Preprints of the 19th World Congress
The International Federation of Automatic Control
Cape Town, South Africa. August 24-29, 2014

Improved Resilience Controllers Using Cognitive Patterns [⋆]

Ricardo Sanz [*] Carlos Hernandez [**] Julita Bermejo [*]
Manuel Rodriguez [*] Ignacio Lopez [*]

[*] *Autonomous Systems Laboratory, Universidad Politecnica de Madrid, Jose Gutierrez Abascal 2, 28006 Madrid, Spain (e-mail: ricardo.sanz@upm.es).*
[**] *Global Incubator, Leganes, Spain (e-mail: carlos.hernandez@globalincubator.com)*

Abstract: Construction of robotic controllers has been usually done by the instantiation of specific architectural designs. The ASys design strategy described in this work addresses the synthesis of custom robot architectures by means of a requirements-driven application of universal design patterns. In this paper we present three of these patterns —the Epistemic Control Loop, the MetaControl and the Deep Model Reflection patterns— that constitute the core of a pattern language for a new class of adaptive and robust control architectures for autonomous robots. A reference architecture for self-aware autonomous systems is synthesised from these patterns and demonstrated in the control of an autonomous mobile robot. The term "autonomous" gains a deeper significance in this context of reflective, pattern based controllers.

Keywords: Resilience; controller architecture; design patterns; cognitive systems; robot controllers; reconfiguration; model-based systems; meta-control.

1. INTRODUCTION

Control architectures for autonomous mobile robots have a long and heterogeneous history (Nilsson, 1969; Brooks, 1986; Lindstrom et al., 2000). In some of these cases, robot controllers have been built from scratch as ad-hoc solutions without any underlying systematic software architecture. The architectural foundation is implicit and the effort is focused on specific, concrete, needed functionalities and component technologies to provide them. These elementary functionalities are then deployed, integrated and operated over a minimal integration platform to generate the robot control system.

An architecture-centric approach is strongly needed in robotics. Focusing on architecture means focusing on the structural properties of systems that constitute the more pervasive and stable properties of them (Shaw and Garlan, 1996). Controller architecture —the core set of organisational aspects— most critically determines the capabilities of robots, resilience in particular. Robot mission-level resilience is to be attained by maximising architectural adaptivity from a functional perspective (Houkes and Vermaas, 2010). In this vein, the *Autonomous Systems* research programme (ASys) tries to leverage a model-based (Miller and Mukerji, 2003) and architecture-centric process for autonomous controller construction.

This paper describes some developments in this direction in the form of *reusable design patterns.* The paper is organised as follows: Section 2 describes the use of design patterns as an architectural strategy; section 3 describes the three target design patterns; section 4 describes the reference architecture generated using these patterns; section V describes its use in a real robot and sections 6 and 7 contain a roadmap for future work and conclusions.

1.1 The ASys Research Programme

The ASys Programme (Sanz and Rodríguez, 2007) is a long term research effort of very simple but ambitious purpose: *develop domain-neutral technology for building custom autonomy in any kind of technical system.* In this context, *autonomy* has a broader meaning that the regular use of the term in autonomous mobile robots. ASys pursues the identification of core architectural traits that enable a system to handle any kind of uncertainty, whether environmental or internal. It is not just a quest for achieving robust movement planning technologies in uncertain environments but *robust teleonomy for unreliable systems in uncertain environments.* Adaptation is a key issue in autonomous robotics to enable the coping with environmental challenges and with internal faults. Architectures enabling dynamic fault-tolerance are an important aspect of the work presented in this paper.

The mainstream direction of the ASys Programme comes from a simple observation: there exists a class of competence that may maximise system autonomy: cognition. We can observe it when technical systems do overcome the unexpected beyond what was technically planned and built into them at design time. It is the idiosyncratic competence of McGyvers or what is shown in the *Apollo XIII* movie. Systems can be more adaptive by making them *exploit design knowledge at run-time.* The ASys strategy is simple: *build any-level autonomy systems by*

[⋆] We acknowledge the support of the European Commission through grants ICEA and HUMANOBS.

using cognitive control loops to make systems that can engineer themselves. A controller of maximal robustness will be able to redesign itself while fielded.

