Dealing with Uncertainty in Contextual Requirements at Runtime

A Proof of Concept

Edith Zavala

Thesis supervised by

Dr. Xavier Franch[1] and Dr. Jordi Marco[2]

[1]Department of Service and Information System Engineering
[2]Department of Computer Science

A thesis submitted for the degree of

Master in Innovation and Research in Informatics

Service Engineering Specialization

Facultat d'Informàtica de Barcelona (FIB)

Universitat Politècnica de Catalunya (UPC) – Barcelona Tech

7th July, 2015

Abstract

Self-adaptive systems are capable of dealing with uncertainty at runtime handling complex issues as resource variability, changing user needs, and system intrusions or faults. If the system requirements depend on context, runtime uncertainty will affect the execution of these contextual requirements. This work presents SACRE, a proof-of-concept implementation of an existing approach, ACon, developed by researchers of the Univ. of Victoria (Canada) in collaboration with the UPC (Spain). ACon uses an adaptation feedback loop to detect contextual requirements affected by uncertainty and data mining techniques to determine the best operationalization of contexts on top of sensed data at runtime. The implementation is placed in the domain of smart vehicles and the contextual requirements involved are aim to provide the functionality of detect and support drowsy drivers. The results obtained with SACRE validate satisfactorily the ACon approach and corroborates that this novel approach is applicable in real world and real-time domains. This prototype could become the seed of more sophisticated tools that eventually could be integrated in real systems. Finally, this work contributes in both research and industry communities. It shows advances in the requirements engineering field for researchers interested in contextual requirements evolution at runtime, at the same time it materializes these advances in a currently trending topic application, demonstrating its potential in the industry.

Acknowledgments

This work would not have been possible without the support of many people. First, the unconditional support of my family and friends, during my studies. Second, the guidance and extremely useful support provided by my supervisors. Third, the financial support during my Master's studies provided by the Mexican organizations: Consejo Nacional de Ciencia y Tecnología (CONACYT) and Instituto de Innovación y Transferencia de Tecnología (I^2T^2). Special thanks also to the Univ. of Victoria, Canada experts for the guidance and support during this project.

The following is the list of some of the persons that helped in one way or another to the completion of this thesis.

Anabel Rodríguez
Arturo Zavala
Evelyn Zavala
Edna Zavala
Ángel Zavala
Alfredo Ríos
Mauricio Ríos
José Ríos

Mario Mechoulam
Ekaterina Mechoulam
Xavier Franch
Jordi Marco
Alessia Knauss
Daniela Damian
Ilsa Torres

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Context

The complexity of current software systems and uncertainty in their environments has led the software engineering community to look for inspiration in diverse related fields (e.g., robotics, artificial intelligence, control theory, and biology) for new ways to design and manage systems and services [1-4]. The first research areas in computer science and engineering in paying attention to the adaptation phenomenon were the control theory and cybernetics. The initial works were based on the physiological homoeostatic principles stated by the early works of Cannon [5] and specialized later by Wiener [6] with the introduction of the closed loop control paradigm that basically consists in the interaction between two elements: an environment and a control system [7].

Nowadays, *self-adaptive systems* are capable of dealing with uncertainty at runtime handling complex issues as resource variability, changing user needs, and system intrusions or faults. This means that they are able to deal with emerging requirements not present at design-time which increases the system availability and its robustness. In order to accomplish this goal, self-adaptive systems execute basic requirements engineering activities by themselves at runtime [12]. While these systems promise to deal with uncertainty, they challenge current software engineering practices, particularly the activities of requirements definition and satisfaction in uncertain environments [24].

Classical requirements engineering is based on the assumption that the environmental context is static and can be understood sufficiently. In the presence of these two environment assumptions, traditional requirements engineering can be performed well. This might be still possible in some environments where the speed of change is slow enough; therefore, existing techniques are still capable of capturing, managing and adapting these contextual changes [8]. However, novel contexts such as smart cities or smart vehicles have to deal with unforeseen conditions at very high speed rate. This dynamicity and uncertainty are the two main obstacles in order to understand, discover, formulate, validate, reason and manage the requirements both at design and especially at runtime [8].

In the self-adaptive software field, the requirements described above are well known as *contextual requirements*. Contextual requirements should be understood as requirements that are characterized to be valid only in a specific context [9-11]. More specifically, based on [12]:

> A *contextual requirement* consists of a 2-tuple of the expected system behaviour and the specific context within which this expected behaviour is valid.

Deeper discussion about self-adaptive systems and contextual requirements challenges as well as proposed solutions to deal with will be presented in later sections.

1.2 Motivation

The capability of the systems, natural and artificial, to adjust their behaviour in response to the environment in the form of self-adaptation has become one of the most promising research directions in the field. In computer science, a great endeavour has been done in order to study the complex combination of some terms present in the nature such as: intelligence, rationality, learning, anticipation and adaptation [7]. Studying them separately seems little promising since the objective of the software engineers is to mimic nature systems.

The computer science community: researchers, developers, industry, etc. is requested to find new and better ways of satisfy the more and more changing and demanding society requirements in uncertain environments. The study, research, development, experimentation and testing of self-adaptive software should be one of the most important topics in the next years. This thesis is motivated by the need of both society and computer science community to advance more and more in the development of systems that minimize the dissatisfaction of requirements due changing contexts.

1.3 Objectives

The ACon (Adaptation of Contextual requirements) approach, presented in [12], has emerged as an approach to address the problem of dealing with runtime uncertainty in the presence of contextual requirements [12] and can represent a great advance for the research and industry communities. This thesis has the objective of validate the ACon approach in a real-time and innovative application domain through the implementation of a proof-of-concept tool. The application domain should be extremely demanding and should have the factor of uncertainty in the environment which it applies.

The validation is done through a proof-of-concept implementation called SACRE (Smart Adaptation through Contextual REquirements) which is the material result and co-lateral objective of this work. SACRE pretends to be the seed of more sophisticated tools that mix both self-adaptive and contextual requirements evolution fields in uncertain environments.

1.4 Methodology

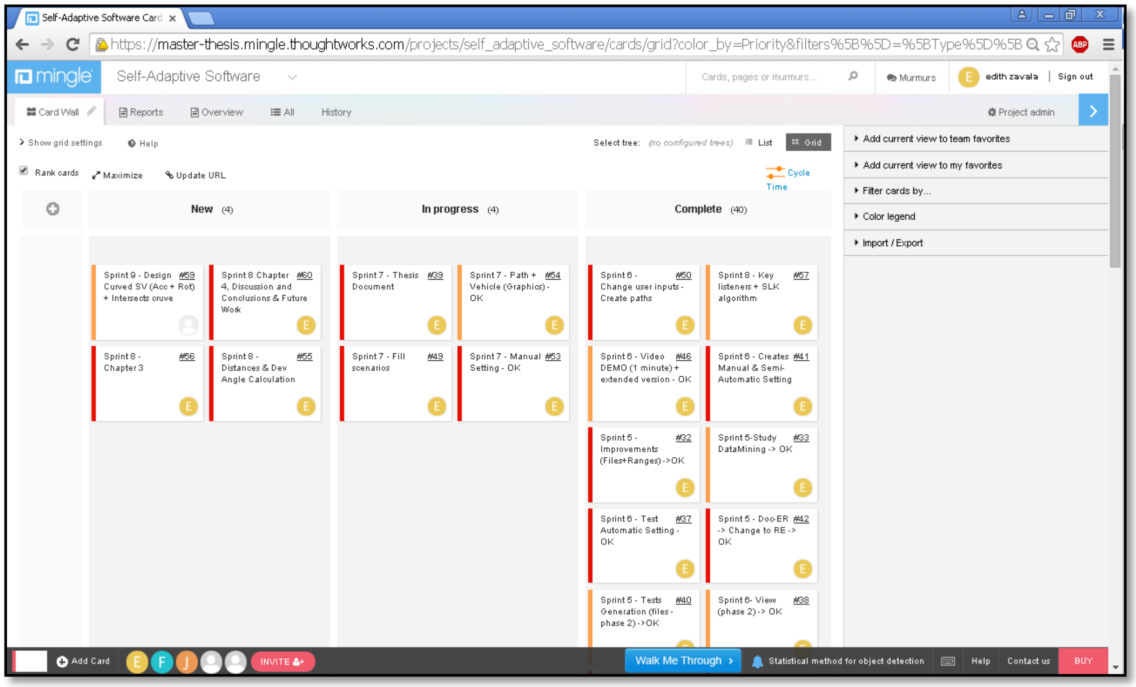
This thesis has been developed in 17 weeks. It has been developed using an *agile* method. Agile methods have demonstrated massive progress in contrast to classic waterfall approaches in the field of software development [13]. Moreover, agile methods are known for reducing time to market, increasing productivity, and gaining cost effectiveness and efficiency of software development efforts [14], which are really desirable characteristics in the current competitive industry.

Furthermore, this approach assumes that requirements changes will occur (at production state) and can be managed effectively. That is why a frequent feedback from the customer is provided, ensuring and enhancing the customer satisfaction. In the case of this thesis the roles of customer, stakeholders as well as product owner (the role that communicates the customers with the work team) have been developed by the thesis advisors and the team role has been played by the student. In order to formalize the approach, the agile manifesto defines four basic core values:

1. Individuals and interactions over processes and tools;
2. Working software over comprehensive documentation;
3. Customer collaboration over contract negotiation;
4. Responding to change over following a plan.

In addition, in [33] there is defined a set of principles that form guidelines for development. *Scrum* is one of the paradigmatic agile methods, it provides a project management framework that focuses development into 15-30 day *sprint* cycles in which a set of *backlog* features are delivered [15]. Scrum defines a daily 15-minutes team meeting for coordination and integration, which in the case of this thesis has been done individually in order to enumerate the activities and objectives planned for each day and in order to evaluate the advances.

The approach only takes into account the process of software development; nevertheless it has been adapted for being applied also to the research phase of this thesis. The deliverables of the research phase have been also visualizations of the advances, just in the same way as a software prototype would be presented. The sprints in this work have been stated as 15-day periods and the backlog features have been recorded in the Mingle[1] tool as it is shown in Figure 1. Moreover, minutes summarizing discussed items and to-do items are produced in each sprint meeting.


Self-Adaptive Software Card
https://master-thesis.mingle.thoughtworks.com/projects/self_adaptive_software/cards/grid?color_by=Priority&filters%5B%5D=%5BType%5D%5B
mingle
Self-Adaptive Software
Cards, pages or murmurs...
Murmurs
edith zavala
Sign out
Card Wall
Reports
Overview
All
History
Project admin
Show grid settings
Help
Select tree: (no configured trees)
List
Grid
Rank cards
Maximize
Update URL
Cycle Time
Add current view to team favorites
Add current view to my favorites
Filter cards by...
Color legend
Import / Export
Add Card
INVITE
Walk Me Through
Statistical method for object detection
Help
Contact us
BUY

New (4)

Sprint 9 - Design Curved SV (Acc + Rot) + Intersects cruve #59

Sprint 8 Chapter 4, Discussion and Conclusions & Future Work #60

Sprint 8 - Chapter 3 #56

Sprint 8 - Distances & Dev Angle Calculation #55

In progress (4)

Sprint 7 - Thesis Document #39

Sprint 7 - Path + Vehicle (Graphics) - OK #54

Sprint 7 - Fill scenarios #49

Sprint 7 - Manual Setting - OK #53

Complete (40)

Sprint 6 - Change user inputs - Create paths #50

Sprint 8 - Key listeners + SLK algorithm #57

Sprint 6 - Video DEMO (1 minute) + extended version - OK #46

Sprint 6 - Creates Manual & Semi-Automatic Setting #41

Sprint 5 - Improvements (Files+Ranges) ->OK #32

Sprint 5-Study DataMining -> OK #33

Sprint 6 - Test Automatic Setting - OK #37

Sprint 5 - Doc-ER -> Change to RE -> OK #42

Sprint 5 - Tests Generation (files - phase 2) ->OK #40

Sprint 6- View (phase 2) -> OK #38

Fig. 1: Backlog of features in Mingle tool

Chapter 2

State of the Art

2.1 Self-adaptive systems

The simultaneous explosion of information, the integration of technology, and the continuous evolution from software-intensive systems to ultra-large-scale systems require new and innovative approaches for building, running, and managing software systems [16]. As a consequence, nowadays, software systems must be flexible, robust, customizable and self-optimizing, among other characteristics usually fulfil by adapting to changing operational context, environments or system features. For the purpose of this work, based on [17], a self-adaptive system is defined as follows:

A *self-adaptive* system is a system able to adjust its behaviour in response to its perception of the environment and the system itself.

Self-adaptive systems have become an important research topic. According to [17], in all the many initiatives to explore self-adaptive behaviour, the common element that enables the provision of self-adaptability is usually software. Moreover, the application areas of this kind of systems are really wide, increasing the importance of its study, development, experimentation, testing and finally the introduction of them in the industry.

Some of the applications mentioned in [17] are: adaptable user interfaces, autonomic computing, dependable computing, embedded systems, mobile ad hoc networks, mobile and autonomous robots, multi-agent systems, peer-to-peer applications, sensor networks, service-oriented architectures, and ubiquitous computing. Furthermore, the combinations and resulting branches of these applications generate a whole world of opportunities for the self-adaptive systems.

Given the application diversity, research initiatives have emerged from different points of view, each of them investigating solutions from its own perspective. It can be mentioned fields such as fault-tolerant computing, distributed systems, biologically inspired computing, distributed artificial intelligences, integrated management, robotics, knowledge based systems, machine learning, control theory, etc. [17]. This thesis explores the idea of merging some of them in order to find synergies that provides new and better solutions for the self-adaptation challenge.

Self-adaptive systems are needed in fields where the environment changes to uncertain states at an unpredictable moment. These changes in the environment include resources availability, user actions or needs, system faults, etc. All these changes that result from a specific operational context at specific moment can be translated as an evolution of the system's requirements. If the requirements depend on context, runtime uncertainty will affect the execution of these contextual requirements.

In the last years the trending solution provided by the research and industry community for designing self-adaptive systems are *feedback loops*, previously used only for controlling and monitoring systems. Currently the systems use these loops in order to execute basic requirements engineering tasks at runtime. One of the most famous feedback loops is the *autonomic element* shown in Figure 2 (introduced by Kephart and Chess [18] and popularized with the IBM's architectural blueprint for autonomic computing [19]); it is the first architecture for self-adaptive systems that explicitly exposes the feedback control loop [20].

Based on [20], an autonomic element consists of a managed element and an autonomic manager with a feedback control loop at its core. The manager or controller is composed of two manageability interfaces, the sensor and the effector, and the monitor-analyze-plan-execute (MAPE-K) engine consisting of a monitor, an analyzer, a planner, and an executor which share a common knowledge base.

The loop described in [20] has the following steps: 1) the monitor senses the managed process and its context, filters the accumulated sensor data, and stores relevant events in the knowledge base for future reference; 2) the analyzer compares event data against patterns in the knowledge base to diagnose symptoms and stores the symptoms for future reference in the knowledge base; 3) the planner interprets the symptoms and devises a plan to execute the change in the managed process through its effectors.

