Three Patterns for Autonomous Robot Control Architecting

Carlos Hernández, Julita Bermejo-Alonso, Ignacio López and Ricardo Sanz
Autonomous Systems Laboratory
Universidad Politécnica de Madrid, Spain
Emails: carlos.hernandez@upm.es, jbermejo@etsii.upm.es, ignacio.lopez@upm.es, Ricardo.Sanz@upm.es

***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 synthesized 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*— *Patterns; autonomous systems; robot controllers; reconfiguration; model-based systems; meta-control.

I. INTRODUCTION

Control architectures for autonomous mobile robots have a long and heterogeneous history [1]–[3]. 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 [4].

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 [5]. Controller architecture —the core set of organizational aspects— most critically determines the capabilities of robots, resilience in particular. Robot mission-level resilience is to be attained by maximizing architectural adaptivity from a functional perspective [6]. In this vein, the *Autonomous Systems* Programme (ASys) tries to leverage a model-based [7], 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 organized as follows: Section II describes the use of design patterns as an architectural strategy; section III describes the three target design patterns; section IV describes the reference architecture generated using these patterns; sections V and VI contain a roadmap for future work and conclusions.

A. The ASys Research Programme

The ASys Programme [8] is a long term research effort of very simple 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 [9]. 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 [10] to enable the coping with environmental changes and with internal faults. Architectures enabling dynamic fault-tolerance is an important aspect of the work presented in this paper.

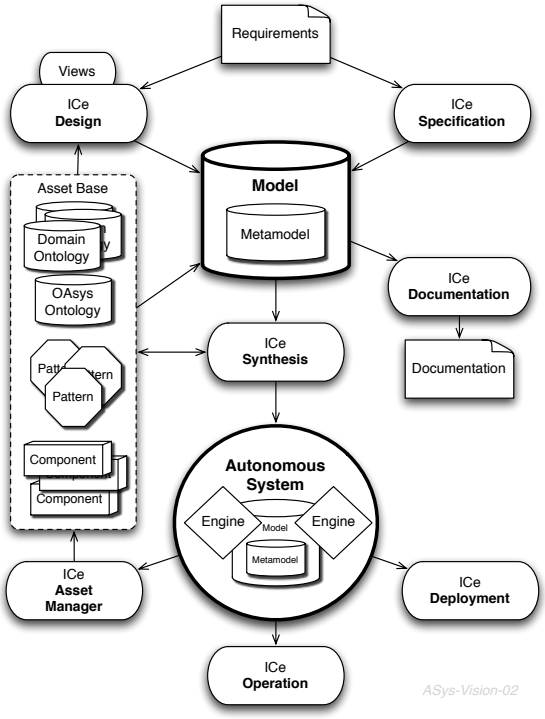

Figure 1. The ASys model-centric systems engineering process. ICe stands for *Integrated Control Environment*.

The mainstream direction of the ASys Programme comes

 ISBN: 978-1-61208-276-9

from a simple observation: there exists a class of competence that may maximize 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 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 generalize and downsize engineer's capabilities to the level of atomic, resilient subsystems in all kinds of operational conditions in technical systems [11]. 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* [12].

B. 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 [13]. 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 [14].
- **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 [15].
- **Break the run-time divide:** There are *design* models that engineers use to build a technical artifact 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 [16] at run-time.

These principles are further developed in section III, 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 generalized adaptation mechanisms by means of run-time reflection in advanced, real-time cognitive architectures.

II. A Pattern-based Strategy

The ASys strategy for building autonomous control systems is the exploitation of reusable assets over architectures defined by means of *design patterns* [17] (see Figure 1). This paper describes the construction and use of three patterns — assets in the *Asset Base*— and their use in the synthesis of an autonomous robot.

A. A Pattern-based Design Approach

A *design pattern* [18] is a reusable solution to a recurring problem. Design patterns are usually not complete designs for whole systems but descriptions of partial designs that offer a solution template of problem solving strategies that may be instantiated for concrete problems. In principle, patterns capture best practices and anti-patterns capture worst practices: things to do versus things to avoid when designing or implementing specific applications [19].