We try to generalise and downsize engineer's capabilities to the level of atomic, resilient subsystems in all kinds of operational conditions in technical systems (Rodriguez and Sanz, 2009). Machines that *deeply know* themselves —their structure, their functions, their missions— will be the mission-level robust machines we need for the future, according to our vision of *self-aware machines* (Hernández et al., 2009).

1.2 ASys Design Principles

This research into general artificial autonomy is driven by some *fundamental design principles* that structure the research and development of our technologies (Sanz et al., 2007). Three of them are of special importance to the work described in this paper:

- **Model-based control:** Cognition is the core competence to develop into robots; a cognitive control loop is based on the exploitation of explicit models of the system under control.
- **Metacontrol:** Teleological robustness —the stubborn prosecution of mission goals— is achieved by means of control loops handling disturbances. When these can happen in the controller itself we need metacontrollers deal with them.
- **Break the run-time divide:** There are *design* models that engineers use to build a technical artefact and *run-time* models that reflective systems may use during their operation. Using the same models for both will break the design/run-time divide and leverage the full potential of model-driven development at run-time.

These principles are further developed in section 3, where we explain how we have reified them in the three patterns that are the core content of this paper. The final objective of this work is the provision of generalised adaptation mechanisms by means of run-time reflection in advanced, real-time cognitive architectures.

2. A PATTERN-BASED STRATEGY

The ASys strategy for increasing autonomy is the exploitation of reusable assets over architectures defined by means of *design patterns* (Sanz and Zalewski, 2003). This paper describes the construction and use of three patterns —assets in the *Asset Base*— and their use in the improvement of a robot controller.

A *design pattern* (Gamma et al., 1995) is a reusable solution to a recurring problem. Design patterns are usually not complete designs for whole systems but partial designs that offer a solution template of problem solving strategies that may be instantiated for concrete problems.

The final objective of this line of work is the creation of a *generative pattern language* to support the construction of intelligent integrated controllers for autonomous systems.

Below we briefly describe the different sections of the *pattern schema* (Sanz et al., 2003) that has been used in this paper:

- *Name*: The name of the pattern.
- *Aliases*: Other names for the pattern.
- *Example*: A real use case of the pattern.
- *Context*: Contextual information regarding the potential application of the pattern.
- *Problem*: The problem that the pattern tries to solve.
- *Solution*: The solution that the pattern provides.
- *Structure*: An architectural description of the pattern using roles and relations between roles.
- *Dynamics*: How system activity happens as sequences of role activations.
- *Related patterns*: Other patterns related with this.
- *References*: Bibliographic references.

3. THREE PATTERNS

The focus of this paper are *three design patterns* that reify some of the ASys principles for the design of autonomous systems (see Table 1). These patterns have been integrated in the OM Reference Architecture for the development of robust controllers for autonomous robots (see Section 4).

The *context* section is identical for the three patterns:

`Context` Development of robust control architectures for autonomous systems; in the current drive toward increased complexity and interconnection and with a need of augmented dependability.

Table 1. Three ASys Design Patterns

Acronym	Name	Content
ECL	*Epistemic Control Loop*	To exploit world knowledge in the performance of situated action.
MC	*MetaControl*	A controller that has another controller as control domain.
DMR	*Deep Model Reflection*	To use the system engineering model as self-representation.

The next three sections describe the three patterns using the schema presented in section 2.

3.1 The Epistemic Control Loop Pattern

`Name` Epistemic Control Loop (ECL).

`Aliases` RCS node, PEIS loop, OODA loop.

`Example` The navigation control system of an autonomous mobile robot.

`Problem` Sometimes controllers are required to implement a closed-loop strategy using an explicit model of the plant —the controlled system, e.g. the mobile robot—, with the possibility to also incorporate feed-forward action or predictive control, by providing design scalability to seamlessly incorporate different algorithms in the same control process.

`Solution` The Epistemic Control Loop pattern defines a loop that exploits world knowledge —i.e. a model of the plant— in the performance of situated action (see Figure 1). This loop is a variant of *Feedback* loop pattern in classical control (Sanz and Zalewski, 2003), but in which the sensory input is used to update an explicit representation of the plant, i.e. the *Model*, through a *Perception* process.

