Mechanisms to Manage Complexity during Object-Oriented Analysis of Large Industrial Automation Systems

Carlos E. Pereira[#]

Univ. Federal do Rio Grande do Sul - UFRGS
Department of Electrical Engineering
Porto Alegre - RS - Brazil
E-mail: cpereira@iee.ufrgs.br

Wolfgang Halang

University of Hagen
Faculty of Electrical Engineering
Hagen - Germany
E-mail:wolfgang.halang@fernuni-hagen.de

Abstract

This paper discusses some topics related to the use of views as an abstraction concept in the object-oriented analysis of large, complex industrial automation systems. The idea behind the use of views is to manage the complexity of the system being automated. They must allow all the people involved in the analysis process - customers, users, problem domain experts, and analysts - to have a better understanding about the problems to be solved by enhancing the specification's clarity. Based on the experience we have gained by applying current object-oriented analysis to the development of industrial automation systems, we can affirm that the proper use of abstraction mechanisms is a core concept that plays a very important role in the development process.

[#] This work has been developed at the Institute for Industrial Automation and Software Engineering (IAS), University of Stuttgart - Germany. It has been partly supported by the Federal Research Council of Brazil (CNPq) under contract number 204495/89.0

1. Introduction and Motivation

Industrial automation systems development is characterized by the necessity to integrate hardware and software systems. As both of these systems grow in complexity and size, the task of system analysis becomes increasingly difficult and time consuming. Power plants, chemical processes, and manufacturing systems are examples of complex, large scale industrial automation systems. The challenge faced by systems analysts is to properly capture, organize, maintain, and understand the large volume of information involved. As stated in [1], analysts tend to use one of two approaches to gather information and document systems: method-driven or model-driven. A method-driven approach consists of a fixed sequence of steps to follow. Unfortunately, a fixed sequence of steps can not allways be followed exactly. So, when problems are founded, analysts' experience is required to adjust steps and to make exceptions to complete the task. The model-driven approach to analysis concentrates on building a model of the system under study. Our experience developing real-time industrial automation systems has shown that for these kind of systems the model-driven approach is more advantageous and effective. Due to the strong interaction of real-time systems with their environment (the technical plant being controlled dictates the behavior of the computer control system), spending time building models and carefully investigating the system under control has proven to be a very important step towards the development of a real-time system that really meets the customer requirements.

Among the different paradigms being adopted by model-driven approaches, the use of object-oriented techniques is receiving an increasing attention. Object-oriented techniques allow a direct mapping of the semantics of the problem domain into the model being generated, leading to an enhancement in the description's readability and understandability. Nevertheless, when trying to put current object-oriented methods 'to work', one can become deeply frustrated, since the great majority of them still lack concepts to tackle large real-time systems.

The opinions stated in this paper have been matured during a research project which was developed at the University of Stuttgart. The project investigated the advantages that could be obtained with the application of object-oriented concepts throughout all development phases of

methods to the development of automation and process control systems were undertaken. We also have compared results obtained with these existing object-oriented methods with non object-oriented ones [9]. The main goal of these comparisons was to achieve a deeper understanding of existing methods and of the concepts underlined, allowing us to determine their strengths and weaknesses. Such case-studies have shown us that the state-of-the-art current methods and their CASE-support have not yet achieved the necessary maturity level in order to be suitable for very complex systems (probably due to the fact that the concepts are not well formalized). Concerning the usage of abstraction mechanisms, techniques proposed in some methods are complementary to those from others, leading to the belief that a combination of them might be interesting (in these aspect we agree with some OO-methodologists, such as Jacobson [10], who claims for a "cease-fire").

2. Complexity and Views

The limitations of the human capacity for dealing with complexity is well-known (see 'the magical number seven plus or minus two' as defined by Miller in [3]), but up to now no 'cook-recipe' has been provided in order to successfully develop complex systems. However, there is a consensus that by applying concepts like decomposition, abstraction, hierarchy, encapsulation, and modularization, the complexity of large systems can be managed and products with high quality can be obtained [11-16].

