Separating Control and Data Flow: Methodology and Automotive System Case Study

Ouassila Labbani, Jean-Luc Dekeyser, Eric Rutten

Separating Control and Data Flow: Methodology and Automotive System Case Study

Ouassila Labbani — Jean-Luc Dekeyser — Éric Rutten
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
Bâtiment M3, Cité Scientifique

59655 Villeneuve d'Ascq Cedex, France

N° 5832

February 2006

Thème COM

ISSN 0249-6399 ISRN INRIA/RR--5832--FR+ENG

Separating Control and Data Flow: Methodology and Automotive System Case Study

Ouassila Labbani , Jean-Luc Dekeyser , Éric Rutten
Laboratoire d'Informatique Fondamentale de Lille
Université des Sciences et Technologies de Lille
Bâtiment M3, Cité Scientifique
59655 Villeneuve d'Ascq Cedex, France

Thème COM —Systèmes communicants
Projet DaRT

Rapport de recherche n° 5832 —February 2006 — 32 pages

Abstract:
In this document we propose to study the control/data flow separation design methodology, using Scade and Mode-Automata, and its application in the design of an automotive system. This methodology allows to facilitate the specifi cation of different kinds of systems and to have a better readability. It also separates the study of the different parts by using the most appropriate existing tools for each of them.

To do that, we study a cruise control system with GPS which makes possible the control of a car speed depending on its position given by a GPS. This system combines both control and data processing and can be specifi ed using our methodology. The goal of this work consists in presenting the application of our methodology on a real system and studing its advantages notably for formal verifi cation.

Key-words: Control/Data Flow Separation, Scade, Mode-Automata, Intelligent Cruise Control with GPS, Simulation, Verifi cation.

Séparation des Flots de Données et des Flots du Contrôle: Méthodologie et Étude de Cas d'un Système d'Automobile

Résumé :

Dans ce document nous proposons d'étudier une nouvelle méthodologie de conception, séparant les flots de données et du contrôle, et son application dans le cas d'un système d'automobile. Cette méthodologie est basée sur l'utilisation de Scade et le concept des automates de modes. Elle permet de simplifi er la spécifi cation des différents types de systèmes et d'avoir une meilleure lisibilité. Ainsi que l'étude séparée des différentes parties du système en utilisant les outils les plus appropriés pour chacune d'entre elles.

Pour ce faire, nous étudions un système de limiteur et régulateur de vitesse intelligent avec GPS. Ce système permet de contrôler la vitesse d'un véhicule en fonction de sa position donnée par le GPS. Son application combine des traitements de données et du contrôle, et peut être spécifi ée en utilisant notre méthodologie de conception. Le but de ce travail consiste à présenter l'application de notre méthodologie de séparation dans un le cas d'un système réel, et étudier ses avantages, notamment pour la vérifi cation formelle.

Mots-clés : Séparation flots de Contrôle/flots de Données, Scade, Automates de Modes, Limiteur et Régulateur de Vitesse Intelligent avec GPS, Simulation, Vérifi cation.

Contents

1 Introduction 4

2 Control/Data Flow Separation Methodology using Scade 5
2.1 Context 5
2.1.1 Reactive Systems and Synchronous Approach 5
2.1.2 Scade 6
2.1.3 Control/Data Flow Combination in Scade 7
2.2 Control/Data Flow Separation Concept 9
2.2.1 Mode-Automata 9
2.2.2 Mode-Automata based Control/Data Flow Separation in Scade 10
2.2.3 Benefits of the Control/Data Flow Separation Methodology 12

3 Automotive System Case Study 12
3.1 Context 13
3.2 Optimization Technique for the Representation of the Roadmap 14
3.3 ICCG System Description 16
3.3.1 `Alarm` Mode 18
3.3.2 `Limit` Mode 18
3.3.3 `Cruise` Mode 18
3.3.4 `LimitGPS` mode 19
3.3.5 `CruiseGPS` mode 19
3.4 System Design without Control/Data Flow Separation Methodology 19
3.5 System Design using Control/Data Flow Separation Methodology 22

4 Experimentation of the System 23
4.1 Simulation of ICCG System 23
4.2 Formal Verification of the ICCG System 24
4.3 Prototype 27
4.4 Field Test 29

5 Conclusion 29

1 Introduction

Complex and critical reactive embedded systems often need reliable and efficient tools and methods for their design. Failures and crashes of these systems can lead to data or time losses, incidents that can potentially be catastrophic. For this reason, several studies have been launched in the reactive systems domain to propose reliable techniques allowing to ensure a good functioning of these systems.

In this field, we often hear about synchronous approach and languages like Lustre [5, 6], Signal [7, 8] and Esterel [9, 10, 11]. These languages use formal techniques having a rigorous semantic allowing to define efficiently a set of tools for simulation, verification and automatic code generation. Synchronous languages are based on different models and can be classified into two main families: *declarative* (data oriented) and *imperative* (control oriented) languages.

However, the most realistic embedded systems combine control and data processing. To specify these systems, several approaches have been proposed like the *multi-languages* approach which combines imperative and declarative languages, and the *transformational* approach which allows the use of both types of languages but, before code generation, the imperative specifications must be translated into declarative specifications, or vice-versa.

In [13], we have studied the transformational approach using Scade (Lustre + Esterel) [14], where Esterel code is transformed into Lustre. We have shown that the currently available syntax of Scade does not follow any clear separation methodology and leads to a mixture of control and data flow representation. This mixture can make difficult the understanding of the system and the re-use of existing applications. To fill this gap, we have proposed to introduce the concept of running modes into Scade specifications to combine the advantages of the two approaches, and to develop a new design methodology separating control and data flows parts.

In this paper, we review the basic concepts of our control/data flow separation methodology, and illustrate its application on a real case of an automotive system. To do that, we propose to study the design of an Intelligent Cruise Control with GPS[1] system (ICCG) which represents a significant automatic contribution in the automotive field.

According to several statistics, high-speed and the no respect of speed limit cause 40% of fatal accidents and increase their severity. In spite of this, more than 60% of drivers do not comply with speed limits on urban roads or trunk roads. The worst situation is when a trunk road passes through a village. There, almost 80% of drivers break the speed limit.

In order to give to drivers the means for controlling the speed of their cars, several constructors have developed various systems such as the speed limiter or regulator already present in some cars. In this field, research continues to give more effective systems, and recently, we often hear about speed regulator systems with GPS which allows to adapt the speed of the car to that authorized in its zone of localization.

The goal of this work consists in presenting the advantages of our methodology in the case of a real system such as the modular development, the readability and in particular, benefits in time and memory capacity in the case of formal verification.

The paper is organized like this: in the second section, we present the main concepts used in our control/data flow separation methodology. In this section, we outline the area of synchronous reactive systems, the Scade environment and the control/data flow combination which represents the basic context of our study. We also present our control/data flow separation methodology based on the Mode-Automata concept and its introduction in Scade. The third section is devoted to the automotive system case study. In this section, we introduce the In-

[1]Global Positioning System

telligent Cruise Control system with GPS and the application of our design methodology for this system. The last section gives some experimental results for this system and shows the advantages of the application of our methodology, in particular for formal varifi cation.

2 Control/Data Flow Separation Methodology using Scade

2.1 Context

2.1.1 Reactive Systems and Synchronous Approach

Reactive Systems are computer systems that react continuously to their environment, by producing results at each invocation [1]. These results depend on data provided by the environment, and on the internal state of the system. In [1], D. Harel and A. Pnueli have given to reactive systems the image of a black box that react to its environment at a speed determined by the latter (fi gure1).

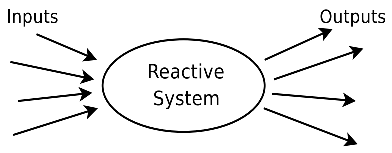


Figure 1: *Reactive System*

Specifi cation of software or hardware reactive systems behavior is complex. It can lead to diffi cult and important errors. Indeed, such systems are not only described by transformational relationships, specifying outputs from inputs, but also by the links between outputs and inputs via their possible combinations in one step [2]. Modeling reactive systems is therefore a diffi cult activity.

In the beginning of the 80's, the family of synchronous languages and formalisms has been a very important contribution to the reactive systems area [3]. Synchronous languages have been introduced to make programming reactive systems easier [4]. They are based on the *synchrony hypothesis* that does not take reaction time in consideration. Each activity can then be dated on the discrete time scale. This hypothesis considers that each reaction is instantanous and atomic.

Synchronous languages are devoted to the design, programming and validation of reactive systems. They have a formal semantics and can be effi ciently compiled, for instance, to C code. These languages can be classifi ed into two main families: *declarative languages* and *imperative languages*.

Declarative or data flow languages like Lustre ([5, 6]) or Signal ([7, 8]) are used when the behavior of the system to be described has some regularity like in signal-processing. Their main task consists in consuming data, performing calculations and producing results.

Imperative or control flow languages like Esterel ([9, 10, 11]) or Argos [12] are more appropriate for programming systems with discrete changes and whose control is dominant: for instance coffee machines. Their purpose is to manage the processing of data by imposing an execution order to operations and by choosing one operation among several exclusive ones.

However, these systems rarely have an exclusively regular or discrete behavior. The most realistic and used embedded systems combine control and data processing. Such global systems may be totally specifi ed with imperative languages, but data dependences between operations can not be clearly specified and furthermore problems may occur due to shared variables. Similarly, they may be totally specified with declarative languages, but the control is hidden in data dependencies making it difficult to specify tests and branchings necessary for verification or optimization purposes. For these reasons, we need efficient tools and methods taking in consideration this kind of systems.

Several approaches have been proposed in this domain. We can find the *multi-languages* approach which combines imperative and declarative languages, like using Lustre and Argos [16]. It is based on a linking mechanism and allows the re-use of existing code. However, when using several languages it is very difficult to ensure that the set of corresponding generated codes will satisfy the global specification. Another design method consists in using a *transformational* approach which allows the use of both types of languages for specification but, before code generation, the imperative specifications must be translated into declarative specifications, or vice-versa, allowing to generate a unique code instead of multiple ones. N. Pernet and Y. Sorel give in [17] an example of this approch which translates SyncCharts, a control flow language, into SynDEx, a data flow language which allows automatic distributed code generation.

The transformational approach is efficient for describing reactive systems combining control and data processing. However, there are systems whose behavior is mainly regular but can switch instantaneously from a behavior to another. They are the systems with running modes. The most adapted method to describe this kind of system consists in using a *multi-styles* approach which makes it possible to describe with only one language the various behaviors of the system. The Mode-Automata [15] represent a significant contribution in this field. Their goal consists in adding an automaton structure to Lustre programs.