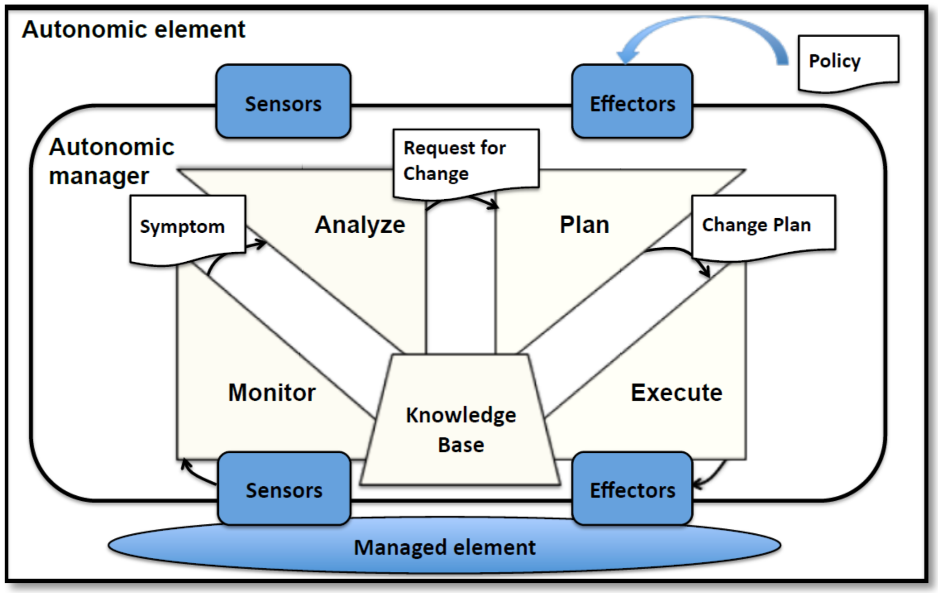


Fig. 2: IBM's autonomic element consisting of a managed element and an autonomic manager with a MAPE-K feedback loop as its core (from [12], [18], [19]).

The manageability interfaces, each of which consists of a set of sensors and effectors, are standardized across managed elements and autonomic building blocks, to facilitate collaboration and data and control integration among autonomic elements. The autonomic manager gathers measurements from the managed element as well as information from the current and past states from various knowledge sources and then adjusts the managed element if necessary through its manageability interface according to its control objective [20]. More details about the feedback loop instance used by ACon will be provided in next sections.

2.2 Contextual requirements

In order to execute an adaptation, self-adaptive systems must continuously monitor changes in their context and react accordingly. As stated in [17], it is obvious that a system cannot monitor everything and must decide what to monitor in order to fulfil a set of high-level goals that should be satisfied regardless the current operational context or environmental conditions. Moreover, non-critical goals could be relaxed in order to minimize the number of non-satisfied goals at every moment, increasing the degree of the system's flexibility and robustness during and/or after adaptation.

Against the classical requirements engineering process, the requirements engineering process for self-adaptive systems it is not focused on what the system should do and under which constraints; instead it focuses on addressing which adaptations are possible and which constraints affect how those adaptations are done [17]. This particular requirement engineering process has two main challenges: 1) not all possible environment conditions are known at design time; 2) in consequence, not all the possible adaptations can be anticipated. The requirements specification for self-adaptive systems, according to [17], should cope with:

- the incomplete information about the environment and the resulting incomplete information about the respective behaviour that the system should expose;
- the evolution of the requirements at runtime

Some efforts have been done in the research community in order to tackle the problem of understanding and modelling these requirements. Currently, the goal-oriented requirements engineering (GORE) and modelling approach seems to be the one that best supports the uncertainty and adaptability that this kind of systems entails. The author in [9] proposes *contextual goal models* to capture the relation between variants for goal satisfaction and context which will be use in this work.

This approach assumes that the context may influence on human's goals and their choices to reach such goals. Moreover, the software developed to meet user's needs has to reflect user adaptation to context, while reaching his/her goals, to derive a set of functionalities to execute. Contextual goal models propose an extension of the basic goal modelling paradigm, being its two main contributions:

- a set of variation points where context might intervene to decide applicable alternatives for goal satisfaction.
- a set of constructs to analyze context. This analysis is for discovering the information that a system needs to capture of its environment to judge if an analyzed context holds.

The main components participating in these models are: actors, goals, tasks, soft goals, and contributions to soft goals, dependencies, means-end, decompositions and contexts. Their definitions could be found in [9], nevertheless due to its relevance in this work the definition of context based on [9] is provided below:

> A *context* is a partial state of the world that is relevant to an actor's goals.

Context analysis should allow for a systematic way in discovering alternative sets of facts an actor may verify trying to judge if a context applies [9]. It is important to note that a context, in order to be evaluated as true or false, should be monitorable by an actor, implying that should be composed by verifiable world predicates (facts) that evidence it. A *fact* should be understood as a world predicate *F* that can be verified by and actor *A* [9]. For example, the world predicate "a student S assists to class C" is a fact since it can be verified revising the current students present in the class.

The approach goes deeper in the explanation and modelling of the contextual requirements, however, thanks to the simplicity of the contextual requirements contained in this work, complex entities are not needed. Figure 3 shows an example of a typical Tropos goal model running the example of a promotion information system, extracted from [9]. This model is useful to realize the contrast with Figure 4, also extracted form [9] which is also a goal model but it considers contexts.

The squares that appear with a letter C plus a number in this last model are the contexts affecting the ways to fulfil the main goal. Depending on the context experienced, the goal will be satisfied with one or another set of tasks. As it has been mentioned before each context could be formed by *statements* and *facts*. Even facts are more desirable because its verifiability, both elements could be present in a context definition. Both entities relate to each other as it is shown in the context model example exposed by Figure 5 extracted from [9].

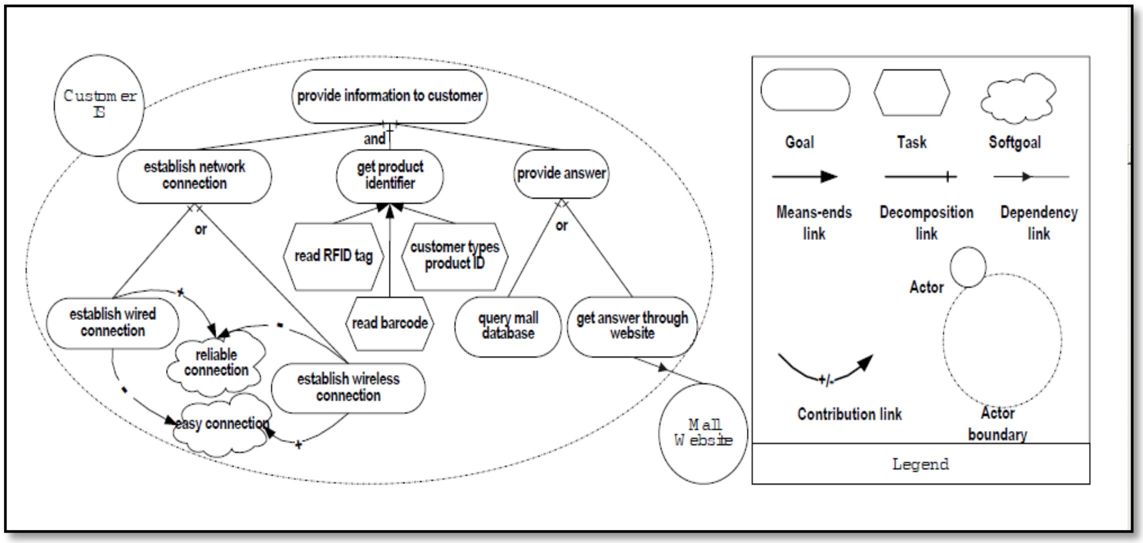


Fig. 3: Tropos goal model example [9]

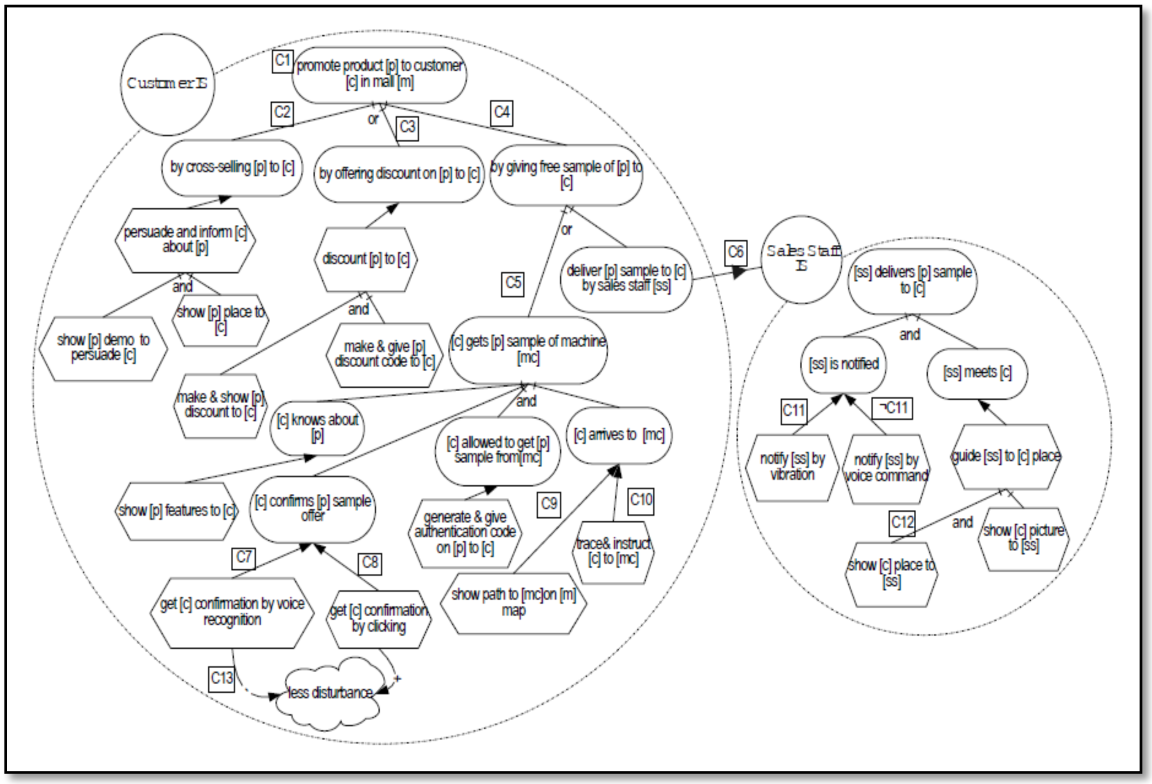

Customer IS
C1 promote product [p] to customer [c] in mall [m]
C2
or
C3
C4
by cross-selling [p] to [c]
by offering discount on [p] to [c]
by giving free sample of [p] to [c]
persuade and inform [c] about [p]
discount [p] to [c]
or
deliver [p] sample to [c] by sales staff [ss]
C5
C6
Sales Staff IS
[ss] delivers [p] sample to [c]
and
show [p] demo to persuade [c]
show [p] place to [c]
and
make & give [p] discount code to [c]
[c] gets [p] sample of machine [mc]
[ss] is notified
[ss] meets [c]
make & show [p] discount to [c]
[c] knows about [p]
and
[c] allowed to get [p] sample from[mc]
[c] arrives to [mc]
C11
¬C11
notify [ss] by vibration
notify [ss] by voice command
guide [ss] to [c] place
show [p] features to [c]
[c] confirms [p] sample offer
generate & give authentication code on [p] to [c]
C9
C10
trace& instruct [c] to [mc]
C12
and
show [c] picture to [ss]
show [c] place to [ss]
C7
C8
get [c] confirmation by voice recognition
get [c] confirmation by clicking
show path to [mc]on [m] map
C13
less disturbance

Fig. 4: A goal model annotated with context at its variations points [9]

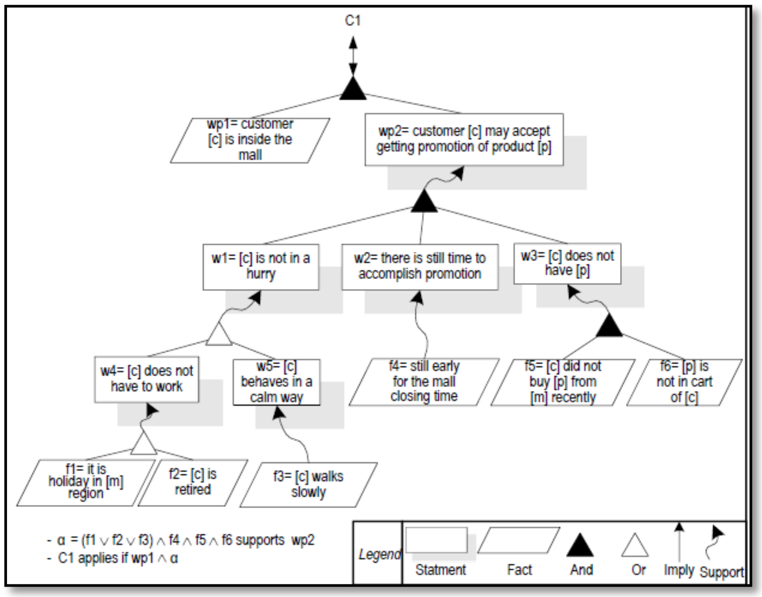

C1
wp1= customer [c] is inside the mall
wp2= customer [c] may accept getting promotion of product [p]
w1= [c] is not in a hurry
w2= there is still time to accomplish promotion
w3= [c] does not have [p]
w4= [c] does not have to work
w5= [c] behaves in a calm way
f4= still early for the mall closing time
f5= [c] did not buy [p] from [m] recently
f6= [p] is not in cart of [c]
f1= it is holiday in [m] region
f2= [c] is retired
f3= [c] walks slowly
- α = (f1 ∨ f2 ∨ f3) ∧ f4 ∧ f5 ∧ f6 supports wp2
- C1 applies if wp1 ∧ α
Legend
Statment
Fact
And
Or
Imply
Support

Fig. 5: The context analysis for C1 [9]

The model shown in Figure 4 will be used later in this document as template for modelling the main SACRE goals. While the model exposed in Figure 5 will be used for describe the contexts considered by the SACRE’s goal models.

2.3 ACon

In [12] the authors propose the ACon approach that emerges as a solution for dealing with the challenges mentioned in the previous sections about self-adaptive systems and the presence of contextual requirements in uncertain environments. The approach depicts the relationship between a contextual requirement, the environment and the context operationalization in terms of the observed environment. In agreement with [12] it can be asserted that changes in the environment result in invalid measured context for a contextual requirement, while changes to the monitoring infrastructure affect system’s ability to recognize the context in which contextual requirements are valid.

Both conditions provoke that the system is not able to satisfy contextual requirements and thus provide the expected system behaviour. In order to fix this problem, the system must adapt the requirement’s context to changes in the environment or in the monitoring infrastructure. ACon has the objective of adapting the measurable context in which contextual requirements are valid to maximize the number of these requirements that are satisfied in the face of runtime uncertainty [12].

2.3.1 Fundamentals

ACon takes the position that self-adaptive systems, in order to respond to unpredictable changes in their operating environment, need to learn from the environmental data that they have available at runtime and execute the adaptation of their contextual requirements in the basis of this apprenticeship [12]. To leverage the great volumes of data available in operating environments (e.g. through sensors), ACon uses machine learning techniques to analyze and detect patterns in the context data. The application of machine learning on contextual data is a novel strategy to detect certain conditions that change over time and to support the reasoning about adaptation decisions.

As authors explain in [12], ACon uses a feedback loop to maintain an up-to-date knowledge about contextual requirements based on up-to date information about the context in which contextual requirements are valid at runtime. Upon detecting that contextual requirements are affected by runtime uncertainty, ACon integrates data mining algorithms that analyze contextual data to determine the context in which contextual requirements are valid and adapt them. ACon includes the interaction with end-users as part of a semi-automatic approach in which the human is in the loop.

As self-adaptive system, ACon aims at reacting to changes that make requirements unsatisfied. In the case of contextual requirements, the particular situation to avoid is having requirements whose context is not operationalized or is not satisfied while the behaviour is which are the two cases that violate the notion of satisfaction of contextual requirements [12]. The operationalization is the set of variables and values for these variables that define a context allowing the system to determine if a context is present or not. In order to accomplish this goal, ACon will continuously try to operationalize contexts not yet operationalized, and will react as soon as possible to unpredictable environment changes and monitoring problems.