A common approach of existing software engineering methods to tackle complexity is to consider the system being automated from different points of view. Each view emphasizes some aspects of the system and neglects others. Regarding these multiple perspectives of a target system, some similarities among current object-oriented analysis techniques can be observed. For instance, the division of the system being modelled into three dimensions - static, behavioral, and functional - has been adopted by some current object-oriented methods, like the OMT-method [4], the OOA-Shlaer/Mellor [2], and others (see [17,14]). Other methods, viz. the OOSA method [1], and the Colbert's method [5], make use of the two first views - object-model and object-behavior , and additionally provide an object-interaction view, where the communication among objects is

described. Shlaer/Mellor have also proposed two models to handle interactions, one for asynchronous communication (object communication model), and another one for synchronous communication (object access model). Objects/classes are key concepts in linking such views. Each object/class represented in the system's static view might have its life cycle described in the behavioral model. Similarly, the functions to be performed by an object/class as well as the information being transformed by them are depicted in the functional model.

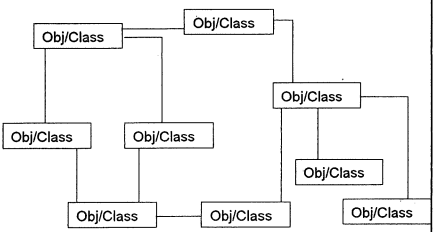
This separation of concerns has been shown to be very useful when developing industrial automation systems. Differents views are described in different models, each one having its own graphical notation and rules suited to the underlined semantics of the aspects under consideration.

Despite the similarities above mentioned, major variations can be found among existing object-oriented methods concerning the application of abstraction mechanisms within the context of these views. As an example, let's consider the static-view as defined in the OMT-Rumbaugh, in the OOA-Shlaer/Mellor, and in the OOSA object-oriented analysis methods. All define the object-model view similarly; the static structure of a real-world system is outlined through the identification of classes, objects, and their relationships to each other. Both make use of a modelling technique similar to the entity-relationship-diagram as proposed by Chen [6], where a diagram depicts all the objects/classes and their relationships. In the OMT method the static view is called object model, the Shlaer/Mellor method prefers the term information model, and the OOSA denotes this view as object-relationship model. Variations among them can be observed in the graphical notation and in some concepts. For the sake of restricting the length of this position paper, we will focus only on aspects concerning the usage of abstractions mechanisms.

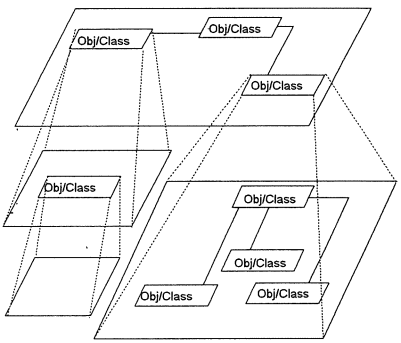
One problem we have encountered when applying the static view to model large systems is that the number of objects and relationships that can be identified in a real system tends to be very large. Methods that do not allow a hierarchical decomposition inside of the view, such as the Shlaer/Mellor-method, lead to a description that is both difficult to understand and to maintain. Therefore, further abstraction mechanisms are needed when handling complex systems in order to have a cardinality of elements at least 'compatible' with Miller's magical number seven.

To manage the complexity of these descriptions as they become large, an interesting solution is the grouping of 'low-level' components (objects/classes or relationships) into high-level

ones, as proposed in the object-relationship model of the OSA-approach and in the Booch method [7][1]. We shall call this kind of decomposition 'vertical abstraction' since one can identify both a top, some intermediate levels, and one or several bottoms in the description (see fig. 1a and 1b). The use of decomposition does enhance the description's understandability. However, it is not the only solution. As an alternative to deal with several objects and relationships in a static view, one might use what we call a '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. The figures 1c and 1d depict these different kinds of use of abstraction concepts only concerning the static view of a system.