In the following, we choose to study the transformational approach using Scade, where Esterel code is transformed into Lustre before any processing of the system, and the introduction of Mode-Automata concepts into Scade models. The goal this work consists in proposing a mixed approach which can facilitate the specification of a variety of synchronous reactive systems.

2.1.2 Scade

Scade (Safety Critical Application Development Environment) [14] is a graphical development environment commercialized by Esterel Technologies[2]. The Scade environment was defined to help and assist the development of critical embedded systems. This environment is composed of several tools such as a graphical editor, a simulator, a model checker and a code generator that automatically translates graphical specifications into C code.

The Scade language is a graphical data flow specification language that can be translated into Lustre. Scade is built on formal foundations. It is deterministic and provides efficient solutions for the development of reactive systems. It has been used in important European avionic projects (Airbus A340-600, A380, Eurocopter) and is also becoming a de-facto standard in this field.

Scade uses two specification formalisms: *block diagrams* for continuous control and *state machines* for discrete control [18]. It adds a rigorous view of these formalisms which includes a precise definition of concurrency and a proof that all Scade programs behave deterministically.

By *continuous control* we mean sampling sensors at regular time intervals, performing signal-processing computations on their values, and outputting values often using complex mathematical formulas. Data is continuously subject to the same transformation. In Scade, continuous control is graphically specified using block diagrams and is based on Lustre Lan-

[2]www.esterel-technologies.com

guage. Scade blocks are fully hierarchical: blocks at a description level can themselves be composed of smaller blocks interconnected by local flows.

By *discrete control* we mean changing the behavior according to external events originating either from discrete sensors and user inputs or from internal program events. Discrete control is generally represented by state machines. A richer concept of hierarchical state machines has been introduced in Scade to avoid the state explosion problems. The Esterel Technologies hierarchical state machines are called *Safe State Machines* (SSMs). These evolved from the Esterel programming language and the SyncCharts state machine [19].

Scade does not give any design methodology. It does not impose a well defi ned technique or rules to follow for the construction of the system, which gives more freedom to users. However, users can specify their system in a not very organized way which makes it diffi cult to understand and to re-use existing specifi cations.

2.1.3 Control/Data Flow Combination in Scade

Large applications contain a mixture of continuous and discrete control parts. To make the specifi cation of such systems easier, Scade makes it possible to seamlessly couple both data flow and state machine styles. Most often, one includes SSMs into block-diagram design to compute and propagate functioning modes. Then, the discrete signals to which a SSM reacts and which it sends back are simply transformed into boolean data flows in the block diagram.

To illustrate this concept, we will study the example of a generic task pattern proposed in [20]. In this example and when the task is active, it can be in three different states: `H`, `M` and `L`. Transitions between states are labelled by conditions c_k, managed by the controller. Figure 2 gives the automaton structure relating to the task specifi cation. These states are differentiated

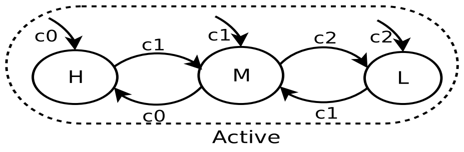


Figure 2: *Simple example of task functioning*

by some characteristics such as for example *time cost*, i.e. the duration taken by one cycle of computation (e.g., each state has a WCET in the reaction) and quality (e.g., precision of numerical computation). For this example, we can think of applications such that we have: `H` (highest quality and time), `M` (medium) and `L` (low). These three modes can be switched between according to the transitions and their conditions c_k. An example is a task computing at each cycle an expression summing three terms: $E = E_1 + E_2 + E_3$, where E_3 and E_2 can be approximated by 0. Each of the modes corresponds to: `H`: the full sum, `M`: an approximation $E_1 + E_2$, `L`: a degraded version E_1.

This example contains both pure control logic and data processing and it can be specifi ed using Scade tool. A possible solution for this system is given by fi gure 3. This specifi cation is mainly inspired by that proposed by Esterel Technologies to specify the `Climate` example [13].

In this example, the system possesses three inputs relative to conditions: c_0, c_1 and c_2, and three inputs relative to parameters E_1, E_2 and E_3. As output, it provides the result E. Input values c_0, c_1 and c_2 pass through a control part represented by the SSM `Control` of fi gure 4.

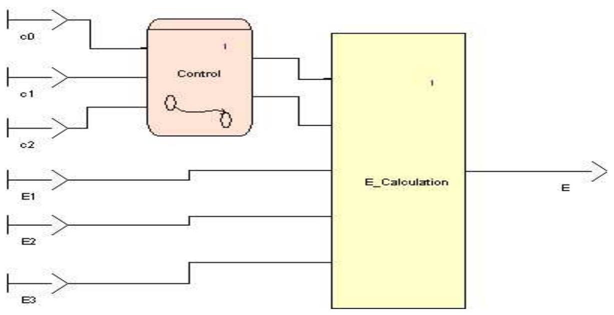


Figure 3: *Active state specification in Scade*

This SSM allows to activate the calculation part `E_Calculation` by two different signals: *high*

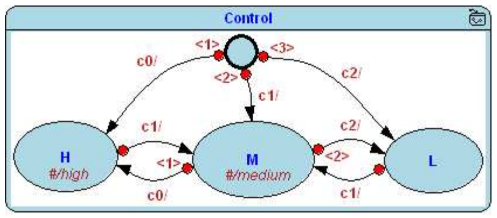


Figure 4: *Control SSM example*

and *medium* which correspond to the activation of state functions *H* and *M* respectively. If the two signals *high* and *medium* are false, the system is in L state.

By descending to a lower level of the hierarchy, the design model corresponding to the `E_Calculation` model is indicated by figure5.

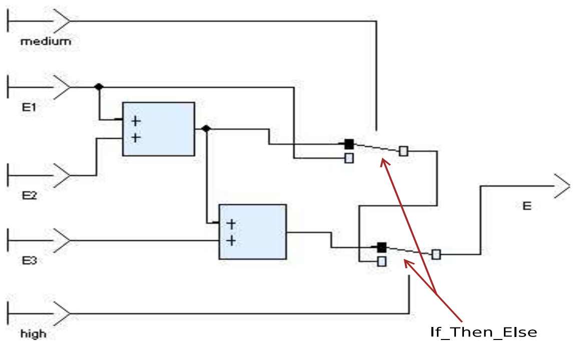


Figure 5: *E_Calculation model example*

In this model, we notice that the calculation part `E_Calculation` contains a mixture of calculation (operator $+$) and control (`If_Then_Else`). This mixture can make difficult the understanding of the system, as well as the use of already existing tools, dedicated exclusively to processing the calculation part or the control part. Furthermore, independently of the values of *high* and *medium* signals, the three calculation parts are always activated and the output value will be chosen depending on signal's values. This corresponds to the strict and compound nature of the conditional structure `If_Then_Else` in Lustre. In this case, the two branches of the conditional structure are always evaluated which can introduce side-effect problems.

In [13], we have proposed a new conceptual model that allows to have a clear separation between control and data flow parts. This will allow us, on the one hand, to avoid the use of the Lustre conditional structure and to have a better readability, and on the other hand, to facilitate the separated study of the different parts by using the most appropriate tools for each category, notably concerning the application of the formal verifi cation techniques.

2.2 Control/Data Flow Separation Concept

The studied example represents a simple case of systems whose behavior is mainly regular but can switch instantaneously from a behavior to another. In this case, the global system is usually composed of a high level control oriented sub-system which executes different data processing for each state of the system. For these systems, it can be important to study separately control and data parts which gives a more structural view of the model and facilitates the modifi cation and the re-use of different parts.

This category of systems is generally known as systems with *running modes*. In [21], F. Maraninchi and Y. Rémond show through a production-cell case study that real industrial applications can be better specifi ed by using a mode-structure if their behavior is mainly regular. These systems can be more easily specifi ed using Mode-Automata.

Several other approaches exist for the specifi cation of regular systems, we quote for example Lucide Synchrone language [22] which is a functional language based on Lustre and OCaml [23]. This language allows to introduce higher order programming concepts in the specifi cation of the reactive systems. An other example is the Ptolemy project [24] which studies heterogeneous modeling, simulation and design of concurrent systems. It allows also to integrate the FSM semantics with concurrently synchronous data flow models [25].

In our work we choose to study Mode-Automata which represent a signifi cant contribution in this fi eld. Based on the example given in section 2.1.3, we will try to introduce the concept of running modes in Scade models to defi ne a new design methodology in Scade which can clearly separate control and data parts.

2.2.1 Mode-Automata

Mode-Automata have been proposed in [15]. They introduce, in the domain-specifi c data-flow language Lustre for reactive systems, a new construct devoted to the expression of *running modes*. It corresponds to the fact that several defi nitions (equations) may exist for the same output, that should be used at distinct periods of time. This concept allows to decompose the specifi cation of the system into several tasks called *modes* by assigning data operations directly to discrete states.

A Mode-Automaton is an input/output automaton. It has a fi nite number of states, that are called *modes*. At each moment, it is in one (and only one) mode. It may change its mode when an event occurs. In each mode, a transfer function determines the values of output flows from the values of input flows. Mode automata can be combined in order to design hierarchical models. They generalize both bounded Petri nets and block diagrams. The structure of Mode-Automta allows to clearly specify where the modes differ and the conditions for changing modes wich makes it possible to better understand the behavior of the system.

Figure 6 represents a simple example of Mode-Automaton. It has two states, and equations attached to them. The transitions are labeled by conditions on X. The important point is that X and its memory are *global* to all states. The only thing that changes when the automaton changes states is the transition function; the memory is preserved.

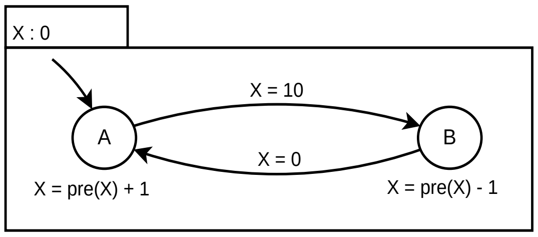


Figure 6: *Mode-Automaton: simple example.*

2.2.2 Mode-Automata based Control/Data Flow Separation in Scade

As indicated in section 2.1.3, the control/data flow combination in Scade can lead to several problems notably for readability and re-use of existing applications. In [20], the generic task pattern example, introduced in section 2.1.3, has been easily specified using multi-mode concept. For these reasons, we think that it becomes necessary to introduce a design methodology and the concept of running modes in Scade models to facilitate the specification, the verification and the re-use of various applications.

