A Pattern Schema for Complex Controllers

Ricardo Sanz, Adolfo Yela and Rafael Chinchilla

Autonomous Systems Laboratory
Universidad Politécnica de Madrid
José Gutiérrez Abascal 2
28006 Madrid, Spain
ricardo.sanz@etsii.upm.es

***Abstract*— Modern control systems are very complex applications that pose special difficulties to systems designers. Design patterns are common in the field of automatic control but in most cases they are restricted to small subfields of application. In this paper we propose a pattern schema with two particular properties: i) it deals with some issues very relevant to control designers and ii) it is generic enough to gather under the same umbrella an heterogeneous collection of patterns ranging from basic, elementary control components to plant-wide systems. This schema is being used in the implementation of a pattern language for the Integrated Control Architecture project.**

I. Introduction

This paper presents a *pattern schema* proposed for the documentation of design patterns in complex control systems. This schema is being used to document design patterns to be supported by the *Integrated Control Architecture* (ICa) technology [1].

ICa is an ongoing development project that tries to provide methods and tools to address the construction of complex control systems in an effective way. Complex control systems are in most cases software-intensive applications that use advanced software technologies and have requirements that go well beyond the knowledge of single disciplines.

ICa is based on the use of control components and domain specific architectures. All them are deployed over real-time middleware based on OMG specifications.

The purpose of this paper is to present this pattern schema and justify its structure under the umbrella of the intended purpose: the transfer of design knowledge between control systems designers and implementors.

II. Complexity in software-intensive control systems

There is an unavoidable tendency in control systems construction that lead us to extreme software complexity [2]. Software complexity is common to other fields with software intensive applications, but in the field of control systems there is an overlapping of complexity factors that make the situation harder.

As an example of a complex control application, the RiskMan system for emergency handling in chemical plants [3] deploys expert system technology for human decision support, fuzzy filtering engines to enhance raw plant data, neural-network models of process units, real-time databases and legacy control applications as a collection of distributed real-time agents over middleware running atop heterogeneous computing platforms. This amounts for millions lines of multilanguage code.

There is progressive awareness in the control engineering field on the need of having a deep knowledge about computing systems. Control engineers must master computer and software technologies to be able to build systems that meet complex requirements.

In many cases, control engineers do develop their own software tools and methods to deal with the special aspects of advanced controllers.

Some of these aspects that induce controller complexity are:

Real-time: Meeting deadlines is always a problem, and providing guarantees about it is extremely difficult, specially when modern systems are based on a enormous amount of software (development tools, operating systems, virtual machines, network protocols, middleware, *etc.*) and hardware (processors, networks, intelligent devices, *etc.*).

Embedded: Running within limited resources is hard and could be harder due to the use of complex software development tools. Fortunately advances in hardware platforms provide some hope for us.

Fault-tolerant: Real-time control systems should demonstrate good behavior under faults. This is critical in some situations/plants where potential damage in case of malfunction is enormous.

Distributed: Present day control systems are inherently distributed over heterogeneous platforms and networks. They run on several interacting computers with varying hardware, operating systems and networking software.

Intelligent: Artificial intelligence is commonly used in solving ill-posed problems in control systems. This means that in many cases it is necessary the integration of third party AI tools in an, otherwise, real-time application.

Large: Many times, these systems (distributed, fault-tolerant, intelligent, etc.) are composed by zillions of lines of code if they meet real world requirements.

Integrated: Even when developed in pieces, using heterogeneous tools, in different time scales and by different people, all these system components should constitute a single, cohesive application in order to be fully functional and manageable.

Open: In many cases plant controllers do interoperate with alien systems to interchange state information and/or

control actions.

Heterogeneous: Real systems run on heterogeneous platforms that are old legacy platforms or new state of the art hardware/software toys.

Ten years ago research on control systems software was mostly focused on just one of these topics. Today, even the simplest real controller falls in several of these categories.

III. The Integrated Control Approach

The ICa approach to this problem is the use of real-time reusable object-oriented components [4] deployed over real-time middleware [5] using an architecture-centric development process [6], [7], [8], [9]. The basic tool for design documentation and architecture specification is the *design pattern*.

Patterns are a critical technology for adequate design knowledge transfer between complex system stakeholders. Patterns can be used to document many design aspects of a systems. While most of the work in this field has been devoted to basic software design patterns in object systems [10], [11], [12], there are many other developments in distribution [13], [14], hardware [15] or even bad practices [16]. There are also scattered works related with the use of patterns in real time systems [17], [18], [19], [20], [21], [22].

