SELF-X: THE CONTROL WITHIN

R. Sanz, I. López, J. Bermejo, R. Chinchilla and R.P. Conde

Autonomous Systems Laboratory
Universidad Politécnica de Madrid, Spain
`ricardo.sanz@upm.es`

Abstract: Trends in complex, software-intensive control systems show a continuous process of incorporating mechanisms of self-representation and reflection. One of the main reasons is to improve system performance and resilience in changing uncertain environments. These approaches employ runtime models of the system itself that, to some extent, are identifiable with other modelling strategies at the system design and construction phases. Systems reflect upon themselves by means of self-models. From partial plant representations like those employed in classic model-based adaptive controllers to more encompassing plant+controller mixed models, the internal model principle is pushing software-intensive control systems designs to a scale that would match the complexity of modern models of human self-awareness and consciousness. *Copyright© 2005 IFAC.*

Keywords: Autonomous systems, software intensive controllers, reflection, machine consciousness, autopoiesis.

1. INTRODUCTION

We all know that computer-based control systems are being applied in a myriad of heterogeneous engineering domains (automobiles, process control, avionics, robotics, *etc.*.) that are —in some sense— the infrastructure of our social lives.

Embedded control systems truly constitute the *mental substrate* of our technified environment. Our daily lives are almost completely dependent of computing systems properly performing their work; *i.e.* being suitable minds for the machines that support many of our activities: talking, cooking, moving, producing, etc. More than having ambient intelligence (EUSAI 2004), *intelligence is becoming our very ambient.*

Embedded control systems is a truly interdisciplinary discipline where automatic control, computer science and electronics meet to solve the complex problems of the domain. But 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 supporting our technified ambient. The construction of complex embedded systems is a major challenge because the standard, mainstream software-intensive system construction techniques are, in many cases, not good enough to deal with such a complex task. Building autonomous embedded systems is a major undertaking for control software engineers.

We can say, without doubt, that control systems complexity is boosting (Åström *et al.* 2000) and, in some sense, software intensive controllers are becoming too complex to be built by traditional software engineering methods (Douglass 1999).

The envisioned path to the solution of this increasing complexity problem is *adaptation.* Adaptation can be seen from different perspectives,

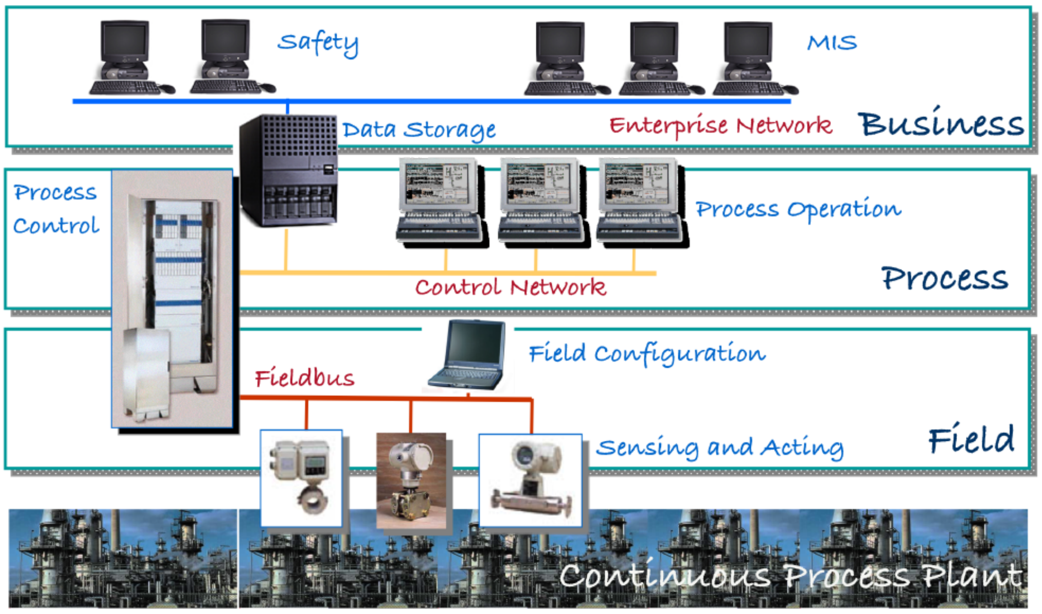

Fig. 1. Having a single, plant-wide integrated control is a major objective of control systems technology and embedded CORBA helps in this task.

but in a rough approximation we can diferentiate between *adaptation during implementation* and *runtime adaptation*. A paradigmatical example of the first type is reusable component retargetting to adapt it to a particular execution platform. A paradigmatical example of the second type is fault-tolerant control.