First, we have tried to apply the concept of separated Control/Data Flow by using Scade. To make this, we have studied the task example by separating control and data parts. The diagram corresponding to our approach is shown on figure 7. In this example, we have divided the

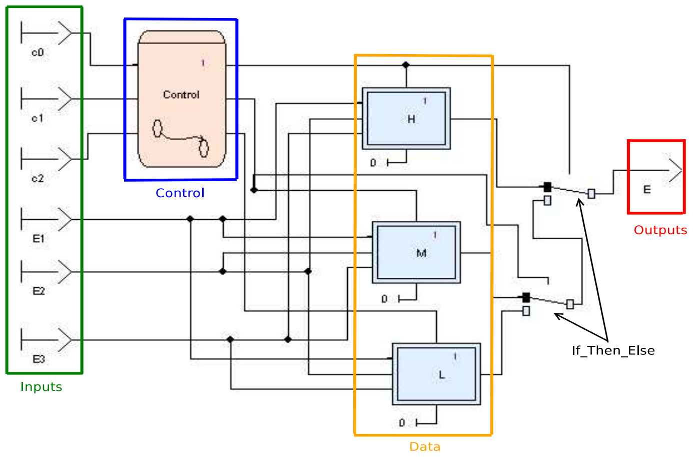


Figure 7: *Trying the separation control/data flow with Scade*

problem into three sub-problems that correspond to the functionality of the different modes of `Active` state: H, M and L. The activation of each part is made by the SSM `Control` depending on the input values of conditions c_0, c_1 and c_2.

In this approach, we can clearly distinguish inputs and outputs of the system, control parts, and data parts. Contrary to what its name indicates, the data part does not only designate an exclusive data processing. It can also contain a SSM followed by a data part, or only the control part. The lowest level in the hierarchy represents an homogeneous part that can exclusively contain the control or the elementary calculation.

The application of this approach in Scade raises some issues. For example, the value of `E` can be modifi ed by the three different calculation parts. However, in Scade it is impossible to link the same output to two different operators. In Scade, each data must have a unique defi nition at a given time, which makes the connection of the same output to several different operators impossible. This requires the introduction of the *If_Then_Else* operators, which complicate the model and break the control/data flow separation concept. To fi ll this gap, we have proposed to add special operators that play the role of `Fork` and `Join` which allow the division of data between several operators. We have also added a *selector* operator that receives as input a value provided by a SSM, according to which it can choose the state to activate (fi gure 8).

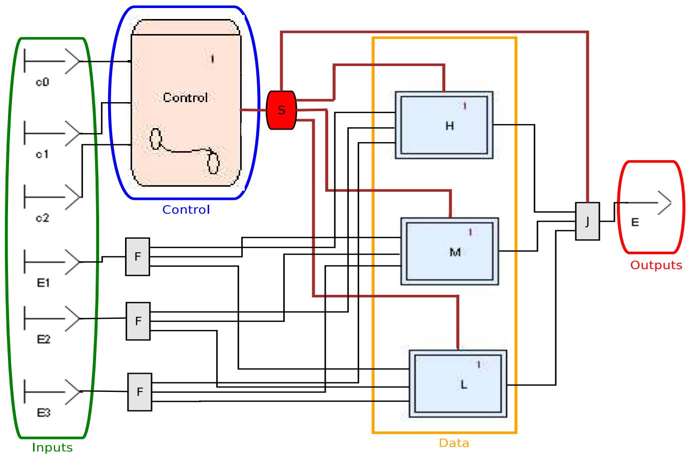


Figure 8: *Control/data flow separation model using Scade and Fork/Join operators*

The function of the *Fork* operator consists in diffusing the input value on all its output points, while the role of the *Join* operator consists in giving an output value among those received as inputs and according to the value provided by the SSM.

Selector and Fork operators represent only an optimization of notations used in Scade because, in this tool, it is possible to connect the same value to several operator's inputs. However, the Join operator replace the conditional structures *If_Then_Else* and *Switch_Case* used in Scade. In this context, one Join operator with n inputs can be used to replace a structure of $n-1$ *If_Then_Else* operators or one *Switch_Case* operator with n inputs.

In the case of the *If_Then_Else* operators, it is obvious that the complexity of the model increases according to the number of inputs which makes diffi cult the comprehension of the model. Thus, if we use the *Switch_Case* operator, calculation blocks are not conditioned and then all inputs must be computed before the operator chooses the selected one. This behavior leads to diffi cult problems and can be very expensive regarding time and memory. Moreover, the default value used in *Switch_Case* operator does not have any interest because we suppose that one and only one component must be activated at a given time[3]. For these reasons, we prefer introducing a Join operator which allows an implicite use of conditional structures and facilitates the comprehension of the model. In figure 9, we give an example of Join operator and its equivalent in Scade.

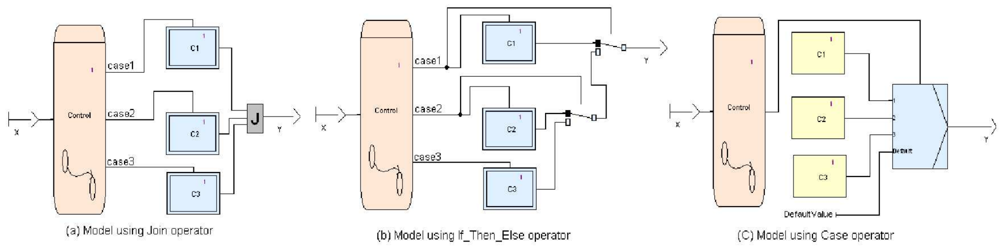

Control
case1
case2
case3
C1
C2
C3
J
X
Y
(a) Model using Join operator
(b) Model using If_Then_Else operator
DefaultValue
Default
(C) Model using Case operator

Figure 9: *Example of the Join operator and its equivalent in Scade.*

We notice that our approach of control/data flow separation is similar to that of the Mode-Automata. The idea consists in introducing the concept of running modes into Scade models to facilitate the specification of the mainly regular systems and to give a more readable design methodology based on the separation between control and calculation parts.

2.2.3 Benefits of the Control/Data Flow Separation Methodology

The introduction of a design methodology separating clearly control and data flow parts allows to have a more readable model. The different parts of this model can be studied separately by using the most appropriate existing tools for each part.

Morever, a modular specification of the different parts of the system allows to benefit from the modular development. It facilitates the re-use of existing applications, the modification, the introduction and the deletion of modes.

This technique allows to simulate and verify separately the different parts of the system, and consequently have a considerable gain in verification time and memory capacities since the number of states of the verified module is much smaller than that of the complete system. The automaton structure is also exclusive, and to each state of the automaton is associated only one activity. The different activities are then exclusive and can be studied separately. This methodology facilitates also the localisation of the different errors while avoding the modification of the whole application which can be time and resource consuming.

3 Automotive System Case Study

In the following section, we study the application and the advantages of our design methodology in the case of a real automotive system.

[3]This concept enable us to avoid the introduction of the default value.

3.1 Context

To illustrate our methodology on a real case, we propose to study the design of an Intelligent Cruise Control with GPS system (ICCG) which represents a significant automatic contribution in the automotive field.

Experts on the roads safety field agreed on the fact that one of the principal causes of the serious road accidents is the high-speed and the no respect of speed limit. For example and according to the french road security[4], each year in France, the number of road accidents exceeds the 120 000 accidents, making more than 7500 dead (7720 in 2001), and more than 40% of these accidents come from the high-speed.

The excess speed is generally due to the lack of responsibility for the drivers and precise informations on the speed limits. For these reasons and in order to give to drivers the means for controlling the speed of their cars, several constructors have developed various systems such as the speed limiter or regulator already present in some cars. In this field, research continues to give more effective systems, and recently, we often hear about speed regulator systems with GPS which allows to adapt the speed of the car to that authorized in its zone of localization.

Many European countries have launched differents projects and experiments on speed regulation systems (denominated for the majority ISA[5] or EVSC[6]). The obtained results are generally diversified and the comparisons remain nevertheless difficult since each experiment has specific objectives and protocols. Thus, technologies used and the nature of systems vary from one country to another. However, some common conclusions indicate the possible benefit of such systems to reduce the number of accidents and their dangerosity.

Among these projects, we can find the french project LAVIA[7] launched in September 2001 [26]. The LAVIA system is an intelligent regulator which automatically adapts the car speed to the speed limit according to the car position. This position is determinated using a navigation device which combines dead reckoning data with the GPS ones to assess the car position and then matches them with a digital map in order to obtain accurate car localisation coordinates.

During the period 1999-2002, the Swedish National Road Administration conducted a comprehensive road information project which included a large-scale trial involving Intelligent Speed Adaptation (ISA)[8] in urban areas [29]. The aim of the trial was to learn more about drivers attitudes and how they use the system, the impact on road safety and the environment, and the integration of the system on the cars.

Another European project, PROSPER[9], has been launched in 2002 [27]. The PROSPER project was developed referring to the council resolution of June 2000. It explicitly identifies that advanced assisted driving technology and technology relating to speed limitation devices can be an important measures for further investigation. The main project output will be the assessment of cost benefit and cost effectiveness of ISA road speed management methods in relation to traditional methods, and a thorough analysis of possible and suitable implementation strategies for different road speed management methods. The results are expected to have a considerable impact on the development and implementation of national and European road transport safety policies, particularly in the short term.

[4]www.securite-routiere.org

[5]Intelligent Speed Adaptation

[6]External Vehicle Speed Control

[7]www.lavia.fr

[8]www.vv.se/isa

[9]www.prosper-eu.nl

Between October 2002 and December 2003, an ISA trial took place in the city of Ghent, Belgium [28]. ISA-Ghent project progressed in parallel with PROSPER project but it is not a part of it in spite of their common objectives. The tested ISA device is the Limit Advisor M2002 developed by the Swedish companies.

Another similar project was developed in Denmark. It is the INFATI project[10], a Danish acronym for "INtelligent FArtTIlpasning" that means "Intelligent Speed Adaptation (ISA)" [30]. This project is mainly bound to the study of the techniques and the means for the realization of the system, system specifi cation, prototype development, tests and informations.

EWGOSC, a European Work Group On Speed Control[11] groups researchers implied in ISA experiments annually. This work group allows researchers to compare their results and experimental protocols to be able to improve their future work.