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

B. A Pattern Schema

Below we briefly describe the different sections of the *pattern schema* [20] that has been used in this paper:

- *Name*: The name of the pattern.
- *Aliases*: Patterns are usually not new; most of them have been discovered and used elsewhere, esp. in the controls domain.
- *Example*: A use case of the pattern; a possible application of the pattern in a real situation.
- *Context*: Contextual information regarding the potential application of the pattern.
- *Problem*: The problem that the pattern tries to solve.
- *Solution*: The form of 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, by structure, by way of use or because they are applied at the same time to a system.
- *References*: Bibliographic references for the pattern.

III. 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 I). These patterns have been integrated in the OM Reference Architecture for the development of robust controllers for autonomous robots (see Section IV). Two of the sections are almost identical for the three patterns; they share a common *context* and are *closely related*:

`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. The design strategy of these systems has to address not only the problem of the uncertainty of the environment, but also of the uncertainty arising from

TABLE I. 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 system itself because of faults or unforeseen, emergent behaviors resulting from the interplay of their components, or their connection with other systems [21]. More traditional approaches such as fault-tolerant control based on redundancy are too expensive and not efficient. The universality of the problem demands a general approach rather than specific solutions for certain applications that are difficult to transfer to other domains.

`Related patterns` These three patterns can be said to constitute a micro *Pattern Language*, sharing a common context of application and being conceived so as to apply them jointly for the development of control architectures for autonomous robots.

The next three sections describe the three patterns using the schema presented in section II-B.

A. 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 2). This loop is a variant of *Feedback* loop pattern in classical control [17], but in which the sensory input is used to update an explicit representation of the plant, i.e. the *Model*, through a *Perception* process. 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 these 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. With an explicit model bearing all the information used in the different elements of the

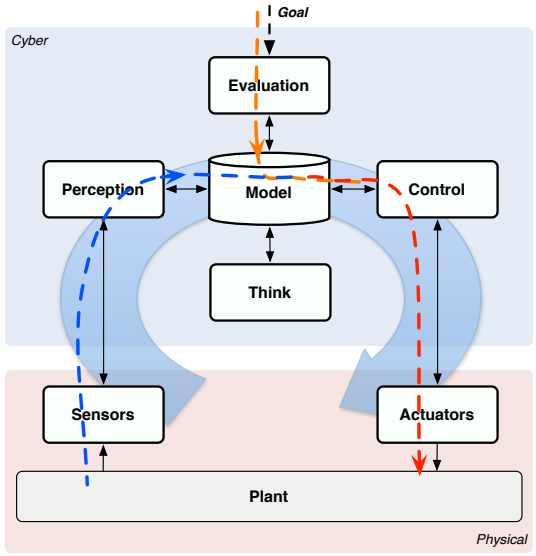

Figure 2. The Epistemic Control Loop Pattern structure. Thin arrows show structural connections between roles, with the arrow head indicating the direction of the data-flow, whereas thicker dashed arrows show the basic flow of information that leads to action generation.

control, the ECL design allows for changing the algorithm of any element of the control, or incorporating a new one, without modifying the rest.

`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 generalization 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. The *Perception* process in ECL corresponds to the first step and may include other reasoning operations as described previously. The *Evaluation* process then generates value from the current estimated state in the *Model*. In the

 ISBN: 978-1-61208-276-9

last phase of the loop the *Controller* produces the most appropriate action based on the information available.

Related Patterns The ECL pattern is rooted on well established control patterns: feedback [17], model-based predictive control [22] or model-based control [23].

References The RCS (Real-time Control System) node [24] defines similar functions that are required in a real-time intelligent control unit. The PEIS (Physically Embedded Intelligent System) loop [25] also considers the aggregation of distributed control components with different functionalities. Boyd's OODA Loop (Observe, Orient, Decide, and Act) [26] is a concept originally applied to the combat operations process, often at the strategic level in both the military operations. It is now also often applied to understand commercial operations and learning processes.

B. 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, *i.e.* it understands its mission, in the sense of detecting when its behavior diverges from its specification; it understands itself, meaning that it can reason about how its own state realizes a certain functional design in order to fulfill its mission; and it can reconfigure itself when required to maintain its behavior convergent towards its mission fulfillment.