2.3.2 Adaptation feedback loop

As it has been mentioned before, feedback loops are first-class entities of engineering self-adaptation. ACon is based on an instance of the IBM's loop cited before, which implements a feedback loop that is known as the MAPE-K loop. The IBM's autonomic element consists of an autonomic manager, and one or more managed elements. The autonomic manager implements two manageability interfaces – sensors and effectors. Through sensors the autonomic manager gathers information from the environment or other autonomic elements. Through effectors, the autonomic manager adjusts the managed element as needed.

An autonomic element itself can be a managed element, therefore consisting of sensors and effectors at the top of the autonomic manager. Through the effectors at the top the autonomic manager can receive policies that drive the adaptation and evolution of the system [12]. The MAPE-K loop, as is explained in [12], contains four components: monitor, analyzer, planner, and executor and the knowledge base. The knowledge base represents the communication channel between the four components of the autonomic manager.

In the case of ACon, the managed element of the feedback loop consists of all contextual requirements of the system. The novel contribution of ACon to this MAPE-K loop is the usage of machine learning to mine context at runtime so that the system can continuously adapt its contextual requirements. According to [12], the purpose of the usage of ACon in the system is the adaptation of the context operationalization in which contextual requirements are valid and differ from previous works that only focus in predefined evolution of requirements without considering uncertainty at runtime. The feedback loop proposed by ACon is called: Adaptation of Contextual Requirements Feedback Loop (short: Adaptation Feedback Loop). Figure 6 gives an overview of the elements and activities taking place in the Adaptation Feedback Loop. In the remainder of this section, based on [12], each component will be described.

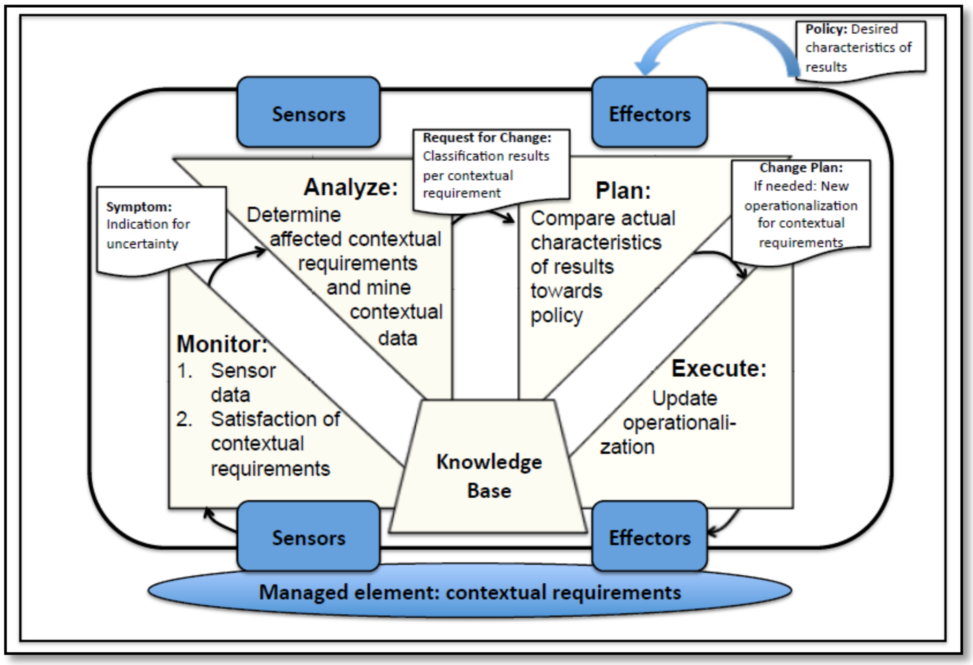

Policy: Desired characteristics of results
Sensors
Effectors
Request for Change: Classification results per contextual requirement
Analyze: Determine affected contextual requirements and mine contextual data
Plan: Compare actual characteristics of results towards policy
Change Plan: If needed: New operationalization for contextual requirements
Symptom: Indication for uncertainty
Monitor: 1. Sensor data 2. Satisfaction of contextual requirements
Execute: Update operationali-zation
Knowledge Base
Sensors
Effectors
Managed element: contextual requirements

Fig. 6: The Adaptation Feedback Loop in ACon responsible for the adaptation of contextual requirements [12]

A. Monitor

The *monitor* senses the managed element and the context, filters the collected sensor data, and decides on relevant events (e.g. requirements violation, sensor relevant change, etc.) that could indicate the need for an action of the autonomic manager. These relevant events are called symptoms and are communicated to the analyzer for further analysis [21]. The monitoring component continuously executes two main tasks:

1) Evaluating the satisfaction of contextual requirements, and
2) Collecting available sensor data related.

ACon considers four different cases that can point to uncertainty affecting the satisfaction of contextual requirements:

- Case 1: No operationalized context: This case will arise typically when the context has not been operationalized at design time and still there is not enough data for the data mining algorithm to propose a context at runtime. Eventually, it also may happen that a context becomes un-operationalized at runtime since no feasible operationalization can be found at a certain moment.

- Case 2: An unpredictable monitoring framework state: When the context is operationalized, some of the sensors might become unavailable temporarily or permanently. Even if the signal is not lost, the data may be incorrect because the sensor requires re-calibration. In any case, the system will not be able to satisfy those contextual requirements which contain the sensor that is malfunctioning in their operationalization. To identify this situation, all sensors that are part of the current operationalized context for the contextual requirements are monitored. Three possible situations are detected in [12]:

    a. The sensor stops sending data.
    b. The sensor is sending wrong data.
    c. The sensor regains a correct state. Either because it gets re-calibrated or comes back into operation again. Requirements that have this sensor in their operationalization are affected, as well as all other requirements that had it at some moment in their history. This second type includes both requirements whose context is not currently operationalized as well as others that are operationalized because it may happen that the current operationalization does not behave as well as the one involving the recovered sensor.

- Case 3: Contextual requirement not satisfied: This situation may occur in two completely different circumstances. First, it may happen that the context has been operationalized in an incorrect or inaccurate way or because the action that leads to the satisfaction of the behaviour has not been executed yet. The reason will be found out in the next phase of the cycle (analysis), determining then the adequate actions for the next loop.

- Case 4: Effects of environmental uncertainty: Cases occur in which the operationalization is not satisfied but the expected system behaviour is. Especially at the beginning, when the data mining classifier is not properly trained because it does not have enough historical data, this case will appear as the system has to learn over time. In this case the user executes an action that makes the specified system behaviour to be fulfilled, triggering the system to re-operationalize the context considering the current context state in the new operationalization.

B. Analyzer

The *analyzer* correlates the received symptoms and in case it decides about the need to adapt the managed element it sends a request for change to the planner. A symptom contains relevant information for the indication on uncertainty so that the analyzer can analyze the situation and make a decision on whether to act on the indications or not. The analyzer uses data mining on the collected sensor data to determine an appropriate operationalization. The decisions will depend on the case that the monitor sends, as follows:

- Case 1: For the case of a symptom of a contextual requirement with no operationalized context no further analysis is needed in the analyzer. It operationalizes the context based on given sensors and collected sensor data up to the current point in time.

- Case 2:

    a. A symptom that points to a sensor failure is ignored up to the point where the analyzer detects symptoms that point to a sensor failure for the same sensor for the *x* last iterations. The analyzer identifies the missing sensor and re-operationalizes the context, making sure that the sensor is not included in the operationalization.

    b. The analyzer handles symptoms for a sensor that is sending wrong data in the same way as case 2 a). It waits for symptoms of type 2 b) for several iterations and re-operationalizes the context for affected requirements excluding the affected sensor.

    c. In case of symptoms for a regained sensor the analyzer waits for several iterations and re-operationalizes the context for all contextual requirements that had this sensor in the operationalization in the past. The analyzer includes the regained sensor when applying data mining.

- Case 3 and Case 4: In both cases the system behaviour is not as expected and the analyzer re-operationalizes the context considering the corrected behaviour.

In the analyzer, ACon relies on lightweight data mining algorithms (i.e. rule-based classifiers) which can be easily rerun many times as newer data instances become available. The outcome of applying such data mining algorithm is the identification of patterns, so called rules which represent the operationalization of context. Rules are produced on contextual data collected by the monitor at runtime and represent the operationalization of the desired context in which the contextual requirement is valid. Many different rules can exist for one contextual requirement, depending on how many patterns the data mining algorithm finds in the contextual data. This means that there will exist many operationalizations for the context of one contextual requirement.

C. Planner

The *planner* defines the activity to execute by considering the policy and create the change plan. In order to do that, the planner compares the error thrown by the rules produced by the data mining algorithm against the policy that is given. The policy can represent either an error threshold given by the system provider or given by the users. In ACon, the users can define this threshold at design time and change it at runtime. If the error is smaller than the threshold, the system updates the contextual requirements with the new rules produced by the data mining algorithm.

D. Executor

The *executor* adapts or evolves the system accordingly following the change plan [22]. The change plan contains the accepted rules produced by the data mining algorithms representing a new operationalization that the executor updates in the knowledge base.

E. Knowledge base

The *knowledge base* stores relevant information about:

1) Contextual requirements. For simplicity all information about requirements is stored in the knowledge base. Every requirement is stored as a 2-tuple <context, expected behaviour>.
2) Sensor data as preparation to apply data mining on. The data used for data mining will come from sensors of different types. Before data mining can be applied on the sensor data, the incoming raw sensor data from the available sensors has to be pre-processed. Pre-processing of the contextual data depends on the characteristics of the sensors and data collected. The frequencies of sensor readings that are taken into account for the operationalization have to be determined and can change at runtime. The sensor data has to be normalized to lay within a range between 0 and 1 so that data mining algorithms that rely on Euclidian distance can be applied for operationalization.

After pre-processing of sensor data, [12] proposes another step necessary to prepare the contextual data for data mining. It consist in the calculation of an additional attribute, the indicator attribute (short: IA), this attribute is stored together with the sensor data in the knowledge base and is needed to determine the operationalization of the context in which a contextual requirement is valid. Every time the expected system behaviour was satisfied it stores the current context together with a value true for the indicator attribute, false otherwise.

2.3.3 Data mining

As it has been mentioned before, ACon relies on lightweight data mining algorithms which can be easily rerun many times as newer data instances become available. Lightweight data mining algorithms, based on [23], are algorithms with low complexity which classify data with relatively fast speed and high transparency of their results classifiers. Fast classification speed is a really important factor in applications with constantly changing environments, as SACRE.

The outcome of applying such data mining algorithm is the identification of patterns, so called rules which represent the operationalization of context. In the application example provided by [12] the rule-based classifier used is JRip. Since SACRE shares similar characteristics (e.g. limited computational resources) with the application in [12], the data mining algorithm considered by this work is also the JRip. This algorithm uses relatively few resources and produces a series of rules that can be converted into a series of if...then statements for system implementation, and which would represent the sensors and associated threshold values that characterize the context conditions for a particular contextual requirement [12].

In [23], an extensive study of the feasibility of integrating data mining algorithms into self-adaptive systems for context awareness and requirements evolution is provided. The author in [23] supports the choice taken by [12] and later taken by this work about the data mining algorithm to use, recognizing the JRip classifier as robust enough in several testing cases providing high levels of accuracy, *precision*, and *recall* even with sensor configuration changes and abrupt changes in normal user behaviour. The *precision* is the measurement that shows how many of the cases that were identified as desired context conditions by the classifier are actually correct; the *recall* shows how many of all desired context instances existing in the sensor data set are also found by the classifier. This work takes into account other measurement: f-measure. This measure is also used by [12] and is defined as the harmonic mean of precision and recall.

In order to obtain the measurements a 10-fold cross validation is applied. The 10-fold cross validation trains and classifies the context conditions on past collected data. Both the algorithm execution over sensed data and the 10-fold cross validation are done in [12] using *Weka*. SACRE takes advantage of the results presented by [12] and uses the Weka tool as well for the same purpose. According to [34], Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing new machine learning schemes. Following sections depict more details about this useful tool and the way it is used by SACRE.

Chapter 3

SACRE

3.1 Description

3.1.1 General purpose

SACRE is a proof-of-concept implementation of the previously explained ACon approach. It is placed in the specific application context of smart vehicles, which is a domain where contextual requirements are constantly affected by uncertainty due to the changing environment. The contextual requirements in SACRE have been designed to provide the functionality of detecting and supporting drowsy drivers. The purpose of the SACRE implementation is to provide the opportunity of validating the ACon approach performance in an extremely demanding situation, as smart vehicles are. This validation is different from the one done in [12] because SACRE runs the approach in real-time providing evidence of its adaptation capabilities at runtime.

The validation in [12] provides a first idea about the ACon capabilities and the advantages of using data mining techniques for adaptation. SACRE provides a complete view of ACon exploiting all the characteristics mentioned in [12]. These changes in the validation process could affect ACon's performance but at the same time could provide evidence of its suitability for be used in real world applications. The results obtained from this work are really relevant for the ACon approach acceptance in the research and industry community.

3.1.2 Application domain

Smart vehicles or intelligent vehicles have become more and more popular in the automotive industry community. In consequence the importance of this domain in the research community has increased. Because smart vehicles are so relevant for industry, their study has been encouraged from the competitiveness between automotive companies which results in an exponentially growing studied field. The domain has interesting but at the same time challenging characteristics such as uncertainty, self-adaptability, self-configurability, high-level of human interaction, etc. The adaptations executed by smart vehicles are not only based on streets' shape, obstacles or weather but also in the user preferences and needs. The smart vehicles, when perform well, improve vehicle and driver safety and performance.

In SACRE just a part of a smart vehicle is implemented because of the thesis' time constraints. Concretely, SACRE offers the functionality of detecting and supporting drowsy drivers at different stages. For this purpose SACRE considers a set of sensors and actuators, graphically shown in Figure 7 and Figure 8, that detects the driver's state and the vehicle position in the lane respectively, and respond with actions in case they are needed. A conceptual model of this domain is provided in Appendix A. The sensors considered are:

1) *Driver's vigilance level*: this sensor is based in the prototype proposed in [25], an image processor for real-time acquisition of a driver's images aiming at monitoring driver's vigilance in real time. This sensor, as shown in Figure 7, is assumed to be placed in the vehicle dashboard. The prototype in [25] proposes to calculate, with this sensor, six parameters: percent eye closure, eye closure duration, blink frequency, nodding frequency, face position, and fixed gaze. In this work, two simple parameters, assumed to be measurable by this sensor, are considered: *eye state* (visible driver's pupil ratio) used by [25] for calculating more complex parameters, and *face position* (frontal/non-frontal).

   The simulation of the eyes state measure, in SACRE, is simplified and supposed to be reported as an average of the state of both eyes. The measure is reported in percentages, i.e. the sensor reports a 46% of visible driver's pupil at a particular moment; in normal conditions this means that the driver's eyelids are covering 54%; or, if the sensor reports 0% of visible driver's pupil at a particular moment, in normal conditions it should be understood that the driver has the eyes closed. Further calculations in the adaptation feedback loop are done over this measure.

About the face position measure, in SACRE it has been simplified to report only two states. Agreeing with [25], SACRE assumes a symptom of driver's fatigue when the driver's face orientation is non-frontal, even sometimes it could means only minor distractions. More details are depicted later in this work.

2) *Electrocardiogram*: this sensor is placed in the driver's seat and provides the measure of driver's hearth beats per minute (hbpm). Some studies such as [25] and [27] have shown that the hbpm value of a person differs considerably when s/he is asleep or awake. Moreover, [27] concludes that the electrocardiogram measure can discriminate with high accuracy between sleep and awake states. Nevertheless, the changes and normal values of this measure differ from one person to another which implies the need of a training phase. SACRE adapts this measure's thresholds at runtime depending on the driver using the vehicle in order to increase its accuracy.

3) *Pressure sensitive*: this sensor is placed in the steering wheel and measures the number of hands on it. This measure relates inversely the number of hands in the steering wheel with the degree of drowsiness of the driver. There are three possible options: 0 hands, 1 hand, 2 hands. This means, less hands on steering wheel more degree of drowsiness the drive is experiencing. Even this is not an exhaustive designed measure; it works well as complementary variable to be taken into account in order to increase the total drowsiness degree detection accuracy of SACRE.

