A Design Pattern meta model for Systems Engineering

François Pfister* Vincent Chapurlat* Marianne Huchard** and Clémentine Nebut**

*LG12P, Ecole des Mines d'Alès, site de Nîmes, Parc Scientifique Georges Besse, 30000 Nîmes, France (e-mail: {forename.lastname}@mines-ales.fr)

**LIRMM, CNRS – Université Montpellier 2, 161 rue Ada, 34095 Montpellier Cedex 5, France (e-mail: {lastname}@lirmm.fr)

Abstract: Design patterns are widely used in software engineering. Building on lessons learned in this domain, this article proposes a suitable design pattern meta model for systems engineering with two goals. The first one is to help designers to leverage and reuse knowledge required in various engineering domains during design phases. The second one is to propose and formalize an approach that allows designers and systems architects to identify relevant design patterns and apply them in a semi-automatic manner. This approach consists to improve model alignment techniques based on model interoperability hypothesis.

Keywords: design-pattern, systems engineering, model alignment, model driven engineering

1. INTRODUCTION

In Systems Engineering (SE) area, the approach called Model Based Systems Engineering (MBSE) (Estefian, 2008) proposes to adopt the principles of MDE (Model Driven Engineering) that treats models as first-class entities and should in particular lead to build sustainable domain models. In addition, Systems Engineering, as software engineering, uses various modelling languages, themselves based on different meta models: the succession of these conceptual foundations is an obstacle to domain models re-usability.

MBSE aims at achieving such re-usability despite technological evolutions by founding it above a stable conceptual pyramid like OMG’s. Systems may be composed of sub-systems which often are heterogeneous, themselves composed of software and hardware components or sub-systems, interfaced with human actors. Therefore, it is necessary to ensure, during the design phase, interoperability between each connected sub-system, in spite of the final system native heterogeneity or complexity. In accordance with (Pingaud 2009), “interoperability can be defined as an aptitude for two foreign systems to interact in order to establish collective, harmonious, and finalized behaviours, without the need of modifying deeply their own structure and behaviour.” Systems efficiency, in terms of their missions, depends on their ability to establish and maintain connections with surrounding systems. Several types of interoperability barriers are emerging (technical, informational, functional, and semantic) (ATHENA, 2003). Our target interests are here the semantic and functional levels. Achieving interoperability of two systems needs, first, to establish correspondences between models that represent the physical architectures of these two systems, and, secondly, to establish correspondences between the functions of both systems. Interoperability at the semantic level is possible if the concepts and then the modelling languages allowing describing systems to interoperate are similar, close, or compatible. These concepts are gathered into various views such as class diagrams, semantic networks, and relational schemas. This interoperability is achieved when it is possible to become able to compare each conceptual entity of the models describing these systems. So, a prerequisite is the semantic alignment of these models.

Interoperability at the functional level is achieved if we can establish a relationship of equivalence between two functions, each belonging to one of the systems to interoperate: two functions are equivalent if they produce identical or compatible outputs for given inputs, resources, and control inputs. So, a prerequisite will be the alignment of functional models of the concerned systems. A function is an operation performed by a resource of the system, which absorbs an input stream (information, matter or energy), and produces a result as an output stream. This transformation needs support mechanisms, and is governed by controls. The relevant views considered here are then activity diagrams for UML and SysML, BPML diagrams, or IDEF0 actigrams (Hafedh, 2003). So, to resolve interoperability issues, it is necessary to perform model matching on the models representing these systems, in order to align them. One key to interoperability is based on the mapping of concepts and processes for each system to interoperate. Matching different but compatible systems (by aligning their models) can be done manually by domain experts, but also in a semi-automatic way. Indeed, current models may contain hundreds or thousands of entities, making fully manual alignment nearly infeasible. To solve this issue, methods and tools have been proposed to assist alignment (Falleri et al, 2010). In the field of Manufacturing Systems, a number of interoperability issues are due to the different and heterogeneous layers and component involved in the production control, through the manufacturing pyramid, going from the enterprise network in...
a decisional environment down to the plant floor layer in a
sensor, actuator, product flow environment. The Pabadis
Promise (Diep, D. et al., 2007) project promoted an agent
based infrastructure, and a dataflow which takes the form of
ontology fragments. The latter is built upon the ISA 95
Standards, and interoperable with it.