Solution MC proposes a separation of the control system into two subsystems (see Figure 3): the *Domain Subsystem*, which consists of the traditional control subsystem responsible for sensing and acting on the plant so as to achieve the domain goal given to the system —e.g. move the mobile robot to a certain location, grab a certain object with a robotic hand...—; 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 [27] for a mobile robot. The pattern imposes the following requirements on the Domain Subsystem: i) its implementation has to provide a *monitoring* infrastructure, providing data at run-time about the processes and elements realizing the domain control, ii) some redundancy, not necessarily physical but more interestingly analytical, in the sense of having alternative designs to realize some functions [28], and iii) the implementation platform shall include mechanisms for *reconfiguration* to exploit that redundancy.

References The separation of the domain control and the meta-control is at the core of AERA, the architecture for autonomous agents developed in the HUMANOBS project

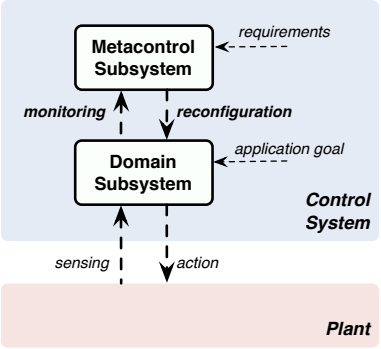

Figure 3. The structure proposed by the MetaControl Pattern.

(Humanoids the Learn Socio-Communicative Skills by Imitation, www.humanobs.org). The issue of metacontrol is also discussed related to reconfiguration of control systems in [29], and in supervisory control in fault-tolerant systems [28].

C. 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.

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 realization 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 4).

References Metamodeling is a core topic in the domain of software modeling [30]. Functional modeling has been addressed in many disciplines, for example in the control of industrial processes [31].

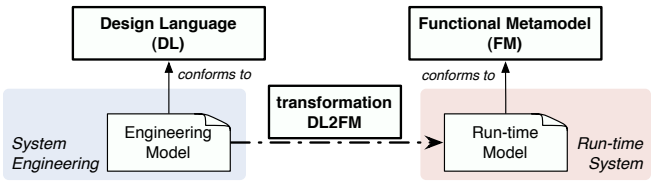

Figure 4. The roles involved in the Deep Model Reflection Pattern.

 ISBN: 978-1-61208-276-9

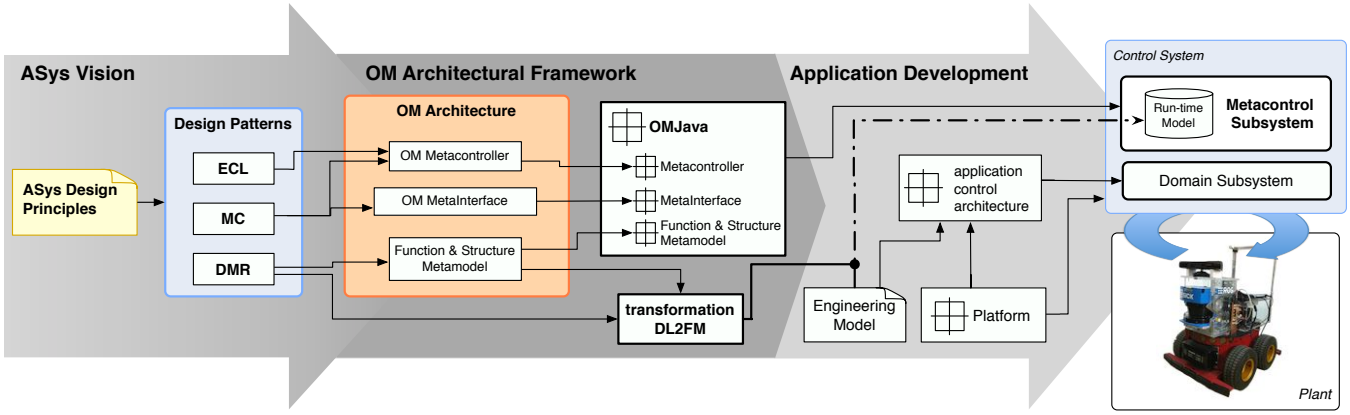

Figure 5. The methodological path followed in the development of the OM Architecture, with the core assets for ASys resulting from it.