4) *Lane keeping control*: these are two sensors assumed to be placed in the edges of the driving mirror. The sensors (referred as right and left sensor) provide information about the vehicle position in lane. In [28] the authors deal with the problem of finding the optimal position and optimal number of these sensors that should be used in a smart vehicle in order to offer a fault-tolerance lane keeping control. For simplicity, and in agreement with current advertisements [35][36], this work considers only 2 sensors situated as explained before. Even the vehicle position calculation is simulated by SACRE, the measures are based in real world physics, for example the ones presented in [29]. The variables measured by these sensors are:

    a. *Lateral distances*: the lateral distances are measured by each sensor regarding its corresponding line side. This means, the right sensor measures the distance between the vehicle's right frontal wheel and the right street's line and the left sensor measures the distance between the vehicle's left frontal wheel and the left street's line.

Because the lane and vehicle dimensions could change, the optimal distances are calculated based on the specific instantiated vehicle placed on the middle of the specific lane instance. Even, if the demonstration of SACRE in this work is done using always the same vehicle instance and road dimensions (but not the same shape), the tool can load other instances as well.

b. *Frontal distances*: the frontal distances follow the same principles as the lateral distances measures, but instead of measuring the distance between the wheels street's lines perpendicularly they represent the distances measured parallel. The desired measure is infinite (not street line in front of the vehicle), meaning that the vehicle rotation angle is exactly the same as the lane angle. Any other frontal distance value would mean that the vehicle has a disposition regarding the lane.

This situation could occur when the driver rotates unnecessarily the steering wheel or when a curve approaches and the driver has not rotated sufficiently the steering wheel. This measure is useful for correct the vehicle angle regarding lane shape changes serving as a predictive measure.

c. *Deviation angle*: Considering the measures presented before, the lane keeping control could provide the deviation angle variable which is not more than the difference between the current vehicle angle and the street angle in the current vehicle position (resulting in a positive or negative value). This measure is also useful for correct the vehicle angle rotating the steering wheel, but differs in its functionality from the frontal distances measure because it is not predictive. More details about its calculation are provided later in following sections.

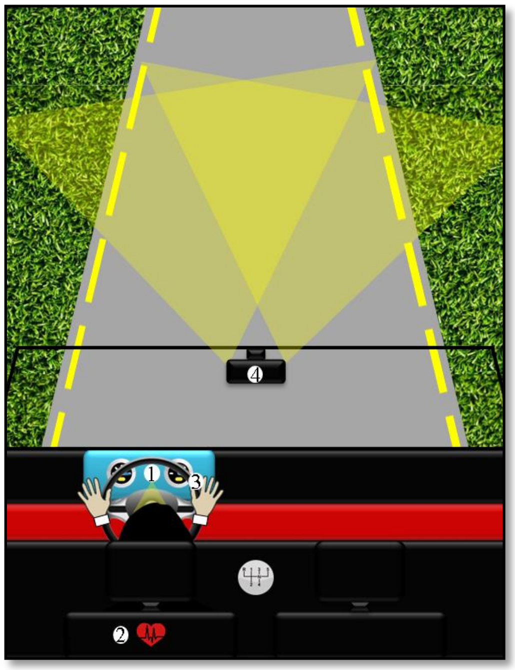


Fig. 7: Vehicle Sensors

The possible stages of a driver defined by SACRE and resulting from some of the sensors' measures mentioned above are: awake, tired, dangerously tired and asleep; ascendingly ordered regarding the driver drowsiness degree each of them represents. In order to support the driver in each of these states, SACRE provides the following actuators:

1) *Support lane keeping*: when the support lane keeping actuator results to be activated it takes the control of the vehicle, basically it controls the accelerator, brake and steering wheel, in order to maintain the vehicle in the lane. The support lane keeping should fulfil some soft requirements such as don't apply abrupt changes in these three devices and ensure the diver's safety. When support lane keeping is activated a target speed is maintained according to the context (e.g. if a curve is approaching the speed is decreased and maintained around 20 km/h). Future iterations of SACRE could include sensors for detecting weather conditions (i.e. detect if it is raining) which would also affect support lane keeping performance. This actuator can be turned on, off and disabled by the driver resulting in candidate adaptations of SACRE.

2) *Vibration alarm*: the vibration alarm is placed in the driver's seat. It is turned on by SACRE in order to alert the driver when s/he is presenting drowsiness symptoms. The vibration of this alarm is supposed to be the adequate for driver's alerting and not for driver's drowsiness encourage. In future versions of SACRE the vibration power could be also adapted regarding context, for example, street's pavement quality which could affect the impact of the

vibration in the driver. For this prototype this is not feasible since no sensor has been considered for detecting that context.

The driver can turn the vibration alarm off (when activated by the system) and disable it when s/he wants to maintain it off indiscriminately. As the same case as with support lane keeping, these actions will result in candidate adaptations of SACRE.

3) *Sound-light alarm*: the sound-light alarm follows the same principles as the vibration alarm, being activated when the driver shows drowsiness symptoms. The decision of activate the vibration or the sound-light alarm depends on the contextual requirements' operationalization that will be presented in following sections. The sound-light alarm could also be calibrated according to the context in future versions of this prototype, for example taking into account the environmental sound and light noise; however, no sensors, for measuring this context, are considered for this tool iteration.

   The sound-light alarm can also be turned off by the driver (when activated by the system) and disabled when s/he wants to maintain it off indiscriminately. The alarms differ from the support lane keeping because they cannot be turned on which is totally obvious because their nature. As it can be supposed the driver's interaction with the sound-light alarm will also result in candidate adaptations of SACRE.

The candidate adaptations, due to driver's interaction with the actuators, have to be later evaluated by the adaptation feedback of ACon in order to accept or reject them. The criteria and procedure followed in this evaluation is explained later in this document.

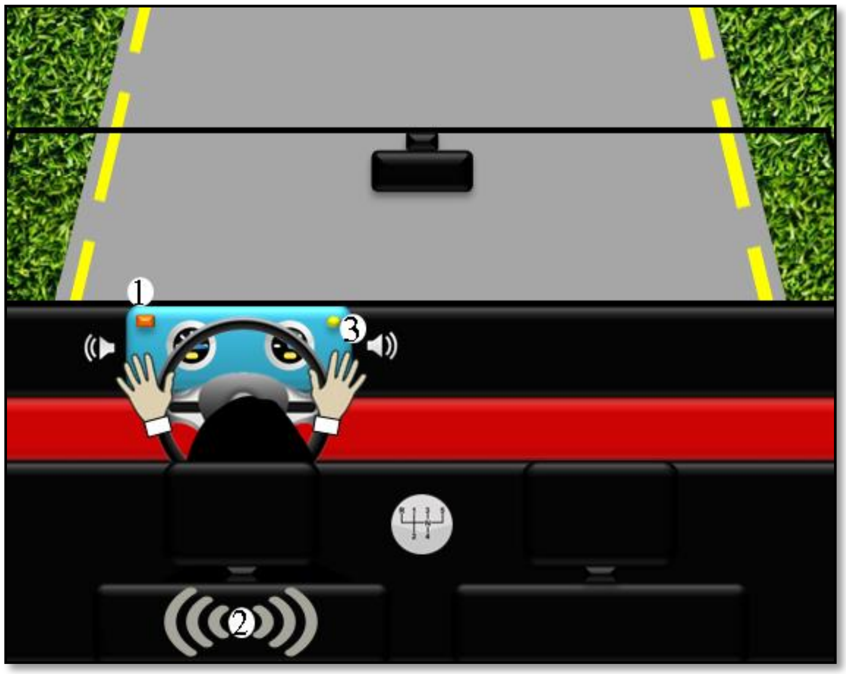


Fig. 8: Vehicle Actuators

3.2 Architecture

In self-adaptive systems, it is necessary to make explicit feedback loops in the architecture design in order to reason about its behaviour and to analyze the adaptation mechanism separately from the target system operation [30]. SACRE keeps, at the higher level of abstraction, the ACon adaptation feedback loop as its core. Nevertheless, the design and implementation of SACRE will be clearer if some modules are explained in detail. At the lower level some architectural decisions have been taken due to the technologies required for the specific domain application. The whole architecture has been conceived following the model-view-controller pattern. The following sections depict the details of SACRE's architecture.

3.2.1 Adaptation feedback loop in SACRE

As it has been mentioned before the adaptation feedback loop proposed by [12] has been taken as SACRE's core. However, some changes have been applied in order to better fit with SACRE's implementation. The version of the adaptation feedback loop used by this work is shown in Figure 9 and the explanation of the changes is presented below. Also, a component-based diagram of the SACRE's architecture is provided in Appendix A.

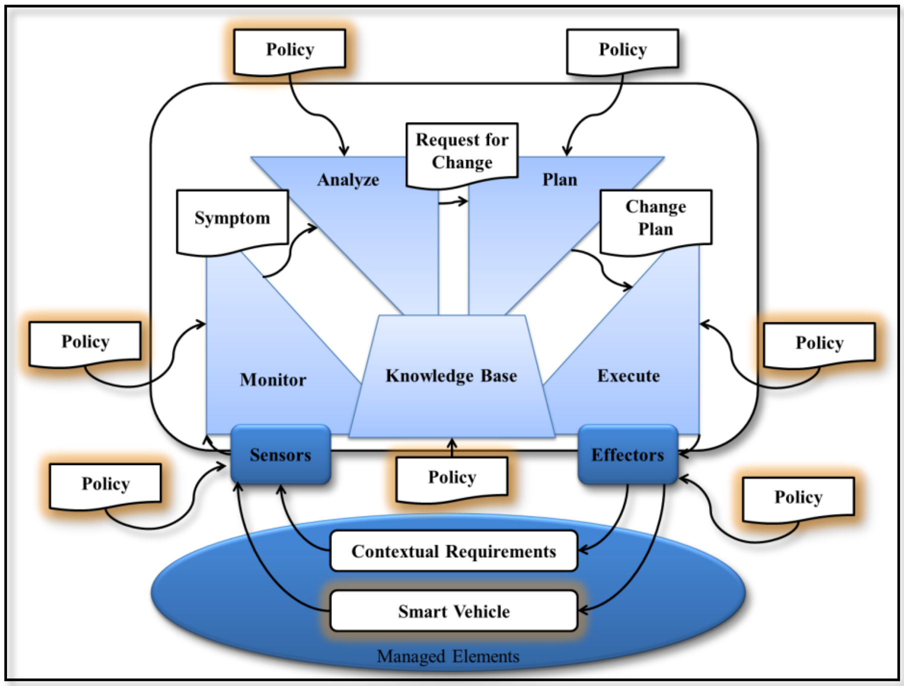


Fig. 9: Adaptation feedback loop in SACRE

The changes applied to the ACon's adaptation feedback loop are:

1) The *policy* element is replicated for all the modules. This change is useful in order to assign user and organization preferences to each element in the loop at design time, increasing the adaptation accuracy at runtime and user satisfaction. This also contributes to the specific domain application abstraction from the loop, increasing the architecture reusability. Although the designed architecture allows changing the policies at runtime, in the implementation this facility has not been provided. The reason is that a smart vehicle, more than a service, is a product that once is out of the providers' ownership they have no control over it unless, for instance, they offer an online support which is not the case of SACRE.

   For this reason, a policy's change at runtime is not considered. Moreover, a driver is intended to interact with the vehicle's actuators and controls and not to know how to manipulate low level characteristics such as the data mining statistics. For instance, imagine that this facility is provided and the user calibrates the data mining acceptable error wrongly; this could cause an unsafety vehicle's behaviour and provoke an accident. Furthermore, the domain specific variables contained in the policies elements are not intended to change over time. The parameters provided, by the policy element in each module, are shown in Table 1.

Table 1: Policy element’s parameters per module

Module	Variables	Observations
Knowledge Base	• Monitor context’s inter-readings time • Minimum number of symptoms before monitor send a symptom document	These variables are supposed to be adapted by the effectors (not done in this demonstration) and communicated to the monitor through the knowledge base.
Sensors	• Specific domain variables • Maximum variables’ values • Minimum variables’ values • Operations to be applied over sensors’ values to calculate variables	The minimum and maximum values are useful for normalizing the variables’ values and for detecting damaged sensors.
Monitor	• Specific domain variables • Maximum normal variables’ values • Minimum normal variables’ values	The minimum and maximum normal values are useful for detecting symptoms (i.e. outliers).
Analyze	• Algorithm to be used • Tool to be used • Parameters for composing the request to the tool; see Appendix B for more details • Data mining algorithm statistics to extract from the response	The request to the tool is prepared for the specifications of the tool used in this demonstration and should be changed if another tool is used instead.
Plan	• Data mining algorithm statistics to check • Data mining algorithm statistics’ thresholds	These thresholds as mentioned before are useful for accept or reject adaptations proposed by the analyzer.
Execute	-	In this implementation the policy element of the executor is empty because the commands’ simplicity it should generate. In more complex implementations instructions about how to translate the change plan into commands would be useful.
Effectors	-	In this implementation this policy is empty because the adaptations are aimed only to the contextual requirements. More complex implementations should use this policy for help the module to effect correctly over the corresponding managed element.

2) In this work, the s*pecific domain application* element has been added to the managed elements module. In this way, both the target application and the contextual requirements element enter in the loop and are adapted. This could be taken as an explicit integration of the application in the loop and not as a really change since, as it has been interpreted by this work, [12] assumes its integration.

3.2.2 Data mining in SACRE

This section depicts more details of the *analyzer* module. As it has been mentioned before, the analyzer component uses data mining techniques in order to determine candidate adaptations finding patterns in the historical data. Moreover it receives the symptoms from the monitor module, interpret them, add or remove sensors from the set to be taken into account in the data mining algorithm and generate the *request for change* document.

In order to increase the architecture's reusability, the data mining module has been designed as an external service letting the opportunity to other implementation to try with other techniques that could provide better results than the machine learning techniques or aggraded techniques to improve the adaptations accuracy. In the case of SACRE the data mining service is offered by the Weka tool API which at the implementation level also implies a clear separation from the rest of the modules. The resulting split of the analyzer module is shown in Figure 10. More details about this service are depicted later in this document.

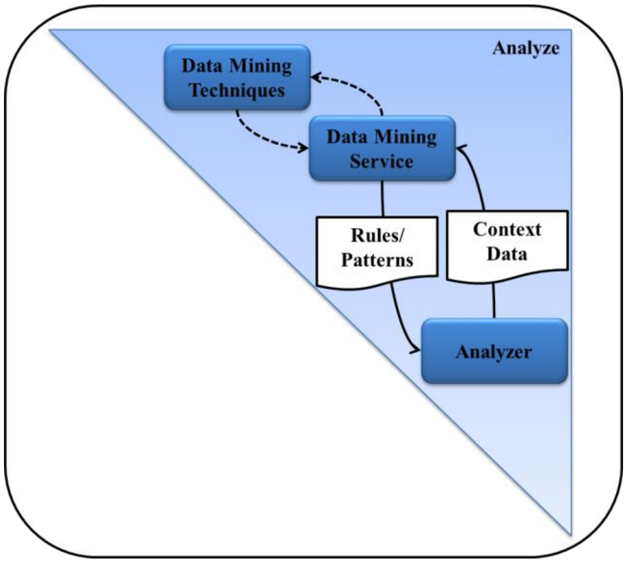


Fig. 10: Expanded analyzer module

In SACRE, the analyzer removes or adds a variable when the variable shows symptoms for 3 iterations as proposed by [12]. In the same way the monitor waits 3 iterations with symptoms before send them to the analyzer. This decision was taken thinking of the calls to the data mining algorithm (JRip) which is well known to provide better results when more evidence is provided, which means more data for finding patterns [12]. This also ensures that the symptom is not an isolate case. The selection of the optimum number of iterations to wait before sending a symptom implies an exhaustive experimental study of data mining results for this application and it is not aim of this thesis. In this first version, SACRE searches for a trade-off between the data mining results' quality and the system adaptations dynamicity.