3.2 Optimization Technique for the Representation of the Roadmap

The presented projects are all based on a GPS system to locate the cars on the roads. They also use a data base representing the speed limits for each zone on the map. However, the imprecision of the cartography and the possible errors of the position given by the GPS make it diffi cult to localize the car on the map, and to fi nd the corresponding speed limit.

To fi ll this gap, several concepts have been proposed using the GPS system to give more precision on the localization of the car. The LAVIA project for example, combines the GPS system with a navigation system to calculate the distances and the course of the car. This approach consists in representing the roads by segments of straight lines. Each segment is defi ned by two points and a speed limit as shown by fi gure10. This needs to transform the

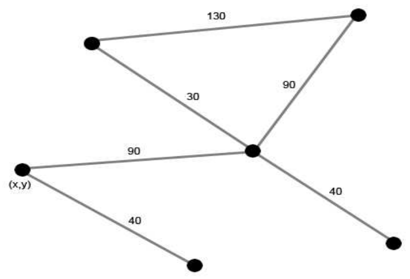


Figure 10: *Road graph with segments*

roadmap into a graph in which the arcs represent the segments and the nodes represent the ends of each segment. Each arc is labelled by a cost representing the speed limit authorized in its zone.

For technical reasons, the segments should not exceed 500 meters. This gives in the majority of the cases a rather signifi cant number of segments which can generate a problem of memorization.

Another technique, used by ISA-Ghent project for example, consists in dividing the roadmap into a set of zones and associating each zone with its speed limit. This approach combines the digital map with polygonal structures called "delimitation boxes" to minimize the number of

[10]www.infati.dk
[11]The ICTCT-extra workshop of 2002 in Nagoya (www.ictct.org)

segments (figure 11). The form of these zones is inevitably not regular which imposes the di-

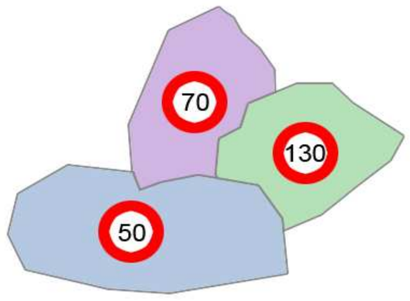


Figure 11: *Definition of different zones*

vision of the roadmap in a set of small identical squares called "grids". Each grid is defined by two points and its speed limit, and the zone is defined by a set of adjacent grids as shown by figure 12.

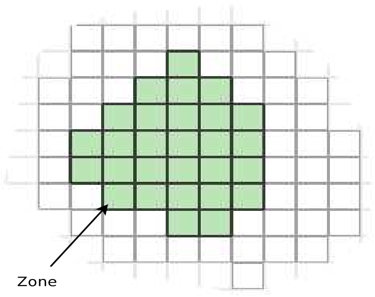


Figure 12: *Definition of the zones through a set of grids*

In order to decrease the number of grids associated to each zone, we propose an optimization algorithm allowing to join the adjacent grids with the same speed limit in order to create a large rectangle. In this case, each zone is represented by a set of rectangles defined by two points[12] and the speed limit as indicated by figure 13.

The roadmap is represented by a set of lines of the form:

$$\underbrace{(X_1, Y_1)}_{point1} \mid \underbrace{V}_{speed\ limit} \mid \underbrace{(X_2, Y_2)}_{point2}$$

At each instant, the GPS gives the position of the car $P(X, Y)$. If $X_1 \leq X \leq X_2$ and $Y_1 \leq Y \leq Y_2$, then P belongs to the rectangle defined by the two points (X_1, Y_1) and (X_2, Y_2), and the car is in the zone limited at the speed V.

This representation technique of the roadmap makes it possible to minimize, in a considerable way, the size of the data base, and consequently the memory capacity. In this case, it becomes optional to know details on the roads and it is not necessary to install the cartography in the car, which has generally a high cost. Furthermore, error risks due to the imprecision of GPS decrease considerably.

[12] Each point is defined by its two coordinate (longitude, latitude)

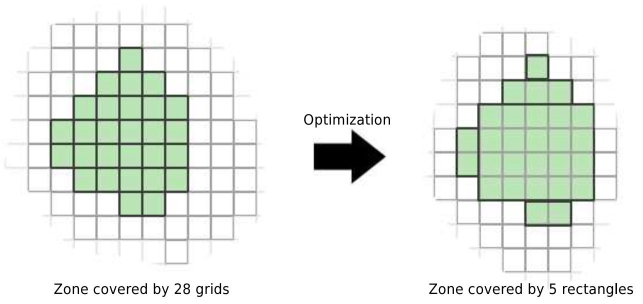


Figure 13: *Optimization techniquefor the representation of the roadmap*

3.3 ICCG System Description

In this section, we propose to study an ISA system that we call ICCG (Intelligent Cruise Control with GPS). This system uses the optimisation technique presented in section 3.2 for the representation of the roadmap. The main goal of our work consists in applying a new design methodology, separating control and data parts, for this type of system. It allows to study the interest of this technique, in particular for the verifi cation of the system.

The functioning of this system is similar to that of ISA. Its main role consists in limiting the car speed automatically to the local prescribed speed given by the driver or by the GPS.

The ICCG can be seen as an electronic help which facilitates the control of a car. It informs the driver about the various changes of speed limits and, in some cases, obliges him to respect them. The study of such a system can be very important since it makes possible to considerably increase the safety of drivers.

The studied system can operate in fi ve different modes: `Alarm`, `Limit`, `Cruise`, `LimitGPS` and `CruiseGPS`. The interaction with the system and the activation or deactivation of different modes are done through a set of buttons:

- `On`: activate the system and set the speed limit to the current speed of the car
- `Off`: stop the system
- `Set`: set the speed limit to the current speed of the car
- `Resume`: reactivate the system after an interruption
- `QuickAccel(+)`: increase the speed limit by a constant `SpeedInc`
- `QuickDecel(-)`: decrease the speed limit by a constant `SpeedInc`
- `GPS`: activate or deactivate the GPS
- `Cruise`: switch the system to `Cruise` mode or return it to `Limit` mode

Informations about the running modes can be displayed on a dashboard to inform the driver about which mode is activated. It is also possible to display the speed limit and the current speed of the car as shown by fi gure14.

Initially, the system is in `Alarm` mode. The activation of the system and the switch to `Limit` mode is done using the `On` button. A more detailed description of the switch between the various modes is given by fi gure15.

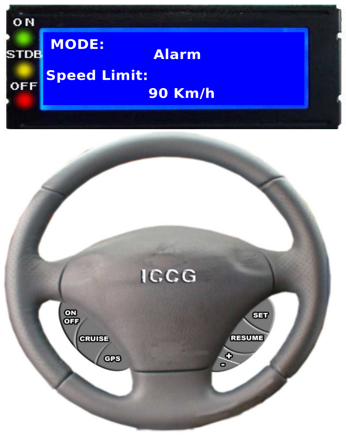


Figure 14: *System representation*

Globally and independently of running modes, the system can be in one of the four states:

- `SysOn`: the system is activated
- `SysOff`: the system is deactivated
- `SysSTDB`: the system is suspended
- `GPSFail`: the GPS signal is lost

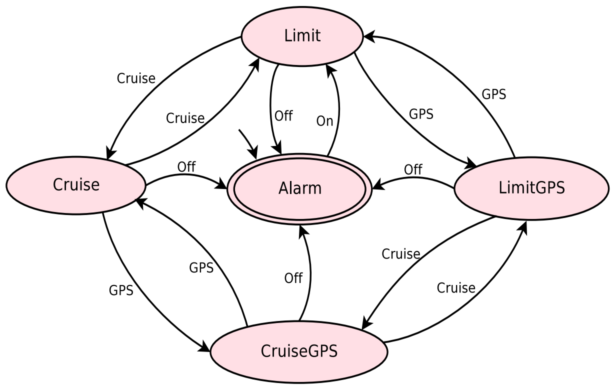


Figure 15: *Different modes of the system*

The switch between states of the system is done according to the brake pedal, accelerator pedal and GPS signal (figure 16). Thus, in this system and for safety reasons, we define an interval speed (`Between=SpeedMax-SpeedMin`) at which the system can be activated. Outside this interval, the system is systematically desctivated and switches to `SysSTDB` state. For example, if the current speed exceeds 160km/h or it is lower than 30km/h, we consider that the

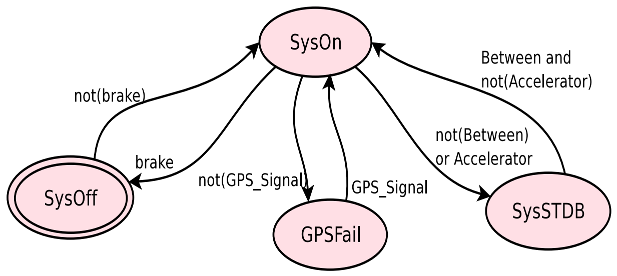


Figure 16: *Different states of the system*

intervention of the system does not have importance and the driver has total control over the car.

3.3.1 `Alarm` Mode

`Alarm` mode is only an informative mode. Its function only consists in indicating the driver, by an audio or luminous signal, if the allowed speed limit is exceeded. This speed is given by a GPS and represents the speed limit authorized in the current zone of the car.

The overspeed alarm does not influence the real speed of the car. The driver has the whole control over the car and the alarm signal represent only a warning signal. In this mode, pedals act normally on the behavior of the car and do not have any influence on the state of the system. However, if the GPS signal is lost, the `Alarm` mode is deactivated, an audio signal is sent and the system passes in `GPSFail` state.

3.3.2 `Limit` Mode

This mode is the most used in marketed cars. In this mode, the GPS is deactivated and the system does not allow to exceed the speed limit fi xed by the driver who can always control the car by accelerating or braking, but cannot exceed the limited speed.

The main functioning of this mode, is that the car cannot exceed the pre-selected speed limit since the accelerator pedal becomes systematically inactive when the driver reaches this limit. However, by necessity (taking over another car for example), the driver can exceed the fi xed speed limit and the system is switched off. This is known as the ”kick-down“ phenomenon which can disable the system temporarily. To do that, we introduce a constant value `MinAccel` beyond which the accelerator pedal is taken into account. In this case, the system becomes inactive, it goes to `SysSTDB` state and becomes active again (`SysOn`) when this pedal is released.

3.3.3 `Cruise` Mode

This mode allows to maintain the car at a constant speed given by the driver. The system forces the car to reach the speed limit and maintain it. In this case, the driver does not control the speed of the car via the accelerator pedal but rather by using a set of buttons.