IV. The OM Architecture

The three patterns presented provide partial solutions to the problem of designing robust control architectures for autonomous systems. They can be used in the construction of more complete architectures, as is the case of the Operative Mind (OM) reference architecture, a complete and general solution, synthesized by integrating the three patterns and realizing them in reusable software.

The conceptual and methodological path to OM is depicted in Figure 5. From left to right: a principled approach to robust cognition and self-awareness is captured as a set of design patterns; they are applied to synthesize the OM Architecture and build an application independent implementation as a Java package; it is then used, together with platform specific available assets, to implement the control architecture of a mobile robot.

A. OM Architectural Assets

OM offers a set of interrelated engineering assets for the development of specific control architectures (see Fig. 6):

1) 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.

2) Metacontroller: The Metacontroller is a refinement of the MetaControl pattern that specifies the design of the Metacontrol Subsystem by application of the ECL pattern. It defines an structure of two nested ECL loops: *ComponentsECL* realizes 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 realizes 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. For their operation, both ECL loops rely on a shared model which captures the engineering knowledge about the domain subsystem.

3) Function & Structure Metamodel: To design the knowledge that the OM Metacontroller exploits for control purposes —*i.e.* its *Model*— we have applied the DMR pattern. This pattern prescribes the explicit use of design-time models. For that purpose we have used an ontological approach to modeling [32]: we have compiled all the necessary concepts required for the explicit representation of the structural and the functional aspects of a system, to later formalize it in the Function & Structure Metamodel. The metamodel contains elements to account for the two referred aspects of an autonomous system:

Structure elements: concepts to represent the configuration of the system in terms of components and their connections.

Functional elements: concepts that capture the teleology of the system, in the sense of Lind [31] of representing the roles the designer intended for the components of the system to achieve the objectives of the system. These representations constitute design solutions for the required functionality.

The metamodel has been specified in UML, and a Platform Specific Model (PSM) has been implemented in Java.

B. Use of OM in an autonomous mobile robot

This section describes the application of the OM Reference Archiecture to develop the control architecture of a mobile robot capable of robustly perform standard navigation tasks.

1) The patrol robot testbed: A patrolling mobile robot testbed has been used to validate the OM Architecture. This application was selected because it involved heterogeneous components, both hardware and software, and had a sufficient level of complexity to prove for generality.

 ISBN: 978-1-61208-276-9

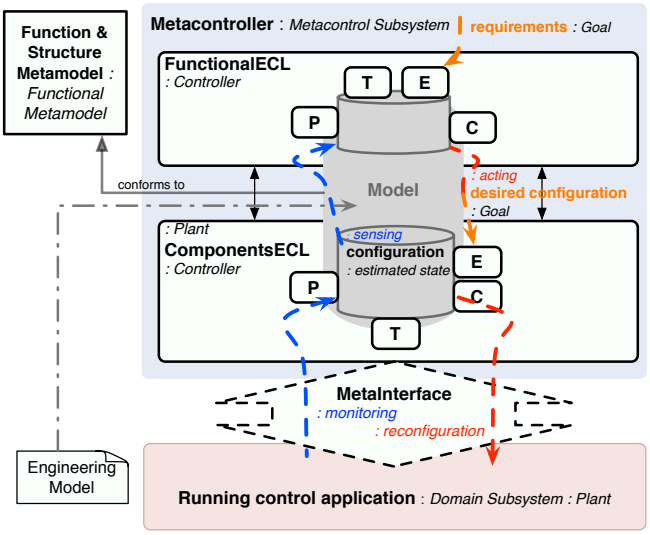

Figure 6. The interplay of the main elements of the OM Architecture Model in the operation of a metacontroller. Functional and Components loops are patterned after the ECL (hence including **P**erception, **T**hink, **E**valuate and **C**ontrol activities around a **M**odel). The roles that each element plays are written in italics after colon marks.

The basic use case consists of the robot (a Piooner 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. The environment has no dynamic obstacles and an initial map is provided. Some runtime deviations from it are possible, *e.g.* chairs may have moved, in order to provide for realistic levels of uncertainty. 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 organization to adapt to the current state of affairs, in order to maintain the mission.

