An Evolutionary Approach to the Design of Supervision and Control Systems

Alberto Coen-Porisini♣
Matteo Pradella♦
Matteo Rossi♦

♣ Dipartimento di Ingegneria dell'Innovazione – Università di Lecce
via per Monteroni, I-73100 Lecce, Italy
Ph: +39-0832-320226 – coen@ultra5.unile.it

♦ Dipartimento di Elettronica e Informazione – Politecnico di Milano
Piazza L. da Vinci 32, I-20133 Milano, Italy
Ph: +39-02-2399-3666 – pradella@elet.polimi.it

(Extended Abstract)

1. Introduction

Every non trivial system is required to evolve in time because the context in which it works changes (end users needs and/or habits, processes under control etc.). Moreover, even if the requirements do not change technology does, and thus an up to date system from a functional point of view can very rapidly become out of date because of changes in software/hardware technologies. Finally, those two major changes are in general correlated since, for example, major changes at the technological level will drive changes at the requirement level and vice versa.

This scenario is particularly true for Supervision and Control Systems (SCS) that is, (real time) systems whose aim is to supervise (i.e., to monitor a predefined set of variables) and to control (i.e., to intervene in order to keep such variables within a pre-determined range of values) a more or less complex process[1]. SCS are long-lived systems involving major economic investments, and thus, they must evolve through several "generations" of technology and requirements. This accounts for the high value that is attributed to specifications *evolvability* and *generality*, that is the ability to adapt them with minor changes to the needs of different cases [1]. Such a need can be viewed at two different levels:

- *Configuration Level*: industrial SCS are often replicated in several instances having a common structure but differing in the number and kind of components and/or parameters, to adjust operating conditions that may vary from plant to plant. For this reason, configurability criteria are carefully considered, starting from the very early phases of system analysis.
- *Functionality Level*: most of the available SCS are currently based on proprietary solutions, which are inherently closed. Adding a new functionality, such as diagnosis capabilities, to an existing system very often results in designing a new SCS devoted to that particular new function. Thus, most of the current SCS are composed of many different sub-systems that are not integrated. Therefore the need for an open platform, on which the different subsystems can coexist and share information, and which can allow one to extend a SCS by adding new components, is becoming more and more relevant.

[1] For example in a thermo-electric power plant, the SCS acquires values representing some physical quantities such as the pressure and the temperature of the steam by means of sensors, and reacts by means of some actuators opening/closing a gauge or augmenting/lessening the amount of fuel burnt.

In [2] the experience gained in introducing formal methods in the development of SCS has been reported with a particular emphasis on configuration issues and how they are tackled by using the formal specification language TRIO [3, 4]. In this paper, instead, we report the work done in the ESPRIT Projects OpenDREAMS and OpenDREAMS-II whose aim is to define an open platform for the design and the implementation of evolvable SCS. The OpenDREAMS projects took CORBA and the OMA [5] as the starting points for defining such a platform. However, introducing a new technology, even if is expected to allow a smooth evolution of SCS, does not solve all the problems by itself. It is necessary to complement it with the appropriate methodologies (and tools) in order to take effectively advantage from the opportunities provided by the technology.

In this paper we discuss how a specification of a SCS written in TRIO can be transformed into a high-level design document in which the target platform (CORBA) is taken into account. The rest of this paper is organized as follows: Section 2 provides a short overview of TRIO and CORBA, while Section 3 discusses the TRIO/CORBA methodology. Section 4 provides a case study based on simplification of a real industrial SCS. Finally Section 5 draws some conclusions.

2. TRIO & CORBA

2.1 The TRIO Specification Language

The TRIO specification language [3, 4] includes a basic logic language for specifying in the small and an object-oriented extension, which supports writing modular reusable specifications of complex systems.

The basic logic language supports a linear notion of time and defines a single basic modal operator, called *Dist*, that relates the *current time*, which is left implicit in the formula, to another time instant: the formula *Dist(F, t)*, where *F* is a formula and *t* a term indicating a time distance, specifies that *F* holds at a time instant at *t* time units from the current instant. Several *derived temporal operators*, can be defined from the basic *Dist* operator through propositional composition and first order quantification on variables representing a time distance.