Because the driver's safety and the driver's comfort factors' importance in this domain is high, the system should respond fast to adaptions minimizing the number of unsatisfied contextual requirements at any time as stated in [12]. At the same time it should ensure high accuracy when adapting, this accuracy depending on the data mining results. The decision of waiting for three iterations is useful for deadlock in case contradictory data exist. For instance, imagine that the last three SACRE iterations have report symptoms over the same variable as follows: outlier, mal-functioning, outlier. If just the two first iterations would be used the system could not know if the variable is in an outlier or mal-functioning case, and the third iteration would introduce a deadlock situation.

3.2.3 Managed Elements

The managed elements module in the adaptation feedback loop of SACRE is composed of two entities, the contextual requirements and the specific domain application. As it has been mentioned above, the specific domain application in SACRE is that of smart vehicles. The managed elements maintain a communication channel with the sensors and effectors modules; nevertheless, in SACRE the communication channel between the effectors and the smart vehicle is not exploited. More details are depicted in the next section.

3.2.3.1 Smart Vehicle

In SACRE, the smart vehicle element has been decomposed due to the variety of functionalities that it provides. The logics of the smart vehicle behaviour are kept in a controller module that communicates directly with the adaptation feedback loop. Meanwhile, the user visualization or view module is kept totally independently and never communicates directly with the adaptation loop. This visualization is created in order to increase the comprehensibility of the SACRE adaptations, to attract users and to facilitate user interaction with the system. In order to communicate the view with the smart vehicle controller an intermediate service has been designed. A second visualization showing the vehicle in the path has been created. Both visualizations communicate directly to each other without using any middleware service.

It is worth to mention that this design is not affecting the reusability of the abstract architecture shown in Figure 6. What is inside of the specific domain application architecture depends only on this domain which will change for other implementations. On the other hand the proposal about keeping the data mining module as an external service increases the reusability of the internal analyzer architecture. Figure 11 shows the smart vehicle managed element architecture in a lower abstraction level.

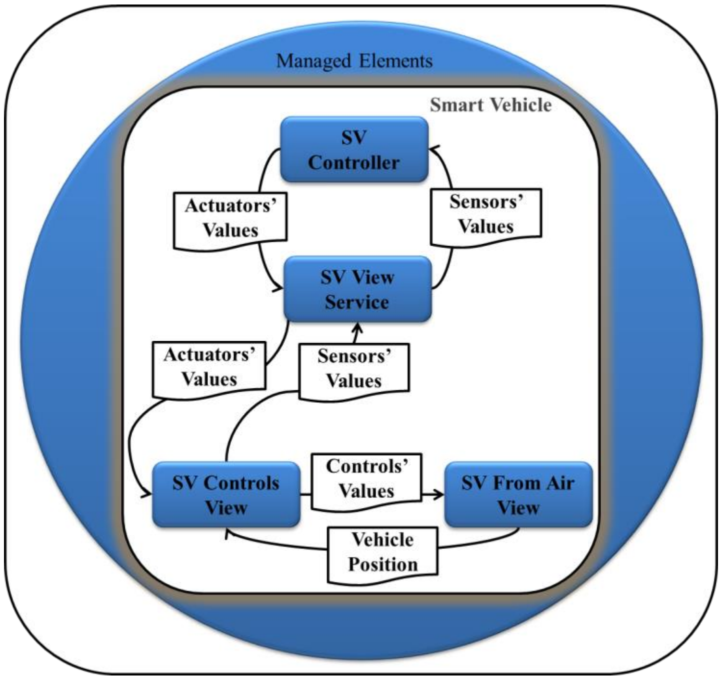


Fig. 11: Expanded application domain module

The entire SACRE cycle flow, taking into account all the considerations mentioned in this section, is shown in Figure 12. Also, a sequence diagram in Appendix A of this process is provided. The cycle can be described to work as follows:

1) The smart vehicle view, containing the controls and variables, communicates the sensors' values changes to the smart vehicle controller through the intermediate service.

2) The smart vehicle controller pushes the sensors' values changes to the sensors module. Analogously, the contextual requirements notify its contextual requirements' operationalization changes to the sensors module.

3) The sensors module for the smart vehicle information performs some calculations in order to generate a set of variables. These variables are the ones that the monitor should use in order to determine if a contextual requirement is satisfied or not. The monitor as well as the sensors module receives the variables and the way to calculate them as initial configuration parameters in their own policy element.

   With the contextual requirements' operationalization changes, the sensors module does nothing, just passes the information to the monitor, when it request it, so it can evaluate the contextual requirements taking into account this new operationalization and the calculated variables.

4) After calculating the variables and in order to enhance the smart vehicle response time to the user, the sensors module communicates the results to the smart vehicle controller.

5) The smart vehicle controller has a communication channel with the contextual requirements element. After receive the variables values from the sensors module it checks the contextual requirements and apply the changes regarding them to the vehicle. Some actions could be not possible to execute since the user intervene, e.g. turn on an actuator disabled by the user.

6) In order to apply the changes the smart vehicle controller not only change its local values but also use the intermediate service for make them explicit in the smart vehicle view module.

7) The smart vehicle view module updates the values and show them to the user through the user view (a display). This smart vehicle view could be updated from the environment (simulating a receptor of hardware changes in the real world) and from the smart vehicle controller being obligated to maintain an aggregated and updated set of changes to the user view. This module discriminates between entities' priorities in order to do the updates. More details about this module are provided in following sections.

8) Meanwhile the steps 4, 5, 6 and 7 are occurring, the monitor module could ask at any configured time the current context to the sensors module. The sensors module could respond that it does not have any data yet or could respond with the variables values and contextual requirements' current operationalization.

9) With the variables' values and an updated version of the contextual requirements' operationalization, the monitor is able to calculate contextual

requirements satisfaction. In case it finds a symptom, the monitor categorizes it in any of the four possible cases considered in Section 2.3.2. Then it should check if the symptom has been found in the last *x* iterations (being *x* a customizable value, currently 3) and in this case, it pushes a symptom document to the analyzer module. The monitor module could find symptoms for each tuple of contextual requirements.

10) The analyzer communicates with the data mining service in order to find patterns in the historical data for the contextual requirements affected by the symptoms and communicate the results to the planner through a request for change document. Using its policy element the analyzer decides the tool and the specific algorithm to use in order to effectuate the data mining.

    The Weka data mining tool and the JRip algorithm are the only initial parameters considered by the demonstration shown in this work; nevertheless, any tool and algorithm could be used if its respective connector module is implemented since each tool and algorithm has its own request/response and data format or API. Creating connectors for alternative tools and algorithms is out of the scope of this work.

11) The planner uses its policy element to discriminate adaptation proposals and communicates its decision to the executor. The statistics used in SACRE, resulting from a 10-fold-cross-validation realized over the JRip algorithm results, are: *precision*, *recall*, *f-measure* and *error*.

12) The executor manipulates the instructions provided by the planner as it is stated in [12] and communicates specific actions to apply over the managed elements to the effectors module.

13) As it has been mentioned above, the adaptations considered by SACRE and ACon are the ones regarding to contextual requirements. The smart vehicle behaviour is not in charge of the loop but the rules of its behaviour that depend on context (contextual requirements) are. Anyway the adapted contextual requirements are later consulted by the smart vehicle.

    The decision of maintaining the smart vehicle functionalities out of the cycle is necessary in order to maintain an adequate response time of the smart vehicle, keeping the requirements that do not depend on context out of the loop. With these considerations, the effector applies the actions indicated by the executor module (that always correspond to the contextual requirements managed element) that are not more than updates of the contextual requirements' operationalizations.

14) If the contextual requirements element changes it announce it to the sensors module closing the loop.

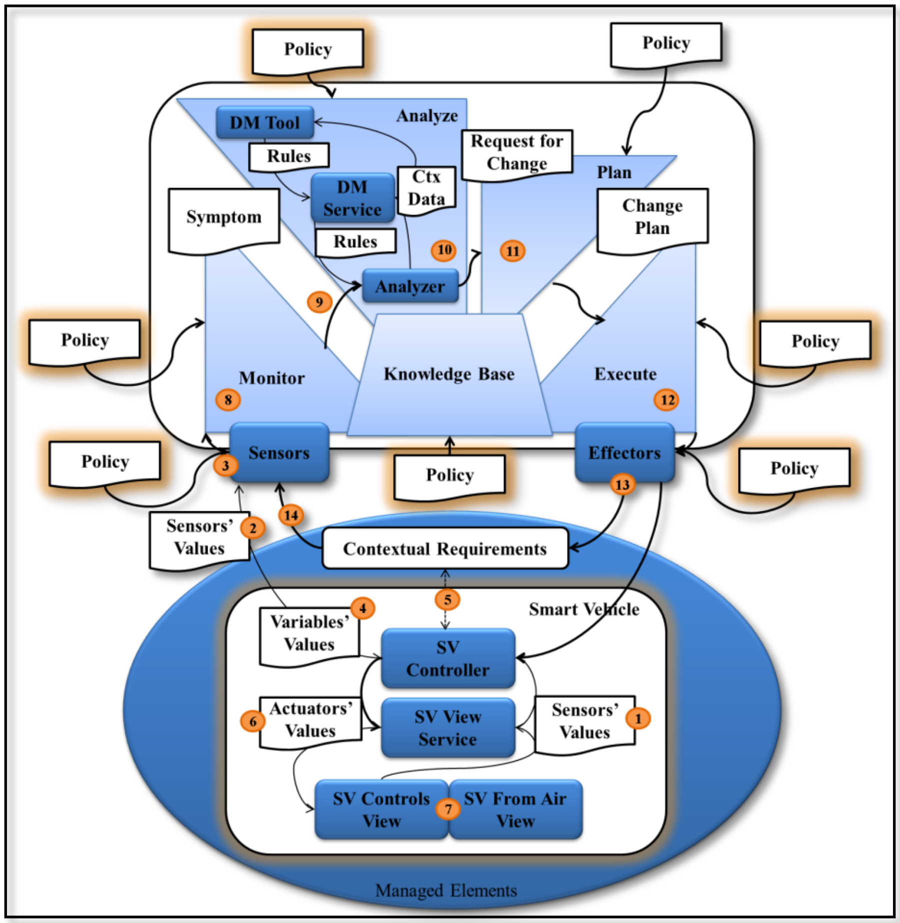


Fig. 12: SACRE's cycle

3.3 Contextual Requirements

The contextual requirements modelling field nowadays is work in progress, not yet standardized or exhaustively studied. Researchers have done some efforts proposing approaches for modelling this kind of requirements, nevertheless further investigation and discussion should be done. Because this work is not aim to provide a proposal of contextual requirements modelling it takes the approach exposed in [9] with this purpose.

For the purposes of this work they have been detected two main SACRE's goals: 1) vehicle keeps in lane, 2) driver does not sleep. Figures 13 and 17 show the models of these two goals annotated with contexts at their variations points. Meanwhile, Figures 14-16 and 18-23 present the models of the contexts considered by each goal model, respectively.

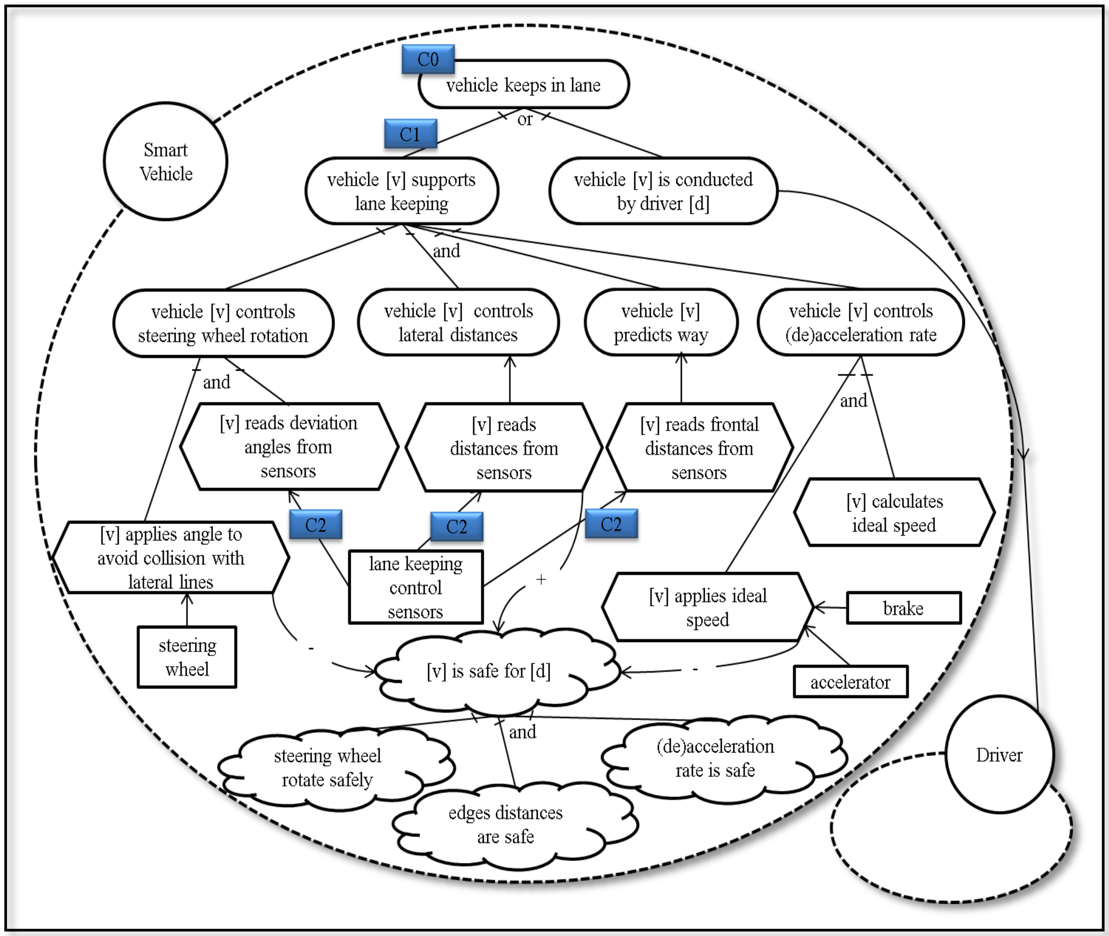

C0
vehicle keeps in lane
or
Smart Vehicle
C1
vehicle [v] supports lane keeping
vehicle [v] is conducted by driver [d]
and
vehicle [v] controls steering wheel rotation
vehicle [v] controls lateral distances
vehicle [v] predicts way
vehicle [v] controls (de)acceleration rate
and
[v] reads deviation angles from sensors
[v] reads distances from sensors
[v] reads frontal distances from sensors
and
[v] calculates ideal speed
C2
C2
C2
[v] applies angle to avoid collision with lateral lines
lane keeping control sensors
+
[v] applies ideal speed
brake
steering wheel
-
[v] is safe for [d]
-
accelerator
and
steering wheel rotate safely
(de)acceleration rate is safe
Driver
edges distances are safe

