Non-functional aspects management for craft-oriented design

and its application to embedded systems

Francois Mekerke, Wolfgang Theurer, and Joel Champeau

ENSIETA
2 rue Francois Verny
29806 BREST Cedex 9
{mekerkfr,theurewo,champejo }@ensieta.fr

Abstract. The most important phase in the development of large-scale system is integration. At model level, some techniques allow us to weave models of different parts of the system into one another so as to obtain a complete model. The problem is that these methods don't guarantee that the properties added to the system by a first aspect won't be invalidated by another one.
We present a technique that enables property checking in the framework of the craft-oriented partition that is the norm in industrial development processes.
In order to validate the consistency of a model distributed depending on the specialities of the teams that have to realise it, we suggest to use an abstract view of the system we call the "pivot" through which craft-oriented "facets" communicate. Since it is the only link among facets, the pivot can check if the properties woven with previous aspects are still valid or not. It can therefore help manage aspects' interference.
This technique provides a good compromise between flexibility and rigour, which allows for free development while validating choices all along.

1 Introduction

As technological changes happen always faster and in an always more fragmented manner, systems' complexity tend to explode in all phases of their life cycle.

The difficulty of dealing with this problem lies not so much in the technologies themselves than in the organisation of the teams that master them, whose number and specialisation increase.

The majority of the troubles encountered by the project manager to follow the state of advance of his system are related to the management of the consistency between the different subsystems.

He must be able to form his own abstract image of the system from the partial data sets given by his partners.

We will place ourselves here in the framework of system modelling, which can help manage the complexity, at least by providing an eagle-eye view of the system. However, consistency management into a set of models remains a full-scale problem.

Partitioning approaches, especially in the MDA framework, provide solutions for the management of "Separation of Concerns" (see [1],[2]). Aspect-Oriented Design (see [3], inspired by Aspect-Oriented Programming (see [4]), or the Composition Filters (see [5]) approach, allow us to separately model functionalities on the one hand, and non-functional concerns on the other hand, then to weave the latter into the firsts.

We introduce here the concept of partitioning by "craft-oriented facets", each dealing with specific functionalities, which responds to the industrial problematics that consists in studying a same system under different lights, with different teams. Indeed, the industrial practise is such that contracting authorities partition the system into subsystems depending on the specialities of their subcontractors (internal or external). We present a methodology enabling the simultaneous management of these two partitions, through a "pivot" that controls exchanges and checks that the properties introduced by the different aspects are always validated.

2 Model management

2.1 Context

During the development of a system, numerous models are created, which each corresponds to a concern or the point of view of a stakeholder. These stakeholders work generally in an autonomous manner, independently from one another, each having its own point of view (models, requirements, specifications, hypotheses…) on the system (see [6]).
However, these points of view can be very far from one another, having very local validities. The question is then for the project manager: how to obtain and moreover maintain consistency between the different models of a system, provided by the stakeholders.
Over this first problematics comes the problematics of the management of cross-cutting concerns, for example fault tolerance or real-time performance, which aspect technologies can help to overcome. The project manager must find means to ensure that aspects are actually taken into account in the distributed model of the functionalities.
This leads to the following problem: we may have to weave every given aspect into the models of several subsystems, since they tackle a cross-cutting concern. However, to do so, we need a distributed aspect whose different parts each operate on a single subsystem.
Another solution would be to merge the local models into a complete model of the system in order to weave the aspects, but this option seems unrealistic because the global model of any system, even small, can become huge, therefore difficult to manage.
In addition to this initial problem, we have to ensure that it is possible to check the properties woven into the system. It is thus necessary to define a set of invariants that will show the persistence of certain required qualities.