For specifying large and complex systems, TRIO has been enriched with concepts and constructs from object-oriented methodology such as classes, inheritance and genericity. Classes denote collections of objects that satisfy a set of axioms. They can be either *simple* or *structured* –the latter term denoting classes obtained by composing simpler ones. A simple class is defined through a set of axioms premised by a declaration of all items that are referred therein. Some of such items are in the *interface* of the class, i.e., they may be referenced from outside it in the context of a complex class that includes a module that belongs to that class.

2.2 CORBA & OMA

In CORBA's approach a system is viewed as a set of objects exchanging messages through a common communication layer. That is, CORBA defines a software bus (ORB - Object Request Broker), where distributed (and possibly heterogeneous objects) may be plugged in. In order to allow objects written in different implementation languages to communicate, CORBA defines an Interface Definition Language (IDL) that must be used to describe the interface of any object connected to the ORB. The IDL is a pure declarative language that describes the operations supported by an object.

According to the OMG's reference architecture (OMA), all objects that contribute to the application are partitioned into four major categories [5, 6, 7]. 1) *CORBAServices* defining the system-level objects that extend the bus; these are collections of objects providing well-defined services, packaged as components. Examples of CORBAServices are Persistency, Query, Transaction, Naming and Event. 2) *CORBAFacilities* defining interfaces and/or objects that provide general purpose capabilities applicable to most application domains. Examples of CORBAFacilities include frameworks for User Interface, Information Management, System Management and Task Management. 3) *CORBADomains* embracing objects that are specific to industrial vertical markets. These high-level components provide functionalities of direct interest to developers in particular application domains (e.g., Finance, Healthcare, Manufacturing, Telecom, Transportation, and Supervision & Control). Finally, 4) *Application Objects* being the ultimate consumers of the CORBA infrastructure are the components specific to end-user applications.

The designer's job is therefore the definition and implementation of Application objects that will communicate among each other and with Services, Domains, and Facilities, through the ORB.

3. The T/C methodology: from specification to architecture

The TRIO/CORBA (T/C) methodology starts from a TRIO specification of the system under design. According to this methodology, the designer smoothly moves from the specification toward a high level design in a step-wise fashion. At each step a different aspect is taken into account so that the complexity of the whole design is kept under control. Moreover, at each step a "design document" is produced in order to keep track of the different choices made. In this way it is possible to modify one or more deign choices without having to modify the whole design.

In order to make the transition from specification to design as smooth as possible every significant aspect is described by means of the T/C design language. During the different steps, the designer gradually introduces every significant architectural aspect, using the T/C language, which is a formal language obtained from TRIO by adding all CORBA basic concepts (operations, attributes, exceptions, interfaces, application objects). Complex concepts (services, frameworks[2] [8]) are built from these basic elements. For instance, T/C includes `Interface Classes` to model IDL Interface, `Application Object Classes` to model CORBA application objects, `Environment Classes` to describe the SCS environment, etc.

The methodology is mainly structured into the following five major steps:

1. identification of architecture-impacting frameworks;
2. identification of data flows between the specification classes;
3. identification of interfaces and application objects;
4. identification of the semantics of operations and attributes;
5. identification of services and non-architecture-impacting frameworks.

[2] Frameworks are usually used to build CORBAFacilities and/or CORBADomains.

3.1 Step 1: Identifying Architecture-Impacting Framework

This step aims at identifying the architecture-impacting frameworks in the specification. Architecture-impacting frameworks are already available components (CORBAFacilities and/or CORBADomains) that must be customized by the designer in order to make them become application objects. Application objects deriving from architecture-impacting frameworks usually have to inherit well-defined interfaces and must be structured in already defined ways. As a consequence, it might be useful to identify in the specification the TRIO classes that will generate T/C classes modeling application objects belonging to the framework. The way in which such generation occurs is described in step 3.

3.2 Step 2: Identifying Data Flows

The second step aims at identifying explicit information exchanges among the specification classes. These are called *data flows* and are a first step for moving from the concept of sharing logical items (Predicates, functions) - typical of TRIO classes - towards the concept of exported operations - typical of CORBA. A data flow can be viewed as a complex merge of TRIO logic items, with an explicit direction of information flow.

Once every data flow is identified it is necessary to characterize it either as an operation or an attribute, which are typical IDL concepts, or as a multicast. Then the designer decides to which TRIO class each operation/attribute belongs.