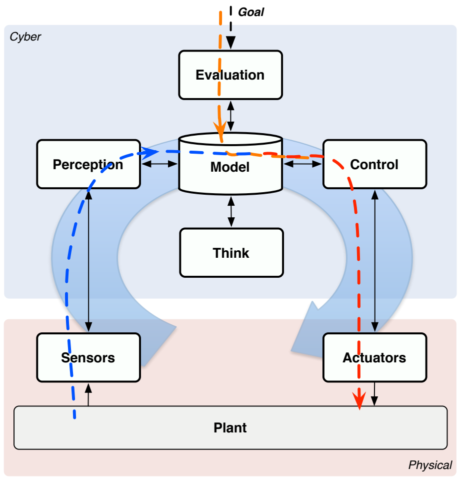

Fig. 1. The Epistemic Control Loop Pattern structure. Thin arrows show structural connections between roles; the arrow head indicates the direction of the data-flow. Thicker dashed arrows show the basic flow of information that leads to action generation.

This model contains both the instantaneous state of the plant and more permanent general knowledge about it. It is the explicitness of this last static knowledge what differentiates the ECL from other control patterns. In many other cases, the static knowledge —i.e. the plant model—, which is application-dependent, is assumed static, being embedded into the controller together with the control algorithm, so it is not possible to change or incorporate an element to the control schema without entirely re-implementing it.

`Structure` The ECL pattern proposes an structural separation of controller roles. The *Perception* process in ECL consists of the processing of the available input form the *Sensors* to update the estimation of the plant state contained in the *Model*. The *Evaluation* process evaluates the estimated state in relation with the current *Goal* of the loop —a generalisation of the error signal of classical feedback control. The *Control* is responsible for generating proper action by using the evaluation result, the information about possible actions contained in the *Model*, and action-generation knowledge, for example planning methods. The action is then sent to the *Actuators* that execute it. The *Think* process include additional reasoning activities operating on the *Model*, e.g. to improve the state estimation, together with any operations that involve the manipulation of knowledge, such as consolidation or prediction. All application specific knowledge is contained in the *Model*. It is accessed and manipulated by the rest of the processes through standard interfaces. This can be implemented using the *Database Management System* pattern, for example.

`Dynamics` The ECL defines a cyclic operation in which each cycle follows the perceive-reason-act sequence, although the *Model* serves as a decoupling element that prevents the blocking of the operation caused by a failure in any of the steps.

`Related Patterns` The ECL pattern is rooted on well established control patterns: feedback, model-based predictive control or model-based robot control (van den Hof et al., 2009).

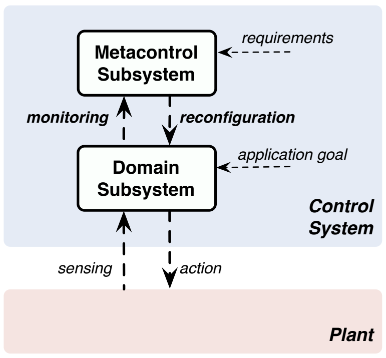

Fig. 2. The structure proposed by the MetaControl Pattern.

`References` The RCS node (Albus and Barbera, 2005) defines similar functions as the ECL. The PEIS loop (Saffiotti et al., 2008) also considers the aggregation of distributed control components with different functionalities. Boyd's OODA Loop (Osinga, 2007) is a concept originally applied to the military operations process that is now also applied to understand commercial operations and learning processes.

3.2 The MetaControl Pattern

`Name` MetaControl (MC).

`Aliases` Meta Architecture (HUMANOBS project).

`Problem` The MetaControl pattern addresses the problem of designing a control system that is self-aware and self-managing, *i.e.* it understands its mission, in the sense of detecting when its behaviour drifts; it understands itself, meaning that it can reason about how its own state realises a certain functional design in order to fulfil its mission; and it can reconfigure itself when required to maintain its behaviour convergent towards its mission fulfilment.

`Solution` MC proposes a separation of the control system into two subsystems (see Figure 2): the *Domain Subsystem*, which is the traditional control subsystem responsible for achieving the domain goal given to the system —e.g. move the mobile robot to a certain location—; and the *Metacontrol Subsystem*, which is a control system whose plant is in turn the *Domain Subsystem*, and whose goal is the system's mission requirements.