2.2 Distributed aspects and invariants

Let us consider the case of an avionics company that want to obtain for each of the commands of its air-plane a fault ratio under 10^{-6} fault per hour of flight. We will also consider that there are two teams working on a command, one for software and the other for hardware.
In addition to their proper problematics (description of the command depending on other data for the firsts, dimensioning of the calculators and integration into the network for the latter), the teams must respect this constraint and coordinate a number of their choices.
If the solution chosen to respect the objective is to apply n-redundancy to functions and calculators, we have to ask the firsts to use the design pattern Main-Rescue (see [7] for the concept and a catalogue of patterns) for the functions, the latter to foresee an adequate number of computational resources, without using two functions linked by this pattern on the same processor. This implies that we give the knowledge of the functions' structure to the "hardware" team at some point of the development process.
However, as it seems counter-productive to provide them with the complete (therefore complex) model of the "software" team, we have to generate an intermediate model, which provides them with the pertinent data and nothing more.
The introduction of an aspect into the system will therefore generate (1) a data flow from a team to the other and (2) a set of invariants to respect (here for example FonctionMain.CPU != FonctionRescue.CPU). These latter can be seen as the expression of a long-term "contract" (see [8]), which will have to be respected from there on.

3 A balanced approach

So as to be able to answer the double requirement induced by the couple separation of concerns / craft-oriented partitioning, we have chosen to keep the craft-oriented partitioning structure and to provide the suitable mechanisms to implement the aspects, which implies guaranteeing their invariants over the whole life cycle.
To summarise, the result we obtain, in terms of structure, is the following: each "craft-oriented facet" presents data to a "pivot", which processes them in order to provide data to other facets or check the properties introduced at weaving time.

3.1 Facets

From now on, we will call a ”facet” both the requirements related to a craft and the models developed in their framework, as well as the physical team in charge of the domain.
Example: *In the case of the development of an air-plane, we could find the following facets: hydraulics, electronics, software, aerodynamics ...*
The facets are the heart of the system, since they are the ones that provide the development effort, and finally realise the product. They are each given their own meta-model, which allow them to provide models to the pivot, and also to manage their own development.
These meta-models manipulate two kinds of objects: (1) internal objects: developed inside the facet, all their data is available, and the facet is in charge of providing their pertinent elements to the pivot; (2) external objects: simple abstractions of objects developed in other facets, the knowledge about them comes from the pivot, they are never refined in the facet.
To realise its own part, a facet is free to use the modelling technology of its choice (if any), but one of its models has to be public. This model, that we call “Public Model”, is the interface of the facet with the pivot. The Public Model is an instantiation of a subset of the facet’s meta-model that has a double functionality: (1) it allows us to define the functional and structural specifications of the facet, and (2) it allows the facet to present its principal characteristics to the pivot whenever necessary. The meta-model of the Public Model is chosen in collaboration with the project manager. This is the only way for him to specify the data he wants to receive and add the necessary consistence between the facets. In the framework of software-intensive systems development, several facets will deal with different software parts. In this case we can consider that each facet will develop a component whose ports’ information will present in the Public Model. The Public Modelof the facet will contain (without being restricted to) the interface of the component.

3.2 Pivot

The pivot is the entity in charge of maintaining the consistency between the facets. On the one hand , some facets’ outputs have to be rewritten to be given as inputs to others, in order to allow these to coordinate each other (see 2.2). On the other hand, the pivot checks that the invariants introduced by the aspects are not violated.
Therefore it contains two kinds of information: (1) those connected to data management (source, processings, destination) and (2) those related to properties checking. The first one implies model transformations techniques, whereas the second requires skills in algebraic computation or formal methods, depending on the situation.
Invariants management is quite a hard task because of the numerous different formalisms we can use, depending on what has to be checked. Our solution to fix the problem is to introduce a set of “layers”, each one containing the information related to a particular aspect, in terms of data flow as well as in terms of invariants. The pivot is thus composed of those layers, which are pushed each time an aspect is added, and poped later in the development process, as soon as they are considered useless.
The pivot meta-model has to lie at the intersection of the facets’ ones in order to make it easier to map the elements of the pivot on those belonging to the facets.
As shown fig 1, the pivot synthesises the relationships among Public Models. It thus enables the uniform usage of heterogeneous data without having to change anything in the Public Models. The most noteworthy property of this construction is that the pivot is the unique link between the Public Models.

