement the perception: is it only a filter (one of the existing or a new one) of the sensory data received?

4. Thinking. What is needed to perform an update, to improve or change the model used?

5. Evaluating. Wha's te function of this module? Just comparing the desired state and the actual state?

6. Is intuition (premonition or the proper word) valuable? How to implement it (case based reasoning)?

7. Shouldn't be treated the emotional sensor as any other sensor? How emotional signals are detected ? How are they evaluated? Do emotions (are usually understood) have to interfere with "objective" , neutral (optimized -up to some point-) decisions? only under some (special, social,....) circumstances?

8. Controlling is understood as restoring or maintaining a desired state, but is this the module in charge of other uses of the model? If a control reconfiguration is the best way to act, does this module the one to detect and propose it? Is the thinking one? the evaluating one?

9. Perhaps other uses out of the control should be external modules, plugged to the architecture? Other possible uses: risk analysis, alarm management, auto diagnostic, reconfiguration assesment, ...

Chapter 6

Glossary and Acronyms

6.1 Glossary

Action: Mapping mental state (model) into physical dynamics.

Agent: The realisation of the epistemic loop.

Goal: Portion of state space.

Meaning: State space partitioning for a value equivalence relation due to a piece of information.

Mission: Reaching a goal keeping some constraints through.

Model: Information structure with a morphism to certain system.

Perception: The mapping from physics into mental state (model).

Process: Dynamical activity of a system.

State: State.

Thought: Ontologically comminted mental content.

6.2 Acronyms

API application programming interface BNF Backus Naur Form FOM Federation Object Model GALT Greatest Available Logical Time HLA High Level Architecture MOM Management Object Model M&S modeling and simulation N/A not applicable OO object oriented OOAD object-oriented analysis and design RTI runtime infrastructure SCADA supervisory control and data acquisition SOM Simulation Object Model XML eXtensible Markup Language

Bibliography

[1] Anthony, R., Rettberg, A., Chen, D.-J., Jahnich, I., de Boer, G., and Ekelin, C. (2007). Towards a dynamically reconfigurable automotive control system architecture. In Rettberg, A., Zanella, M. C., Dömer, R., Gerstlauer, A., and Rammig, F.-J., editors, *Embedded System Design: Topics, Techniques and Trends*, volume 231 of *IFIP*, pages 71–84. Springer.

[2] Aubin, J.-P. (1991). *Viability Theory*. Systems & Control: Foundations & Applications. Birkhäuser.

[3] Balmelli, L., Brown, D., Cantor, M., and Mott, M. (2006). Model-driven systems development. *IBM Systems journal*, 45(3):569–585.

[4] Barandiaran, X. (2004). Adaptive behaviour, autonomy and value systems. normative function in dynamical adaptive systems. Master's thesis, COGS, University of Sussex.

[5] Barwise, J. and Etchemendy, J. (1998). A computational architecture for heterogeneous reasoning. In Gilboa, I., editor, *Theoretical Aspects of Reasoning about Knowledge: Proceedings of the Seventh Conference (TARK 1998)*, pages 1–14, San Francisco, California. Morgan Kaufmann.

[6] Bertschinger, N., Olbrich, E., Ay, N., and Jost, J. (2006). Autonomy: An information-theoretic perspective. Working Paper 06 10 035, Santa Fe Institute.

[7] Braun, C. (1993). Domain specific software architectures – command and control.

[8] Calinescu, R. (2007). Model-driven autonomic architecture. In *ICAC'07. Fourth International Conference on Autonomic Computing*, Jacksonville, Florida, USA.

[9] Cleaveland, R. (2004). Model-based development for control software development. In Wirsing, M. and Ronchaud, R., editors, *Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems*, Edinburgh, UK.

[10] Clements, P. C. (1996). Coming attraction in software architecture. Technical Report CMU/SEI-96-TR-008, Carnegie-Mellon Software Engineering Institute.

[11] Collier, J. (2001). What is autonomy? In *Proceedings of Computing Anticipatory Systems, CASYS'2001*.

[12] Collier, J. (2002). What is autonomy? *International Journal of Computing Anticipatory Systems*, 12:212–221.

[13] COMPARE (2005). Embedded component framework architecture and artefacts. Technical Report ASL-R-2005-1, Autonomous Systems Laboratory, Madrid, Spain.

[14] Craik, K. (1943). *The Nature of Explanation*. Cambridge University Press, London.

[15] Foreman, J. (1996). Product line based software development- significant results, future challenges. In *Software Technology Conference*, Salt Lake City.

[16] Gery, E., Harel, D., and Palachi, E. (2002). Rhapsody: A complete lifecycle model-based development system. In *Proc. 3rd Int. Conf. on Integrated Formal Methods (IFM 2002)*, pages 1–10.

[INCOSE] INCOSE. Guide to the systems engineering body of knowledge – g2sebok: A comprehensive guide and a singular resource for understanding the extent of the practice of systems engineering. Online: http://g2sebok.incose.org/.

[17] Klir, G. C. (1969). *An approach to general systems Theory*. Litton Educational Publishing, Inc.

[18] Klir, G. J. and Elias, D. (2003). *Architecture of Systems Problem Solving*, volume 21 of *IFSR International Series on Systems Science and Engineering*. Kluwer Academic Publishers, 2 edition.