IV. Classical Pattern Schemata

To make patterns reusable, adequate descriptions for them should be available. Authors present their patterns in ways suitable for their intended use. They are known as pattern schemata, pattern forms, pattern templates, etc. Examples of these schemata can be found in the literature [11] [23] [13].

In the Patterns FAQ maintained by Doug Lea, he proposes this format based on Alexander's work:

```
IF    you find yourself in CONTEXT
      for example in EXAMPLE
      with a PROBLEM
      entailing some FORCES
THEN  for some REASONS
      apply a DESIGN RULE
      to construct a SOLUTION
      leading you to a NEW CONTEX
               and OTHER PATTERNS
```

The pattern schema used by Buschmann *et al.* in their architectural patterns book [12] is composed of the following sections:

1) **Name**: The name of the pattern.
2) **Also known as**: Aliases.
3) **Example**: Example of use.
4) **Context**: When to use it.
5) **Problem**: Problem to solve.
6) **Solution**: Solution proposed to the problem.
7) **Structure**: How solution is achieved.
8) **Dynamics**: Behavior of the entities involved.
9) **Implementation**: Sample implementation.
10) **Example resolved**: Example with solution applied.
11) **Variants**: Patterns that share structure.
12) **Known Uses**: Examples of real use.
13) **Consequences**: What would you expect to happen.
14) **See also**: Related patterns.

If you want to write a pattern describing a design solution, all you need to do is to adequately fill the relevant sections for your pattern.[1]

V. A Pattern Language

A pattern language is a collection of interrelated patterns that can be used to describe or design a system [23]. The objective of the Integrated Control Pattern Language (ICa PL for short) is to provide an extensible framework to express control application designs (mostly architecture in the domain of complex controllers).

It is not intended as a general pattern language for software architecture description even when most of the software patterns are generic enough to be applied in other domains. ICa PL offers a collection of patterns to address common problems in the implementation of advanced controllers for complex systems. This means that while the patterns are generic to be reusable they must be specific to concrete problems or tasks that appear in this type of systems.

VI. ICa Pattern Schema

This section contains the pattern description structure (the *pattern schema*) that was built to provide easy conceptual understanding by control designers and immediate applicability of the pattern.

Obviously not all sections of this schema are equally applicable to all possible patterns, mainly because patterns usually refer to different architectural levels. In specific cases, some sections can disappear and other new sections can appear for specific to address concrete issues related with that class of patterns.

Name The name of the pattern. Names are important if things named are going to be reused, inter-changed and discussed. Their name should be simple but with a clear relation with the pattern itself.

Aliases Patterns are usually not new; most of them were employed during long time using different names for them.

Example The example provides a sample framework of application of the pattern. This section identifies a possible use of the pattern in a real situation.

Problem The problem the pattern tries to solve.

Solution What the pattern does. What type of solution it provides.

Forces The competing factors that makes difficult the decision of the system architect.

Context Contextual information regarding the framework of application of the pattern.

Structure Architectural description of the pattern. Roles and relations between roles. It is usually a diagram. It includes a description of all the roles that appear in the pattern.

[1] Well, it is not completely true. Writing *good* patterns is very difficult and requires a lot of work previous to the put-on-paper task and, perhaps, even more after it.

Dynamics How system activity happens. Sequences of role activation. It is important to note that adequate documentation tools are critical for this section. Message sequence charts or finite state machines are common tools for this work but there are many other alternatives like for example use case maps that are very suitable to high level dynamics description.

Implementation Practical issues regarding the put on practice of the pattern to a real situation.

Timing Temporal considerations for the pattern. It is of special importance to control systems and/or real-time applications because they can fail if action timing is not adequate.

Example application Issues regarding the application of the pattern to the framework presented in the example section.

Variants Common modifications to the basic pattern structure. Known deviations and heresies from the orthodox use.

Known Uses Examples of the use of the pattern in real life.

Consequences Implications derived from the use of the pattern in an application. Both unavoidable consequences (desirable or not) and those that should be stressed by the system designer.

Related with Other patterns related with this, by structure, by way of use or -very important- be-cause they are applied at the same time to a system.

References Bibliographic references for the pattern.