3.3 Layers

The layers are the cement that holds the pivot and the facets together. They are the only entities to have the knowledge of both the Public Models and the pivot. Through the abstract representation of their information, the distributed architecture of the system is transparent for them. They can simply work on some elements of the models as if they were parts of a unique homogeneous model.
A layer has the knowledge of all the elements it has to monitor. It computes either the value of abstract pivot attributes from attributes of facets’ internal objects, or the values of facets’ external objects from attributes in the pivot. Thanks to this symmetric organisation, two facets never have to (and must not) exchange information directly.
Once pivot’s attributes are computed, constraints can be checked on them.
Figure 2 shows an example of how a layer can be defined using the UML syntax. We define two main types of constraints: (1) local constraints that guarantee the consistency between facet elements and

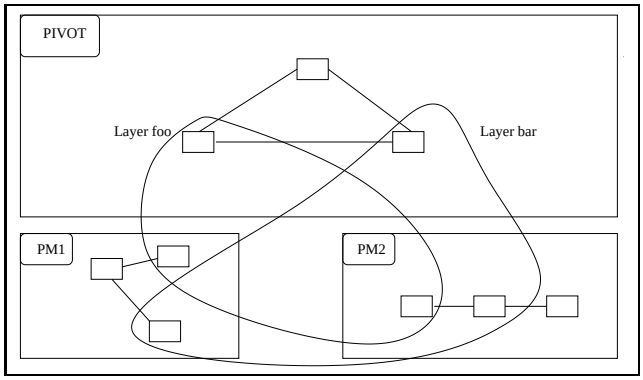

Fig. 1. Pivot general organisation

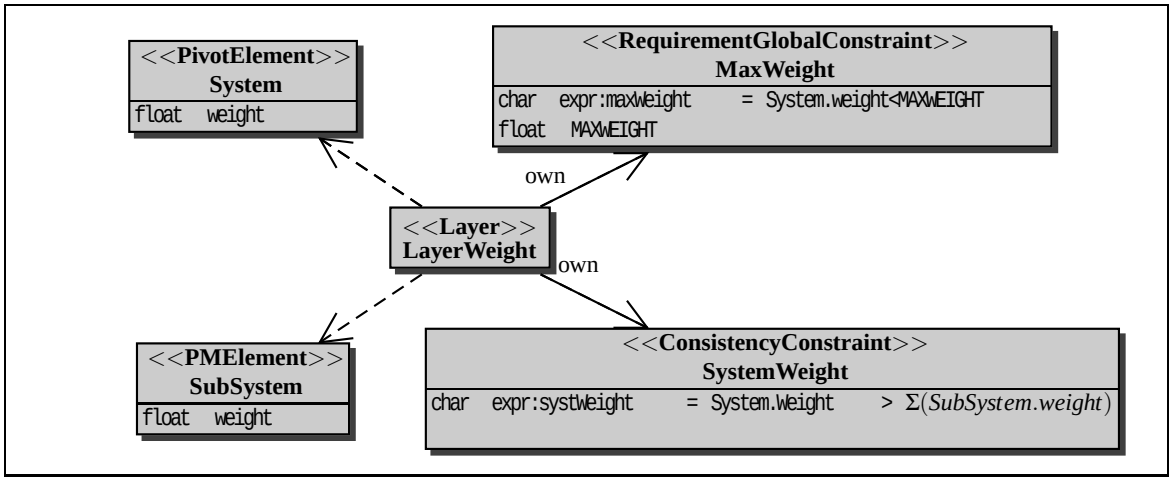

Fig. 2. Layer: example

their representation in the pivot, they also provide a way to compute the values in the pivot from those of the facets, and (2) global constraints that express global system requirements.
With this organisation, we emulate the weaving of distributed aspects, and we add the continuous checking of the sets of constraints that characterise them.
When developing collaborating components, layers will be dedicated to checking that the inputs' and outputs' formats of the components are compatible.

4 Development process

4.1 Two phases

The process induced by such a structure for managing models and their relationship can be split in two main stages: first, a phase of negotiation and then a phase we called "stable models' phase". This behaviour is close to the industrial usage, where the choice of the partners then the technological choices and, finally the tasks' allocation are negotiated.

- The first stage includes drafting a consensus on the choice of the meta-models to be used by the different stake-holders to communicate (those of the Public Models), then on the determination of the pivot meta-model. Follows a phase where the pivot own structure is defined, through analyse of the preliminary solution elements provided by the facets.
- The second stage consists in an enrichment of this structure, *i.e.* an improvement of the pivot attributes precision and a convergence towards the solution model. Whenever we have to add a new facet at this stage (which means we have forgotten it at the requirement analysis stage) we have to go back in negotiation phase. We then may have to first rebuild the pivot adding the new facet object, and add force the existent facets to change their Public Modelin order to provide the information relevant to the new one.

