An Object-Oriented Approach to Handle Complex Real-Time Industrial Automation Projects

Pablo Darscht
darscht@ias.uni-stuttgart.de,
Institute for Automation and Software Engineering
Stuttgart, Germany

Carlos Eduardo Pereira
cpereira@iee.ufrgs.br
Universidade Federal do Rio Grande do Sul (DELET-UFRGS)
Porto Alegre, Brazil

Abstract

This paper describes an object-oriented approach to handle complex industrial automation systems. The proposed approach relies on relationships between modeling perspectives and integrates different models of existing object-oriented methodologies. So, it supports the multi-disciplinary character of industrial automation projects. A framework based on the concept of viewpoints is also presented.

1 Introduction

Computer-based industrial automation systems (CBIAS) demand an increasing amount of embedded real-time software and hardware. They frequently include distributed, heterogeneous networks, and are driven by requirements on performance, security, fault-tolerance, and real-time behavior, among others. The satisfaction of these conflicting requirements increases the complexity of CBIAS development, since it requires managing tradeoffs among them. In addition CBIAS development process involves several people with different backgrounds and expertise, such as process and control engineers, software engineers and problem domain experts, among others. Each one of these groups has a different perspective of the system under development and usually adopts distinct modelling techniques to describe/analyse the system. Consider for example the automation of a chemical plant: It must be decided how to control the plant (process engineering, control engineering), which hardware should be used to implement the control strategies (hardware engineering), the software must be developed (software engineering), and so on. Each project participant has its own perception of the system under development: process engineers will see the plant in the form of a process flow diagram, control engineers will work on block diagrams representing control loops, hardware engineers will probably prefer to highlight electrical interfaces of actuators and sensors, and software engineers would like to have an abstract model of the plant. Managers will see the problem in terms of resources needed, Gantt diagrams, project matrixes, etc.

All these persons have to work harmonically to build up cost-effective and high quality industrial automation systems. Taking also into account the very well known limitation that humans have when dealing with information [1], concepts like decomposition, hierarchy, encapsulation, and modularization, are essential to manage the complexity of large systems.

Some development methods have been published, proposing graphical notations and guidelines to help developers in the task of managing the complexity of CBIAS development. Among the techniques that have been published , those based on the object-oriented (OO) paradigm have gained special interest in the academia and industry. One of the main advantages achieved by OO methods is to organize all the information contained in the different perspectives around objects that exist in a real-world system, allowing a semantic mapping of problem domain concepts and enhancing the readbility and understandability of the different perspectives. Experience also has shown that there are several benefits which can be achieved when applying object oriented techniques to the development of CBIAS [2].

This paper describes an object-oriented approach to handle complex industrial automation applications. The approach attempts to overcome some of the problems encountered when trying to scale up object-oriented methods to capture complex real-time systems, mainly: (1) the lacking of mechanisms to describe timing aspects and check consistency; (2) the weak connection between different partial views or perspectives that come up when developing complex applications. Rather than introducing yet another object-oriented method, the approach intends to provide a formalism to integrate useful development techniques, such as those present in several existing object-oriented methods.

2 Object-oriented methods: state of the art

A large number of OO methods have been published over the last few years. In 1994 Stein listed 43 (!) different methods [3]. However, even OO-methodologists recognize that differences between several leading methods are minimal [4] and many organizations have found that most of them tend to be more complementary than competitive.

Traditionaly, object-oriented methods attemp to manage complexity by focusing the system under consideration from different perspectives. Typically, structural, behavioral, and functional views are defined using entity-relationship-like diagrams, state-transition and data flow diagrams respectively.

To manage the complexity of these descriptions as they become large, some methodologists propose the grouping of 'low-level' components (objects/classes or relationships) into high-level ones, such as the object-relationship model of the OOSA-approach [5] and in the Booch method [6]. This 'vertical decomposition' does enhance the description's understandability. However, a strategy to manage complexity is needed, since abuse of hierarchical decomposition can lead to descriptions much more unreadable and hard to maintain than flat ones.