Fig. 13: Vehicle keeps in lane - Goal Model

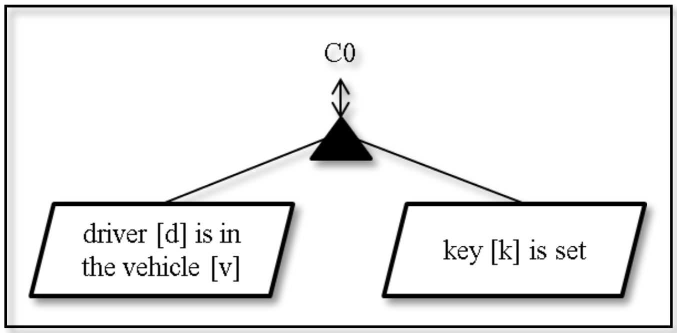


Fig. 14: Smart vehicle is On – Context Model

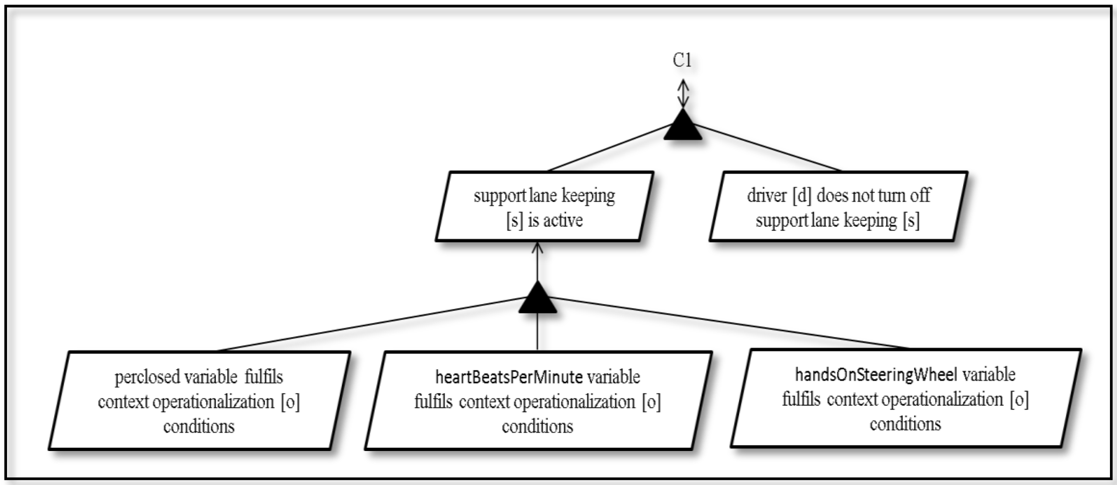

C1
support lane keeping [s] is active
driver [d] does not turn off support lane keeping [s]
perclosed variable fulfils context operationalization [o] conditions
heartBeatsPerMinute variable fulfils context operationalization [o] conditions
handsOnSteeringWheel variable fulfils context operationalization [o] conditions

Fig. 15: Driver is asleep and support lane keeping is On – Context Model

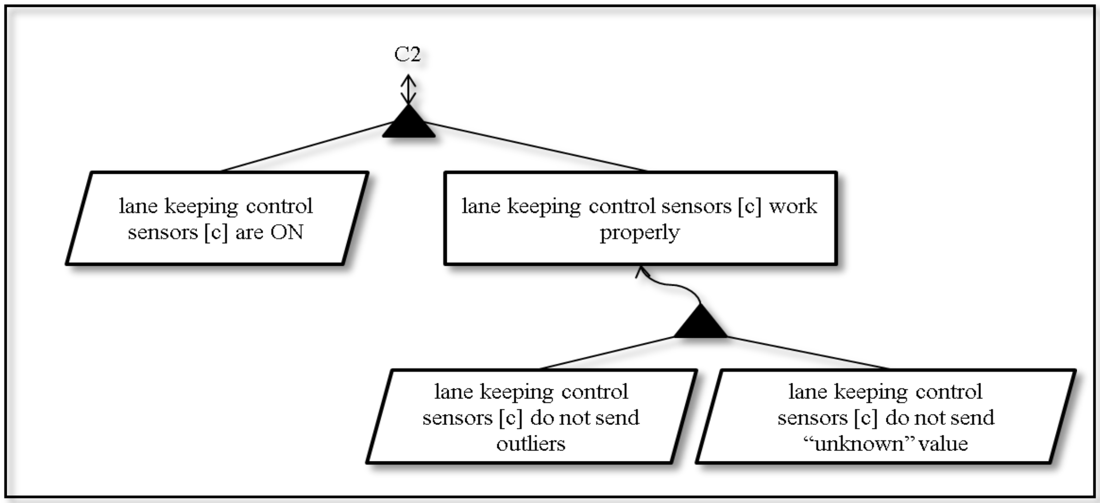

C2
lane keeping control sensors [c] are ON
lane keeping control sensors [c] work properly
lane keeping control sensors [c] do not send outliers
lane keeping control sensors [c] do not send "unknown" value

Fig. 16: Lane keeping control sensors are OK – Context Model

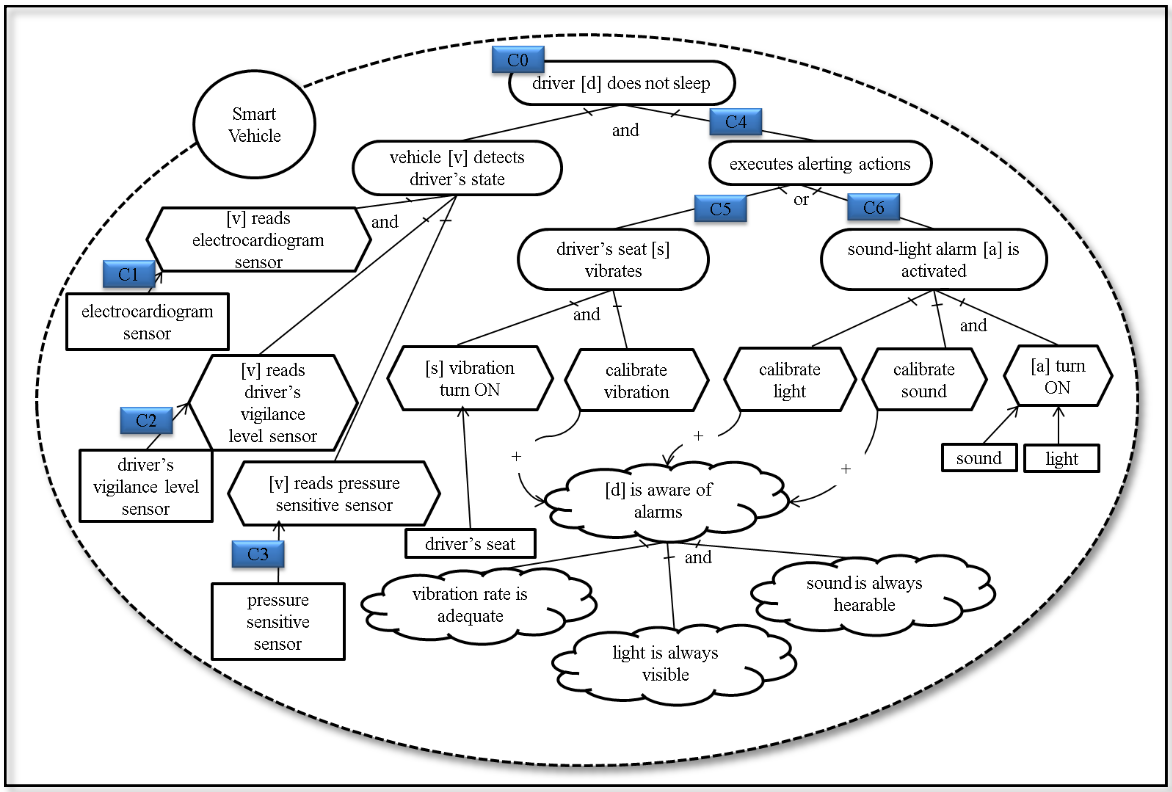

C0
driver [d] does not sleep
Smart Vehicle
and
C4
vehicle [v] detects driver's state
executes alerting actions
or
C5
C6
[v] reads electrocardiogram sensor
and
driver's seat [s] vibrates
sound-light alarm [a] is activated
C1
electrocardiogram sensor
and
and
[v] reads driver's vigilance level sensor
[s] vibration turn ON
calibrate vibration
calibrate light
calibrate sound
[a] turn ON
C2
+
+
sound
light
driver's vigilance level sensor
[v] reads pressure sensitive sensor
+
[d] is aware of alarms
+
C3
driver's seat
and
pressure sensitive sensor
vibration rate is adequate
sound is always hearable
light is always visible

Fig. 17: Driver does not sleep - Goal Model
*Context C0 is the same context than C0 in goal model: Vehicle keeps in lane

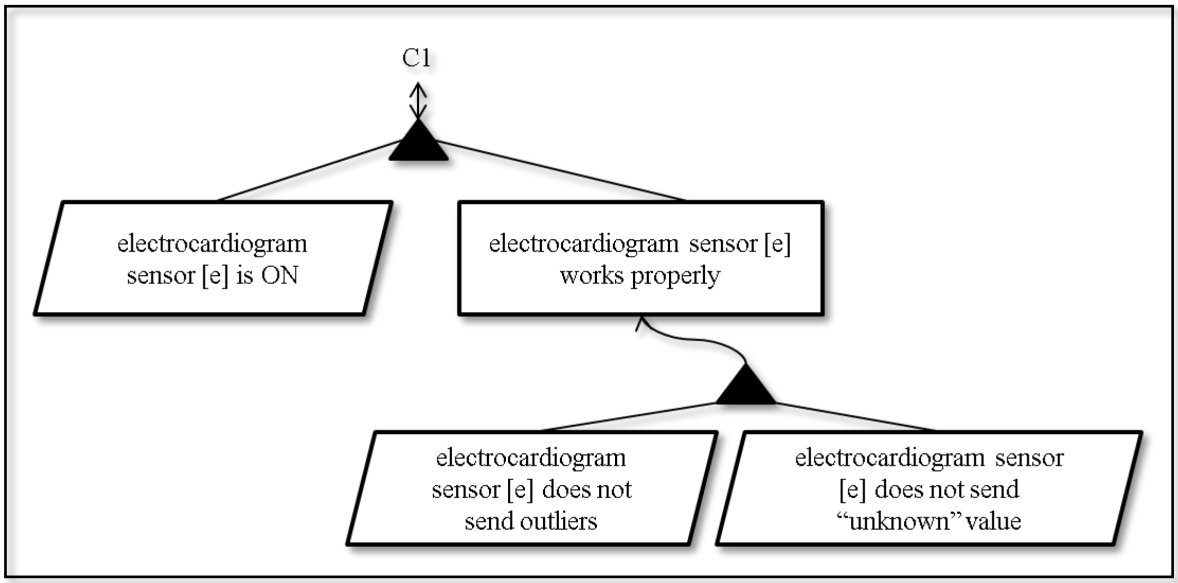

C1
electrocardiogram sensor [e] is ON
electrocardiogram sensor [e] works properly
electrocardiogram sensor [e] does not send outliers
electrocardiogram sensor [e] does not send "unknown" value

Fig. 18: Electrocardiogram sensor is OK – Context Model

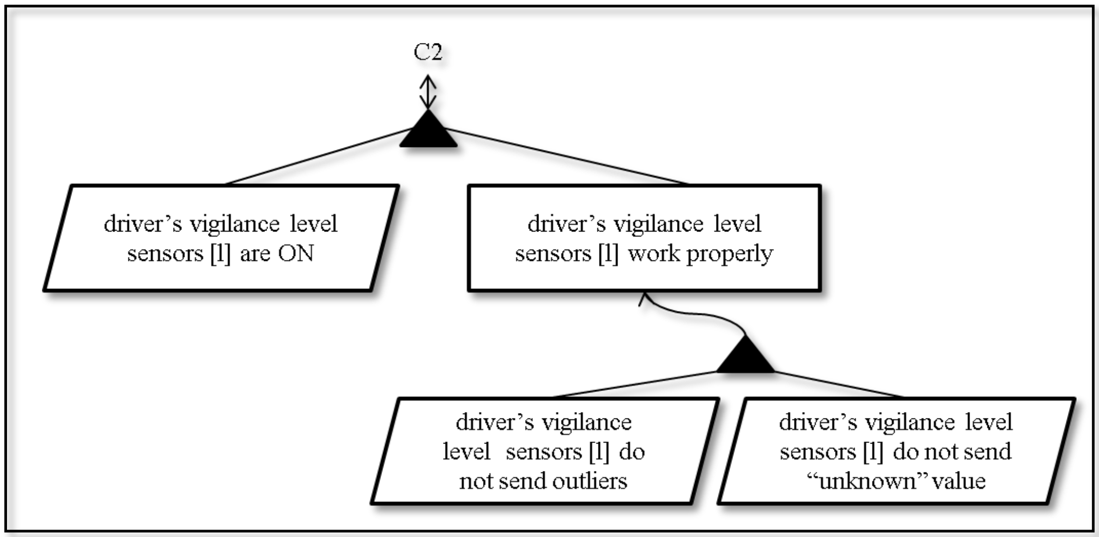

C2
driver's vigilance level sensors [l] are ON
driver's vigilance level sensors [l] work properly
driver's vigilance level sensors [l] do not send outliers
driver's vigilance level sensors [l] do not send "unknown" value

Fig. 19: Driver's vigilance level sensors are OK – Context Model

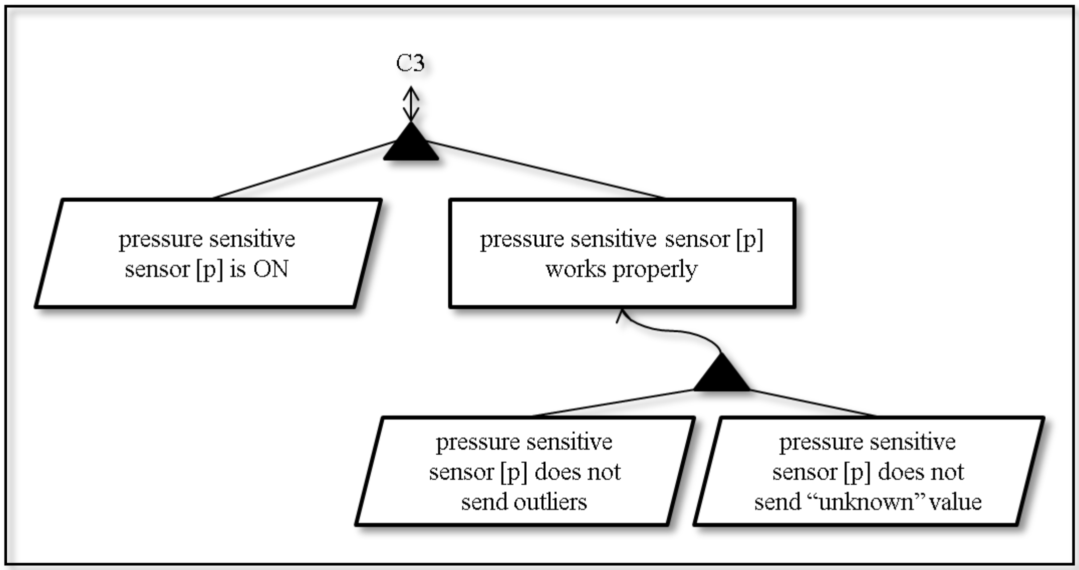

C3
pressure sensitive sensor [p] is ON
pressure sensitive sensor [p] works properly
pressure sensitive sensor [p] does not send outliers
pressure sensitive sensor [p] does not send "unknown" value