`Structure` The two subsystems in which the control system is to be divided operate in different domains. The Domain Subsystem operates in the application domain, and could be patterned after any arbitrary control architecture, for example the navigation architecture proposed in (Marder-Eppstein et al., 2010). The pattern just imposes the following requirements on the Domain Subsystem: i) its implementation has to provide a *monitoring* infrastructure, providing data at run-time about the elements realising the controller, ii) some redundancy, not necessarily physical but analytical, in the sense of having alternative designs to realise some functions (Blanke et al., 2006), and iii) the implementation platform shall include mechanisms for *reconfiguration* to exploit that redundancy.

References The issue of metacontrol is also discussed related to reconfiguration of control systems in (de la Mata and Rodríguez, 2010), and in supervisory control in fault-tolerant systems (Blanke et al., 2006).

3.3 The Deep Model Reflection Pattern

Name Deep Model Reflection (DMR).

Problem This pattern addresses the problem of how to use the engineering model of a control system as a self-representation, so the system can exploit it at run-time to adapt its configuration in order to maintain its operation converging to its goal.

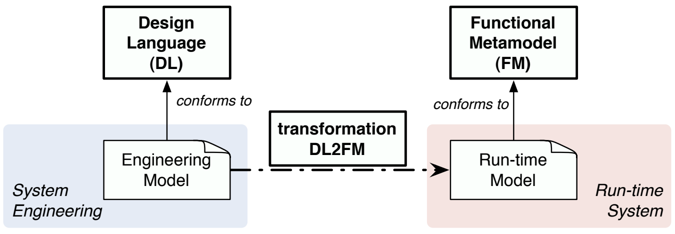

Fig. 3. The roles involved in the Deep Model Reflection Pattern.

Solution Develop a metamodel capable of explicitly capturing both i) the static engineering knowledge about the system's architecture and functional design, and ii) the instantaneous state of realisation of that design. This *Functional Metamodel* has to be machine readable to be usable by a model-based controller. A mapping from the languages used to design the system to this metamodel is necessary, in order to generate the run-tim model of the system from the engineering model of it. Automatic generation is possible if both conform to a formal metamodel, and a transformation between both metamodels exists (Figure 3).

References Metamodelling is a core topic in the domain of software modelling (Kühne, 2006). Functional modelling has been addressed in many disciplines, for example in the control of industrial processes (Lind, 1994).

4. THE OM ARCHITECTURE

The three patterns presented provide partial solutions to the problem of designing robust control architectures for autonomous systems. They have been used in the construction of a more complete architecture, the *Operative Mind Reference Architecture* (OM) (Hernández, 2013), reified in the form of reusable software.

OM offers a set of interrelated engineering assets for the development of specific control architectures (see Fig. 4):

MetaInterface: The application of the MetaControl pattern to our reference architecture has resulted in an interface that specifies the contract between the Domain and Metacontrol Subsystems' implementations.

Metacontroller: The Metacontroller realises the MetaControl pattern, using the ECL pattern to specify the Metacontrol subsystem. It defines an structure of two

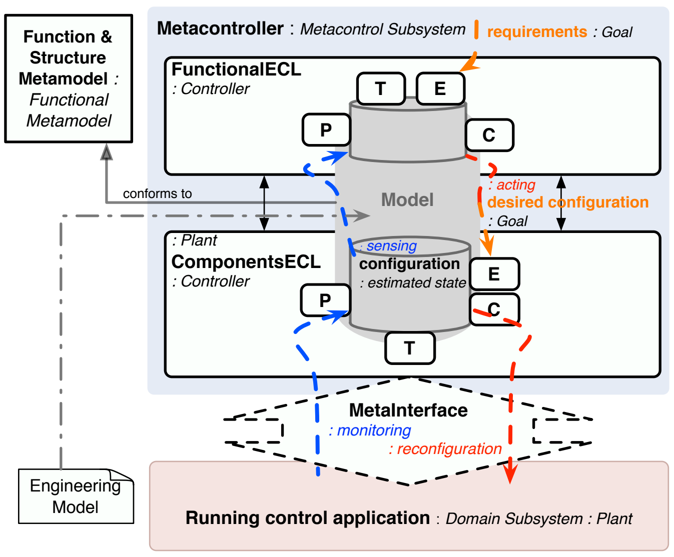
Function & Structure Metamodel : Functional Metamodel
Metacontroller : Metacontrol Subsystem
requirements : Goal
FunctionalECL : Controller
T
E
P
C
conforms to
Model
: acting
desired configuration : Goal
: Plant
ComponentsECL : Controller
sensing
configuration : estimated state
E
P
C
T
MetaInterface
: monitoring
: reconfiguration
Engineering Model
Running control application : Domain Subsystem : Plant