This paper shows how the identification and use of design
patterns can guide the model alignment process. The goal of
the presented research work is to make a model pre-handling,
in order to find design pattern correspondences for each
model, as described further. Such a pre-handling consists of a
decomposition which will be useful to facilitate model
alignment. This paper considers also the potential benefits
gained using design patterns in Systems Engineering.

Section 2 gives some design pattern definitions. Section 3
describes the design pattern application mechanisms. Sections 4 and 5 distinguish between idiomatic and domain
specific design patterns and locate them within the Systems
Engineering processes. Sections 6 and 7 expose the research
contributions: a design pattern metamodel, and the benefits of
applying design-patterns in the field of model-mapping
before concluding in section 6.

2. DESIGN PATTERN DEFINITION

Design patterns are one of the approaches used to abstract
individual constructions within an overall architecture. This
idea, originally proposed by Christopher Alexander
(Alexander et al., 1977), has been widely used in software
ingineering (Gamma et al., 1994). A design pattern is a way
to represent invariant knowledge and experience in design by
practitioners. It can help human actors to identify and solve
problems by drawing or imitating such knowledge and
experience. The objectives are: to gain performance
(comprehensiveness, relevance), reliability (proven solutions,
justified and contextual argued), to gain economic value
time savings and, finally, to facilitate collaborative work by
sharing design patterns repositories. These objectives can be
achieved by leveraging and integrating such knowledge, good
practices and lessons learned, and by formalizing them for
reuse. So design patterns are a mean to formalize and create
standard solutions repositories in response to known and
frequently encountered issues in a particular field.

We propose the use of design patterns in Systems
Engineering, not in response to technical implementation
problems as proposed in (Gamma et al, 1994) but to
structure the functional and physical architectures. In this
way, a design pattern is a simple and small artefact, rarely
isolated and therefore correlated with others. A design
pattern is described by an abstract model (physical or
functional) to be imitated by models describing the target
system to design. It defines the collaboration of some system
components or some system functions to contribute to a
given mission. A design pattern is not a creativity method (by
definition, it exists only if the proposed solution is well
known and frequently used in the field and, therefore, is not
innovative). In the same manner, it is not a reusable
component. It is destined to be imitated and adapted to a
particular context.

According to the Model-Based/Model-Driven initiative
(Soley 2000), a design pattern should be represented using a
formalised language, handled through a design tool and able
to be involved in model transformations. Patterns (and
models on which they apply) can be then expressed with
languages such as SysML (OMG), which is an UML profile,
or STEP (ISO), based on the EXPRESS schema (Schenk,
1994), and also OWL (W3C). Within models, design patterns
apply a crystallization process (Baudry, 2003) (Jézéquel et
al., 2005). Impacted entities fit together in a configuration to
meet specific roles defined in the pattern. In the proposed
work, design patterns are defined in UML or SysML as
parameterized collaborations. They specify a set of classes
and objects that have specific roles and interactions.

Many proposals for formalizing patterns exist. These include
for example the P-Sigma formalism (Conte et al., 2002), and
also (Gizara 2000) contribution who proposes the application
of design patterns for product data management, that is, to
structure a model of physical system components. Many
efforts have been undertaken by the promoters of MBSE to
integrate design patterns into the models developed by
Systems Engineering (Cloutier, 2007). The AFIS
(Association Française d'Ingénierie Système - French
Association of Systems Engineering), the French chapter of
INCOSE, has been mandated to define a design pattern
methodology in System Design. Our participation in the
Technical Committee MBSE within AFIS gives us the
opportunity to begin a formal reflection to represent design
patterns on the basis of SysML or its implementation by the
ISO 10303 - STEP AP233.

3. APPLICATION MECHANISMS

UML uses the term pattern as a synonym for parameterized
collaboration. A parameterized collaboration is used when, in
the basic way, the current classes of a model work in the
same way as the collaboration classes, but classes and
operations are named differently. Collaboration is a name
given to the interaction among two or more classes. Typically
this is described in an interaction diagram (Larman, 2001).
Transposed to SysML, a system design pattern could be
described as an internal block diagram, coming with a
sequence diagram. An activity diagram (functional view) and
a state diagram (behavioural view) can be supplied.