In this paper we will consider some of the aspects of this complexification process to discover that the various technologies already used in this field somewhat converge into a common perspective that may be described as *adaptive, model-based, reflective complex controllers* (Sanz 2004). We will also discover that this vision is also consilient with today's ideas in the field of machine consciousness (Holland 2003).

This paper is divided into five parts. After this introduction the paper summarily analyses the increase of control system complexity; the following section deals with some of the techniques used to simplify complex systems construction or increase its resilience; the final section enters into the very core of this paper: the generalised convergence into controllers that exploit self models. A final section with some conclusions closes the paper.

2. COMPLEXITY RAISING

Control software complexity is increasing due to many factors of different origin: newly required functionalities, fault tolerance for dependability, operation over heterogeneous platforms, augmented intelligence, openness and distribution, *etc.*. These factors have had many effects on control systems but we will focus here in just two: the increase in size and the decrease in dependability.

2.1 Dealing with size and complexity

The very first effect of these factors is that embedded software shows a steady increase in code size and design complexity. Embedded control system size is increasing as a result of the convergence of increases of initial systems size, the construction by composition and the code contributed during the evolution of the system.

At the end, the actual size of a real software intensive controller (a distributed control system in a refinery or a flight control system in an airplane) cannot be easily determined. System hierarchies, embedded components, integrated legacy systems, redundancies —intentional or not– make the problem of system size estimation, engineering and even understanding a challenging task.

From the maintenance point of view this leads to unmaintainable systems when newly induced errors overpass solved bugs. In most of these situations architecture recovery is impossible and hence system refactoring cannot be performed.

From the point of view of system design and construction the size problem leads to excessive costs and time-to-market of new products.

The main approach taken so far to address this complexity in construction problem come from the reuse-centric community. Many techniques, like component-based implementation (Szypersky

1998), reusable generic frameworks (Blum *et al.* 2003) or aspect oriented programming (Lieberherr *et al.* 2001) tried to exploit existing partial system designs and implementations to easy this construction process. Reuse by adaptation is a promising technology that has the potential of solving many of the problems of systems construction.

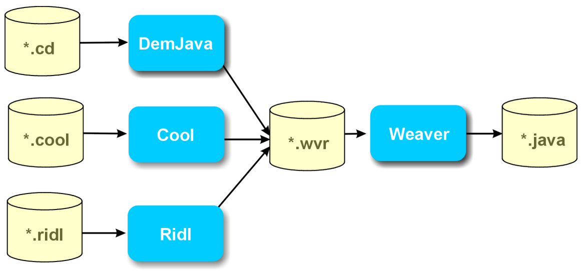

Fig. 2. Example of a aspect-based software adaptation process using the Demeter/Java aspect weaver.

But, besides the many advances, the reuse problem is still open and much work is still necessary to make of reusability the usual and almost universal method of software system construction (as it is in the case of hardware). Many problems must be solved before reaching the maturity status needed for embedded control system application: existence of truly reusable components, specification of component taxonomies, platform architecture and abstraction guidelines, consideration of non functional aspects, embeddable component description, component certification, smart component libraries, component change management, *etc.*

2.2 Dealing with dependability and resilience

Besides dealing with systems size, achieving the required system dependability is another major need. There is a manifest shortage of engineering capability to build the dependable complex control systems of tomorrow. This is a very serious problem because complex information systems are required not only to provide increased levels of performance (for example in plant-wide optimisation) or enhance the constructability and maintainability of systems (for example in X-by-Wire systems) but also to increase or at least maintain the levels of dependability of the smaller, reduced intelligence, embedded systems of the past.

Machines are becoming much more skilful thanks to the incorporation of massive doses of information technology at the expenses of deteriorating global system quality, which is a major threat that forbids the application of intelligent technology in some fields where dependability and/or adaptability is essential.

Valuable contributions are coming from the systematic, model-based and proof-driven software engineering approaches (e.g. formal methods for correct system design (Benveniste *et al.* 2003)); but they have a serious limitation concerning our capability of designing and analysing very complex systems in the presence of state combinatorial explosion and uncertainty and in guaranteeing that the implementation process does not induce system property violations.

3. THE APPROACH FROM AUTONOMY

An 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 system perceives changes and adapts to these changes to keep the mission assigned to it during the design phase.

Some interesting alternatives being currently explored are based on the implementation of architectural mechanisms for self-organisation and self-repair. These systems are built and started in a base state and they follow adaptive lifecycles based on the circumstances of the environment that surrounds the computing system. Of major interest are the research efforts in fault-tolerant systems, automatic learning, genetic programming and autonomic computing. Some of these approaches lack, however, the capability of incorporating reasoning about the physical world surrounding the controller in the process of mental adaptation because they are too focused onto the computing system itself.

