# A model based architecture for the semi-automatic generation of multiplatform applications: Brake system diagnostic application

Francisco Jurado, María José Santofimia, Miguel A. Redondo

Grupo de Investigación CHICO
Escuela Superior de Informática. Universidad de Castilla – La Mancha
Paseo de la Universidad, 4. 13071 Ciudad Real.
Spain
Francisco.Jurado@uclm.es, i72sarom@yahoo.es, Miguel.Redondo@uclm.es

**Abstract**. The diagnostics of electro-pneumatic brake system for industrial vehicles is a process dependent on the system manufacturer. With regard to this, the tools developed by manufactures are usually aimed to steady platforms devoid of standards. In order to lighten this drawback, we propose the design of model based tools for the semi-automatic generation of multiplatform applications, from domain, task, and platform (PC, Tablet PC, PDA, "proprietary devices", etc.) models. This paper describes the architectural approach for the diagnostic application generator tool. This paper also points out some of the problems that need to be solved, and lines of work are proposed on this regard.

## Introduction

The diagnostics of electro-pneumatic brake system for industrial vehicles is a process dependent on the system manufacturer. Therefore, a specific device is required in order to verify each kind of system.

In the diagnostic process of ABS systems four families can be distinguished, namely: WABCO, HALDEX, KNOOR Y BOSCH. Each one is composed of a range of subfamilies, with their own communication protocol for the diagnostic process. In a first attempt to join the different diagnostic capabilities of the different ABS system families, Cojali[1] designs the JalTest. However, JalTest is a closed product, with little chances of expansion. Everytime a new brake system is released, it is necessary to modify this device software so that it can support the communication protocol, commands, and services introduced by the new system. This is an expensive task, in terms of time, effort, and therefore cost. The same happen when a new device is proposed to support the application (for example, a PDA connected to the brake system control unit through Bluetooth).

[1] http://www.cojali.com/

It could be interesting to have a modelling language, for the protocol specification used in the different diagnostic system, and a language for describing the user tasks related to the diagnostic processes, obtaining the code that implement them in the programming language, suitable for the target platform. Therefore, it will be possible to reach a level of abstraction high enough for focusing on the requirements at domain, communication, and user interface levels. These requirements are to be solved in the target platform.

Therefore, our hypothesis is that this approach saves us from having to rewrite the whole code, and allows us the use of high level specifications in order to generate code for different platforms, promoting the use of mobile devices for the diagnostics of these systems.

In this paper we present an approach for facing the problem of the semi-automatically generated code for applications aimed to diagnose advanced electro-pneumatic brake systems, different brands, for industrial vehicles, using a model based approach, by means of task, domain and platform models. This paper is organized as follows: the first section presents the proposed architecture, along with the main problems that have to be tackled; the second section describes how to solve the problem of the specification and development of the user interface for the application; the third section describes our approach for the development of code aimed to process the communication protocol for the diagnostic process; finally, some ideas derived from the present work, remarking the future lines of work.

## Proposed architecture

Analysising the proposed issue, three main aspects have to be taken into account:

- Target platform, this matter provides the requirements regarding the operating system, display and interaction capabilities, hardware and communication interfaces, etc.
- Graphic user interface, which will depend on the different tasks that the user has to carry out on the application.
- Diagnostic protocol, which sets both, the services that can be provided to the user interface layer, and the communication process with the different control units of the electro-pneumatic system.

In order to tackle this problem, a first approach consists in creating two different branches of work: on the one hand, a tool for the automatic generation of code that implements the communication protocol with the electro-pneumatic control unit system; on the other hand, a tool that generates code for the user interface. These tools use the most appropriate models for the aspects that need to be specified.

The platform model sets strong restrictions for both, the code generation for the communication protocol implementation, and the user interface. Moreover, the application domain layer, or at least part of it, is specified by the services offered by the diagnostic protocol. The application domain layer sets the link between the

generated code for the implementation of the communication protocol and the generated code for the user interface. The diagnostic application is made up from the code generated by these two tools.