When designing a model, some part of it, revealing a design
challenge, may require the application of a design pattern.
Once the design pattern is chosen from a model repository by
the domain expert who is currently working on this model, a
pattern instance is parameterized with involved model entities
(i.e. blocks, components, objects). As soon as the design
pattern instance is applied to the model, the model entities are
reconfigured to comply the design pattern (their structure and
composition may be amended, supplemented or renamed, constraints can be added). It works as a local model modification; the latter mimics the applied design pattern. The design pattern instance persists within the model as a parameterized collaboration, and can be used to improve the model alignment. If the current domain is mechanics for example, model entities will represent mechanical components. Applying some pattern could allocate particular constraints (structure, geometry, surface aspect, non-functional requirements) on the model components. These fit together so that they assume the roles of the parameters defined for the collaboration that represents the pattern in the model. These roles represent associations between current model components and the pattern participants (i.e. the components composing the patterns). If the pattern is involved in a functional architecture, the impacted elements are functions or activities.

4. IDIOMATIC PATTERNS

We will distinguish two major types of patterns: idiomatic patterns and domain patterns. An idiomatic pattern describes low level, structuring elements of a model, governing associations between these elements, defining aggregation and containment strategies. These are low-level structures that do not affect the architectural semantics. The GOF patterns (Gamma et al., 1994) could be classified as such, in the software engineering domain. Similarly, in Systems Engineering, physical components are sometimes expressed with ISO10303 AP214 or AP210 STEP models (physical components in the fields of mechanics and electronics).

![Fig. 1. Loose Coupling Assembly design pattern](image)

The examination of an idiomatic pattern applied in the STEP product representation (Fig. 1) allows us to find that the Loose Coupling design pattern is far from the Composite design pattern described by (Gamma et al., 1994). Instead, we can detect a composition design pattern based on a bipartite graph. One graph represents the structure of the product, and the other represents its geometry. In such a model, it will be possible to stack, as layers, many sub-models involved in various aspects of the component representation (functional model, performance model, geometric model ...). Patterns at this level, participate in the concern of model representation, and have no effect on the nature of the end systems. It is necessary to detect those patterns in the physical architecture that our alignment algorithms must walk through, but these patterns are not involved in the macroscopic system semantics we are targeting.

5. DOMAIN SPECIFIC AND SYSTEM ENGINEERING PATTERNS

(Pont, M.J., 2001) provides a set of patterns to support the development of embedded software systems. Though its main domain is software, proposed patterns may be material ones (e.g. RC Reset), or involving a mix of material elements and algorithmic principles (e.g. Software Switch Debouncer). Such patterns are not idiomatic as GOF patterns, they are specific to real-time software. (Sanz, R. et al, 2003) proposes design patterns in the field of control systems, in order to manage their complexity, improve many non-functional features such as fault-tolerance, embedding, time reactivity, heterogeneity. (Brandl, D., 2006) describes the ISA-88 industry standards as a set of design-patterns in the field of batch manufacturing (S88 design pattern) and continuous and discrete manufacturing (NS88 design pattern for non-stop production). His pattern models are founded on a petri-net based language (procedural function charts). The target domains are, for example, pharmaceutical industries. Thus, contrary to the idiomatic patterns, this research seeks design patterns that capture the various domain concerns about the mission and the functions achieved by the system under design. So, the example proposed in Section 7 describes a case of applying a regulation design pattern within the functional and physical architecture of motor vehicles.

Design pattern mobilization occurs, in Systems Engineering, during the functional and the physical architecture design processes as proposed in (ISO 2008), while describing the solution carried out by the pattern. The solution will be adapted from a generic case described by the design pattern to the current specific context of the model under study. Functional solutions will be provided, but also non-functional features such as reliability, security, and performance.

