

# A Rationale and Vision for Machine Consciousness in Complex Controllers

Ricardo Sanz, Ignacio López and Julita Bermejo-Alonso

*Autonomous Systems Laboratory*  
Universidad Politécnica de Madrid

## Abstract

The science and technology of computer-based control is facing an enormous challenge when increased levels of autonomy and resilience are required from the machines that are supporting our ambient. We can say, without any doubt, that control systems complexity is boosting. In some sense, software intensive controllers are becoming too complex to be built by traditional software engineering methods, this complexity appearing as an increase in size and a decrease of system dependability; this last being the most frightening one. Systems must be more aware and more responsible of themselves that they are today. A question emerges: Can artificial consciousness be a solution to the performance/dependability problem? This paper describes the initial ideas guiding our research. We try to develop software intensive controllers for autonomous robust behaviour based on self-awareness, qualia being left for future endeavours.

## Complexity raising in control systems

Control systems are becoming extremely complex [Aström 2000]. Science and technology of computer-based control is facing an enormous challenge when increased levels of autonomy and resilience are required from the machines that are supporting our ambient: electricity networks, cars, telecommunications, etc. All these systems include as a necessary component for the provision of the system functionally an enormous amount of embedded control software. We can say, without doubt, that control systems complexity is boosting and leading to a construction problem. We may even ask the question: Are control systems becoming too complex to be built under the required performance and safety constraints?

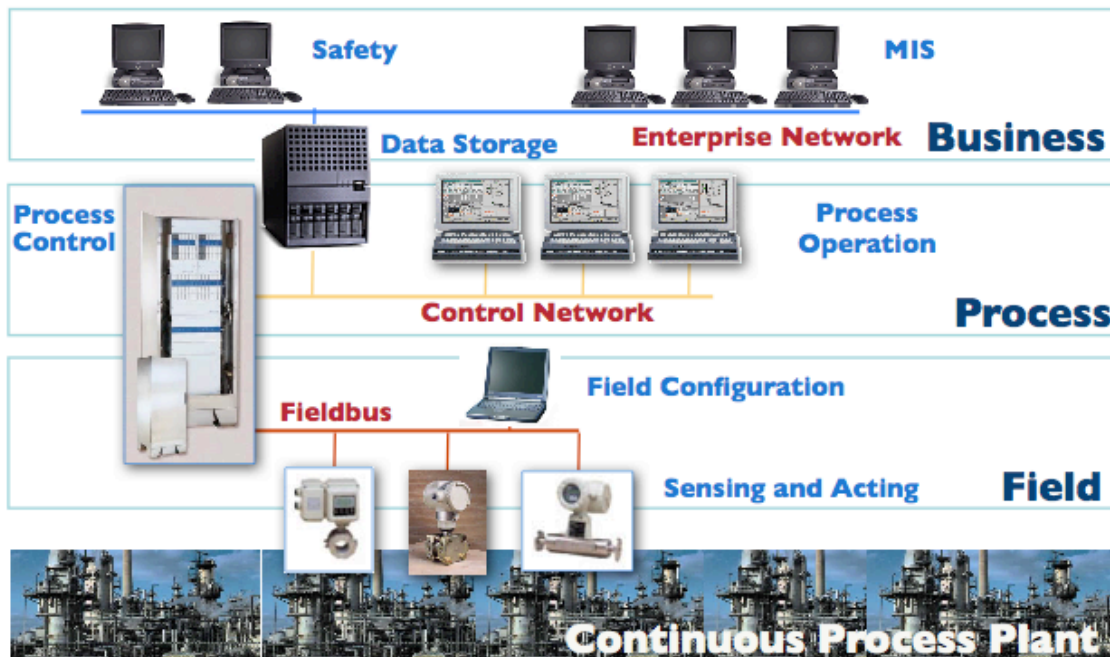
In some sense, software intensive controllers are becoming too complex to be built by traditional software engineering methods. Old age engineering processes fall short when dealing with increased levels of required functionality, performance, connectivity, robustness, adaptability, openness, etc. Non-functional requirements are pressing towards a world of control systems that cannot be built by conventional human-based engineering processes.

There are two main effects of this increased complexity that we would like to consider:

- Increase in size, that supposes a necessary increment in effort to build the system and that implies a whole new bunch of development and management practices.

- Decrease in designability (i.e. the capability of effectively calculate and/or predict the characteristics of the system when built). This may suppose a decrease in performance or more worrying a major decrease of dependability.