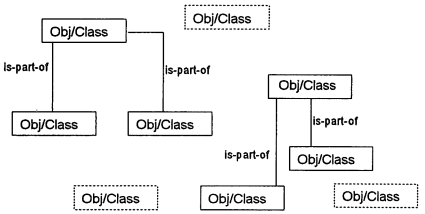


'flat' object-oriented static view
Fig. 1a

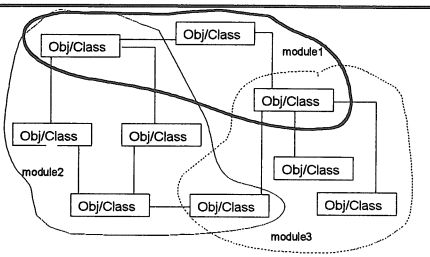


object decomposition - 'vertical abstraction'
Fig. 1b

[1] Even though the Booch-method does not explicitily mention a static-view, the class- and instance-diagrams have a similar meaning, and they include a object decomposition as a natural way of clustering a complex system.



filtering out the *is-part-of* relationships
Fig. 1c



grouping classes and relationships into modules
Fig. 1d

The example above has only considered the use of abstraction concepts within the context of the static model. Nevertheless, similar mechanisms can also be applied to the other views. For instance, instead of adopting only planar state-transition diagrams to describe the desired system's behavior, some methods rather use hierarchical approaches like ObjectCharts[8], and the OOSA-method[1]. As mentioned before, objects/classes are considered key concepts when applying these views, and may be used as a link from one view to another, for instance, through a CASE-tool support.

Another alternative adopted by some object-oriented methods for dealing with a system with high complexity consists in first breaking it down into subsystems, even before starting to identify objects and relationships. The idea is quite similar to one of the key concepts of the structured analysis, the functional decomposition. Because of this, it has been seen with some criticism by the object community.

For instance, Shlaer and Mellor propose the division of a large system under analysis into different domains, each one being related to a different subject matter. A domain is defined in [2] as 'a *separate* real, hypothetical, or abstract world inhabited by a *distinct* set of objects that behave according to rules and policies characteristic of the domain'. Examples of such domains are application domains, architectural domains and implementation domains. Large domains may also be partitioned into subsystems, where the latter are obtained by identifying 'clusters' of objects, i.e. group of objects interconnected with one another by several relationships. The approach suggests first to identify the domains needed for a system before beginning with the construction of the

other models (static, behavioral, functional and interaction), i.e. before of identifying the objects/classes that belong to the system ([2] page 140).

One of the major lessons learned during the development of (real-time) industrial process control systems is that the use of object-oriented techniques really means an improvement in the development process of these systems. The main advantage is obtained from the organization of all the information around objects/classes. The direct mapping of relevant entities in the problem domain of industrial automation systems (like tanks, pipes, valves, sensors, etc.) to elements in the model being built not only enhances the readability and understandability of the specification, but also contributes to a better maintenance of the system.

Undoubtedly, the incorporation of some ideas coming from functional- and data-oriented modelling techniques on an object-oriented process - like the utilization of a context diagram to better delineate the boundary system-environment or even the use of functional decomposition - should be (and really are being) considered. However, in our opinion, by first defining domains and only later on identifying objects/classes inside loosely connected domains, one of the great advantages of the object-oriented paradigm might be lost: the concept of objects as a single logical location around which all the project information is organized. Such approach may lead to a description containing one of the major drawbacks of the structured analysis, i.e. the loose connection among analysis documentation components (processes and domains) and the real-world objects.

We do agree that a 'functional classification' inside an 'object-oriented' model can be fruitful. However, as the name already indicates, in order to maintain the denomination *object-oriented*, a development process should always consider objects/classes as key points.

Historically, the field of industrial automation systems has been characterised as a multi-disciplined one, since people with different backgrounds take part in the development process. Curiously, the areas of knowledge of the involved persons are commonly classified according to their functionality: there are experts in the field of process control, experts in fault diagnosis, in data acquisition, in visualization, etc. Terms and concepts used by these groups may vary widely. While people involved with fault diagnosis probably are more familiar with concepts of the artificial intelligence field - like backward-chaining, inference mechanisms,... - control engineers

are more likely to think about PID-controllers, stability proofs, and so on. Because of this, to define a development process as well as to deliver a system requirements specification that is clear for all these people is not an easy task.