As an alternative to deal with several objects and relationships in a static view, one might use 'horizontal abstraction': the different kinds of relationships (is-part-of, is-a, associations, etc.) and objects/classes are 'filtered out' in order to reduce the amount of elements being shown and to focus on different aspects of the system. This idea is similar to that proposed in the OMT method where modules and sheets are used as logical constructs for grouping classes and relationships. Another approach consists in first breaking the system under consideration down into subsystems, even before starting to identify objects and relationships.

Information contained in different perspectives of a system may intersect and overlap. That implies a requirement for coordination and consistency maintenance between them (inter-views checking). Unfortunately, such aspects are not properly taken into account by existing methods.

3 Formalizing the concept of multiple perspectives

3.1 ViewPoints

The computational appearance of a modeling perspective is what Finkelstein et al. call *a ViewPoint.* A ViewPoint (VP) is defined as *„a loosely-coupled, locally managed object encapsulating representation knowledge, development process knowledge and partial specification knowledge about a system and its domain"* [7]. ViewPoints can be classified according to the representation knowledge and development process knowledge they contain. A ViewPoint type is characterized by a set of *notation elements* (NE). They define the bricks that the application developer uses to build a ViewPoint. For instance, if he is constructing an Information Model, he can use objects, relationships, associative objects, subtyping, and the like. In order to obtain an architecture able to support these ideas, two *metaclasses* have been defined: the ViewPoint-templates (VP-templates) and the notation element templates (NE-templates).

3.2 Structural relationships among ViewPoints

VP-templates and NE-templates are used to describe the internal structure of a ViewPoint. However, a development method does not consist of the aggregation of a set of independent modeling perspectives, but it defines also relationships among them.

A (small) set of structural relationships between ViewPoints was defined, providing each of them with a strong semantic. Each type of relationship established between ViewPoints determines the kind of interactions that are allowed between the parts involved in it. The structural relationships that have been defined are: *superposition, derivation and context-part.*

The concept of ViewPoints **superposition** allows the idea of *horizontal abstraction* presented before to be formalized. Each time a ViewPoint shows more than one dimension of the model, it may be convenient to consider it as composed by many 'atomic layers'. The *superposition* formalizes this concept. A Superposition is a relationship between a *basic ViewPoint* and a set of *overheads*, that results in a new composite VP.

The *basic* ViewPoint is a normal one, but an *overhead* only makes sense if it is considered together with the basic ViewPoint. Figure 2 shows an information model as a composite VP consisting of a basis VP and two overheads: attributes and temporal annotations.

One VP 'B' is said to **derive** from a set of VP 'A', if a significant portion of the information contained in 'B' can be deduced from the information contained in 'A'. Consider the behavior description: Shlaer and Mellor [8] use state models and event lists: for each object having a state model, an event list giving all events it expects can be automatically generated (or *derived*). Another interesting variant of derivation are those ViewPoints that summarize the information contained in more than one original ViewPoint. The timed graphs presented in next seccion exemplify this relationship.

Beside this 'complete derivation' -that could be seen as the generation of a *redundant* ViewPoint-, there are also cases of partial derivation: consider for example the *scenarios* used in OMT [9]: much of the information contained there will be used later in the behavior model, and thus an initial scaffolding for it can be derived from the scenarios. However, the state diagrams used to specify this modeling perspective also contain new information. Code generators build another example of derivation.

Context-part relationship: Figure 1 shows an information model containing two objects and their associated state models. The information model is the *context* in which the state models make sense. This relationship is not the usual refinement, because the 'inferior' ViewPoints (in the example, the state models) contain a different kind of information than the 'superior' ViewPoint. The binding element between both ViewPoints is the object whose behavior is described by the state model. We call *porthole* the notation element of the context-ViewPoint that plays this role in the relationship. In Figure 1, each instance of the notation element *object* in an information model can be the porthole to a state model

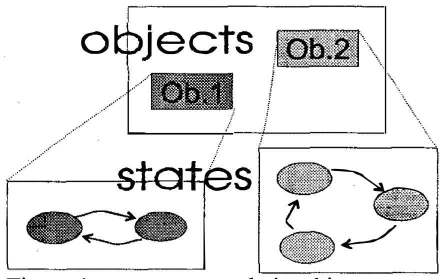

Figure 1: context-part relationship

describing the behavior of the object.