A decrease in dependability in office software is not a very big problem –as we all know. This is not the case of technical systems where dependability (both as reliability and maintainability) is a critical aspect. A chemical plant control system can be composed of thousands of computers –from intelligent sensors to top level management information systems– and kill thousands of people when not working properly (see Figure 1).



**Figure 1:** A simplified diagram of a distributed control system for an industrial continuous process plant. These systems may include thousands of interconnected computers.

Providing the required dependability is a major problem and the origin of specific information technology practices to build safety-critical systems. The problem is that these methods do not scale well for complex applications and that limitation forces a trade-off between features and robustness [Gunderson 2003].

Common control system failures may be associated to unexpected events –changes in the plant– or to control design errors –in a sense similar to the former from an epistemological point of view. Many control strategies have tried to cope with this kind of change: robust, self-tuning, adaptive controllers are well known examples. However, in complex, software intensive controllers, the failure modes due to faults or errors in the software are becoming a very significant part. In these systems the unexpected not only comes from the plant under control and its environment but from the controller implementation itself.

A possible path to the solution of this increasing control software complexity problem is to extend the adaptation mechanism from the core controller to the whole implementation<sup>1</sup> of it.

Adaptation of a technical system like a controller can be seen from two main perspectives: adaptation during construction and runtime adaptation. A paradigmatic example of the first type (construction) is reusable component retargeting to adapt it to a particular execution platform. A paradigmatic example of the second type (run-time) is fault-tolerant control.

We have reached the conclusion that the continuously increasing complexity makes almost impossible the only use of construction-time techniques because they do not scale and prove robust enough. Designers cannot guarantee by design the correct operation of a complex controller [Benveniste 2003].

The alternative approach is to move the responsibility for correct operation into the system itself. That means moving the adaptation from the implementation phase into the runtime phase. During runtime the control system perceives changes –not only in the plant– and adapts to these changes to keep the mission assigned to it during the design phase.

Analysing the characteristics of this problem –action by reflection– we have identified a common problem structure –isomorphism [Bertalanffy 1969]– between this situation and the situations analysed in consciousness studies. We hence wonder whether artificial consciousness can be a potential solution to the performance/dependability problem in artificial control systems.

## The Business Case

It may sound strange to claim for the existence of a business case for conscious machines when there is even disagreement on the role that consciousness plays in natural systems and its evolutionary value. This is clearly shown in the fact that there is even a school of thought that claims that consciousness is an epiphenomenon, *i.e.* nothing we can't live without.

Obviously, we do not think that way, and the best proof of its evolutionary value is our own everyday perception of consciousness: What do you prefer? a conscious or an unconscious taxi driver? What do you prefer? a conscious or an unconscious neurosurgeon? It is quite clear that consciousness does play an important role in the correct execution of tasks, in the exercising of adequate behaviour in the presence of uncertainty.

But beyond exploring ideas on what consciousness is and how can it be manifested by an artificial agent we may consider the question of real needs for this technology. Is there any real business case for it?

---

<sup>1</sup> The percentage of code that corresponds to the implementation of the real controller is less than 1% for controllers of minimal complexity. The rest of the code is supporting and peripheral software.

Indeed it is. Not one but many business cases. Let's mention just two in quite different niches and then do some analysis that may serve as a general business drive for this technology.

The first business case is the case of the software systems we use to support human activity processes; our laptops, PDAs and mobile phones are full of software and communications because today's computing environment is changing from the isolated isles of productivity around a productivity toolset to the open sea of web services and dynamic applications. There is no longer a central deity that decides when to release a new complete update of our suite. There's nobody in charge of our whole environment anymore. We must strive for keeping our working environment in line with evolving realities out there. And the task is not easy at all: this new Flash 8 media file that can't be properly executed on my Linux Firefox browser; this just-released sequencer plug-in that my OS X music software rejects to incorporate. All they are changing in the world out of a coherent configuration management. There is no single authority that can do that.

The second business case is the case of electrical system internetworking between countries. National power production plants, transport and distribution grids are operated by companies or governments that have quite different objectives and strategies. Cross-border interconnection seems necessary from many points of view (e.g. robustness, efficiency, policies, etc.). But, from a purely technical point of view, the task of controlling such a system is hopeless. There's nobody in charge anymore. Subsystems are built using different technologies and operated under different ruling agencies. While the technical issues may be solved relatively easily (standardisation bodies do help in this) the main problem remains: integrated and unified decision-making. The local operation decisions can be contradictory at a system-wide scale. Autonomous distributed decision making processes mine the technically sound operation of the global network. These processes are not only political or commercial decision processes but also include technical, even automatic, decision processes that happen ubiquitously in the network and that can produce electrical ripples that may manifest catastrophically in a remote place. We are engineering systems that suffer butterfly effects. That old wildly free character of some natural realities is becoming a daunting fact of our infrastructures.