Fig. 20: Pressure sensitive sensor is OK – Context Model

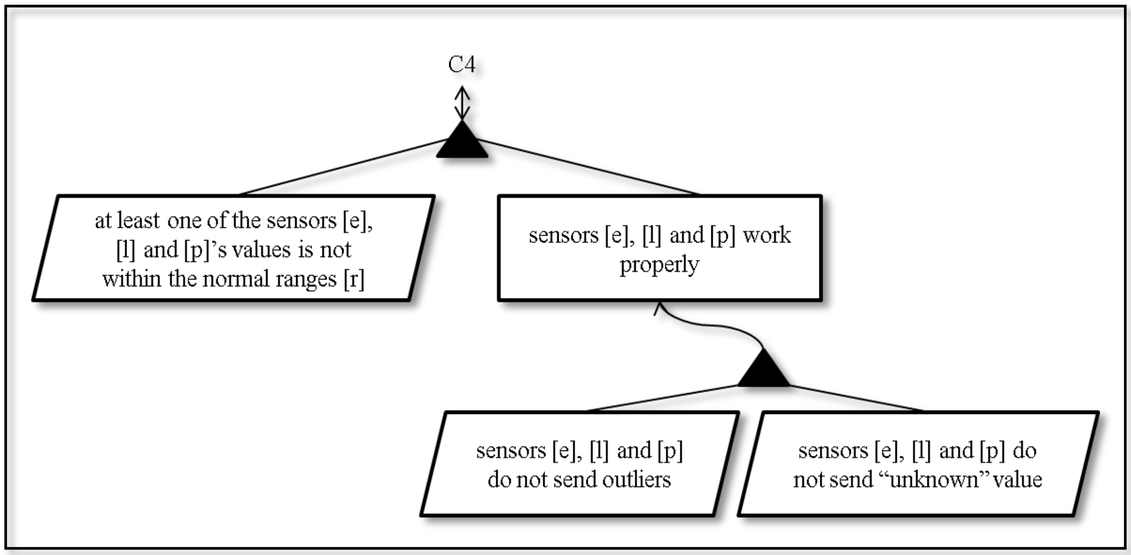

C4
at least one of the sensors [e], [l] and [p]'s values is not within the normal ranges [r]
sensors [e], [l] and [p] work properly
sensors [e], [l] and [p] do not send outliers
sensors [e], [l] and [p] do not send "unknown" value

Fig. 21: Driver shows drowsiness symptoms – Context Model

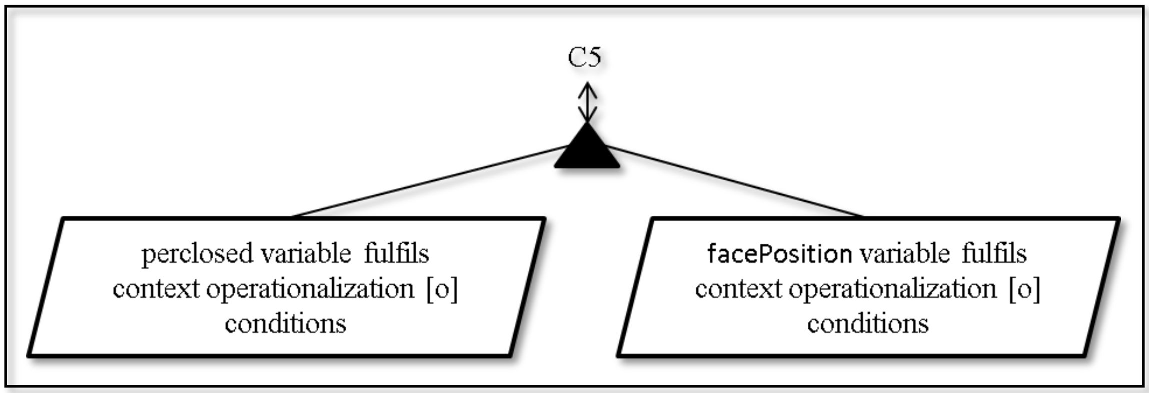

C5
perclosed variable fulfils context operationalization [o] conditions
facePosition variable fulfils context operationalization [o] conditions

Fig. 22: Driver is tired – Context Model

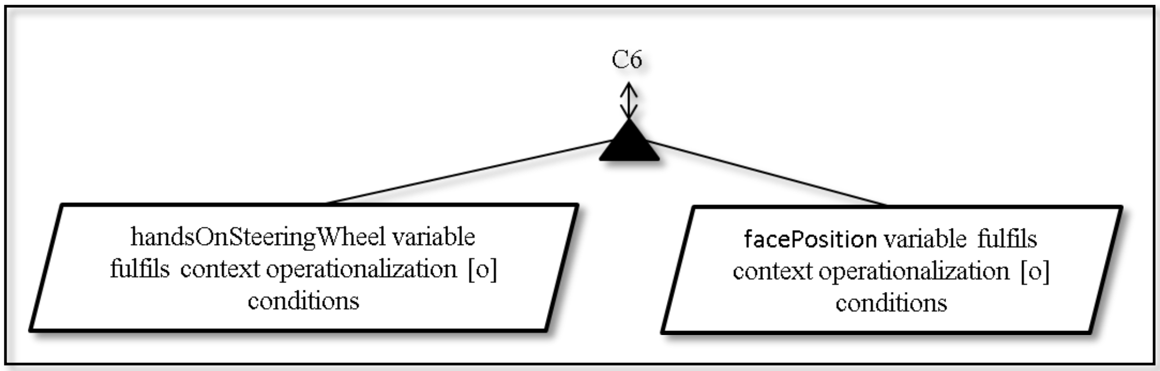

C6
handsOnSteeringWheel variable fulfils context operationalization [o] conditions
facePosition variable fulfils context operationalization [o] conditions

Fig. 23: Driver is dangerously tired – Context Model

For the demonstration of SACRE presented in this work and taking into account the models above, three contextual requirements have been derived. These contextual requirements as well as their operationalization are introduced as an initial configuration parameter to the tool. The operationalization of a contextual requirement consists of all the variables that should be considered (called *variables involved*) in order to describe the context of the contextual requirement and the values (that can be range of values) of the variables that allow the system to decide whether a context holds or not [12]. The operationalization could be represented as an arbitrary formula but always should result in a true or false response. In SACRE, the operationalizations' formulae are logical aggregations (ANDs) of the evaluation of the variables values (normalized).

The set of contextual requirements as well as their operationalization are shown in Table 2. The rightmost column of the table shows the variables' values before the normalization. The normalization has been done taking into account the same minimum and maximum variables' values stated in the sensors module's policy explained before.

Table 2: SACRE's contextual requirements

CR	Context	Behaviour	Operationalization	Operationalization (non-normalized)
cr_1	Driver is tired	Activate seat vibration alarm	Perclosed>0.15 AND FacePosition=1	Perclosed>15% AND FacePosition=Non-frontal
cr_2	Driver is dangerously tired	Activate sound-light alarm	FacePosition=1 AND HandsOnSteeringWheel<1	FacePosition=Non-frontal AND HandsOnSteeringWheel<2
cr_3	Driver is sleeping	Activate support lane keeping	Perclosed>0.25 AND HeartBeatsPerMinute<0.46 AND HandsOnSteeringWheel<1	Perclosed>25% AND HeartBeatsPerMinute<55 AND HandsOnSteeringWheel<2

The contextual requirements of SACRE are aimed to detect and support drowsy drivers. Since the description of a context when a driver is tired, dangerously tired or asleep is subjective and depends a lot on the context (user, sensors measures accuracy, etc.) it is a great example in order to validate ACon. The operationalization of these three contextual requirements is changing constantly at runtime, since the interaction between the driver and the actuators use to be high. Moreover, the drivers' state could be misunderstood easily and should be adapted frequently.

It can be noticed that the sensors related with the support lane keeping are not involved in the contextual requirements. This work considers that the requirements related to maintaining the vehicle in the lane don't depend on context; what is depending on context is the activation or not of the support lane keeping capability. This means that the functionality of maintaining the vehicle in the lane is in the same set of requirements as for example that the user can open automatically the windows or that the vehicle must has an USB input, etc.

The variables describing the contexts of the contextual requirements are the ones calculated by the sensors module and do not always match exactly with the sensors in the specific domain application. For example, the driver's vigilance level sensor used to measure the eye state is not present in the contextual requirements table directly. The perclosed variable (percentage of the time the eyes have been "closed"), based on [25] but renamed for this work because the calculations differences, accumulates the eyes state along the whole execution and reports the percentage of this accumulation where the eyes were in closed state.

In order to calculate the perclosed, the following steps are done:

1. Based on the visible driver's pupil ratio at *x* time reported by the driver's vigilance level sensor:
    - An open state is accumulated if the visible driver's pupil ratio is more or equal than 40%.
    - A closed state is accumulated if the visible driver's pupil ratio is less than 40%.
    - No transition states are considered for this work.

2. A ratio between the accumulated closed and open states regarding the total elapsed time is calculated and reported as the perclosed measure.

For example, if have elapsed three iterations and after apply the discriminator criteria (i.e. 0%-40% the eyes are closed and 40%-100% the eyes are open, considering that the eyes blink at the same time or that the percentage represent an average of both eyes states) the eyes state result in: open, closed, open; the perclosed would report 33.33% which is the percentage of time (simplified by discrete iterations) that the eyes were closed.

For the SACRE implementation some assumptions not clarified in [12] should be used in order to manage the translation of the sensors values in variables. For example, the sensors module based on its policy element revises if a sensor is malfunctioning; if it is malfunctioning the sensors' module does not consider it for the variables' calculation. Later on, the monitor based on its own policy element decides if the variables' values do not correspond to normal values and classify these ones into the corresponding symptom (outlier, malfunctioning, etc.).

With this assumption, the translation of the symptoms to specific sensors is discarded being its relation one to many and not one to one. This factor could limit the SACRE's adaptation performance when trying to explain with precision what is exactly wrong in the specific domain application. The values of the variables in Table 1 have been not exhaustively studied since in any case they will be adapted at runtime, however they are based on the literature mentioned before in which each sensor interpretation is based (section 3.1.2).

3.4 Functionalities

The SACRE tool is presented to the user as an interactive dashboard. This display shows the most important variables of the simulated part of a smart vehicle. Moreover a second display is provided showing graphically the position of the vehicle in the road. The communication between both displays is really important since a change in any of the displays related with the position of the vehicle in the lane should be communicated to the other display immediately in order to create a sense of realism to the user. Furthermore the interactive display should communicate with the smart vehicle controller as it has been explained in previous sections.

The changes in the controls' display basically correspond to accelerate, brake and rotate the steering wheel; while the changes in the second vehicle and path display correspond to the calculation of lateral and frontal distance and the deviation angle regarding the lane shape (graphically approximated) taking into account the new values of the accelerator, brake and steering wheel provided by the controls' display. In order to keep it simple, the small vehicle position changes are not considered. The accelerator, brake and steering wheel can be manipulated from different sources, more details are explained below.

SACRE offers a set of user-manageable sensors and actuators. The degree of the manageability depends on the setting selected at the initial stage of the tool, shown in Figure 24. There are two options:

a) *manual*, which allows the user to have full control of the application, changing the values of the variables as s/he wants, except the dependent variables such as lateral and frontal distances and the deviation angle which are calculated based on the input of the user for the rest of the variables. In this setting the user accelerates, decelerates and rotates the steering wheel using the keyboard.

b) *automatic*, an option without user assistance through a set of predefined scenarios running automatically in the tool. The independent variables such as: acceleration, brake and steering wheel rotation are read from a file, simulating the user input.

In both cases the user can manipulate the actuators (support lane keeping, vibration alarm, and sound-light alarm) which will trigger the adaptations and for commodity the values regarding the user state are extracted from a file. Also in both cases the user receives the feedback about the contextual requirements evolution through the variables visualization and can evaluate the system performance comparing the decisions taken by the tool against the expected ones. However, at runtime sometimes it is difficult to follow the adaptations; thus SACRE, for evaluation purpose, provides a historical record of the adaptations (among other data) generated during the execution, showing them in console. Figure 25 shows an example of this output.

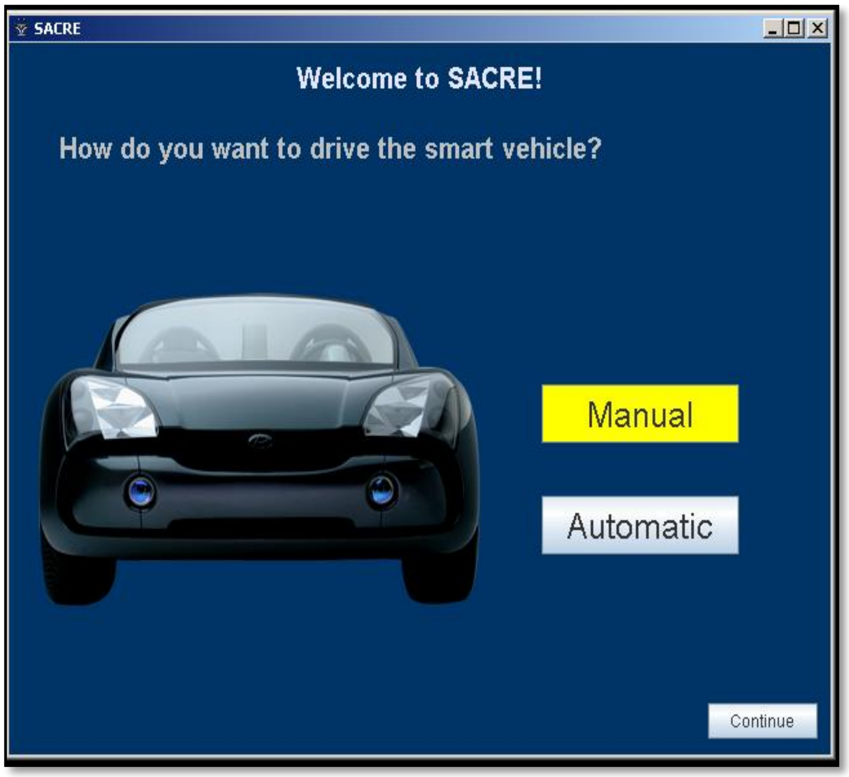
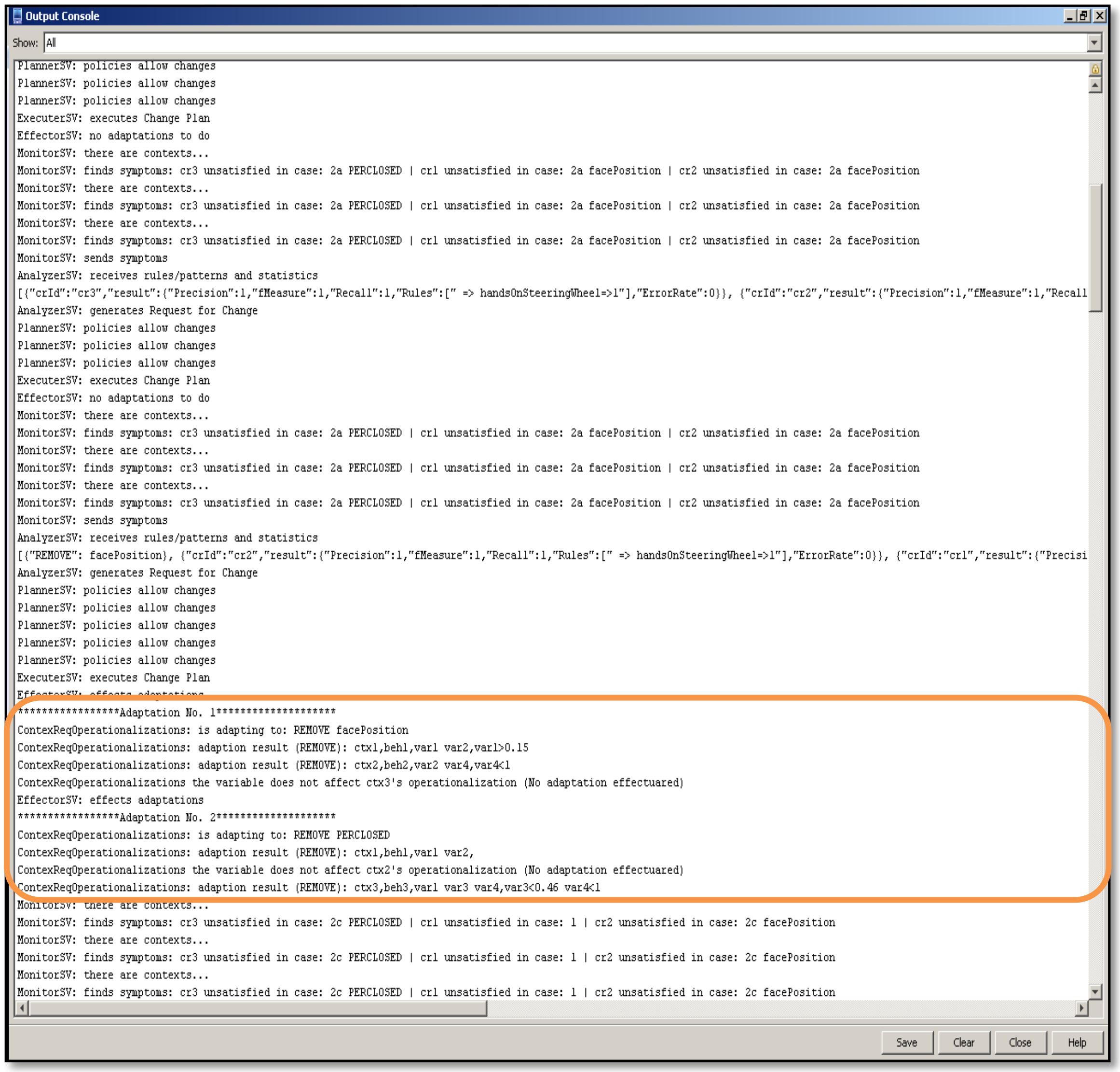