4.2 Two applications

Consistency The main usage of the pivot is to dynamically check the consistency of the data provided by the different facets through their Public Models. Each time a value is modified in a Public Model, all the layers related to this element are evaluated again. Whenever a constraint is violated by the new value, it is rejected, and the faulty facet is notified as well as the project manager.
Example: *If we consider two facets "software" and "hardware", the pivot can check that the CPU power provided by the "hardware" team will allow the processings defined in the "software" team to respect their real-time requirements.*
More generally, it can easily detect inconsistencies between quality of service requested and quality of service provided.
Basically, in this case, the pivot can be seen as a checker which verifies that the data stored in the Public Models satisfy the constraints' system composed of the layers.

Dimensioning As a practical application, the pivot can also be used the other way round, as a methodological tool for system dimensioning. In this case, we don't just check that a set of values satisfies the system: given values for a subset of the system's variables, we solve the system in order to obtain a set of solutions for the rest of them.
Actually, by implementing processings in the layers other than those imposed by the aspects, it is possible to add a reflexive flavour to the pivot, which could then provide elements of solution by itself. For example, by first determining the CPU power needed by a set of functions to respect their real-time specifications, the pivot could determine the appropriate set of on-the-shelf CPUs able to execute them properly.

5 Real-time embedded software

Although our approach aims at being as general as possible, and applicable to all sorts of systems, thus to an undefined number of facets of different kinds, we mainly focused on software-intensive embedded systems composed of a "software" facet and a "hardware" facet. Our choice of these two particular facets can be argued as follow:

First, most embedded (software intensive) systems, are built upon a “software+hardware” core (sensor/control/actuator chain), closed enough to be studied as a completely independent system.
It seems also interesting to show how the hardware platform can be integrated in software development.
Finally, the real-time performance aspect is a cross-cutting concern for the two facets.

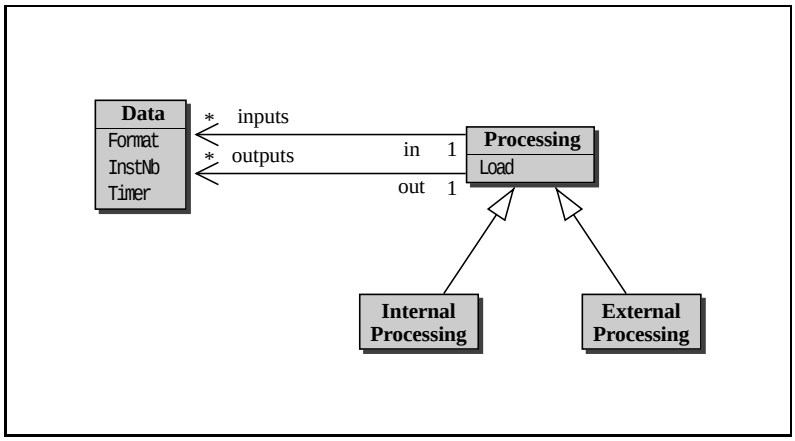

Fig. 3. “Software” facet concept model

The “software” facet is in charge of developing the whole high level software (drivers, schedulers and low level protocols are considered to be part of the hardware). Its meta-model, shown figure 3, is composed of processings, which exchange data. Each processing is viewed as a sequential unit by the rest of the system. That sets the Public Model granularity, fixed from the first stage till the end of the development.