2) Platform development: The platform selected to implement the domain control system is ROS [4] 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 modelable 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 [27], 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 multiplatform implementation of the OM Architectural Model (see Figure 5), including a complete implementation of the Function & Structure Metamodel, the OM Metacontroller, and a Java specification of the MetaInterface.

In order to integrate the OMJava implementation of the Metacontroller with the ROS-based navigation control architecture, an application-independent ROS implementation of the MetaInterface has been developed as a set of ROS nodes. These nodes, together with another one which wraps the OMJava Metacontroller as a ROS node, constitute a PSM of the OM Architecture for the ROS platform.

3) Application development and validation: So far we have described 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 is 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 realize the core functionality through certain configurations of its structure.

Following we briefly describe how the application independent OM processes we have developed exploit the aforementioned knowledge in each of the 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 possible 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 fulfills 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 its control architecture using available components.

V. Future Work

Our current pattern language contains only a small number of patterns centered on core ASys issues. It is necessary to complete it with more common, practical patterns to achieve a full *generative pattern language* [33].

The current implementation of the ICe —the Integrated Control Environment— is just a collection of engineering

 ISBN: 978-1-61208-276-9

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 behavioral 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 [34] 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 Domain Subsystem it controls.

The ambition of ASys is of universality; and hence there is a need for domain generalization, i.e. the extension of theoretical concepts and technological assets to other domains. Current efforts are centered around *autonomous robots* and *continuous process control systems* [11], 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 artifacts, their content may find strong biological roots in animal cognition [35]. 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 [36].

VI. Conclusions

This work shows a pattern-based approach to the construction of sophisticated, self-aware control systems in the domain of autonomous robots. 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 for autonomous systems.

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

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 three patterns offered solutions to very different problems both at design time and runtime, but as the OM Reference architecture and the OMjava realization demonstrate in the testbed, they are easily integrable and can successfully collaborate in generating better system architectures.

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 runtime 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 system adaptation.

Acknowledgements

The authors would like to thank the European Commission for grant HUMANOBS (Humanoids the Learn Socio-Communicative Skills by Imitation).

References

[1] N. Nilsson, “A mobile automaton: An application of artificial intelligence techniques,” AI Center, SRI International, 333 Ravenswood Ave, Menlo Park, CA 94025, Tech. Rep. 40, Mar 1969, sRI Project 7494 IJCAI 1969.

[2] R. A. Brooks, “A robust layered control system for a mobile robot,” IEEE Journal of Robotics and Automation, vol. 2, no. 3, 1986, pp. 14–23.

[3] M. Lindstrom, A. Oreback, and H. I. Christensen, “Berra: a research architecture for service robots,” in IEEE International Conference on Robotics and Automation, 2000. Proceedings. ICRA ’00., San Francisco, CA, USA, April 2000, pp. 3278–3283 vol.4.

[4] M. Quigley, K. Conley, B. P. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, “Ros: an open-source robot operating system,” in ICRA Workshop on Open Source Software, 2009.

[5] M. Shaw and D. Garlan, Software Architecture. An Emerging Discipline. Upper Saddle River, NJ: Prentice-Hall, 1996.

[6] W. Houkes and P. Vermaas, Technical Functions. On the Use and Design of Artefacts. Springer, 2010.

[7] J. Miller and J. Mukerji, “MDA guide v1.0.1,” Object Management Group, Tech. Rep. omg/03-06-01, 2003.

 ISBN: 978-1-61208-276-9

[8] R. Sanz and M. Rodríguez, "The ASys Vision. Engineering Any-X autonomous systems," Universidad Politécnica de Madrid - Autonomous Systems Laboratory, Technical Report ASLAB-R-2007-001, 2007.

[9] W. Meeussen, E. Marder-Eppstein, K. Watts, and B. P. Gerkey, "Long term autonomy in office environments," in ICRA 2011 Workshop on Long-term Autonomy, IEEE. Shanghai, China: IEEE, 05/2011 2011.

[10] B. Hayes-Roth, K. Pfleger, P. Lalanda, P. Morignot, and M. Balabanovic, "A domain-specific software architecture for adaptive intelligent systems," Software Engineering, IEEE Transactions on, vol. 21, no. 4, Apr 1995, pp. 288–301.

[11] M. Rodriguez and R. Sanz, "Development of integrated functional-structural models," in 10th International Symposium on Process Systems Engineers, Salvador, Brasil, August 16-20 2009, pp. 573–578.