VII. Pattern Example

A. Name

LAYERS[2]

B. Aliases

Layered control, hierarchical control, control loop nesting.

C. Example

Cement kiln control is a complex problem because there are some ill conditions. The two principal are:

1) The sensing system is not as good as it should be because there are unmeasurable magnitudes (for example the most critical one: burning zone temperature) or the sensing process suffers great delays .
2) The controllability of the kiln is quite limited. For example, raw material input composition can only be approximately set. Responsiveness of the kiln to control actions is also limited. Control of kilns is done by a mixture of simple controllers (usually PLCs) in charge of maintaining basic process magnitudes, and a human operator that takes all decisions concerning the setpoints of the basic controllers. Operator experience has a critical influence on the state and performance of the kiln. Continuous attention to critical magnitudes (for example contents of NO_x in exhauster gases) is also of extreme importance.

[2]This pattern has been extensively described in the literature using the same name. But in each case the focus is slightly different due to the orientation of the intended use. The foundation, however, remains the same in all cases.

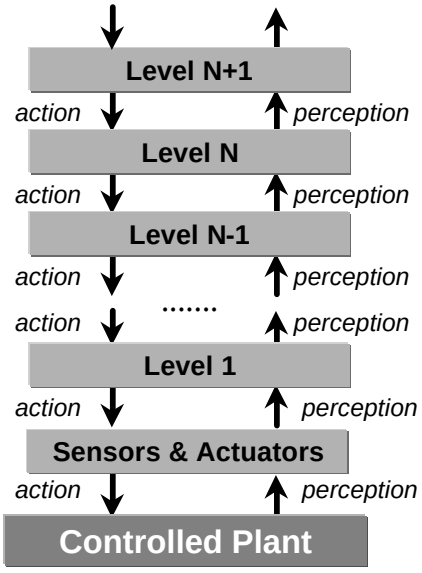

Fig. 1. Layer relations in the LAYERS control pattern are just structured perceptions and actions with neighbors.

D. Task

Provide increasingly complex intelligent behavior without sacrificing performance and/or reliability.

E. Problem

The problem that LAYERS tries to solve is the complex integration issues that can appear when control making components have different cognitive and action capabilities.

Some authors refer to the principle of increasing precision with decreasing intelligence [25]. In most cases, we should talk not only about precision, but speed, trustability and implementation technology.

F. Context

Plants with dynamic problems that are not well addressed using a single control technology approach.

In most cases, heterogeneous control systems are required to provide different levels of intelligence and speed.

G. Solution

A complete gradation of authority is specified and controllers attain to it.

Each cognitive level interacts only with the adjacent levels. Interactions with upper layers are *perception* interactions; interactions with lower layers are *action* interactions.

Additional advantages are obtained from the use of LAYERS. For example, the clear decomposition simplifies development of the components (the layers) of the system, because interaction and dependency are limited.

H. Structure

Interaction between layers take the form of perception/action pairs. Perception goes upwards toward increasing intelligence. Actions go downwards towards the real plant under control. Layer L_N get information about the state of the virtual plant it sees from the immediately lower layer. For layer L_N the plant under control is the real plant plus all the layers up to it: The unique control actions available to L_N are those control inputs admissible for L_N.

I. Dynamics

There are two typical layer behavior variants in these type of systems:

- *Active layers* that actively get state information from the lower layer and perform actions based on this information.
- *Passive layers* that perform their activity only upon reception of an activity request. The two typical activation policies are all active layers and event driven activity triggered by state reception from lower layers.

J. Implementation

Layers are usually implemented as sets of processes. Couplings between layers are simple enough to be implemented using whatever type of integration mechanism is available. In the example described in this pattern TCP/IP message based interaction between processes was used.

K. Timing

It depends clearly on the application and the layers involved. Lower layers are usually faster than upper layers. There are some typical timings for continuous process plants ranging from milliseconds in sensing an basic control loops to hours in strategic decision making. In the mobile robotics area, times are -roughly- the same. In faster applications (avionics, machine control, etc) times can be one tenth of these.

L. Example application

The example for this pattern is the CONEX [26] application (See Figure 2). It is and agent-based, distributed intelligent control system developed for a cement kiln. This system uses the LAYERS pattern to implement a multilayered control system based on heterogeneous control technologies. The objective of the controller is the optimization of clinker burning conditions (temperature, time, oxygen, etc). The system was designed to maximize control capability to maintain kiln conditions under strict margins.