What do these two cases have in common? The answer is relatively easy to identify: the behaviour of the global system is driven by interaction of local systems that are not any longer under a common change authority. It may look like the problem is that of proper physical integration, but not only. The main issue, the really daunting thing, is that the bottleneck, or to be more precise, the key of the path to the solution is the capability of cognitive level metareasoning and integration. The question is for a technical system to be able to reason about i) how it is able to think and act ii) how others do the same and iii) how can I communicate with them to achieve my objectives. Some of these topics have been addressed in the agents community, but agent technology still lacks the level of self-awareness that is needed to properly vehiculate these processes.

The business case is clear: software intensive systems -real-time or not- are getting so complex that we're no longer in the position of fully controlling them and their environments to make them robust enough. The classic zero-defect engineering or

replicative fault-tolerance approaches do not scale well to systems of such a size and such a rate of uncoordinated change.

The possibility we envision is also clear: make systems responsible for providing their function. Instead of having a single production engineer -producing either software or electricity- in charge of change let the systems take care of themselves. Make the systems self-aware. This is somewhat happening in the field of software (IBM's autonomic computing or Sun's conscientious software). We need it to also happen with physically embedded systems.

It looks like it is better if machines are conscious but, obviously, machines are not humans and hence machine consciousness need not be the same as humans consciousness. For example, we must be aware of the apparent differences between *sensing*, *perceiving* and *feeling*. Perception and feeling seem tinted with the colours of phenomenology, that seems to be a private, agent specific, issue (at least in biosystems).

We see three major motivations for research on artificial consciousness:

1. *Building artefacts like us*: consciousness, emotion and affect, experience, imagination, etc. This may be the main motivation for consciousness in Robotics.
2. *Studying natural systems with computer laboratory models*: this is one of the main strategies of Cognitive Science.
3. *Making effective machines*: the pursuit of Intelligent Control technologies.

In any case, a deeper, cleaner theory of consciousness is needed to support these objectives.

The focus of our work on machine consciousness is hence on software intensive controllers for autonomous robust behaviour based on self-awareness. Qualia and related issues are left out for future endeavours.

## Approaching autonomy

As we said before, the selected approach is to simplify the engineering work of building robust systems by moving the responsibility for correct operation into the system itself. This will require complex real-time architectures to support this reflective capability but hopefully they will improve performance and, more importantly, increase resilience of large-scale systems.

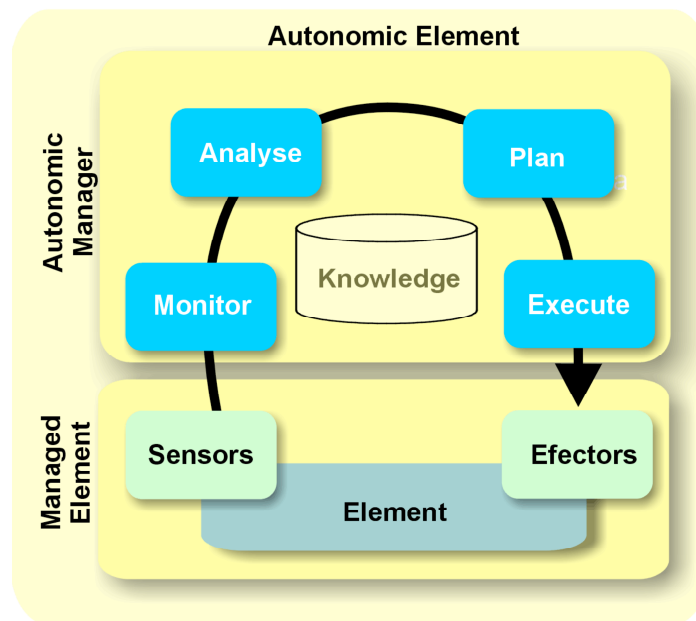
Moving the adaptation from the implementation phase into the runtime phase will make necessary for the system to gather self-information to perceive the changes that the systems is suffering: i.e. during runtime the system perceives changes and adapts to these changes to keep the mission assigned to it during the design phase.