3. A proposal

A modelling technique for industrial automation systems which combines object-oriented concepts with the concept of functional decomposition has been proposed in .despite the fact that this might look like a contradiction. Functional views are applied to abstract out the aspects of a specification which are of relevance to a given group. However, all the information is still organized around objects and classes, maintaining the direct mapping of the semantics from the problem domain to the specification.

In order to illustrate this discussion, consider the example of a tank class where chemical reactions can be conducted. Suppose a tank has a attribut called fluid_in_tank, which represents the actual level of a fluid inside the tank. Developers responsible for fulfilling different functional requirements shall probably have different views of the tank class and of the *fluid_in_tank* attribute:

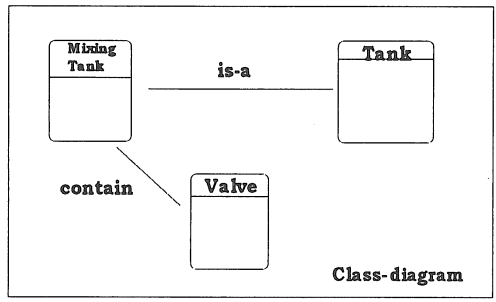
- from the point of view of an user-interface builder, a tank is an icon consisting of some geometric figures,and its attribute fluid_in_tank delimitates an area to be filled on the screen;
- from the point of view of a control engineer, a tank may be seen as an element that has to be heated according to some recipe, by using different control strategies. The attribute fluid_in_tank can be used as an input in a feedback control network.

These different views are taken into account in our framework (fig.2), in a manner quite similar to the 'horizontal decomposition' presented before. They act like filters, neglecting some unnecessary information and stressing others of relevance to the subject matter under study. However, maintaining the tank element as a unit is very important. Our experience has shown that these kind of elements really tend to be more stable than functions (or domains based on functional views). Moreover, the consistency of the information encapsulated in a class/object can be more

easily ensured. This is true not only for the analysis and design phase, but also during the implementation. For instance, the attribute fluid_in_tank of an instance of the tank class might be used either by a controller to determine a inlet valve opening degree, as well by a fault diagnosis system to check some premises, or even by a user interface system to show graphically the actual level. All of these operations could be running concurrently. The use of views provides mechanisms for synchronizing the access of the object's attributes, additionally avoiding the splitting of information through several instances.

4. Conclusion

The use of abstraction mechanisms in order to manage the complexity of large systems is a core concept that should be properly taken into account in object-oriented methods. In this paper we have discussed how some existing object-oriented methods define and apply these mechanisms. Based on our experience developing industrial automation examples, an approach to handle the objects/ classes through functional views was briefly described. The classification of these functional views are related to the areas of knowledge of people involved in the development process. The main idea is to enhance the understandability of the specification, while ensuring consistency of the information described.