Fig. 4. The interplay of the main elements of the OM Architecture Model in the operation of a metacontroller. The roles that each element plays are written in italics after colon marks.

nested ECL loops: *ComponentsECL* realises a servo-control loop of the configuration of the Domain Subsystem, to which it is connected for sensing and acting through the MetaInterface. The ComponentsECL goal is to keep a certain desired configuration, given by the action of the outer loop, the *FunctionalECL*, whose sensory input is the current configuration as estimated by the ComponentsECL and its goal is the system specification. The FunctionalECL evaluates the observed configuration by determining how well it realises the functions designed to address the application requirements —*i.e.* the *mission*—, which is the goal of the FunctionalECL loop, and acts by producing a reconfiguration when necessary. Both ECL loops rely on a shared model which captures the engineering knowledge about the domain subsystem.

Function & Structure Metamodel: For the knowledge that the OM Metacontroller exploits for control purposes —*i.e.* its *Model*— we have applied the DMR pattern. We have used an ontological approach to modelling (Bermejo-Alonso, 2010), compiling all the necessary concepts required for the explicit representation of the structural —components and their connections— and the functional —teleology of the system— aspects of a system, to later formalise it in the Function & Structure Metamodel. The metamodel has been specified in UML, and a Platform Specific Model (PSM) has been implemented in Java.

5. ENHANCING A MOBILE ROBOT CONTROLLER

The OM Reference Architecture has been used to build the control architecture of a patrolling mobile robot capable of robustly perform standard navigation tasks. This application was selected because it involved heterogeneous components, both hardware and software, and had a sufficient level of complexity to prove the ASys/OM generality. The basic use case consists of the robot (a Pioneer 2AT8 platform instrumented with a SICK LMS200 laser sensor, a Kinect and a compass) navigating through an indoor office environment to sequentially visit a number of given waypoints. Two scenarios were envisaged to test the benefits of the application of the OM Architecture in terms of robustness and autonomy: i) a temporary failure of the laser driver, ii) a permanent fault in the laser sensor. In both cases the system should be able to detect it and reconfigure its organisation to adapt to the current state of affairs, in order to maintain the mission.

The software selected to implement the domain control system is ROS (Quigley et al., 2009) for several reasons: i) its middleware infrastructure provides with mechanisms for *monitoring* and *reconfiguration* as prescribed by the `MetaControl` pattern, ii) its computation model of nodes that publish and subscribe to message channels or topics fits in a component-based architecture model, being thus modellable with our Function & Structure Metamodel; and iii) there are open source ROS implementations of components for navigation of mobile robots available.

For the Domain Subsystem of the control architecture we have used the ROS navigation stack (Marder-Eppstein et al., 2010), which we have tuned for our Pioneer mobile platform, and complemented with other available ROS packages for the robot Kinect and Laser sensors, and additional ROS necessary for the patrolling mission.

The metacontroller is implemented using the OMJava package. It provides a domain independent and multi-platform implementation of the OM Architectural Model, including a complete implementation of the Function & Structure Metamodel, the OM Metacontroller, and a Java specification of the MetaInterface.

The application development included the implementation of: i) a Java library which provides an implementation of OM Metacontroller (OMJava), ii) a ROS PSM of the OM Architecture (OMrosjava), which includes ready-to-deploy OM-based metacontrol assets for any robot application implemented with ROS.

Thanks to the architectural model-based approach, to implement the concrete testbed it was only necessary to generate the application model according to the Function & Structure Metamodel, in order to provide the Metacontroller with explicit knowledge about the system: i) its mission —core functionality required—, ii) its structure —its components and their properties—, and iii) its functional design —available design solutions that realise the core functionality through certain configurations of its structure.

The application-independent OM processes exploit the aforementioned application-specfic knowledge in the two scenarios envisaged:

Scenario 1: the Metacontroller detects the failure of the laser driver and repairs the component it by re-launching the software process. Only the ComponentsECL intervenes, since the solution is achieved at the structure level.