The `On` button allows the selection of the speed limit to respect, the two buttons `QuickAccel` and `QuickDecel` allow, respectively, to increase or decrease this speed by a constant value `SpeedInc` and the `Set` button assigns the current speed of the car to the speed limit.

If the driver presses the accelerator pedal, the functioning of the system is interrupted and the system becomes inactive (`SysSTDB` state). The system can take again its functioning and pass to `SysOn` state only if the accelerator pedal is released. Thus, pressing the brake pedal completely deactivates the system which goes in `SysOff` state. In this case, the system can be reactivated only through the `Resume` button.

3.3.4 `LimitGPS` mode

The behavior of this mode is similar to that of `Limit` mode. In `LimitGPS` mode, the car cannot exceed a speed limit defi ned by the system. This limit depends on the speed required by the driver, the maximum speed authorized in the current zone and the maximum speed authorized in the next zone. In this case, the speed limit always takes the minimum value of these three speeds to ensure the safety of driver.

3.3.5 `CruiseGPS` mode

This mode has the same behavior as the `Cruise` mode. The only difference is that the speed limit is calculated in the same way as in `LimitGPS` mode since we take the GPS into account. If the GPS signal is lost, the system is automatically stopped and goes in `GPSFail` state.

For safety reasons, in addition to the current position of the car, we take into account another position called `NextArea`. This position represents an anticipation of the next position of the car ($\sim$ 3 or 4 seconds in advance). According to the current position of the car, its speed and its direction, we calculate the anticipation point `NextArea` which allows to deduce the speed limit of the following zone `NextAreaSpeedLimit`. This safety measure is only useful if `NextAreaSpeedLimit`$<$`AreaSpeedLimit`. For example, if the car is on a road limited to 130km/h, and it moves towards an exit road limited to 90km/h, its speed must go down to 90 before being on the exit road.

3.4 System Design without Control/Data Flow Separation Methodology

The specifi cation of the ICCG system consists of two essential parts. The fi rst part (`ICCG`) represents the behavior of the cruise control with GPS, and the second part (`CarSimple`) specifi es the behavior of the car with which our system will interact.

The ICCG system takes as input a set of values representing the differents buttons (`On`, `Off`,...), the pedals of the car, the speed limit of the current zone and that of the following zone. As output, the system provides an information on its state which can be active (`SysOn`), inactive (`SysOff`) or suspended (`SysSTDB`). It also provides an information about the selected mode, the fi xed speed limit (`SpeedLimit`), the speed requested by the driver (`DriverSpeed`), the current speed of the car (`CurrentSpeed`), an alarm signal (`AlarmSignal`) to indicate that the speed limit was exceeded, the `StopAccelerator` signal to block the accelerator pedal, and the `GPS_Fail` signal indicating the loss of the GPS signal or the nonidentifi cation of the zone. A global view of our system is given by fi gure17. The main functioning of the system allows, from command buttons, to specify running modes, to save the speed limit requested by the driver, and to act on the speed of the car according to the selected mode.

A possible solution for the specifi cation of the ICCG system has been proposed in [31]. The corresponding design model is given by fi gure 18. This model was achieved in a very intuitive way since the main goal was just to develop a functional model without following any control/data flow separation methodology.

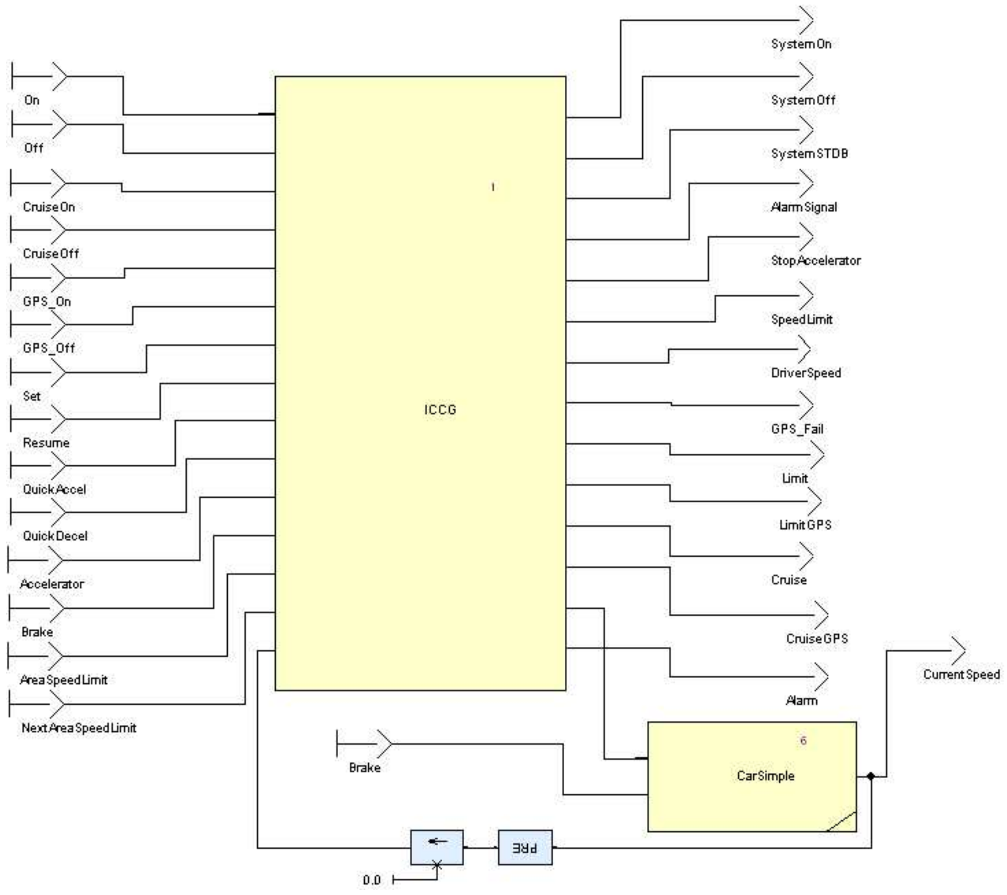


Figure 17: *ICCG system: global view*

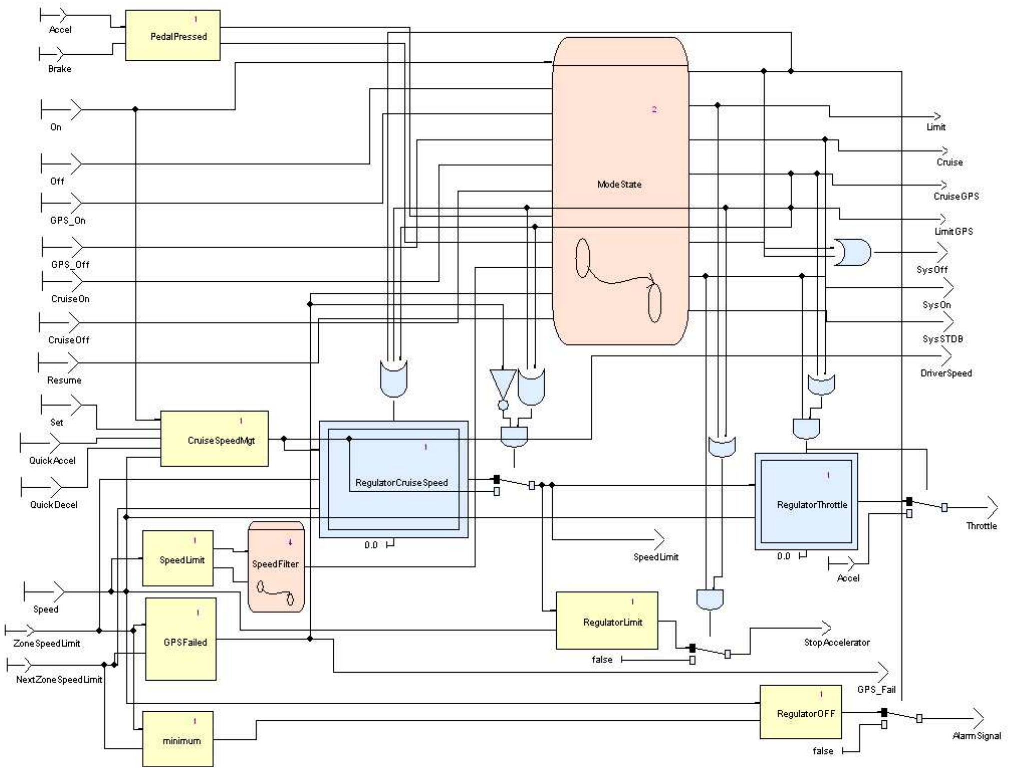


Figure 18: *Internal structure of ICCG model without following any control/data flow separation methodology*

3.5 System Design using Control/Data Flow Separation Methodology

As shown by figure18, the model contains a mixture of control and data processing. It does not allow to clearly distinguish the various modes of the system and the switch conditions between modes. This model is very ambiguous since a small modification in the behaviour of a given mode requires the modification of the whole application. Indeed, it is difficult to extract the mode from the total specification. This is also valid for the introduction of a new mode or the deletion of an existing one. Thus, the application of formal verification techniques on such models is very difficult and even impossible. Errors are more and more serious and the resulting system will be unstable

For these reasons and to test our design methodology, we have modified the specification by adopting our methodology which allows a good separation between control and data flow parts, and makes it possible to clearly distinguish the different modes of the system. The specification diagram relating to ICCG system, following this methodology, is given by figure19.