In terms of functional blocks, the proposed architecture is shown in figure 1

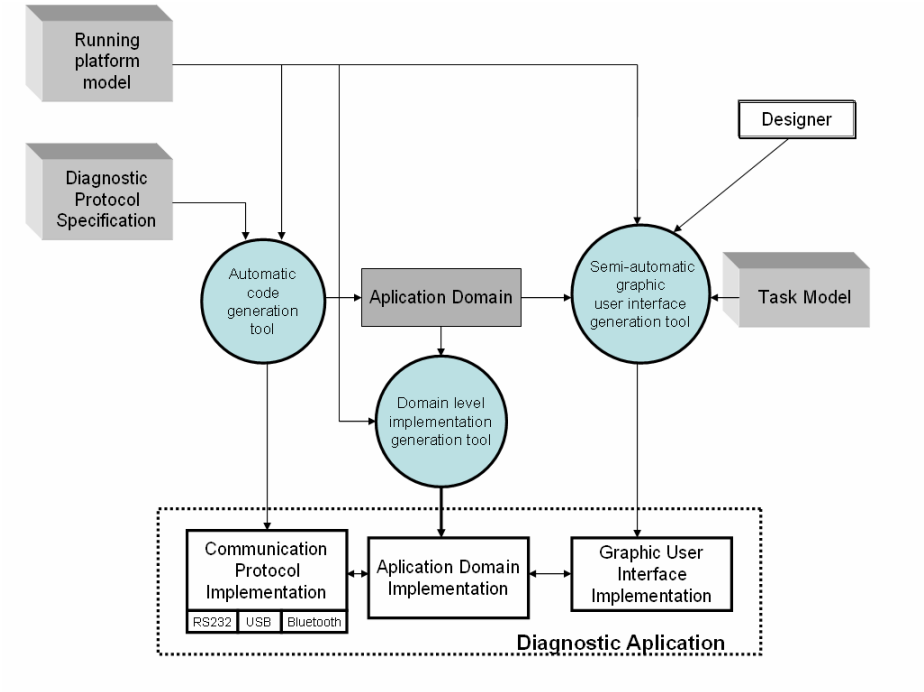


**Fig. 1.** Functional architecture of the diagnostic application generator tool.

As it can be noticed in the above diagram, there are two starting points. One point is the model that collects the relevant features of the target platform (PC, Tablet PC, PDA, etc.). The other point is the specification of the diagnostic protocol that provides information about the diagnostic commands and the issues regarding the communication. These two models are first processed by a code generator tool that provides an implementation of the diagnostic protocol in terms of the communication channel specified in the target platform model. Moreover, it is also generated an application domain model that contains information regarding the commands supported by the diagnostic protocol, and the results that these commands can provide.

The application domain model is independent of the target platform. In order to characterize it as a command interpreter for the target platform this model is processed by a code generator tool that provides its implementation.

Diagnostic tasks are understood as a sequence of actions that users carry out in order to diagnose the different aspects of a brake system. The relationship between these tasks and the domain model is taken into account in order to generate the application user interface. Therefore, it is necessary to develop an interface generator tool. This tool basically provides information about the elements that the interface should contain, but there is not information about how these elements have to be placed. This decision is taken by the designer.