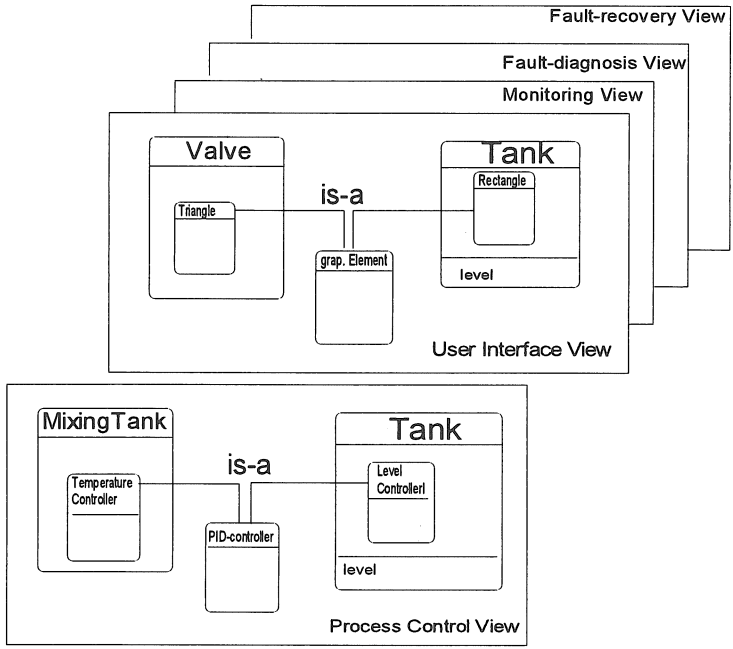


Fig.2: Differemt functional perspectives of objects

5. References

1. Embley, D., Kurtz,B. and Woodfield,S.: *Object-Oriented Systems Analysis : A Model-Driven Approach*. Yourdon Press, Englewood Cliffs, NJ, 1992.
2. Shlaer,S. and Mellor,S.: *Object Life Cycles: Modeling the World in States.* Yourdon Press, Englewood Cliffs, NJ, 1992.
3. Miller,G. *The Magical Number Seven, Plus or Minus Two: Some Limits in Our Capacity for Processing Information*. The Psychological Review, vol. 63(2), March 1956.
4. Rumbaugh, J. et all:*Object-Oriented Modelling and Design.* Englewood Cliffs, NJ, Prentice-Hall 1991.
5. Colbert, E.:*The Object-Oriented Software Development Method: A Practical Approach to Object-Oriented Development* In: ACM Tri-ADA Conference 1989, pp. 400-415.
6. Chen, P.P.: *The entity-relationship model - toward a unified view of data*. ACM Transactions on Database Systems, 1(1):9-36.
7. Booch,G.: *Object-Oriented Design with Applications*. Benjamin/Cummings, Menlo Park, CA, 1990.
8. Coleman,P. ,Hayes, F., and Bear,S.: *Introducing Objectcharts or How to Use Statecharts in Object-Oriented Design*. IEEE TSE,18(1),January1992. pp.9 - 18.
9. Pereira, C. and Darscht, P.: "*Using Object-Orientation in Real-Time Applications: An Experience Report*". in: Proc. of TOOLS Europe 94. Versailles, France. 1994
10. Jacobson, I.: „*Time for a cease-fire in the methods war*". Guest Editorial, Journal of Object-Oriented Programming, Vol. 6, No. 4, July/August 1993. pp. 6 and 84.
11. Parnas, D.; Clements, P.; Weiss, D.: „*The Modular Structure of Complex Systems*" IEEE Transactions on Software Engineering, Vol. SE-11, No. 3, March 1985, pp. 259 - 266

12. Harel, D.: *"Statecharts: A Visual Formalism for Complex Systems"*. Science of Computer Programming 8 (1987) 231-274, North-Holland.
13. Champeaux,D.: *„Object-Oriented Analysis and Top-Down Software Development"*. Proceedings of ECOOP 91, Lecture Notes in Computer Science 512, Springer Verlag, NY, 1991, pp. 360-376.
14. Champeaux,D. and Faure,P.: *„A comparative study of object-oriented analysis methods"*. JOOP-Journal of Object-Oriented Programming, Vol. 5. No.1, March/April 1992.
15. Coleman, D. et all.: *"Object-Oriented Development: The Fusion Method"*. Prentice-Hall, 1994.
16. Davis, A.: *„Software Requirements. Analysis & Specification"*. Prentice Hall Inc., Englewood Cliffs, New Jersey, 1990.
17. Drake,J. et all: *„A Case Study to Evaluate Three Object-Oriented Analysis Techniques"*.University of Minnesota, Dept. of Computer Science, Technical Report Feb. 1992
18. Pereira, C. *„An Object-Oriented Approach for Real-Time Requirements Engineering"*. Ph.D. thesis, University of Stuttgart, 1995. (in German).