Scenario 2: this time it is not enough to relaunch the laser driver because the error is due to a permanent fault in the laser device. the ComponentECL detects this and the situation scales to the FunctionalECL loop. The functional failure caused by the unavailability of the laser driver is assessed, and an alternative configuration that fulfils the functionality required to maintain the mission is generated. This configuration makes use of the Kinect sensor instead of the laser. The new configuration is commanded to the ComponentsECL, which reconfigures the navigation systems accordingly. In summary, the robot redesigns at runtime its control architecture using available components.

6. FUTURE WORK

Our current pattern language contains only a small number of patterns centred on core ASys issues. It is necessary to complete it with more common, practical patterns to achieve a full *generative pattern language.*

The current implementation of the ICe —the Integrated Control Environment— is just a collection of engineering resources atop the Rational Software Development Platform. Our current effort is to use the Eclipse RCP to create a specific IDE for the ASys engineering process. There is an ongoing work in the automation of some of the transformation processes. For example, concerning the current implementation of the OM Architecture Model, the automatic transformation from the engineering design language to our Function & Structure metamodel is still under development. The MDD transformations necessary to complete the ICe toolchain are still in early stages.

The Functions & Structure metamodel now only models basic structural aspects of control systems. It is necessary to incorporate behavioural aspects to our current metamodel so the Metacontrol will be able to handle function-centric, dynamical issues in the Domain Subsystem. It is necessary to improve the metamodel by including the full ECL and OASys (Bermejo-Alonso, 2010) concepts to further specify functionality. An specially important ongoing work is the self-closure of the MC pattern: the application of the MetaControl pattern to the Metacontrol Subsystem, so it becomes also part of the subsystem it controls.

The ambition of ASys is of universality; and hence there is a need for domain generalisation, i.e. the extension of theoretical concepts and technological assets to other domains. Current efforts are centred around *autonomous robots* and *continuous process control systems* (Rodriguez and Sanz, 2009), but other technological domains are under consideration —e.g. utilities, telecoms or maintenance systems. Even more, while the patterns described in this paper are technological designs for autonomous artefacts, their content may find strong biological roots in animal cognition (Hernández et al., 2011). In this sense, the ASys research programme may have impact not only in how engineers build autonomous robots, but also in how cognitive scientists understand the mind (Sanz, 2010).

7. CONCLUSIONS

This work shows a pattern-based approach to the construction of sophisticated, self-aware control systems in the domain of autonomous systems. The three patterns —Epistemic Control Loop (ECL), MetaControl (MC), and Deep Model Reflection (DMR)— offer valid reusable design assets for the implementation of custom architectures.

The development of the testbed robotics application demonstrated the benefits of following a pattern-based approach in the implementation of a resilient control architecture.

The patterns, as instantiated in the OM Architecture, were easily applicable thanks to the availability of a domain neutral implementation (OMjava). From it, the production of the ROS platform-specific model was straightforward, and only slightly hampered by the lack of a formal Platform Definition Model for ROS. Considering strictly only the development of the testbed application, the addition of our reference architecture produced only a minor extra-effort when compared with a standard development of the control architecture for the mobile robot.

The pattern-based, model-centric approach to the construction of autonomous controllers proposed by ASys can offer possibilities —both for engineering and run-time operation— that go well beyond current capabilities of intelligent autonomous robots. In this direction, the application of our OM Architecture, rooted on the three design patterns described in the paper, has provided the robot with deep run-time adaptivity based on a functional understanding, hence demonstrating enhanced robust autonomy through cognitive self-awareness.

Remember that the ASys programme seeks *robust teleonomy for unreliable systems in uncertain environments.* The model-based, self-aware adaptivity approach supported by these patterns departs from conventional robust control approaches, offering a more open-ended pathway for universal system adaptation.

REFERENCES

Albus, J.S. and Barbera, A.J. (2005). RCS: A cognitive architecture for intelligent multi-agent systems. *Annual Reviews in Control*, 29(1), 87–99.

Bermejo-Alonso, J. (2010). *OASys: An Ontology for Autonomous Systems.* Ph.D. thesis, Departamento de Automática, Universidad Politécnica de Madrid.