[19] Kruchten, P., Selic, B., and Kozaczynski, W. (2001). Describing software architecture with UML. In *Proceedings of the 23rd International Conference on Software Engineering, ICSE 2001, 12-19 May 2001, Toronto, Ontario, Canada*, page 0715, Los Alamitos, CA, USA. IEEE Computer Society.

[20] Kumar, V., Cooper, B. F., Eisenhauer, G., and Schwan, K. (2007). Enabling policy-driven self-management for enterprise-scale systems. In *Proceedings of Workshop on Hot Topics in Autonomic Computing, HotAC II*, Jacksonville, Florida, USA.

[21] Luo, X., Zhang, C., and Leung, H. (2001). Information sharing between heterogeneous uncertain reasoning models in a multi-agent environment: a case study.

[22] Maturana, H. and francisco Varela (1980). *Autopoiesis and cognition: The realization of the living*. D. Reidel, Boston.

[23] Minsky, M. (1968a). Matter, mind and models. In *Marvin Minsky (Ed.) Semantic Information Processing*. MIT Press.

[24] Minsky, M., editor (1968b). *Semantic Information Processing*. MIT Press.

[25] Mittal, S., Zeigler, B. P., Martín, J. L. R., Sahin, F., and Jamshidi, M. (2008). Modeling and simulation for systems of systems engineering. In Jamshidi, M., editor, *System of Systems - Innovations for the 21st Century*. Wiley.

[26] OMG (1998). A discussion of the object management architecture. Technical report, Object Management Group, Falls Church, USA.

[27] Petkos, G. and Vijayakumar, S. (2007). Context estimation and learning control through latent variable extraction: From discrete to continuous contexts. In *IEEE International Conference on Robotics and Automation (ICRA '07)*.

[28] Ray, A., Cleaveland, R., Jiang, S., and Fuhrman, T. (2006). Model based verification and validation of distributed control architectures. In *Proceedings of Convergence Convergence International Congress and Exposition on Transportation Electronics*, Detroit, USA.

[29] Rodríguez, M. and Sanz, R. (2001). HOMME: A modeling environment to handle complexity. *International Journal of Modeling and Simulation*.

[30] Roth, E. M., Patterson, E. S., and Mumaw, R. J. (2001). Cognitive engineering: Issues in user-centered system design. In Marciniak, J., editor, *Encylopedia of Software Engineering*. John Wiley and Sons, New York, 2 edition.

[31] Russell, D. and Xu, J. (2007). Service oriented architecture in the delivery of capability.

[32] Sanz, R. (1990). *Arquitectura de Control Inteligente de Procesos*. PhD thesis, Universidad Politécnica de Madrid.

[33] Sanz, R. (2003). Against biologism. contribution to the panel on intelligent control imitating biology: Promises, challenges and lessons. In *IEEE International Conference on Intelligent Control*, Houston, USA.

[34] Sanz, R., Alarcón, I., Segarra, M. J., de Antonio, A., and Clavijo, J. A. (1999). Progressive domain focalization in intelligent control systems. *Control Engineering Practice*, 7(5):665–671.

[35] Sanz, R., Jiménez, A., and Galán, R. (1991). CONEX: A distributed architecture for intelligent process control. In *Proceedings of the World Congress on Expert Systems*, Orlando, FL.

[36] Saridis, G. G. and Valavanis, K. P. (1988). Analytical design of intelligent machines. *Automatica*, 24(2):123–133.

[37] Schrödinger, E. (1944). *What is life?* Macmillan, New York.

[38] Senge, P. M. (1994). *The Fifth Discipline*. Currency/Doubleday.

[39] Shaw, M. and Garlan, D. (1996). *Software Architecture. An Emerging Discipline*. Prentice-Hall, Upper-Saddle River, USA.

[40] Simon, H. A. (1996). *The Sciences of the Artificial*. MIT Press, third edition.

[41] Stewart, J. (2004). Autopoiesis and cognition. *Artif. Life*, 10(3):327–345.

[42] Thereska, E., Ailamaki, A., Ganger, G. R., and Narayanan, D. (2007). Observer: keeping system models from becoming obsolete. In *Proceedings of Workshop on Hot Topics in Autonomic Computing, HotAC II*, Jacksonville, Florida, USA.

[43] Torngren, M., Henriksson, D., Redell, O., Kirsch, C., El-Khoury, J., Simon, D., Sorel, Y., Zdenek, H., and Årzén, K.-E. (2006). Co-design of control systems and their real-time implementation — a tool survey. Technical Report TRITA-MMK 2006:11, Royal Institute of Technology.

[44] Wirsing, M. and Ronchaud, R. (2004). *Report on the EU/NSF Strategic Workshop on Engineering Software-Intensive Systems*. Edinburgh, UK.

Title: The ASys Vision
Subtitle: Engineering Any-X autonomous systems
Author: Ricardo Sanz, Manuel Rodríguez

Date: February 11, 2008
Reference: R-2007-001 v 0.3 Draft

URL: http://www.aslab.org/documents/R-2007-001.pdf

Autonomous Systems Laboratory

UNIVERSIDAD POLITÉCNICA DE MADRID
C/JOSÉ GUTIÉRREZ ABASCAL, 2
MADRID 28006 (SPAIN)

Document based on class aslreport.cls v 2.2

ASLab CVS *Revision* : 1.25