4 Multiple perspectives of temporal aspects

Among the different perspectives proposed by exisitng OO methods, the behavioral view is specially important, since timeliness is a core concept in the development of industrial automation systems. The behavior of the real-time part ought to be predictable, since some timing requirements must be met under any circumstances, in order to avoid severe damages or loss of human lives. Because of this, the description and consistency checking of timing requirements plays a key role in the requirements of industrial automation systems. Unfortunately, exisiting OO methods lack notations for describing real-time aspects, specially quantitative timing requirements, such as deadlines, time-out, etc. Attention must also be paid to avoid temporal inconsistency inside and between views. For instance, the temporal relationships contained in a process model diagram must be consistent with those specified in the state models.

An object-oriented framework for the specification and analysis of timing information at the requirements engineering phase of the (real-time) industrial automation systems development was developed and is described in details elsewhere [10, 11]. Basically, the proposed framework enhances state transition diagrams with temporal predicates written in an event-based language. In order to check consistency, temporal information present in the different diagrams has to be extracted and translated into a common formalism. For that purpose, timed graphs were introduced. Nodes in a timed graph correspond to events (i.e. points in time), edges are related to temporal relations among events, and the weights denote quantitative time intervals among events. The description is validated by means of a temporal reasoning process through an inference mechanism that takes into account the relationships among the classes and objects defined.

Intersections between different views are far from being obvious because the knowledge within each perspective is represented in different ways [7]. By means of mapping the temporal knowledge from the different views proposed by different OOA methods into timed graphs, an unified temporal view can be achieved, within which a validation of the temporal specification can be applied. This 'temporal view' can be considered as a projection of all the temporal information contained within all the other views (see Fig. 10). As depicted in Fig. 11, the timed graphs may be seen as a new ViewPoint which can be derived from other views. It is important to note that instead of considering the timed graph as a flat structure without hierarchy, abstraction mechanisms provided by object-oriented techniques are taken into account, such as components hierarchies (is-part-of relationships), classification, views,

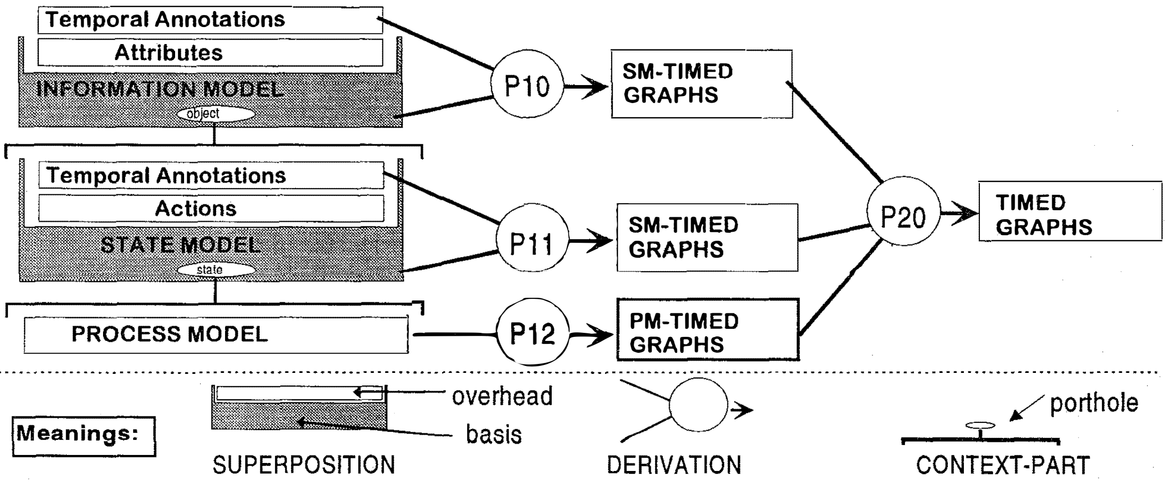

Figure 2: Integrating timed graphs to the Shlaer-Mellor method: timed graphs are obtained as derivation from the other ViewPoints that contain timing information

and objects, as a means to achieve better information hiding. This also contributes to minimizing the complexity of the inference process applied to checking the consistency of the specification as discussed in [11]. Experience shows that, during the development of very large systems, consisting of lots of objects with several relationships among them, such temporal contradictions, even the more trivial ones, are likely to be overlooked.