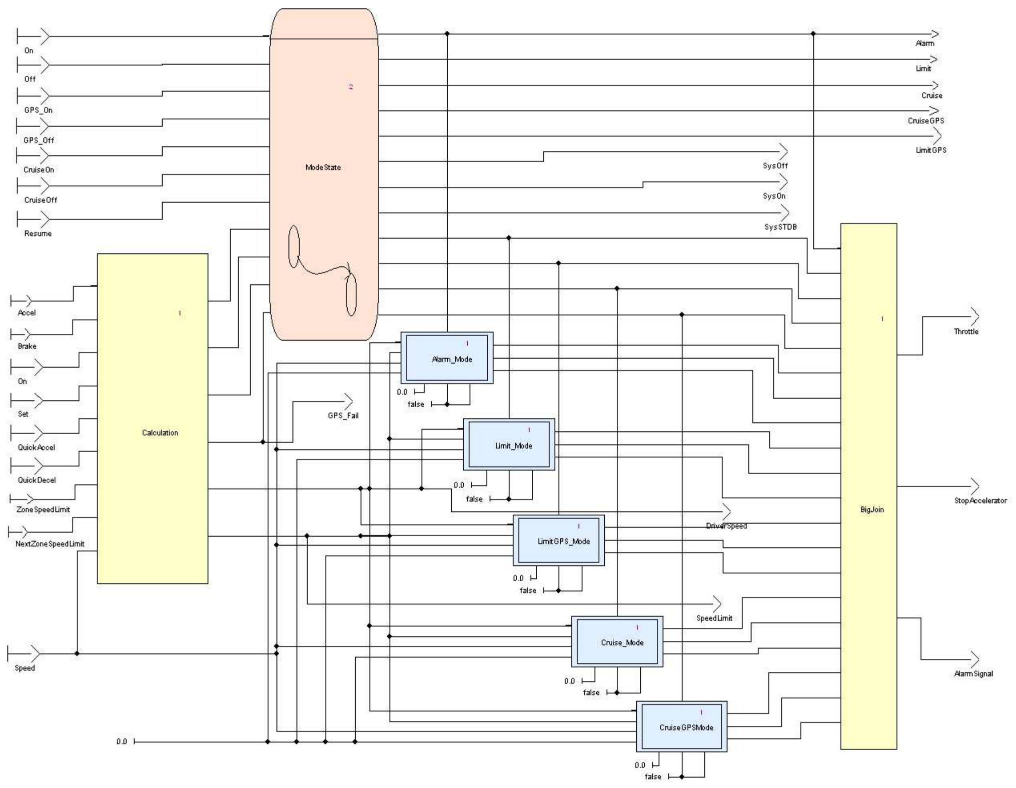


Figure 19: *Internal structure of ICCG model according to control/data flow separation methodology*

This model is composed of three main parts. The first part is a *pre-calculation* part represented by `Calculation` model which contain the common processing for the different modes. This calculation part is always executed independently of the selected mode. It gives as output a set of values which can be used by the control part, the different execution modes, or be directly displayed on the dash-board. The second part is the *control* part represented by the

SSM ModeState. This part allows to select the mode to be activated according to the value of input buttons. The third part is a *mode-calculation* part. It is composed of five calculation parts relating to the different modes of the system. The execution modes have always the same interface which can facilitate the introduction, the deletion and the modification of modes.

4 Experimentation of the System

4.1 Simulation of ICCG System

In this section, we present a simulation example for the different running modes of the ICCG system. For simplification reasons, we divide our study into two parts: the first part studies the simulation of the Alarm, Limit and LimitGps modes, and the second part studies the simulation of the Alarm, Cruise and CruiseGPS modes.

Figure 20 represents the simulation example of the three modes: Alarm, Limit and LimitGps. In this figure, we present six simulation steps:

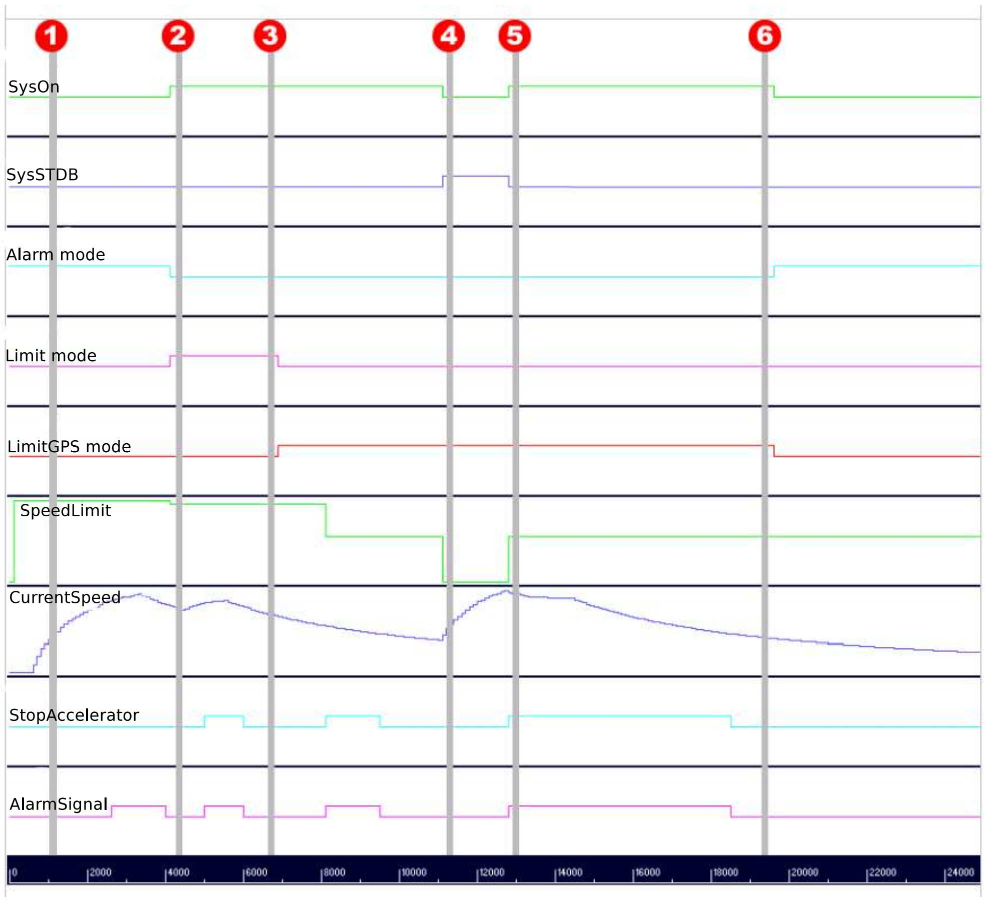


Figure 20: *Simulation example of modes: Alarm, Limit and LimitGPS*

1. The `Alarm` mode is activated and the `SpeedLimit` is fi xed to the minimum value of current speed and next area speed. When `CurrentSpeed` exceeds the `SpeedLimit`, the `Alarmsignal` is activated until the `CurrentSpeed` becomes less than the `SpeedLimit`.
2. The `On` button is pressed and the `Limit` mode is activated. In this case, the `CurrentSpeed` is equal to `SpeedLimit` and if the driver wants to exceed this limit, the accelerator is stopped and the alarm signal is sent.
3. The `GPS` button is pressed and the `LimitGPS` mode is activated. In this mode, the `SpeedLimit` takes the minimum value between the speed given by the GPS, the current speed and the next area speed. Also in this case, if the `CurrentSpeed` exceeds the `SpeedLimit` then the accelerator is stopped and the alarm signal is sent.
4. The accelerator pedal is strongly pressed (kick-down phenomenon) and the system goes to the `SysSTDB` state.
5. The accelerator pedal is released and the ICCG system is reactivated. The `SpeedLimit` takes its last value before pressing the accelerator pedal, the `StopAccelerator` and the `AlarmSignal` are activated because the `CurrentSpeed` exceeds the `SpeedLimit`.
6. The `Off` button is pressed, the ICCG system is deactivated and the `Alarm` mode is activated.

Similarly, fi gure21 represents the simulation example of the three modes: `Alarm`, `Cruise` and `CruiseGPS`. In this fi gure, we also present six simulation steps:

1. The `On` button is pressed and the ICCG system is activated. In this case, the accelerator pedal is strongly pressed (kick-down phenomenon), the `Alarm` mode is deactivated and the system goes to the `SysSTDB` state.
2. The accelerator pedal is released and the system goes to the `SysOn` state (the `Limit` mode is activated).
3. The `Cruise` button is pressed, the `Cruise` mode is activated and the `CurrentSpeed` reaches the `SpeedLimit`
4. The `GPS` button is pressed and the system goes to `CruiseGPS` mode. In this case, the value of `SpeedLimit` changes since the speed limit of the zone is lower than the current `SpeedLimit`.
5. The accelerator pedal is pressed (kick-down phenomenon) and the system goes to `SysSTDB` state. When the pedal is released, the system is reactivated and `SpeedLimit` takes its last value.
6. The brake pedal is pressed and the system is stopped (`SysOff` state). In this case, only the `Resume` button allows to reactivate the system.

4.2 Formal Verification of the ICCG System

The goal of this study consists in verifying some properties of our ICCG system. This verifi cation process is applied for the two design models, without and with separation methodology, to

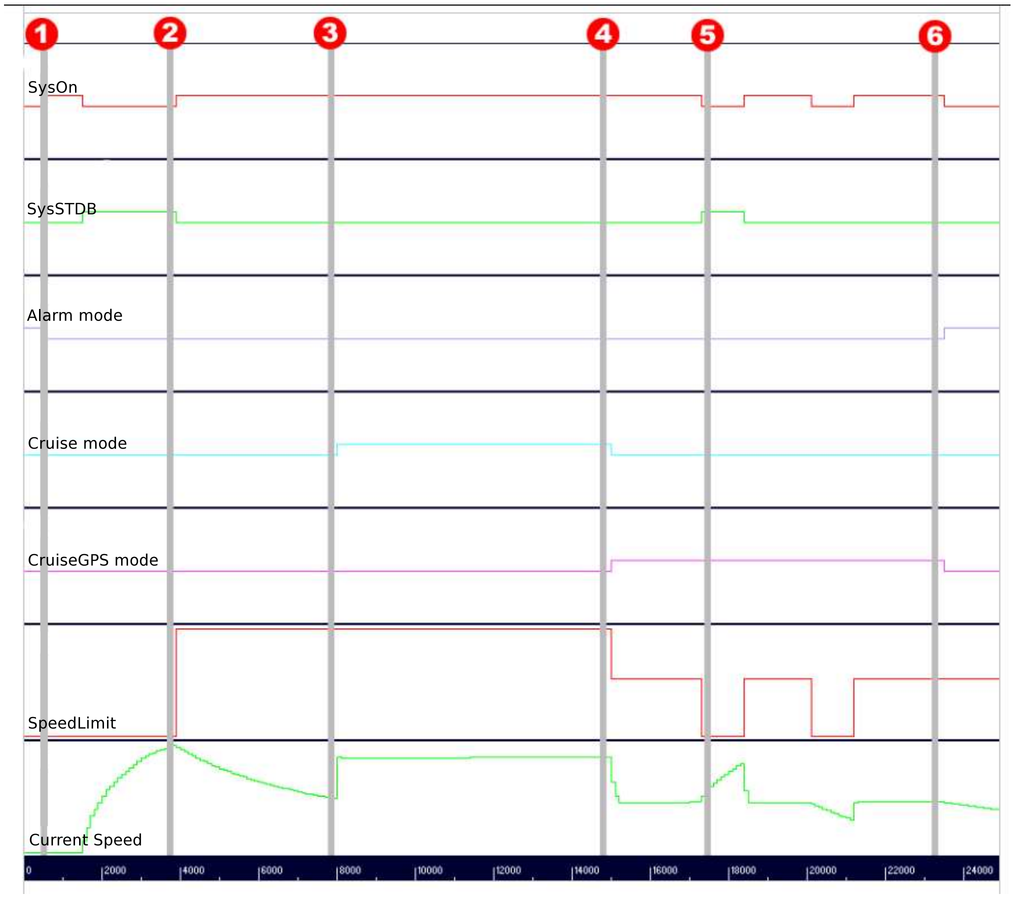


Figure 21: *Simulation example of modes:* $\mathtt{Alarm}$, $\mathtt{Cruise}$ *and* $\mathtt{CruiseGPS}$

compare the obtained results by each model. In the following, we will note the first specification model, which does not follow a clear separation design methodology, by `Model1`, and the second model, which follows our design methodology and clearly separates control and data parts, by `Model2`. The set of properties that we choose to verify is as follows:

- *Property1*: at each moment, one and only one running mode is activated at once
- *Property2*: if the `Alarm` mode is activated then the `AlarmSignal` is *True* if and only if `CurrentSpeed`>`SpeedLimit`
- *Property3*: if the `Alarm` mode is activated then the system does not have any effect on the accelerator (`StopAccelerator`=*False*)
- *Property4*: if the `Limit` mode is activated and `CurrentSpeed`>`SpeedLimit` then the accelerator is stopped (`StopAccelerator`=*True*)
- *Property5*: if `LimitGPS` or `CruiseGPS` modes are activated, the speed limit value is that of the minimum of `DriverSpeed`, `AreaSpeedLimit` and `NextAreaSpeedLimit`

As indicated in section 3.4, `Model1` mixes control and calculation parts which can complicate the understanding of the application and the distinction of the different running modes of the system. In this model, if we want to verify some properties of a given mode, we must verify all the system since the distinction of the concerned mode is very difficult and even impossible. Contrary to this model, `Model2` allows a clear separation between control and calculation parts and therefore a good distinction of the different running modes of the system. In this model, if we want to verify some properties of a given mode, only the concerned mode will be verified and the property formula is less complicated than that used for the verification of the same property in `Model1`. For example, if we want to verify *Property2* for `Model1`, we must verify that the system is in `Alarm` mode. However, the verification of this same property for `Model2` does not require the verification of the mode since we know that the verified mode is `Alarm` mode. This separation methodology also allows to easily detect and locate the specification errors, and avoids remaking all the specification in the case of errors.

We give in table 1 a comparison of verification times obtained for the different properties of each model. These results are obtained by using the verification tool of Scade. This tool is used to exhaustively verify complex functional properties, detect hard-to-find corner-case bugs, and explore complex design behaviors. The Scade formal verification module is called *Design Verifier*[13] and it is a *model-checker* based on a powerful proof engine developed by Prover Technology AB[14]. The property to verify is generally described in a separated node which must be linked to the system without modifying it and act as an *observer* of the system.

Model-checker procedures are useful tools for the verification of finite-state systems. The first model-checker algorithm has been developed by Clarke and Emerson in 1981 [33]. It consists in confronting an automaton structure, called *model*, with a temporal logic formula, called *property*. The automaton represents the behavior of the studied system, while the formula makes the expression of the property to verify for this system possible.

In our study, table 1 shows that the separated verification of the different modules gives the same result if we verify all the system. Moreover, the verification time of separated modules is much faster since the number of states of the checked automaton is much smaller. This difference in time relates also to the fact that the formulas of the verified properties are simpler

[13]www.esterel-technologies.com/products/scade-drive/design-verifier.html
[14]www.prover.com

Property	*Verified module for `Model1`*	*Verified module for `Model2`*	**Valid**	**Time** `Model1`	**Time** `Model2`	**Speed up**
Property1	Whole system	SSM `ModeState`	$\surd$	0.460*s*	0.080*s*	**5.75**
Property2	Whole system	`Alarm` mode	$\surd$	0.430*s*	0.030*s*	**14.33**
Property3	Whole system	`Alarm` mode	$\surd$	3.424*s*	0.010*s*	**342.40**
Property4	Whole system	`Limit` mode	$\surd$	0.440*s*	0.040*s*	**11.00**
Property5	Whole system	`LimitGPS` mode	$\surd$	8.702*s*	0.080*s*	**108.77**
		`CruiseGPS` mode			0.090*s*	**96.68**

Table 1: *Verification times of different properties*

in the case of modular verifi cation. The obtained result is interesting since it makes it possible to separate the specifi cation of the system into several modules according to a given methodology, and to verify these various modules separately. This verifi cation technique, that we call *modular verification*, makes it possible to well locate the errors, to gain time, to gain memory space and to reduce state explosion problems.

4.3 Prototype

To test the ICCG system in a real situation, we have developed a simulation prototype using Visual Studio .Net and the C# language[15].

The fi rst application of our prototype is a simulation map allowing to defi ne a roadmap and to create the data base relating to this map. Generally, this application consists in representing graphically on a roadmap the speed limit of the different zones as we have shown by fi gure 22. This application allows also to apply the optimization algorithm introduced in section 3.2, and to generate the data base from the obtained result (fi gure 23). For safety reasons, the critical points of the roadmap such as the exits of motorways can be treated in a more precise way. We can zoom on these zones to increase the accuracy and then apply the same optimization process.

The second application is developped to simulate the fonctionning of the ICCG system in a real conditions. To do that, we have connected our system to a GPS of type Garmin[16]. The GPS uses the communication protocol NMEA (National Marine Electronic Association) [32] to send informations about the car position each one second to the ICCG prototype. In this case, the prototype computes, in real time, the speed of the car, its direction and its current position using geographical data (latitude and longitude). According to these informations and to the data base of the roadmap generated by the simulator, the prototype locates the current and the next position of the car and then the speed limits to be respected.

[15]www.hitmill.com/programming/dotNET/csharp.html
[16]www.garmin.com

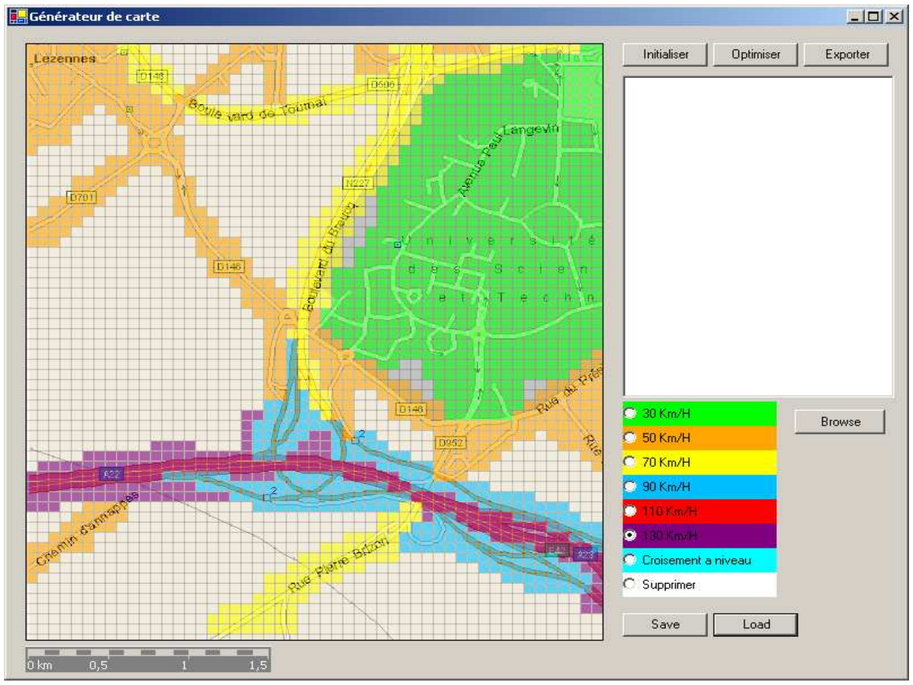
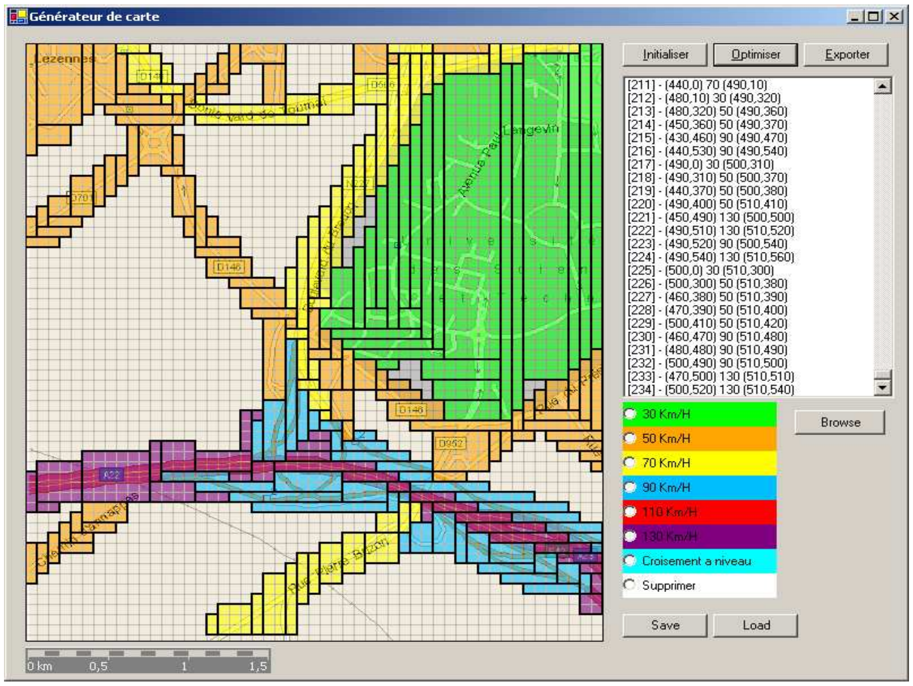