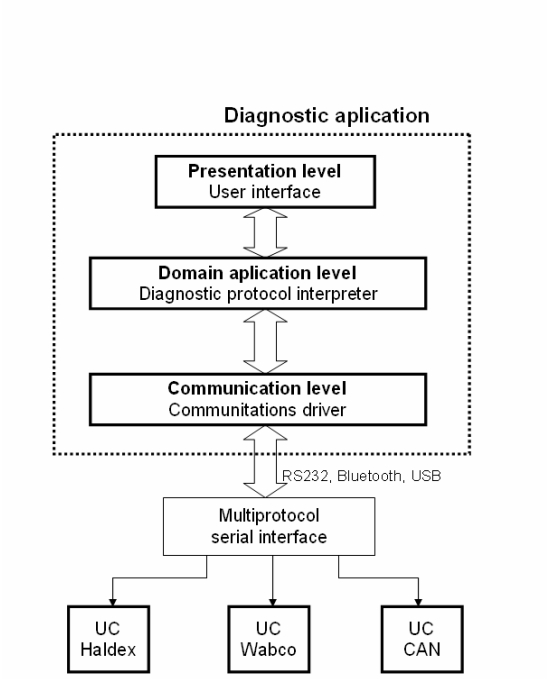


**Fig. 2.** Layers of application of diagnostics and their connection with the most frequently used brake system .

By doing so, we would have obtained a diagnostic application, organized in three layers: communication, application domain, and presentation (figure 2). It has to be noticed that at communication layer it is important to remember that the channels, included on standard devices such as PCs, PDAs, etc., are RS232 port, Bluetooth or USB. These channels generally are not compatible with the electro-pneumatic system ones. This drawback is overcome using a protocol adapter device, designed by COJALI. Therefore, the diagnostic application, automatically generated, is independent of the aspects regarding the communication with the control units of the brake systems.

In order to materialize the tools here proposed, it is necessary to accomplish two basic tasks, having always on mind the use of models: (1) specification of the user

tasks and generation of the user interface code; (2) specification of the diagnostic protocol and generation of the code that processes it.

## Specification and generation of the user interface

In recent years, the user interface development has been a discipline with an increasing importance. Historically, the interface design was considered as an artistic task, lacking methodology. The Web technology, pervasive computing, and mobile devices made the interface development more and more complex. This context arises the need of establishing a methodology that supports the interface design as an engineering process.

Studies about the possibility of automating the process of interface generation soon appeared, although the effort required doing so was quite high. Several lines of investigation appear regarding this aspect. The Model Based User Interface (MBUI) paradigm has been the most important of all those lines of investigation. This paradigm provides information about the user interface, such as the tasks that the user accomplishes on the interface, how the interaction with the interface is carried out, etc.

Generally, the tools that support the interface design can be classified into three big categories:

- Language based tools. These tools require the use of a special purpose language.
- Interactive tools for the graphic specification. These tools support an interactive design of the interface.
- Model based generation tools: These tools make use of high level specifications in order to automatically generate the interface.

The MBUI methodology establishes the use of several models. A model is understood as a declarative specification of the structure and behaviour of a software element. However, we talk about declarative models, and therefore these models do not contain code but only high abstraction descriptions. These declarative models are characterized for being really expressive.

There is a wide range of models, each one aimed to model a specific feature of the system. The MBUI design paradigm has several variants, in terms of the models used. However, some of these models are common to most of these variants, namely:

– Task Model
– Domain Model
– Platform Model
– Presentation Model
– Dialogue Model

Among these models, and for a first approach to our solution, we just use the task and domain models. One of the most important design primitives intended to obtain

usable interactive systems is focusing on the users and their tasks. The identification and comprehension of the user activity are indispensable, so that it can be formulated in terms of a set of tasks. The task model describes the static and dynamic distribution of the user work. Usually, this information is described by a hierarchical organization of the objectives, tasks, subtasks, and the relationship established among them. The CTT language (Concur Task Tree) is proposed for the task specification and task modelling [1,2].

CTTE is a tool that supports the CTT language, for the automatic generation of applications. This tool supports the analysis and development of tasks for cooperative applications. CTTE counts with a graphical interface that allows the manipulation of a set of elements (user actions), hierarchically grouped and connected by temporary commands. These elements and commands are the ones provided by CTT.

Once the task model has been developed, TERESA[4] is used for the user interface generation. TERESA is a transformation-based tool that semi-automatically generates multi-device interfaces. This tool uses the XML file, generated from the task model specified by CTT. This XML specification is automatically generated by CTTE.

This method has already been used and validated in an evolution process of a computer supported collaborative learning (CSCL), based on the desktop metaphor in order to include tasks by mobile computing [5,6].

## Specification of the diagnostic protocols

Revising the bibliography, there can be found several languages for the specification of communication protocols. Among the most relevant systems we mention SDL [7], Lotos [8] and Estelle [9].

SDL[2] is the one recommended by the ITU[3], with two representation possibilities: text and graphic. It consists of a model based on the communication between processes, described as finite state machines. Lotos[4] is widely used in industrial and university environment for the specification of OSI protocols. It is the ISO IS8807 standard, established in February 1989, and it is a technique for the formal specification of distributed and concurrent systems. It is made up from a language for process specifications and another language for algebraic specifications that support the system definition, at different abstraction levels. Estelle[5] is the second formal description technique developed by ISO, based on a finite state transition model and languages such as Pascal and Ada. This is the ISO IS9074 standard, established in July 1989.