Several alternatives have been explored in the past, based on the implementation of architectural mechanisms for self-organisation and/or self-repair. These systems are built and started in a base state and they follow adaptive life-cycles based on the

circumstances of the environment that surrounds the computing system (e.g. fault-tolerant systems, automatic learning, genetic programming or autonomic computing)

As an example, let's quote IBM's description of their autonomic computing initiative [IBM 2003]:

*"Autonomic computing is the ability of an IT infrastructure to adapt to change in accordance with business policies and objectives. Quite simply, it is about freeing IT professionals to focus on higher-value tasks by making technology work smarter, with business rules guiding systems to be self-configuring, self-healing, self-optimizing, and self-protecting."*



**Figure 2:** The IBM's vision on autonomic computing is based on the implementation of reflective control loops to increase operational autonomy of managed computing elements.

To achieve all these self-x properties, autonomic computing is based on the implementation of reflective control loops to increase operational autonomy of managed computing elements (see Figure 2). The IT elements –computers, printers, switches, *etc.*– are augmented with sensors, actuators and knowledge-based controllers termed autonomic managers. Computing infrastructures are composed by collections of autonomic elements that are able to respond to change autonomously.

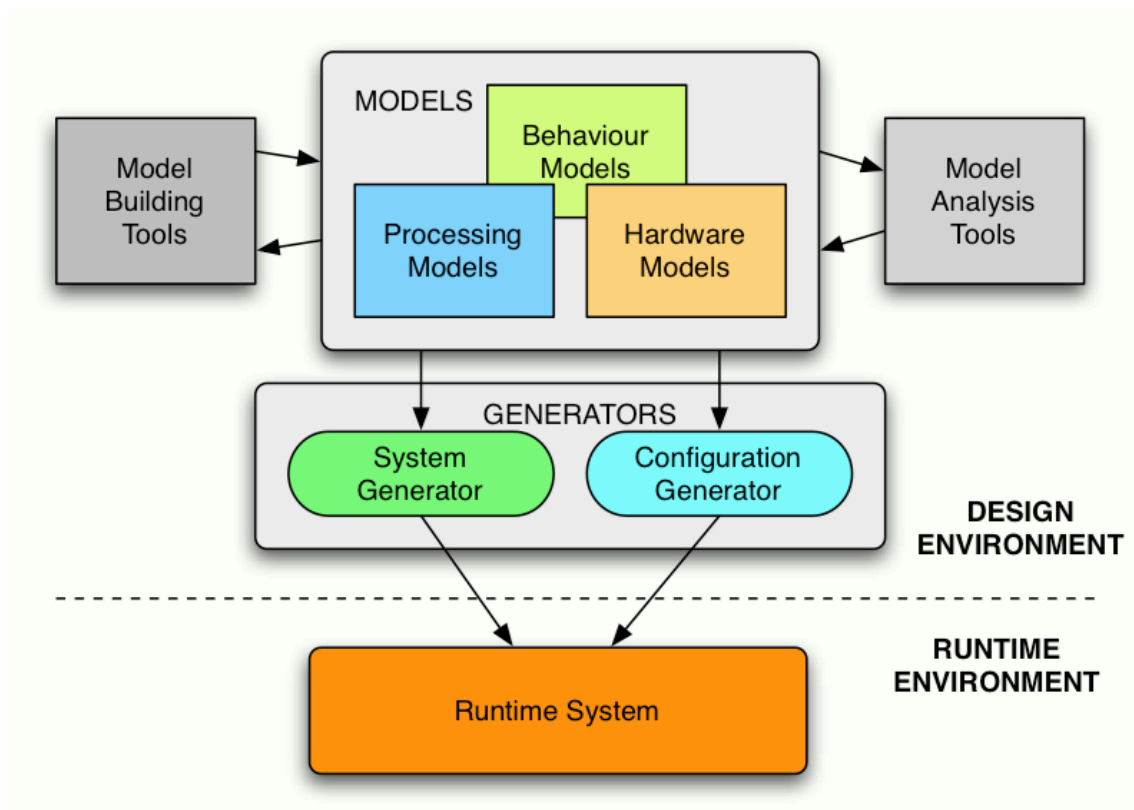
## The Modelling Approach to System Development

The approach that we try to follow is to implement our equivalents of the autonomic managers using system models as the main knowledge representation of the system itself. Model-driven development (MDD) is a strategy to build systems that is gaining wide acceptance in the construction of complex embedded systems. And we try to use the MDD engineering models as runtime models for the autonomous agent.