IBM describes their autonomic computing initiative as (Jacob *et al.* 2004):

”*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.*”

As we can see in Figure 3, this approach is fundamentally based in the implementation of inner —homeostatic— control loops inside the software-hardware system. It is interesting to see the similarities between this architecture and advanced intelligent control architectures (Albus and Meystel 2001).

Similar approaches, even while more focalized into specific tasks and disturbances, are those of fault-tolerant computing (Shrivastava *et al.* 1993) or reflective middleware (Gilani *et al.* 2004).

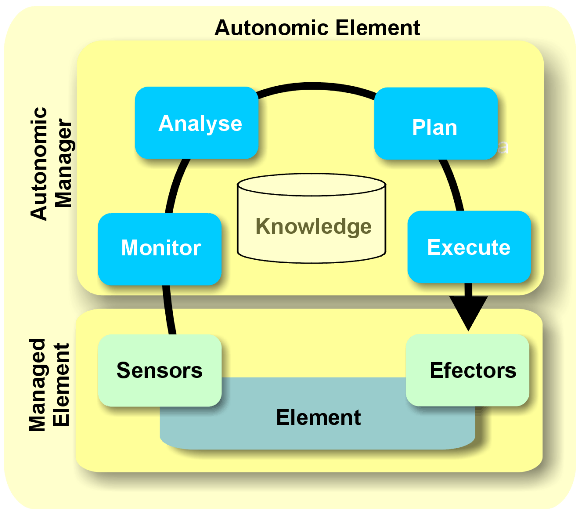

Fig. 3. Control loops in autonomic computing architectures (adapted from (IBM 2003)).

4. THE APPROACH FROM MODELING

Another recent approach to build complex embedded systems is model integrated computing. In the words of Jonathan Sprinkle, Model-Integrated Computing (MIC) 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.*"

MIC addresses the problems of designing, creating, and evolving information systems by providing rich modeling environments including model analysis and model-based program synthesis tools. MIC is expected to simplify the process of creation and evolution of complex systems using integrated, multiple-aspect models. These models include, but need not be limited to— processing models, behavior models and hardware models (see Figure 4).

Deep system models are built and analysed using tools provided by the modeling environment. These models are later used to automatically generate the final system (see Figure 4). This means that the models must necessarily capture the semantic aspects of the final system and not just the structural properties.

5. REFLECTION AND MODEL-BASED REFLECTION

In some sense, the problems of the different approaches to adaptivity can be traced back to a deep deficiency of control systems: artificial control systems do not fully understand their role

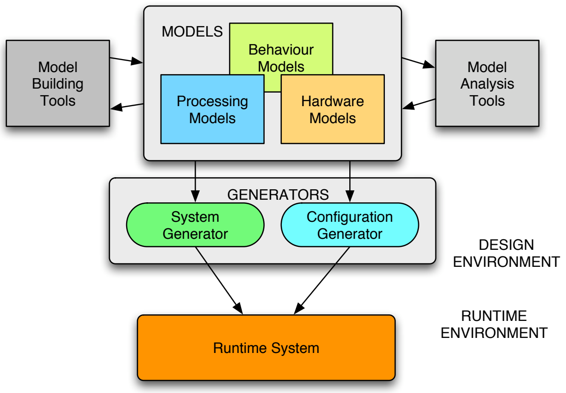

Fig. 4. Model-integrated computing can simplify the construction and evolution of complex systems.

and their perceptual flow in the terms and up to the level necessary to provide the robust performance required for critical systems. They are trapped into their particular, engineering-made, all-surrounding Matrix.

Our research project SOUL tries to overcome this limitation. The objective of this research is to explore new architectural approaches in the design of intelligent controllers that attribute meaning to complex patterns of sensory stimuli and are able to exploit those meanings into the generation of actions that, hopefully, will satisfy high-level, meaningful goals.

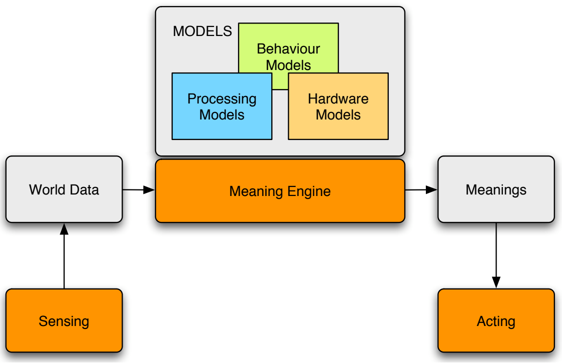

Fig. 5. The SOUL architecture uses explicit meaning representation in the generation of control actions.

Our consideration into the development of SOUL is that this meaning-based control should be done not only in terms of perception/action over the external world but also in terms of perception/action over the internal world, i.e. the mind and body of the agent itself.