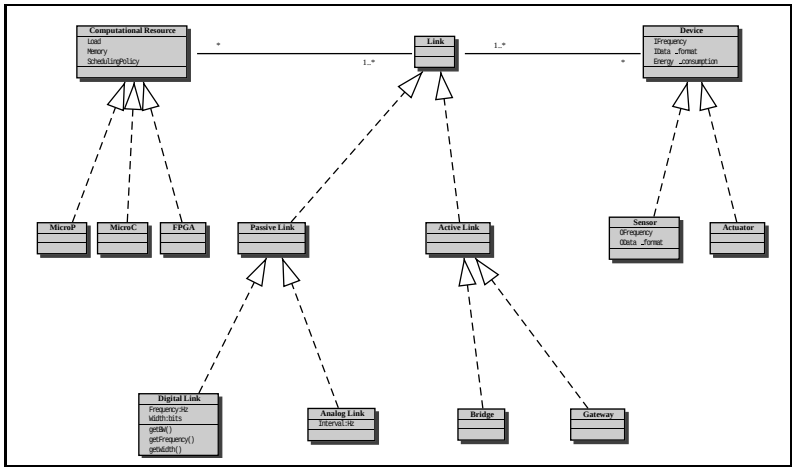

Fig. 4. “Hardware” facet concept model

The “hardware” facet describes the platform which runs the processings. The meta-model of this facet, shown 4, includes computational resources which communicate with each other or with devices via communication links. The pivot is composed of functions which exchange data on communication links.

The pivot elements describe the same entities than the facets ones, but from a different point of view.

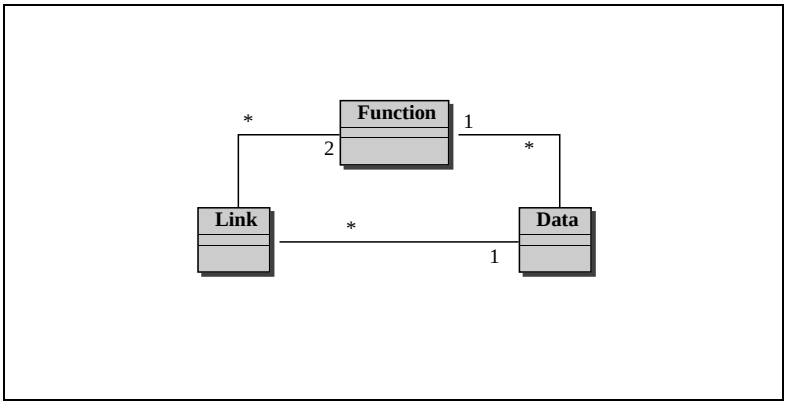

Fig. 5. Pivot concept model

5.1 Formal property validation

One of the major problems that arises in the scope of distributed modelling, is the real-time behavioural validation of the system. Actually, despite the existence of numerous formal techniques (model-checking, formal test, etc see [9]) for modelling and validating real-time softwares, evaluating the impact of the hardware platform on software is very laborious. Taking this impact into account needs a wide human intervention (because of undecidability problems see [10], [11]). These troubles are generally bypassed by introducing timing information into the models of process (like timed automata see [12]), *i.e.* building a unique model containing simultaneously behavioural (functional) information and executive information. Even if these methods have already proved themselves, they don't get along with our wish to separate models by crafts. We thus propose creating a validation layer which builds on the fly a set of communicating timed automata from information provided by the "hardware" and the "software" facets. The previously mentioned techniques can be applied on this set. This layer is based on an abstract syntax associated with an operational semantics and a translation algorithm to a chosen target formalism (UPPAAL's timed automata, for instance). The Public Model of the "software" facet has to provide its processings' behavioural models, the "hardware" facet as for it, has to give the pivot the values related to its computational resources and links (power, bandwidth ...).

The abstract syntax related to this layer is defined as the parallel composition of functions sets:

$S := \mathcal{F}$

$\mathcal{F} := \mathcal{F} \parallel_F F \mid F$

$F := < I, O, gs, ge, gd, \Delta, A, T >$

A function is a tuple $< I, O, gs, ge, gd, \Delta, A, T >$ where:

- I: the set of inputs
- O: the set of outputs
- $gs : O \rightarrow 2^{Q^+}$ (set of parts of Q^+, set of positives rational numbers) mapping that associates to any given data from the set of outputs, an interval for its emission delay (expressed in cycles from the effective launch of the function).
- $ge : I \rightarrow 2^{Q^+}$ mapping that associates to any given data from the set of inputs, an interval for its reception delay (expressed in cycles from the effective launch of the function).
- $gd \in 2^{Q^+}$: interval for the function launch delay.
- $\Delta \subseteq 2^{Q^+}$ interval for the function clock offset δ, set at system launch .
- A: untimed automata describing formally the function behaviour.
- T: the function activation period.