In the words of Jonathan Sprinkle:

*[MDD is] “A design methodology used to create and evolve integrated, multiple-aspect models of computer- based systems using concepts, relations, and model composition principles to facilitate systems/software engineering analysis of the models and automatic synthesis of applications from the models.”*

In MDD, deep and multiple system models are built and analysed using tools provided by the modelling environment. These models are later used to (semi)automatically generate the final system (see Figure 3). This means that the models must necessarily capture the semantic aspects of the final system and not just the structural properties. Functional modelling help capture the intention [Leveson 2000] of the designer and the finality of the artificial system [Bertalanffy 1969].



**Figure 3:** In the model-driven approach to system development, multiperspective, semantically-rich models of the system under construction are specified in advance and used to generate automatically the running system, typically by a set of transformation-based generators.

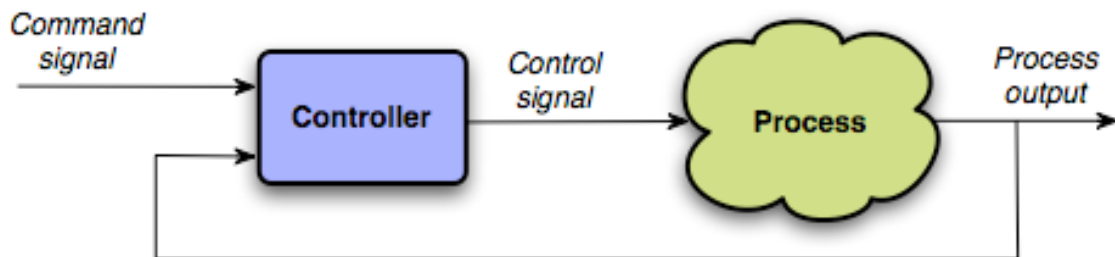
If the models capture the semantics of the final system it is possible to use model-based reasoning techniques to make run-time decisions about the system itself. In a similar way to fault-detection systems, discrepancies between system behaviour and expected behaviour may be pointed out identifying differences between reality and model-predicted outputs.

## Model-based self-aware control systems

## Analysis of extant control engineering perspectives

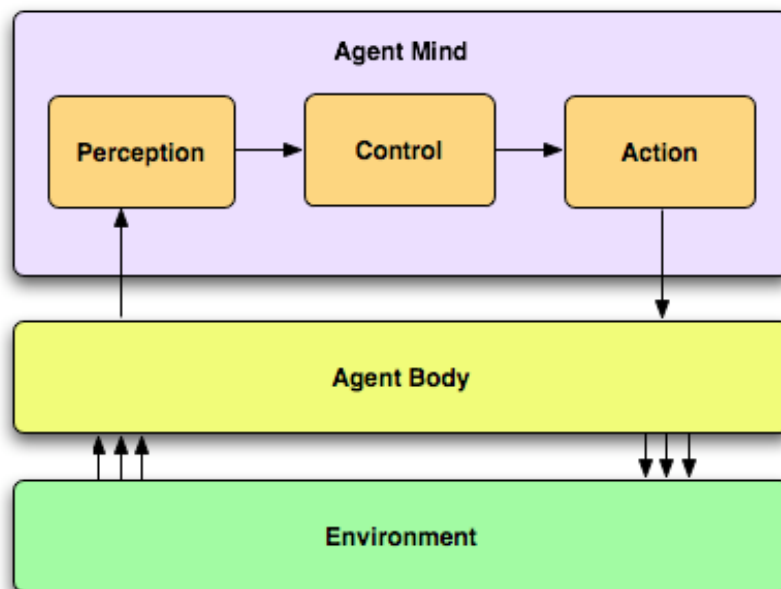
Before introducing our approach to this problem let's consider some basic control patterns that will help set the ground for our work.

Control systems exploit knowledge about the process under control to provide inputs to it that can drive the dynamics of the process to reach certain output state of interest (see Figure 4).