6. CONTRIBUTION: A DESIGN PATTERN METAMODEL FOR SYSTEMS ENGINEERING

The first contribution of this paper is then to propose a design pattern metamodel. It relies the Cloutier proposal (Cloutier, 2007) but considers more formally the design pattern metamodel (Fig. 3 & Fig. 4) within a (candidate) System metamodel (Fig. 2). The design pattern for Systems Engineering (SystemPattern) has participants who can be physical components, or functions in the case of functional patterns. The fragment of an actual model impacted by a design pattern imitates the latter if the actual model’s components or functions play the role of the design pattern participants, and also if the model’s dynamic behaviour mimics the dynamic behaviour described by the pattern.

Fig. 3 shows the design pattern meta-class (SystemPattern) associated with related, requested or mandatory, equivalent and anti-patterns. It can have multiple aliases (aka, also-known-as). A set of keywords allows its localization, and it has a rationale. A design pattern is legitimated by citing
known application cases (known-uses). Its participants are components or functions, each component being associated with one another through interfaces that convey informational or physical flows (Item on Fig. 2).

Fig. 2. Systems Engineering patterns within a System metamodel (overall vision)

As shown in Fig. 4, a design pattern embeds a controlled vocabulary depending on the domain to which it belongs. The application context of the design pattern is described by defining the set of constraints and contradictions that the design pattern solves (forces). The impact of the design pattern implementation is evaluated. Finally, a model containing the problem solved by the pattern, and a model containing the solution to be imitated by the current model are provided. These models can be described with different languages, each language offering different views (static, dynamic, functional, behavioural).

7. PATTERNS MAY IMPROVE MODEL ALIGNMENT

The second contribution of this work is about model alignment mechanisms that become possible to use. Model transformation, according to the MDA paradigm, is based on a transformation model instance of a meta-model as QVT (Bézivin, 2001). This transformation model formalizes a mapping of correspondences between the source model elements and the target ones. When matching rules are not trivial, the transformation model formalizes more complex imperative rules. The transformation model can be constructed manually by an expert of the relevant modelling domain. Actually, these models may contain hundreds or thousands of items, so it would be useful to rely on a tool to assist the expert. The proposal consist to use the proposed meta model of design patterns to improve the automated discovery of mapping alignment, in combination with existing techniques (Falleri et al. 2010) (Euzenat et al., 2004). This is done by analysing the pattern instances which are embedded, when explicitly applied, or buried and detected (Tonella et al., 1999) (Arevalo & al. 2004) (Gueheneuc, 2008).

Fig. 3. System design pattern meta-model (A).

The example shown in Fig. 5 describes a cruise-control pattern, applied to an electric car, on the one hand, and on a conventional thermic-engined car on the other. Each model mimics this design pattern to define its own cruise-control subsystem. Thus, this case represents the need for a system architect to make interoperable two automotive systems helped by aligning their models. The design pattern is described by its organizational structure (functional and dynamic views have been omitted). The design pattern participants (throttle, brake, transmission control unit, etc.) that compose a solution-type for a cruise control, are associated with the roles played by the components of the two current models that imitate the pattern. A classical alignment model based on the Similarity Flooding (Melnik et al., 2002), which is a method of propagation of similarities in a labelled graph to determine a match, will be used to generate a mapping between two models. This algorithm will be effectively complemented by analysing the roles connecting the design pattern to both models.
8. CONCLUSIONS

This article promotes first a design pattern metamodel for Systems Engineering (SE) that complete the process initiated by INCOSE. This metamodel takes advantage of lessons learned in software engineering, but adapts the principle to SE area. If a model keeps trace of patterns it imitates, the latter will contribute to document it and to promote interoperability of the represented system, with surrounding systems. The system components are modelled with architectural languages as block diagrams, or other languages used to represent products throughout their life cycle, such as STEP. In continuation, the current work focuses to implement design patterns within this family of languages, and study the consequences in terms of interoperability in these specific areas. Finally, from a methodological point of view, we will propose to help designers to describe domain patterns and to pass from problems to solutions by implementing the metamodel described above. Second, such a methodology will lead us to contribute to existing Systems Engineering methods, with the proposition of a design pattern application functional model, expressed in SysML activity diagrams.
REFERENCES


Brandl, D., 2006. Design Patterns for Flexible Manufacturing, ISA.


Gamma, E. et al., 1994. Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley.