[12] C. Hernández, I. López, and R. Sanz, "The operative mind: a functional, computational and modelling approach to machine consciousness," International Journal of Machine Consciousness, vol. 1, no. 1, June 2009, pp. 83–98.

[13] R. Sanz, I. López, M. Rodríguez, and C. Hernández, "Principles for consciousness in integrated cognitive control," Neural Networks, vol. 20, no. 9, 2007, pp. 938–946.

[14] R. C. Conant and W. R. Ashby, "Every good regulator of a system must be a model of that system," International Journal of Systems Science, vol. 1, no. 2, 1970, pp. 89–97.

[15] C. Landauer and K. L. Bellman, "Meta-analysis and reflection as system development strategies," in Metainformatics. International Symposium MIS 2003, ser. LNCS. Springer-Verlag, 2004, no. 3002, pp. 178–196.

[16] L. Balmelli, D. Brown, M. Cantor, and M. Mott, "Model-driven systems development," IBM Systems journal, vol. 45, no. 3, 2006, pp. 569–585.

[17] R. Sanz and J. Zalewski, "Pattern-based control systems engineering," IEEE Control Systems Magazine, vol. 23, no. 3, June 2003, pp. 43–60.

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

[19] J.-M. Perronne, L. Thiry, and B. Thirion, "Architectural concepts and design patterns for behavior modeling and integration," Math. Comput. Simul., vol. 70, no. 5-6, Feb. 2006, pp. 314–329.

[20] R. Sanz, A. Yela, and R. Chinchilla, "A pattern schema for complex controllers," Emerging Technologies and Factory Automation, 2003. Proceedings. ETFA '03. IEEE Conference, vol. 2, Sept. 2003, pp. 101–105 vol.2, .

[21] N. G. Leveson, Engineering a Safer World Engineering a Safer World: Systems Thinking Applied to Safety. The MIT Press, 2012.

[22] A. Pike, M. Grimble, A. O. M.A. Johnson, and S. Shakoor, The Control Handbook. CRC Press, 1996, ch. Predictive Control, pp. 805–814.

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

[24] J. S. Albus and A. J. Barbera, "RCS: A cognitive architecture for intelligent multi-agent systems," Annual Reviews in Control, vol. 29, no. 1, 2005, pp. 87–99.

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

[26] F. P. Osinga, Science, Strategy and War: The Strategic Theory of John Boyd. Routledge, 2007.

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

[28] M. Blanke, M. Kinnaert, J. Lunze, and M. Staroswiecki, Diagnosis and Fault-Tolerant Control. Springer-Verlag Berlin, 2006.

[29] J. L. de la Mata and M. Rodríguez, "Accident prevention by control system reconfiguration," Computers & Chemical Engineering, vol. 34, no. 5, 2010, pp. 846 – 855.

[30] T. Kühne, "Matters of (meta-)modeling," Software and System Modeling, vol. 5, no. 4, July 2006, pp. 369–385.

[31] M. Lind, "Modeling goals and functions of complex industrial plants," Applied Artificial Intelligence, vol. 8, 1994, pp. 259–283.

[32] J. Bermejo-Alonso, R. Sanz, M. Rodríguez, and C. Hernández, "An ontological framework for autonomous systems modelling," International Journal on Advances in Intelligent Systems, vol. 3, no. 3, 2010, pp. 211–225.

[33] D. Brugali and K. Sycara, "Frameworks and pattern languages," ACM Computing Surveys, March 2000.

[34] J. Bermejo-Alonso, "OASys: An ontology for autonomous systems," Ph.D. dissertation, Departamento de Automática, Universidad Politécnica de Madrid, 2010.

[35] C. Hernández, R. Sanz, J. Gómez-Ramirez, L. S. Smith, A. Hussain, A. Chella, and I. Aleksander, Eds., From Brains to Systems. Brain-Inspired Cognitive Systems 2010. Springer, 2011.

[36] R. Sanz, "Machines among us: Minds and the engineering of control systems," APA Newsletters - Newsletter on Philosophy and Computers, vol. 10, no. 1, 2010, pp. 12–17.

 ISBN: 978-1-61208-276-9