**Figure 4:** A simple feedback control system provides a small amount of operational autonomy [Gancett 2004].

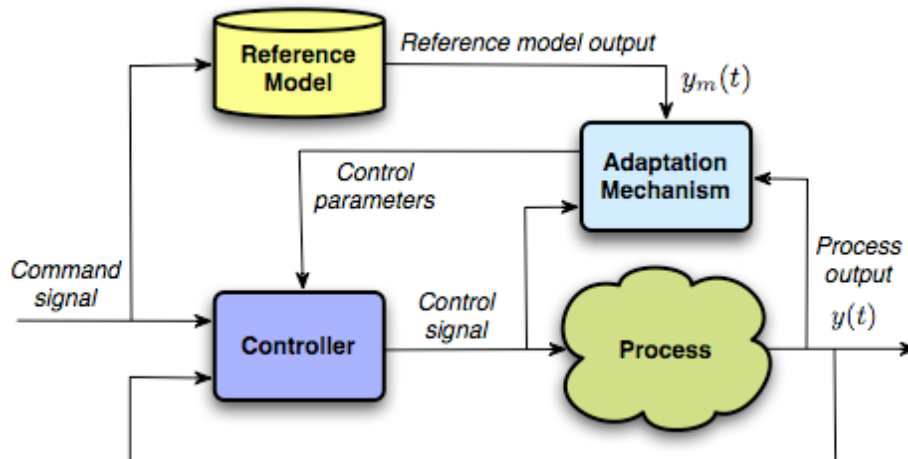
In general, in autonomous agents, we can consider the agent body as the process to be controlled, and the output measurement and control signal generation are mapped into sensing and action processes (see Figure 5). This must be done carefully because the division agent-environment may be not very clear depending on the embodiment and the function of the agent (consider for example the cases of mobile robots or environment control systems).



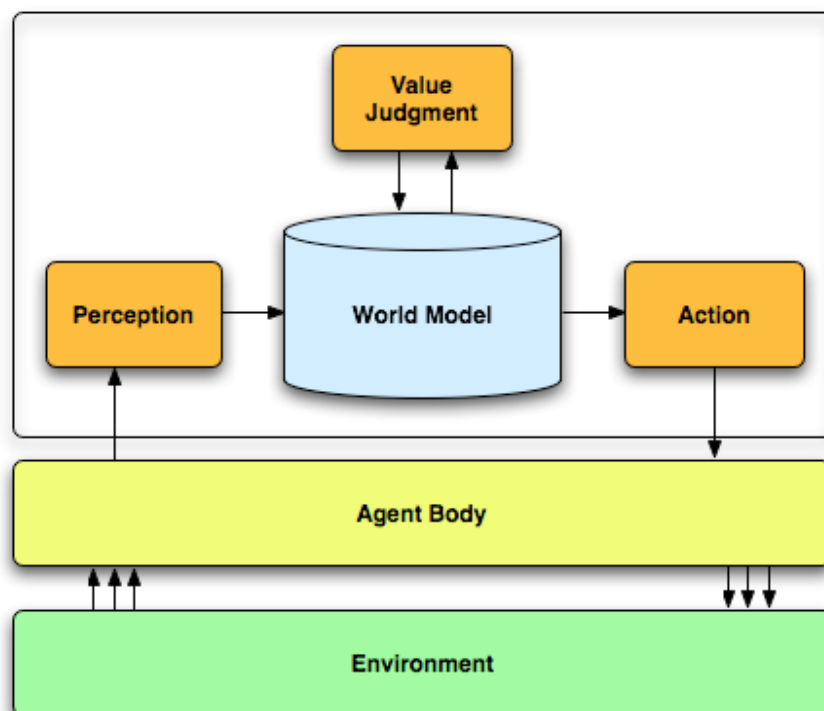
**Figure 5:** The controller vision of the autonomous agent.

In the case of model-based controllers, the control box uses an explicit model of the plant under control to improve the performance of the controller going beyond the calculation of the simple control action of a feedback controller. There are many examples of this

type of control, e.g. model-reference adaptive controllers and the RCS control model of autonomous agents (see Figures 6&7).



**Figure 6:** A model-reference adaptive controller used to track process changes and adapt the controller to them using a reference model of the process.



**Figure 7:** The basic structure of the RCS intelligent controller exploits a model of the agent and the world in the determination of control actions [Albus 2001].

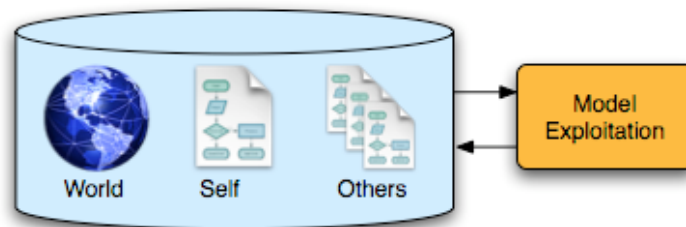
A truly adaptive system (e.g. a system with the autonomic properties) must necessarily do a semantic processing of the information it has about itself. This is the type of

reasoning that MDD tools help perform over their models if they are semantically precise enough.