[2] Specification and Description Language. Lenguaje de especificación y descripción
[3] International Telecomunciation Union
[4] Language of Temporal Ordering Specifications. Lenguaje de especificaciones de ordenamiento temporal
[5] Extended State Transition Language

There are compilers for these languages that support, in a semi-automatic way, the generation of C or Pascal code. However, these are limited solutions that can only be used by those who know which the access point to the software is, that is, those who know the programming interface offered by the compiler. In other situations, compilers generate the verification code.

Another relevant example is the Promela[6] language. Starting from a system modelled by Promela, there are tools, such as Spin [10], which are able to run simulations of the modelled system, or to generate C code that can execute an efficient verification of the system properties. Along the simulation and verification processes, Spin checks for the lack of inter-block, send and receive errors, and never run code. Furthermore, it is able to check the system invariants. However, both Promela and Spin are specially aimed to the verification and do not offer any mechanism for the code generation that could be used for the implementation of the modelled protocol.

These techniques, along with other specification protocols, such as state diagrams or Petri nets, provide us with a graphical way of representing and formalizing communication protocols. However, nowadays, the development of tools for the code generation that implements the communication automaton, from these languages, has decreased in importance.

It is obvious that there is a need for a mechanism that simplifies the way the communication protocol is specified, independently of the programming language used for the implementation, the operating system, and the communication channel (serial port, TCP connection, etc.). Furthermore, it is necessary to provide a way of knowing the services offered by the communication protocol. This has to be done in an easy way and with little detail about the implementation.

Our purpose is to develop a language (or sublanguage) for the communication protocol specifications, independent of the implementation language and the platform on which it will be run. This approach detaches the operating system and the communication channel used. Once the protocol has been specified, the next step is the generation of the implementation code, after revising the translation to the programming language, and identificating of the physical communication channel.

The protocol specification proposed should follow two requirements:

- Identifying the primitives that can be directly invoked from the automaton that implements the communication, that is, the services offered to the rest of the system by the software component that implements the protocol.
- The communication protocol specified should not make any asumption about the channel used in the communication with the other part.

The protocol code generator carries out a translation or mapping from the designed protocol specification language to the target programming language, on which the automaton will be implemented. The main advantage of this approach is that we are

[6] Protocol/Process Meta Language, Metalenguaje de protocolos/procesos

not required to re-write the whole code for each different protocol, relieving us from the other side we are saved from considering the peculiarities of the language and target platform of the application.

The specification protocol language has been designed using XML, in order to achieve that:

- The language was open, so that it can be extended as new requirements were identified, and in a try to support future works.
- Quicker test and verification process, since there is no need to modify the code but the specification affected.
- Easy to integrate with off-the-shelf protocol validation mechanism, such as Promela
- High abstraction mechanism that simplifies the process of identification of the incongruence which are difficult to point out in the code.
- There is no need to write language analysers since there are plenty of APIs.

Focused on the achievement of a multi-device code, it is clear that there is a need of interaction with the communication hardware, and at a higher level with its driver, from the first stages of the development process. How can the issue of adapting the generated code to the primitives offered by the communication interface be solved?

A software pattern can be the solution for this specific situation, the facade pattern [11]. We generate an intermediate object, placed in the middle of the generated code and the driver, in such a way that the same set of primitives is always used over the generated code.

There is another drawback in the achievement of a multi-device solution: the hardware. The communication with the different brake system control units is usually carried out through the serial port. The problem is that the serial communication must be accomplished through a channel handle by both UART[7] (for Haldex and Wabco subfamilies), or a CAN[8] channel; in cases, the physical layer and the MAC are totally different.

The use of CAN ports provides little portability, since it means buying an off-the-shelf solution (usually a controller card), with drivers that will be linked to a specific operating system and platform. Once again the solution to this problem consists of the use of a facade pattern, this time on the hardware. The hardware designed to this purpose is called MPSI (Multiprotocol Serial Interface). It is in charge of accomplishing the adaptations required to the physical and MAC layers, required for the communication. This device can be seen in figure 2, and it connects the communication layer of the diagnostic application and the different control units supported. Therefore the device (PC, PDA, TabletPC, etc.) establishes the communication with the MPSI through the RS232, Bluetooth, USB or any other communication channel. These communication channels are far more common on conventional computing and mobile devices.