Figure 22: *Application of the simulation map*



```
[211] - (440,0) 70 (490,10)
[212] - (480,10) 30 (490,320)
[213] - (480,320) 50 (490,360)
[214] - (450,360) 50 (490,370)
[215] - (430,460) 90 (490,470)
[216] - (440,530) 90 (490,540)
[217] - (490,0) 30 (500,310)
[218] - (490,310) 50 (500,370)
[219] - (440,370) 50 (500,380)
[220] - (490,400) 50 (510,410)
[221] - (450,490) 130 (500,500)
[222] - (490,510) 130 (510,520)
[223] - (490,520) 90 (500,540)
[224] - (490,540) 130 (510,560)
[225] - (500,0) 30 (510,300)
[226] - (500,300) 50 (510,380)
[227] - (460,380) 50 (510,390)
[228] - (470,390) 50 (510,400)
[229] - (500,410) 50 (510,420)
[230] - (460,470) 90 (510,480)
[231] - (480,480) 90 (510,490)
[232] - (500,490) 90 (510,500)
[233] - (470,500) 130 (510,510)
[234] - (500,520) 130 (510,540)
```

Figure 23: *Optimization and data base generation of the roadmap*

4.4 Field Test

A real test of our application was performed in Villeneuve d'Ascq[17] using a laptop and a Garmin GPS (figure 24). It represents a simple test of the system since the application does not react

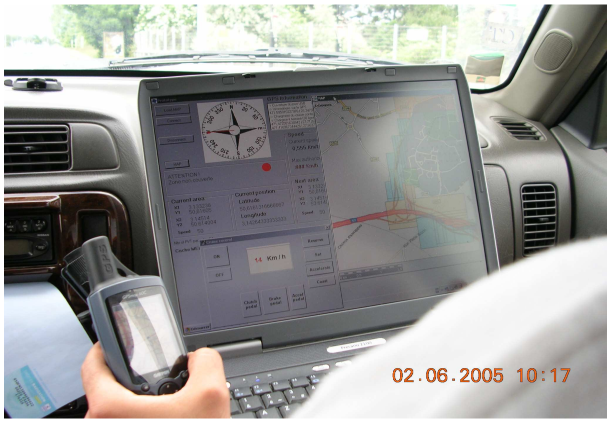


Figure 24: *Real test*

directly on the speed of the car. In our case, the ICCG system informs only the driver by an alarm signal if the speed limit is exceeded.

To display the different informations on the state of the system and the speed changes, we have proposed a graphical interface giving the roadmap, the car position, the next car position, the car speed, the speed limit,. . . . We have also proposed an interface for the different buttons of the ICCG System as shown by figure 24.

In this test and to be much more interested by the speed limits of the roads, we have fixed the speed requested by the driver at 140km/h. In this case, the authorized speed limit will be always that of the current zone of the car. This test gave satisfactory results and showed that the detection of the different changes of zones is precise. For example, the passage from a 90km/h zone to a 70km/h zone was very quickly announced by the system and the alarm signal was activated until the speed of the car went below 70Km/h.

5 Conclusion

In this paper we have studied the application of a control/data flow separation methodology in the case of an automotive system. First, we have shown that the specification environment Scade does not follow a clear separation design methodology and leads to a mixture of control and data flow representation. This mixture can make difficult the understanding of the system and the re-use of existing applications. For these reasons, we have proposed a design methodology allowing to separate clearly control and data parts, and consequently give a good modular development and an easy re-use of the different components of the system.

To illustrate the advantages of our methodology, we have studied its application on a real automotive system example (ICCG). This study has shown that the obtained specification model is more readable, it is easier to maintain and to re-use. Also, this model allows a modular formal

[17] 59650, France

verifi cation since the differents modules of the system are easily locatable. This verifi cation technique makes it possible to gain time and memory capacities and allows to detect more easily errors in the system.

Acknowledgment

The authors would like to thank Ahmed Jerbi and Yousri Miled who developed the preliminary specifi cation and the on road prototype of the ICCG system.

References

[1] David Harel and Amir Pnueli, *On the development of reactive systems*, In K. R. Apt, editor, Logics and Models of Concurrent Systems, **13**, NATO ASI Series,springer-Verlag, pages 477-498, New York, 1985

[2] L. Zaffalon and P. Breguet, *Conception de Systèmes Réactifs*. Revue Scientifi que de l'EIVD, 2001

[3] N. Halbwachs, *Synchronous programming of reactive systems*. Kluwer Academic Publishers, ISBN: 0792393112, 1993

[4] G. Berry and A. Benveniste, *The synchronous approach to reactive and real-time systems*. Proceedings of the IEEE, **79(9)**, pages 1270-1282, September, 1991

[5] P. Caspi, D. Pilaud, N. Halbwachs and J. A. Plaice, *Lustre, a declarative language for real time programming*. Proceedings of the 14th ACM SIGACT-SIGPLAN symposium on Principles of Programming Languages, pages 178-188, Munich, West Germany, 1987

[6] N. Halbwachs, P. Caspi, P. Raymond and D. Pilaud, *The synchronous data-flow programming language LUSTRE*. Proceedings of the IEEE, **79(9)**, pages 1305-1320, September, 1991

[7] Albert Benveniste, Patricia Bournai, Thierry Gautier and Paul Le Guernic, *SIGNAL: a Data Flow Oriented Language for Signal Processing*. INRIA Technical Repport, RR-0378, Centre de Rennes IRISA, March, 1985

[8] P. Le Guernic and T. Gautier and M.Le Borgne and C. Le Maire, *Programming Real-Time applications with SIGNAL*, Another Look at Real-Time Programming. Proceedings of the IEEE. **79(9)**, pages 1321-1336, September, 1991

[9] Frédéric Boussinot and Robert De Simone, *The Esterel Language*. Another Look at Real-Time Programming. Proceedings of the IEEE. **79(9)**, pages 1293-1304, September, 1991

[10] Gerard Berry and Georges Gonthier, *The Esterel Synchronous Programming Language: Design, Semantics, Implementation*, Science of Computer Programming, **19(2)**, pages 87-152, November, 1992

[11] Gérard Berry, *The Foundations of Esterel*. Proofs, Languages, and Interaction, Essays in Honour of Robin Milner. MIT Press, 2000

[12] F. Maraninchi and Y. Rémond, *Argos: an Automaton-Based Synchronous Language*. Computer Languages. Elsevier. **27**, pages 61-92, April, 2001

[13] Ouassila Labbani, Jean-Luc Dekeyser and Pierre Boulet, *Mode-automata based methodology for Scade*, In Springer, Hybrid Systems: Computation and Control, 8th International Workshop, LNCS series, pages 386-401, Zurich, Switzerland, March 2005

[14] Esterel Technologies, *SCADE Language Reference Manual*, 2004

[15] Florence Maraninchi and Yann Rémond, *Mode-automata: About modes and states for reactive systems*, European Symposium On Programming, Springer verlag, LNCS 1381, Lisbon, Portugal, March, 1998

[16] M. Jourdan and F. Lagnier and F. Maraninchi and P. Raymond, *A multiparadigm language for reactive systems*. IEEE International Conference on Computer Languages (ICCL). Toulouse, France, 1994

[17] Nicolas Pernet and Yves Sorel, *Optimized Implementation of Distributed Real-Time Embedded Systems Mixing Control and Data Processing*. International Conference: Computer Applications in Industry and Engineering, Las Vegas, USA, November, 2003

[18] Esterel Technologies, *Efficient Development of Airborne Software with SCADE Suite*^{TM}, 2003
url: www.esterel-technologies.com/technology/WhitePapers/overview.html

[19] Charle Andrés, *Representation and Analysis of Reactive Behaviors: A Synchronous Approach*. Computational Engineering in Systems Applications (CESA). IEEE-SMC, pages 19-29, Lille, France, July, 1996

[20] Hervé Marchand and Eric Rutten, *Managing multi-mode tasks with time cost and quality levels using optimal discrete control synthesis*. In Proceedings of the 14th Euromicro Conference on Real-Time Systems, ECRTS'02, Vienna, Austria, June, 2002

[21] Florence Maraninchi and Yann Rémond, *Applying Formal Methods to Industrial Cases: The Language Approach (The Production-Cell and Mode-Automata)*. Proc. 5th International Workshop on Formal Methods for Industrial Critical Systems, Berlin, April, 2000

[22] Paul Caspi and Marc Pouzet, *Lucid Synchrone, a functional extension of Lustre*. Technical Report, Université Pierre et Marie Curie, Laboratoire LIP6, 2000

[23] Xavier Leroy, *The Objective Caml System Release 3.0.8: Documentation and user's manual*. Institut National de Recherche en Informatique et en Automatique, July, 2004
url: caml.inria.fr/pub/docs/manual-ocaml/

[24] Edward A. Lee, *Overview of the Ptolemy Project*. Technical Memorandum UCB/ERL M03/25, University of California, Berkeley, July, 2003
url: ptolemy.eecs.berkeley.edu/publications/

[25] A. Girault, B. Lee, and E. A. Lee, *Hierarchical Finite State Machines with Multiple Concurrency Models*. IEEE Transactions On Computer-aided Design Of Integrated Circuits And Systems, **18(6)**, June, 1999

[26] Michel Marchi, Jacques Ehrlich and Laurent Salesse, *LAVIA: the French ISA project, main issues and first results on technical tests*, Proceedings of the 10th ITS Congress, Madrid, Spain, 2003

[27] Veerle Beyst, *PROSPER: Project for Research On Speed adaptation Policies on European Roads, Final Report on Stakeholder Analysis*. Technical Report, version 1.4, 2004

[28] Jean-Manuel Page, *A final technical report on the Belgian Intelligent Speed Adaptation (ISA) trial*. Technical Report, Project and research engineer, Belgian Institute for Road Safety, 2004

[29] Torbjörn Biding and Vägverket, *Intelligent Speed Adaptation (ISA), Results of large-scale trials in Borlänge, Lidköping, Lund and Umeå during the period 1999-2002*. Technical Report, 2002

[30] C. S. Jensen, H. Lahrmann, S. Pakalnis and J. Runge, *The INFATI Data*. TimeCenter Technical Report, 2004

[31] Ahmed Jerbi and Yousri Miled, *Limiteur-Régulateur de Vitesse Intelligent (LRVI)*, Projet de Fin d'Études d'Ingénieur en Informatique, Faculté des Sciences de Tunis, Organisme d'accueil: Laboratoire d'Informatique Fondamentale de Lille, Juin, 2005

[32] David R. Morse, Henrik S. Gedenryd, Simon Holland, *A Simple, Technology-neutral Lingua Franca for Location Systems, Applied To Combined Indoor-Outdoor Navigation*, Technical Repport 2002/10, 2002

[33] E. M. Clarke and E. A. Emerson, *Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic*. In proceedings of the IBM Workshop on logics of programs, **131** of LNCS, pages 52-71, Springer Verlag, 1981

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399