### ***The ASys vision: Model-based self-aware control systems***

The vision of the ASys project of our research group can be summarized as follows: *make artificial controllers operate with the same models that humans use to build them.*

In this way, the system would operate with an explicit model of itself and of its environment: an integrated model of its world. A system working in this way would exploit the model for its operation not only in the classical sense of control, but even for interaction with its environment and the other agents of its world.



**Figure 8:** *Model contents and use.*

This vision implies that the tools and models that are being developed to be used by builders at the implementation phase can find their “autonomic” use by the system itself at the runtime phase. So, the engineering models do constitute the very self-model of the running system and the value system used to make decisions (both *auto* and *allo*) may be included with the model itself because it contains information not only about the world but also about the agent itself and other agents that interact with it (and the system provides value to them).

In this way, the control setpoints of conventional control systems find their place at the same level of any relation of authority between intelligent agents (human or artificial). Model exploitation strategies will not only drive the status of the body –the plant– but also all the interactions with other agents based on semantically sound representation of their own value systems.

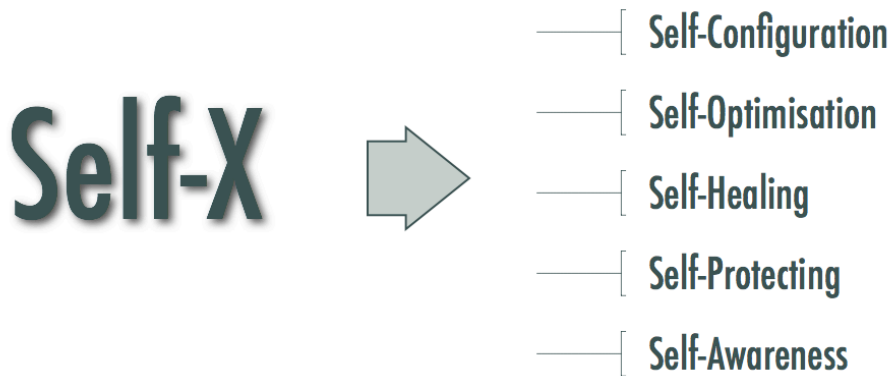
This vision goes a step beyond current technologies: Classic adaptive control or fault-tolerant control use plant models to adapt the control to new plant conditions. Fault-tolerant computing uses computation models to keep the computation ongoing. Reflective, model-based autonomous systems will have deep model-based introspection capabilities that will let them achieve a high degree of adaptability and resilience.

Models (of the world, of the self, of others) may be executed by the agent over appropriate virtual machines for different purposes (Self-X operations), for example:

- Decision-making and action calculation
- Diagnosis and reconfiguration

- What-if (imagination) and risk evaluation
- Abduction, retrodiction and causal analysis
- Coordination, negotiation and agreement
- *etc.*

We call this technology Self-X. Self-X functionality exploits models of the system itself in the performance of model-based action.



We expect that this vision can break the problematic barrier between the process and the product and bridge the gap between construction and run-time phases to enable life-cycle adaptation. In summary, we can say that Self-X makes the system adaptive using self-models and real-time reflection.

Obviously this vision fall short of explaining the multifarious nature of human consciousness, but one of the main problems we confront there is the lack of a systematic characterisation of it with some relevant exceptions [Alexander 2003].

## Conclusions

The rise of control system complexity claims for new technologies that provide the required scalability to address the problems of the technical infrastructure.

We consider that the capability of adaptation is a key issue but without losing the constraining capability of design-based engineering processed. Bounded, semantically-rich self-adaptation is needed for many reasons (in particular for increased dependability). With the required levels of autonomy and complexity, the phases of systems engineering and construction necessarily run short. Complex systems must be able to adapt to their real scenarios of activity as part of their operation.

In this light, the ASys Vision considers that model-driven development technology is a crucial technology for complex controllers. Model-based control is not new at all, but it can be extended to the control system itself and not be restricted to model the plant. The way to design and to build systems that can operate efficiently and with high degrees of dependability is by making systems exploit their knowledge not only for

controlling the plant (as traditionally,) but for controlling themselves. This implies reflexive capabilities, which still have to be highly developed for the artificial world.

Conscious, self-aware controllers will exploit self models embedded in models of their world for maximising mission-level effectiveness. A question remains, however: *What is it like to be a model-based reflective predictive controller?* There may be something it is like to be such exploiter of a self-model linked to such a world-model in a machine with a mission.