3.3 Step 3: Identifying Application Objects and Interfaces

During this step the designer will first identify application objects possibly restructuring the TRIO specification by splitting and/or grouping TRIO classes. For instance, TRIO classes identified at step 1 will most likely be restructured in order to fit the already defined architecture of the selected frameworks.

`Application Object classes` will then originate from the classes that are touched by at least one data flow. Notice that, it is possible that not all classes of a TRIO class diagram represent application objects since some of them might represent physical devices (for example, sensors), that could possibly interact with application objects, but not by means of operations and/or attributes. Starting from the description of the `Application Object` classes, it is possible to automatically build the connections between the new classes. TRIO axioms can be automatically moved from the old classes to the new ones, once the composition of the latter is known.

The sub-sequent phase is the identification of interfaces: since every `Application Object` class owning at least one attribute and/or operation is a CORBA server, it must support at least one interface. Moreover, every operation, attribute and multicast owned by a server must belong to one interface of the `Application Object` class.

3.4 Step 4: Identifying the Semantics of Operations and Attributes

In this step a semantic classification of operation and attributes is performed according to CORBA “informal semantics”. For instance, operations can be blocking or non blocking; attributes can be 'readonly' or not. As a result a set of axioms describing the semantics is added to the T/C design definition.

3.5 Step 5: Identifying Services and Non-Architecture-Impacting Frameworks

In the last step, CORBAServices and non-architecture-impacting frameworks used by the application objects are added. The transformations performed during this step are mainly of two kinds: 1) introduction of stereotypes (e.g., a multicast implemented by means of the Event service is labeled <<event>>) and 2) introduction of *standard objects* (i.e., objects supporting standard interfaces and whose behavior is defined by a service/framework).

4. A Case Study: the IMS application

During the OpenDREAMS-II project, we applied the methodology to specify and develop an Instrumentation and Maintenance System (IMS) for an energy plant as a part of an ASCS (Advanced Supervision and Control System). The IMS is in charge of monitoring the state of the devices (transducers, actuators), and notify the operator about possible malfunctions.

A more detailed description of this case study is presented in the full paper.

5. Conclusions

This paper discusses the T/C methodology that has been defined to provide a smooth transition from specification to high level design, in the field of SCS. The main advantages of this approach are:

Being a step-wise refinement it naturally fits the requirements of evolvability since any major design decision is taken at different steps. Thus modifying one of them will cause the modification of only the subsequent phases and not of the whole design process. Moreover, adding new functionalities to an already existing system can be done in an incremental way. At first the specification needs to be complemented with the description of the new functionality; then, a new design must be done, by adding the new parts to the already existing design. Of course this may require some major modification of already made choices, but this is more likely not to spread across the whole design.

Finally, the TRIO language (along with its design extensions - T/C) is already proved to be effective to handle configurability issues, which are very relevant for SCS.

References

[1] R. Lutz, "Reuse of a Formal Model for Requirements Validation", Proc. 4th NASA Langley Formal Methods Workshop, C. Holloway and K. Hayhurst, eds., Hampton, 1997.

[2] E. Ciapessoni, A. Coen-Porisini, E. Crivelli, D. Mandrioli, P. Mirandola, A. Morzenti, “From Formal Methods to Formally Based Mthods: An Industrial Experience” ACM Trans. on Software. Engineering and Methodologies, vol.8, no. 1, January 1999, pp79-113.

[3] C.Ghezzi, D.Mandrioli, A.Morzenti, "TRIO, a logic language for executable specifications of real-time systems", The Journal of Systems and Software, Elsevier Science Publishing, vol.12, no.2, May 1990.

[4] A. Morzenti, P. San Pietro, “Object-Oriented Logic Specifications of Time Critical Systems”, ACM Transactions on Software Engineering and Methodologies, vol.3, no.1, January 1994, pp. 56-98.

[5] CORBA: Architecture and Specification, OMG book, OMG editor, Framingham, USA, August 1995.

[6] CORBAServices, *OMG book*, OMG editor, Framingham, USA, August 1995.

[7] CORBAFacilities, *OMG book*, OMG editor, Framingham, USA, August 1995.

[8] Object Oriented Application Frameworks, Special Issue of the Comm. Of the ACM, M. Fayad, D. Fayad Eds., vol.40, no. 10, October 1997