In some sense, we can say that these systems do have a true understanding of what is the mission and what is going on around them. They will be able to provide increased levels of system-wide resilience required to properly maintain a stable metahuman environment. Embedding the capability of understanding the mission, and, in

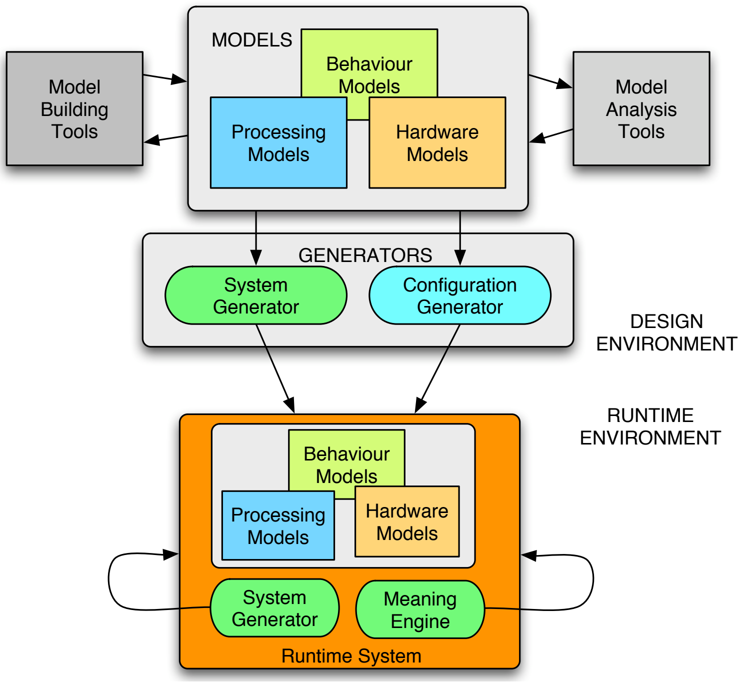

Fig. 6. The SOUL variant of the baseline model-integrated computing process includes runtime model-based reflection.

human-machine systems, being able to properly agree on this meaning with the human user, is a critical step for providing a dependable computer-based platform for human life.

We believe that if we want to construct a truly adaptive system (a system with the autonomic properties described in section 3) it must necessarily do a semantic analysis of the information it has about itself. This is the type of reasoning that MIC tools can perform over their models.

In some sense, tools 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. Reflective, model-based autonomous systems will have deep introspection capabilities that will let them achieve a high degree of adaptability.

Classic adaptive control, or fault-tolerant control uses plant models to change the control to new plant conditions. Fault-tolerant computing uses self models to keep the computation ongoing. These advanced autonomous controllers will exploit models of themselves integrated into models of the world to maximise mission-effectivity by means of meaningful adaptation. We call them *conscious controllers*. Conscious controllers will exploit self models embedded in world models.

Interestingly enough, the complex control model of SOUL, converges with recent trends in mental architectures for machine consciousness (see for example the work of (Holland and Goodman 2003), (Haikonen 2003) or (Aleksander and Dunmall 2003)).

6. CONCLUSIONS

Trends in complex, software-intensive control systems show a continuous process of incorporating mechanisms of self-representation and reflection to improve performance and resilience. From partial plant representations like those employed in classic model-based adaptive controllers to more encompassing plant+controller mixed models, the internal model principle is guiding control systems architectural design to a scale that would match the complexity of modern models of human awareness and self-consciousness.

Following IBM's description of their "autonomic computing" initiative, we are aimed at developing computer-based controllers which are self-aware, self-configuring, self-optimizing, self-healing, self-protecting, and self-adapting. That means that we expect to attain an increase of functionality, constructibility and resilience in many system functions by means of the incorporation into the very controller of mechanisms for having a "self".

Self-X functionality exploits models of the system itself in the performance of model-based control tasks that perceive, think about and act over the system itself. These system models constitute the very essence of meaning generation, the core of machine consciousness.

An important problem being faced here is the proper characterisation of what constitutes the body of each agent or agency and what constitutes the surrounding world (very different perspectives can be found today e.g. in mobile robotics and process control systems). This is a process of engineering of selves, trying to achieve the capability of building ad-hoc artificial persons to manage complex situations. Machine consciousness research is, in fact, starting to look over what was considered an uncrossable barrier between the natural and the artificial. Using Ryle's words, we envision an artificial, scalable, version of our own *ghosts in the machines.*

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.).

Blum, Alex, Vaclav Cechticky, Alessandro Pasetti and Walter Schaufelberger (2003). A java-based framework for real-time control systems. In: *Proceedings of* 9^{th} *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/.

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.

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.

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

Sanz, Ricardo (2004). Co^7: 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.