All delays are computed in the pivot from the quantitative information provided by the "software" facet (code size, number of instructions needed to produce a data, etc.) and the qualitative information extracted from the "hardware" facet (power, rate, etc.). The communication delays are included in the functions' input/output delays.

The operational semantics associated to this syntax, is defined as a timed labelled transitions system (TLTS) expressed by rules of the following form:

$$\frac{\text{premise}}{\text{conclusion}}$$

where premise is a conjunction of time assertions and assertions specifying the state of the automaton of the considered function.
The operational semantics, as well as the translation algorithm are not provided is this article, for readability (and size) reasons.
The set of invariants associated to this layer are formal real-time behavioral properties (deadlock-freeness, safety, liveness, etc) expressed in a temporal timed logic (such as TCTL). The set of communicating timed automata built by the translation algorithm (a process we may implement soon) has to satisfy all invariants. To check this, we will need an external model-checker for the specified formalism: we consider using UPPAAL (see [13]) for this purpose.

6 Conclusion and prospects

We have expounded a structure for managing the developments of systems involving craft-distributed models. We also explained how to make sure that cross-cutting concerns are taken into account, through a set of layers. These layers contain information on the identity of the elements they have to deal with, and the set of constraints to apply on these elements. Thus, this method allows us to maintain consistency over the different facets, through guaranteeing the validity of cross-cutting properties all along a system's life cycle.
We have already applied this method to a simple dimentioning application with some success. We now study a bigger application: we work on the modelling of an autonomous underwater "glider" developed at ENSIETA[1], we defined three facets, "hardware", "software" and "control", and a number of simple layers (QoS, mass, volume, ...).
Through this experience, our goal is to develop, in the near future, tool support for this specific application.

References

1. Hürsch, W.L., Lopes, C.V.: Separation of concerns. Technical Report NU-CCS-95-03, College of Computer Science, Northeastern University, Boston, MA (1995)
2. Ossher, H., Tarr, P.: Multi-dimensional separation of concerns and the hyperspace approach. In: Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development, Kluwer (2000)
3. Suzuki, J., Yamamoto, Y.: Extending uml with aspects: Aspect support in the design phase. In: Proceedings of the third ECOOP Aspect-Oriented Programming Workshop. (1999)
4. Kiczales, G., Lamping, J., Menhdhekar, A., Maeda, C., Lopes, C., Loingtier, J.M., Irwin, J.: Aspect-oriented programming. In Akşit, M., Matsuoka, S., eds.: Proceedings European Conference on Object-Oriented Programming. Volume 1241. Springer-Verlag, Berlin, Heidelberg, and New York (1997) 220–242
5. Aksit, M., Tekinerdogan, B.: Solving the modeling problems of object-oriented languages by composing multiple aspects using composition filters (1998)
6. Hilliard, R.: Using the UML for Architectural Description. In France, R., Rumpe, B., eds.: UML'99 - The Unified Modeling Language. Beyond the Standard. Second International Conference, Fort Collins, CO, USA, October 28-30. 1999, Proceedings. Volume 1723 of LNCS., Springer (1999) 32–48
7. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1994) ISBN: 0-201633-61-2.
8. Meyer, B.: Applying design by contracts. IEEE Computer **25** (1992) 40–51
9. Laroussini, F.: Automates temporisés et hybrides. In: École d'Été Temps Réel (ETR2003), Toulouse (2003) 155–166
10. J. Ermont, F.B.: La vérification des systèmes temps réel soumis la préemption de processus est indécidable. In: MSR2003 (Modélisation des Systèmes Réactifs). (2003)
11. Ermont, J.: Une algèbre de processus pour la modélisation et la v érification des systèmes temps-réel avec préemption. PhD thesis, ENSAE (2002)
12. Alur, R., Dill, D.L.: A theory of timed automata. Theoretical Computer Science **126** (1994) 183–235
13. Larsen, K., Pettersson, P.: Timed and Hybrid Systems in UPPAAL2k, ("http://www.cs.auc.dk/ paupet/talks/MOVEP2k.html/")

[1] http://www.ensieta.fr/dtn/automatique/glider/gliders.htm