## References

Albus, James and Alexander Meystel (2001). *Engineering of Mind: An Introduction to the Science of Intelligent Systems*. Wiley Series on Intelligent Systems. Wiley. New York.

Aleksander, Igor and Barry Dunmall (2003). Axioms and tests for the presence of minimal consciousness in agents. *Journal of Consciousness Studies* 10(4-5), 7–18.

Åström, Karl, Isidori, Alberto, Albertos, Pedro, Blanke, Mogens, Schaufelberger, Walter and Sanz, Ricardo, Eds.) (2000). *Control of Complex Systems*. Springer. Berlin.

Benveniste, A., L. P. Carloni, P. Caspi and A. L. Sangiovanni-Vincentelli (2003). Heterogeneous reactive systems modeling and correct-by-construction deployment. In: *Proc. Int. Conf. Embedded Software (EMSOFT)* (R. Alur and I. Lee, Eds.).

Bertalanffy, Ludvig von (1969). *General System Theory*. George Braziller, 1969.

Blum, Alex, Vaclav Cechticky, Alessandro Pasetti and Walter Schaufelberger (2003). A Java-based framework for real-time control systems. In: *Proceedings of 9th IEEE International Conference on Emerging Technologies and Factory Automation*. Lisbon, Portugal. pp. 447–453.

Douglass, Bruce Powell (1999). *Doing Hard Time. Developing Real-Time Systems with UML, Objects, Frameworks and Patterns*. Object Technology Series. Addison-Wesley. Reading, MA.

EUSAI (2004). European symposium on ambient intelligence. <http://www.eusai.net/>.

Gancet, Jeremi and Simon Lacroix (2004). Embedding heterogeneous levels of decisional autonomy in multi-robot systems. *7th International Symposium on Distributed Autonomous Robotic Systems (DARS'04)*, Toulouse (France) 23-25 June 2004.

Gilani, Wasif, Nabeel Hasan Naqvi and Olaf Spinczyk (2004). On adaptable middleware product lines. In: *Proceedings of the 3rd ACM Workshop on Adaptive and Reflective Middleware*. Toronto, Canada.

Gunderson, Lance H. and Jr., Lowell Pritchard, Eds.) (2002). *Resilience and the Behavior of Large-Scale Systems*. Island Press. Covelo, CA.

Haikonen, Pentti O. (2003). *The Cognitive Approach to Conscious Machines*. Imprint Academic. Exeter.

Holland, Owen and Ron Goodman (2003). Robots with internal models - a route to machine consciousness?. *Journal of Consciousness Studies* 10(4-5), 77–109.

Holland, Owen, Ed.) (2003). *Machine Consciousness*. Imprint Academic. Exeter, UK.

Huang, Hui-Min (Ed.) (2004). *Autonomy Levels for Unmanned Systems (ALFUS) Framework. Volume I: Terminology. Version 1.1*. National Institute for Standards and Technology (NIST), 2004.

IBM (2003). *An architectural blueprint for autonomic computing*. Technical report. IBM.

Jacob, Bart, Richard Lanyon-Hogg, Devaprasad K. Nadgir and Amr F. Yassin (2004). *A Practical Guide to the IBM Autonomic Computing Toolkit*. RedBooks. IBM.

Leveson, Nancy (2000), *Intent Specifications: An Approach to Building Human-Centered Specifications*. *IEEE Trans. on Software Engineering*, January 2000.

Lieberherr, Karl, Doug Orleans and Johan Ovlinger (2001). *Aspect-Oriented Programming with Adaptive Methods*. *Communications of the ACM* 44(10), 39–41.

Meystel, Alexander (2000). *Measuring Performance of Systems with Autonomy: Metrics for Intelligence of Constructed Systems*. White Paper for the Workshop on Performance Metrics for Intelligent Systems. NIST, Gaithersburg, Maryland, August 14-16, 2000.

Sanz, Ricardo (2004). *CO<sup>7</sup>: Converging trends in complex software-intensive control*. In: *Sixth Portuguese Conference on Automatic Control*. Faro, Portugal.

Shrivastava, S.K., L. Mancini and B. Randell (1993). *The duality of fault-tolerant system structures*. *Software: Practice and Experience* 23(7), 773–798.

Szypersky, Clemens (1998). *Component Software. Beyond Object-Oriented Programming*. ACM Press / Addison-Wesley. Reading, MA.

Taylor, John G. (1999). *The Race for Consciousness*. MIT Press. Cambridge, MA.