[7] Universal Asynchronous Receiver/Transceiver
[8] Controler Area Network

## Final remarks and further work lines

In this work we have started from the hypothesis that it is feasible to semi-automatically generate multi-device code, using the models that better describe the parts that have to be specified, using design patterns as well. Therefore, it will be possible to describe the features of an application, even a relatively complex one, in such a way that by modifying the platform model, this application would be available for another different platform such as PC, Tablet PC, PDA, etc.

On this paper we have proposed an architecture for a specific context, this aspect has allowed us to evaluate our approach with a high level of realism. The proposed architecture counts with two main drawbacks that need to be overcome:

1. User interface specification and generation
2. Communication protocol specification and the code generation for its process.

The solution proposed for the first issue lies on the use of a Model Based approach, along with the code generation assisted by existing tools adapted to new technologies. For the second problem, we propose the design of a new language (or sublanguage) XML based and especially oriented to the generation of an open implementation for protocol processing but not oriented to its verification. Therefore, this work has to continue its evolution in two different lines:

1. Extending the possibilities offered by TERESA in order to support different technologies and devices for the implementation of user interfaces.
2. Precisely defining the protocol specification language and validation for different communication protocols oriented to system diagnostics, in our case, of electro-pneumatic brake systems.

## References

[1] Paternò, F., Mancini, C. and Meniconi, (1997). ConcurTaskTrees: A diagrammatic notation for specifying task models. In IFIP TC 13 International Conference on Human-Computer Interaction Interact'97. (Ed, S. Howard, J. H., and G. Lindgaard) Kluwer Academic Publishers, Sydney, pp. 362-369.

[2] Paternò, F., Santoro, C. and Tahmassebi, S. (1998). Formal Models for Cooperative Tasks: Concepts and an Application for En-Route Air Traffic Control. In 5th Int. Workshop on Design, Specification, and Verification of Intractive Systems DSV-IS '98(Ed, Johnson, P. M. a. P.) Springer-Verlag, Abingdon, pp. 71-86.

[3] Mori G., Paternò F., & Santoro C. (2002). CTTE: Support for Developing and Analyzing Task Models for Interactive System Design, IEEE Transactions on Software Engineering, pp.797-813.

[4] Mori G., Paternò F. & Santoro C. (2004). Design and Development of Multidevice User Interfaces through Multiple Logical Descriptions, IEEE Transactions on Software Engineering, pp.507-520.

[5] Molina, A.I., Redondo, M.A., Ortega, M., Analyzing and modelling user task in DomoSim-TPC system for adapting to mobile devices. Reviewed Selected Papers from the 5th Conference of the Spanish Association for HCI (AIPO) Interacion 2004. Springer-Verlag (In press).

[6] Molina, A.I., Redondo, M.A., Ortega, M. (2004), Task modeling in Computer Supported Collaborative Learning environments to adapt to mobile computing. In Antonio Laganà, Marina L. Gavrilova, Vipin Kumar, Youngsong Mun, Chih Jeng Kenneth Tan, Osvaldo Gervasi (Eds.), Computational Science and Its Applications - ICCSA 2004, International Conference. Proceedings, Part III. Lecture Notes on Computer Science, 3045, pp. 786-794. Springer Verlag.

[7] CCITT SG X: Recommendation Z.100: Specification and descriptón language SDL. Contribution Com X-R15-E. (1987).

[8] Information processing system – Open Systems Interconnection – LOTOS, a formal description technic based on temporal ordering of observation behaviour. International Standard ISO 8807. (1989)

[9] Information processing system – Open Systems Interconnection – Stelle: A formal description technique based on an extended state trasition model. International Standard ISO 9074. (1989)

[10] Holzmann, G.J. (1991) Design and validation of computer protocols. Prentice-Hall.

[11] Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1998), Design pattern. Elements of reusable object-oriented software. Addison-Wesley.