Conclusions and future work

The approach has been successfully applied to the development of some real-time applications, such as a package router system ([2]), a modular production system ([12]), among others. It allowed the detection of several inconsistencies that had not been discovered by existing tool support. Moreover, since the approach takes into account the multi-disciplinarity and the resulting multiple perspectives that each project member may have from the whole industrial system, it promotes the necessary team work. Abstraction mechanisms, such as hierarchy and modularity, have shown to be an essential part of the framework developed.

The benefits achieved with the use of object-oriented concepts are quite encouraging. On the one hand, the mapping of problem domain objects to the different diagrams have enhanced significantly the communication between different expert groups. On the other hand, the introduction of ViewPoint classes and instances have eased the consistency checking between diagrams.

Within the context of a cooperation project with the company MarkV Systems from the USA, a prototype version of a CASE-tool supporting the ideas presented is currently being implemented using the Tool Development Kit of the METACASE-tool ObjectMaker.

Acknowledgments

The authors would like to acknowledge the financial support provided by the German Service for Academic Exchange (DAAD, grant 322/512/251/2) and by the Research Council of Brazil (CNPq) under contract number 204495/89.0

References

[1] Miller, G.: „The Magical Number Seven, Plus or MinusTwo: Some Limits in our Capacity for Processing Information“. *The Psychological Review*, vol. 63(2), March 1956

[2] Pereira, C. and Darscht, P.: „Using Object-Orientation in Real-Time Applications: An Experience Report or (1 Application + 5 Approaches = 1 Comparison)“. *Proceedings of TOOLS EUROPE '94* Versailles, France, Ed. by B. Magnusson, B. Meyer, J. Nerson and J. Perrot, Prentice Hall, Englewood Cliffs

[3] Stein, W.: *„Objektorientierte Analysemethoden: Vergleich, Bewertung und Auswahl“*. BI Wissenschaftsverlag, Angewandte Informatik; Bd. 12, Mannheim, Leipzig. 1994 (in German)

[4] Jacobson, I.: „Time for a cease-fire in the methods war“. *Journal of Object-Oriented Programming*, Guest Editorial, Vol. 6, No. 4, July/August 1996.

[5] Embley, D., Kurtz, B. and Woodfield, S.: *"Object-Oriented Systems Analysis. A Model-Driven Approach"*. Yourdon Press, Prentice Hall, Englewood Cliffs, 1992

[6] Booch, G.: *„Object Oriented Analysis and Design with Applications“*. Second Edition, Benjamin/Cummings, Redwood City, 1994

[7] Finkelstein, A., Kramer, J., Nuseibeh, B. Finkelstein, L. and Goedicke, M.: "ViewPoints: a Framework for Integrating Multiple Perspectives in System Development“. *International Journal of Software Engineering and Knowledge Engineering*, Vol.2, No.1 (1992) 31-57, World Scientific Publishing Company

[8] Shlaer, S. and Mellor, S.: *„Object Life Cycles: Modeling the World in States“*, Yourdon Press, Englewood Cliffs, NJ, 1992.

[9] Rumbaugh, J. et all, *„Object-Oriented Modelling and Design“*, Englewood Cliffs, NJ, Prentice-Hall 1991.

[10] Pereira, C.:"On Describing Timing Requirements within a Real-Time Object-Oriented Requirements Engineering Phase“. *Workshop on Object-Oriented Real-Time Systems Analysis and Design Issues, OOPSLA '93*, Washington, USA, 1993

[11] Pereira, C.: „Applying Object-Oriented Concepts to the Description and Consistency Checking of Temporal Aspects of Real-Time Industrial Automation and Process Control Systems“. To appear in *6th IFAC Symposium on Information Control Problems in Manufacturing, INCOM 95*, Beijing, China, ocotber 1995.

[12] Darscht, P., Frigeri, A. and Pereira, C.: „Building up Object-Oriented Industrial Automation Systems: Experiences interfacing Active Objects with Technical Plants“. To appear in *IEEE International Workshop on Factory Communication Systems (WFCS'95)*, Switzerland, october 1995