All these layers were implemented using independent active agents. Each agent was composed by two or more VMS processes using message-based interactions over TCP/IP. This approach made possible the execution of trustable (in logical and temporal behavior) agents in one computer and non-trustable agents in other computer. This division was of extreme importance to provide 365×24 availability of the lower layers.

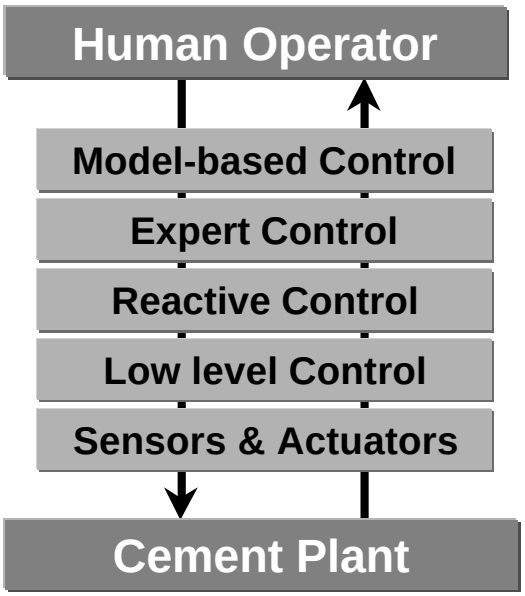

Fig. 2. CONEX: An example of a layered intelligent controller for a cement plant using the LAYERS pattern.

M. Variants

Minor deviations are usual in the application of this pattern.

Sometimes, specific layers do violate basic pattern design guidelines to provide specific performance enhancements.

It is usual to find superimposed patterns in the same system, because each design provides solutions to specific problems. In the CONEX example, the *Model and Simulator* agent interacted with all the layers. It provided a three tiered model of the plant (numerical, qualitative and knowledge based) to support the whole range of activities of the rest of the system.

A variant that is included as another pattern in the ICa PL is the CONTROL TUNER pattern.

N. Known Uses

Layers is used in all types of control systems with a minimum of complexity.

VIII. CONCLUSIONS

Design patterns are a very useful vehicle for design expression and design knowledge transmission.

In the field of complex control systems this is of *extreme utility* as the knowledge involved in the construction of any complex applications spans several disciplines.

Patterns can effectively be used to transfer design knowledge from different but complementary engineering perspectives (control, software, reliability, *etc.*) using a common language understood by all system stakeholders.

This paper has presented a *pattern schema* used in the Integrated Control Architecture Pattern Language to capture design knowledge of complex control systems.

This schema is being used to capture the wide design knowledge used in the construction of complex control systems and used to build new systems based on reusable components.

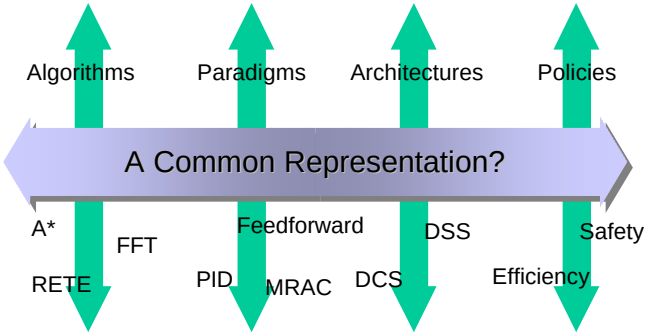

Fig. 3. The big challenge is finding a common pattern schema that can be used for different types of complex controller design knowledge.

REFERENCES

[1] R. Sanz, M. J. Segarra, A. de Antonio, and J. A. Clavijo, "ICa: Middleware for intelligent process control," in *IEEE International Symposium on Intelligent Control, ISIC'1999*, Cambridge, USA, 1999.

[2] R. Sanz, W. Schaufelberger, C. Pfister, and A. de Antonio, "Software for complex control systems," in *Control of Complex Systems*, K. A. ström, P. Albertos, M. Blanke, A. Isidori, W. Schaufelberger, and R. Sanz, Eds. Springer, 2000, ch. 7.

[3] R. Sanz, M. Segarra, A. de Antonio, I. Alarcón, F. Matía, and A. Jiménez, "Plant-wide risk management using distributed objects," in *IFAC SAFEPROCESS'2000*, Budapest, Hungary, 2000.