Blanke, M., Kinnaert, M., Lunze, J., and Staroswiecki, M. (2006). *Diagnosis and Fault-Tolerant Control.* Springer-Verlag Berlin.

Brooks, R.A. (1986). A robust layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, 2(3), 14–23.

de la Mata, J.L. and Rodríguez, M. (2010). Accident prevention by control system reconfiguration. *Computers and Chemical Engineering*, 34(5), 846–855.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software.* Addison-Wesley Professional Computing Series. Addison-Wesley Publishing Company, New York, NY.

Hernández, C. (2013). *Model-based Self-awareness Patterns for Autonomy.* Ph.D. thesis, Escuela Técnica Superior de Ingenieros Industriales - Universidad Politécnica de Madrid.

Hernández, C., López, I., and Sanz, R. (2009). The operative mind: a functional, computational and modelling approach to machine consciousness. *International Journal of Machine Consciousness*, 1(1), 83–98.

Hernández, C., Sanz, R., Gómez-Ramirez, J., Smith, L.S., Hussain, A., Chella, A., and Aleksander, I. (eds.) (2011). *From Brains to Systems. Brain-Inspired Cognitive Systems 2010*, volume 718 of *Advances in Experimental Medicine and Biology.* Springer, New York.

Houkes, W. and Vermaas, P. (2010). *Technical Functions. On the Use and Design of Artefacts.* Springer.

Kühne, T. (2006). Matters of (meta-)modeling. *Software and System Modeling*, 5(4), 369–385.

Lind, M. (1994). Modeling goals and functions of complex industrial plants. *Applied Artificial Intelligence*, 8(2), 259–283.

Lindstrom, M., Oreback, A., and Christensen, H.I. (2000). Berra: a research architecture for service robots. In *IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA '00.*, 3278–3283 vol.4. San Francisco, CA, USA.

Marder-Eppstein, E., Berger, E., Foote, T., Gerkey, B., and Konolige, K. (2010). The office marathon: Robust navigation in an indoor office environment. In *Robotics and Automation (ICRA), 2010 IEEE International Conference on*, 300 –307.

Miller, J. and Mukerji, J. (2003). Mda guide. Technical Report omg/03-06-01, Object Management Group.

Nilsson, N. (1969). A mobile automaton: An application of artificial intelligence techniques. Technical Report 40, AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025. SRI Project 7494 IJCAI 1969.

Osinga, F.P. (2007). *Science, Strategy and War: The Strategic Theory of John Boyd.* Routledge.

Quigley, M., Conley, K., Gerkey, B.P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A.Y. (2009). Ros: an open-source robot operating system. In *ICRA Workshop on Open Source Software.*

Rodriguez, M. and Sanz, R. (2009). Development of integrated functional-structural models. In *10th International Symposium on Process Systems Engineers*, 573–578. Salvador, Brasil.

Saffiotti, A., Broxvall, M., Gritti, M., LeBlanc, K., Lundh, R., Rashid, J., Seo, B., and Cho, Y. (2008). The PEIS-ecology project: vision and results. In *Proc of the IEEE/RSJ Int Conf on Intelligent Robots and Systems (IROS)*, 2329–2335. Nice, France.

Sanz, R., Yela, A., and Chinchilla, R. (2003). A pattern schema for complex controllers. *Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference*, 2, 101–105 vol.2.

Sanz, R. (2010). Machines among us: Minds and the engineering of control systems. *APA Newsletters - Newsletter on Philosophy and Computers*, 10(1), 12–17.

Sanz, R., López, I., Rodríguez, M., and Hernández, C. (2007). Principles for consciousness in integrated cognitive control. *Neural Networks*, 20(9), 938–946.

Sanz, R. and Rodríguez, M. (2007). The ASys vision. engineering Any-X autonomous systems. Technical Report R-2007-001, Universidad Politécnica de Madrid - Autonomus Systems Laboratory.

Sanz, R. and Zalewski, J. (2003). Pattern-based control systems engineering. *IEEE Control Systems Magazine*, 23(3), 43–60.

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

van den Hof, P.M., Scherer, C., and Heuberger, P.S. (2009). *Model-Based Control: Bridging Rigorous Theory and Advanced Technology.* Springer.