[4] R. Sanz, "ORC::Industry," in *Proceedings of the IEEE International Symposium on Real-time Object-Oriented Computing, ISORC'2001*, Magdeburg, Germany, 2001.

[5] D. C. S. et al., *Pattern Oriented Software Architecture: Patterns for Concurrent and Distributed Objects*. Chichester: Wiley, 2000.

[6] M. Alarcón, P. Rodrí guez, L. Almeida, R. Sanz, L. Fontaine, P. Gómez, X. Alamán, P. Nordin, H. Bejder, and E. de Pablo, "Heterogeneous integration architecture for intelligent control," *Intelligent Systems Engineering*, 1994.

[7] J. S. Albus, "A reference model architecture for intelligent systems design," in *An Introduction to Intelligent and Autonomous Control*, P. Antsaklis and K. Passino, Eds. Boston, MA: Kluwer Academic Publishers, 1992, pp. 57–64.

[8] P. Bernus, L. Nemes, and T. J. Williams, Eds., *Architectures for Enterprise Integration*. London: Chapman & Hall, 1996.

[9] S. Bhansali, "Software synthesis using generic architectures," *Automated Software Engineering*, vol. 1, no. 3/4, pp. 239–279, 1994.

[10] K. Beck, J. O. Coplien, R. Crocker, L. Dominick, G. Meszaros, F. Paulisch, and J. Vlissides, "Industrial experience with design patterns," in *Proceedings of ICSE*, 1996, pp. 103–114.

[11] E. Gamma, R. Helm, R. Johnson, and J. Vlissides, *Design Patterns. Elements of Reusable Object Oriented Software*. Reading, MA: Addison-Wesley, 1995.

[12] F. Buschmann, R. Meunier, H. Rohnert, P. Sommerlad, and M. Stal, *Pattern Oriented Software Architecture. A System of Patterns*. Chichester, UK: Wiley, 1996.

[13] T. Mowbray and R.C.Malveau, *CORBA Design Patterns*. Chichester: Wiley, 1997.

[14] D. C. Schmidt and F. Kuhns, "An overview of the real-time CORBA specification," *Computer*, vol. 33, no. 6, pp. 56–63, June 2000.

[15] M. Pont, *Patterns for Time-Triggered Embedded Systems: Building Reliable Applications with the 8051 Family of Microcontrollers*. London: Addison-Wesley, 2001.

[16] W. Brown, R. Malveau, H. McCormick, and T. Mowbray, *Antipatterns: Refactoring Software, Architectures and Projects in Crisis*. Chichester: Wiley, 1998.

[17] D. Lea, "Design patterns for avionics control systems," State University of New York, Oswego, Tech. Rep. ADAGE-OSW-94-01, November 1994, dSSA ADAGE Project.

[18] B. Selic and G. Gullekson, "Design patterns for real-time software," in *Proceeding of Embedded Systems Conference West*, 1996.

[19] S. Kuikka, T. Tommila, and O. Ventä, "Distributed batch process management framework based on design patterns and software components," in *Proceedings of the 14th World Congress of IFAC*, Beijing, China, 1999.

[20] R. Sanz, M. J. Segarra, A. de Antonio, F. Matía, A. Jiménez, and R. Galán, "Design patterns in intelligent control systems," in *Proceedings of IFAC 14th World Congress*, Beijing, China, 1999.

[21] J. Eker and A. Blomdell, "Patterns in embedded control systems," Department of Automatic Control. Lund Institute of Technology, Techreport ISRN LUFTD2/TFRT–7567–SE, October 1997.

[22] J. Zalewski, "Real-time software architectures and design patterns: fundamental concepts and their consequences," *Annual Reviews in Control*, vol. 25, no. 1, pp. 133–146, July 2001.

[23] J. O. Coplien and D. C. Schmidt, *Pattern Languages of Program Design*. Reading, MA: Addison-Wesley, 1995.

[24] R. Buhr, "Understanding large-scale behavior patterns in complex systems," Department of Systems and Computer Engineering, 1996.

[25] G. G. Saridis and K. P. Valavanis, "Analytical desing of intelligent machines," *Automatica*, vol. 24, no. 2, pp. 123–133, 1988.

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

[27] A. Aarsten, D. Brugali, and G. Menga, "Designing concurrent and distributed control systems," *Communications of the ACM*, vol. 39, no. 10, pp. 50–58, 1996.