Fig. 24: SACRE setting selection



```
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: no adaptations to do
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":1,"fMeasure":1,"Recall":1,"Rules":[" => handsOnSteeringWheel=>1"],"ErrorRate":0}}, {"crId":"cr2","result":{"Precision":1,"fMeasure":1,"Recall
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: no adaptations to do
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"REMOVE": facePosition}, {"crId":"cr2","result":{"Precision":1,"fMeasure":1,"Recall":1,"Rules":[" => handsOnSteeringWheel=>1"],"ErrorRate":0}}, {"crId":"cr1","result":{"Precisi
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
*****************Adaptation No. 1*********************
ContexReqOperationalizations: is adapting to: REMOVE facePosition
ContexReqOperationalizations: adaption result (REMOVE): ctx1,beh1,var1 var2,var1>0.15
ContexReqOperationalizations: adaption result (REMOVE): ctx2,beh2,var2 var4,var4<1
ContexReqOperationalizations the variable does not affect ctx3's operationalization (No adaptation effectuared)
EffectorSV: effects adaptations
*****************Adaptation No. 2*********************
ContexReqOperationalizations: is adapting to: REMOVE PERCLOSED
ContexReqOperationalizations: adaption result (REMOVE): ctx1,beh1,var1 var2,
ContexReqOperationalizations the variable does not affect ctx2's operationalization (No adaptation effectuared)
ContexReqOperationalizations: adaption result (REMOVE): ctx3,beh3,var1 var3 var4,var3<0.46 var4<1
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
```

Fig. 25: SACRE historical adaptations record

SACRE provides a second menu where a user can select the path style s/he wants the vehicle to follow and the context conditions for different driver states (e.g., tired driver). An instance of this menu is shown in Figure 26. As it can be noticed the number of configuration parameters in SACRE is high and if the interaction with the user is aggregated it is obvious that each execution of SACRE will be different. The possible adaptations as well as their order are unfeasible to calculate, recurrent characteristic in self-adaptive software.

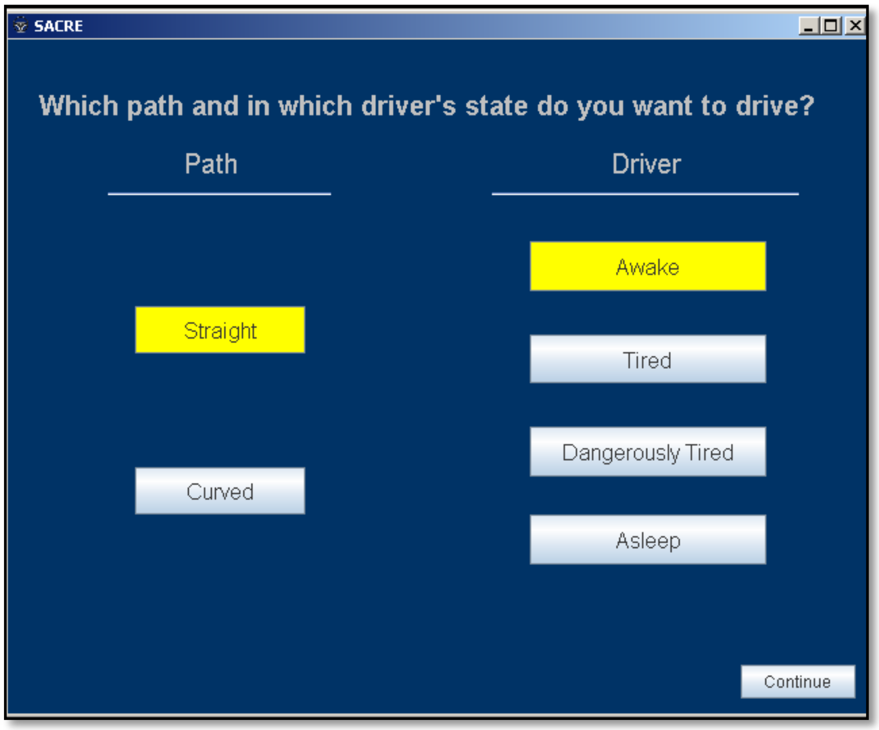


Fig. 26: SACRE path and driver's state menu

Figure 27 shows the controls' display of SACRE, managed by the smart vehicle view module. This display contains the values of the following variables:

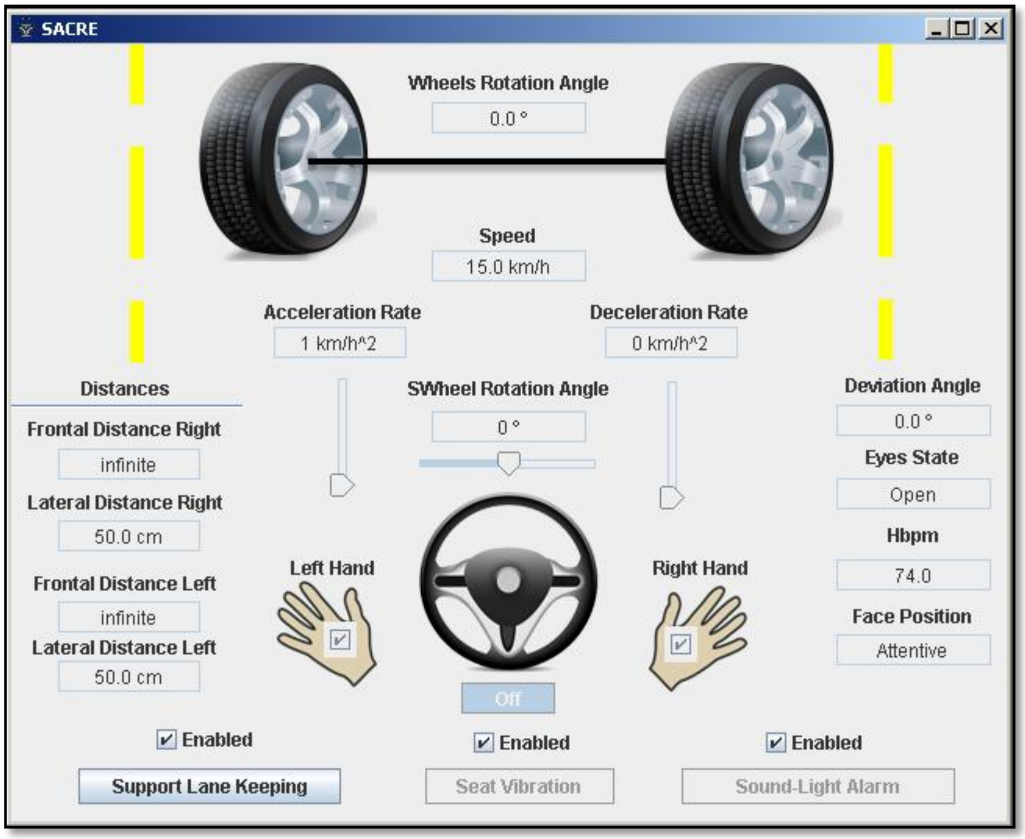
- Regarding the driver:
  - eyes state
  - face position
  - hands on the steering wheel
  - driver's heart beats per minute (hbpm)

- Regarding the vehicle's position:
  - lateral and frontal distances between the vehicle and the street's lateral lines
  - deviation angle of the vehicle regarding the estimated correct position it should have

- Regarding the vehicle's controls:
  - speed
  - acceleration rate
  - deceleration rate

- Regarding the actuators' state:

- support lane keeping
- sound-light alarm
- seat vibration alarm


SACRE

Wheels Rotation Angle: 0.0 °

Speed: 15.0 km/h

Acceleration Rate: 1 km/h^2

Deceleration Rate: 0 km/h^2

SWheel Rotation Angle: 0 °

Distances

Frontal Distance Right: infinite

Lateral Distance Right: 50.0 cm

Frontal Distance Left: infinite

Lateral Distance Left: 50.0 cm

Left Hand ✓

Right Hand ✓

Off

Deviation Angle: 0.0 °

Eyes State: Open

Hbpm: 74.0

Face Position: Attentive

✓ Enabled — Support Lane Keeping

✓ Enabled — Seat Vibration

✓ Enabled — Sound-Light Alarm

Fig. 27: SACRE controls display

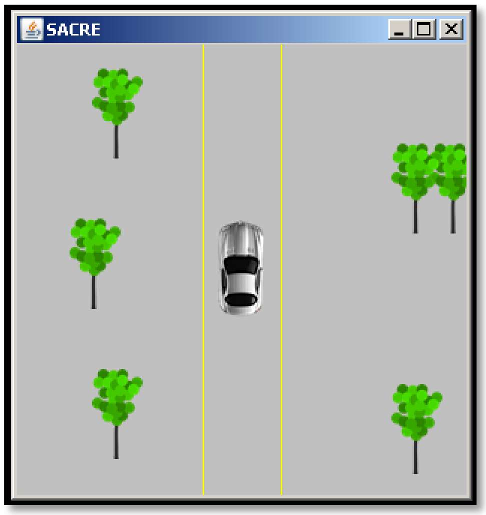


Fig. 28: SACRE air street view

The secondary SACRE's display is provided in order to illustrate the vehicle position on the street; an instance of this display is shown by Figure 28. This display responds to the accelerator, brake and steering wheel changes moving graphically the vehicle in the road according them. This display should be understood as a complementary evidence of SACRE intended to arise the interest of the stakeholders involved in the topic of self-adaptive systems and to demonstrate its applicability in real world implementations. Once the support lane keeping is activated it is in charge of maintain the vehicle in the lane taking into account the calculation graphically generated by this display. Later, in the implementation section more details about this display are provided. Moreover, a use case diagram describing all of these functionalities is provided in Appendix A.

3.5 Implementation

SACRE is a proof-of-concept implementation of the ACon approach and as it has been mentioned in Section 3.1.2 it is placed in the application domain of smart vehicles. The software developed for smart vehicles is usually done in the environment of embedded systems. The smart vehicles as well as other smart embedded devices nowadays should provide more and better functionalities to the users using less and less amount of resources.

3.5.1 Technologies

In the case of SACRE's implementation the resources limitation is taken into account; nevertheless, for demonstration and testing purposes a part of the tool should be implemented as a desktop application. Moreover, a way to simulate the embedded device is desirable in order to monitor its adaptation performances. Finally, an important fact should be taken into account, the data mining tool Weka offers an API not suitable for embedded environments because its size (~6MB).

3.5.1.1 Rationale

According to [31], embedded system applications were once written in assembly language. But as embedded computer hardware resources grew, pressures such as time-to-market and development cost drove a shift to the more portable and less error-prone C and C++ languages. These languages required more memory and more CPU cycles, but declining hardware costs and rising software complexity made the trade worthwhile.

The C language currently is the most commonly used programming language for embedded software development and gives the programmers a relatively extensive writing flexibility, accordingly to [32]. However, this flexibility could affect the system's performance since the code quality will highly depend on the programmer. On the other hand, Java language has become more and more popular in the last years, gaining a great worldwide population of developers trained in enterprises and schools. The consideration of Java in the embedded world also has increased and now is an attractive alternative.

Some advantages of Java over C and C++ in embedded systems, based on [31], are:

- Java is a modern, object-oriented, language without the error-inducing complexity of C++. Objects are natural representations for sensors and actuators.

- Java is less prone to errors than C. For example, there are no pointers, and memory management is automatic.

- Highly portable; Java classes do not need to be recompiled to run on a different CPU or operating system.

- Security, oriented. Java libraries support encrypting sensitive data sent to or from embedded devices, and validating digitally signed code downloaded to update or extend embedded applications in the field.

- Multi-threaded, enabling the natural expression of parallel activities and their simultaneous execution on platforms whose operating system thread model takes advantage of multiple CPU cores.

- Equipped with a large collection of OS-independent libraries including database access and graphical user interfaces.

- Tunable to match hardware resources and application needs. For example, there are multiple options for runtime compilation to native instructions, and for how and when unused (garbage) memory is reclaimed.

- Extensible with native methods written in C that interact with special purpose embedded system hardware.

- Debuggable on the desktop and remotely. A desktop Java Runtime Environment has the same APIs as one that runs on an embedded computer, except for hardware-specific interfaces and behaviours. Most functional debugging can be done on a desktop computer. An embedded system that has a network connection can be debugged and profiled remotely.

As it is stated before Java's popularity is really high and the complexity of its use in embedded systems is relatively low against C and C++ according to the points enlisted before. Taking into account these considerations Java seems the better choice for SACRE's implementation. Moreover other reasons could be provided:

1) The stronger programming abilities of the author of this work are placed in the Java world. Even a special edition for embedded system should be used, the learning curve is preferable than the one for starting with other languages.

2) The Weka tool API is offered to be used in Java code, so the use of Java in all the modules will facilitate the integration of SACRE's.

3) Using Java, the simulation of the embedded device as well as the creation of the view modules are trivial.

3.5.1.2 SACRE

3.5.1.2.1 Modules

Taking into account the justification presented in the previous section, a mixture of Java technologies are used for the SACRE's implementation which is presented in this section. The advantage of this implementation is its proximity to a real world embedded application since it has been developed in an emulated embedded device. This argument corresponds to the core of SACRE (the adaptation feedback loop and managed elements controllers), but not to the graphical part of the smart vehicle (the smart vehicle view modules) which in theory simulate the device hardware visualization. Code for controlling the real hardware, i.e. LEDs, sounds, etc. should be developed in order to substitute the simulated actuators in the SACRE's view.

SACRE was implemented as follows:

➢ Java ME (<u>Java</u> <u>M</u>icro <u>E</u>dition Embedded) 8.1 has been used for the emulated embedded application that contains the adaptation feedback loop modules, specifically the MAPE-K loop modules plus the policies documents, and the Managed Elements Controllers.

➢ Java SE (<u>Java</u> Platform, <u>S</u>tandard <u>E</u>dition) 1.8 has been used for develop the smart vehicle view modules (both displays) and for develop two RESTful services which fulfil the following tasks:
    - *A smart vehicle middleware service* designed and developed for communicate the Java ME smart vehicle controller module with the Java SE smart vehicle view module containing the controls; and,

- *A data mining service* with the objective of communicate the Java ME data mining module that controls the data source and symptomatic contextual requirements with the Java SE data mining module that manage the Weka API usage in order to execute the JRip algorithm over the data.

The sensors, contextual requirements and smart vehicle controller and view modules were been designed to follow the pattern of observer-observable in order to keep updated context data and to manage the changes in the displays. In the case of the Java ME modules, this pattern has been developed from scratch (as well as other functionalities) because it is not included in the Java libraries available.

3.5.1.2.2 Data persistence

Another important topic to discuss is the data persistence process which in the case of SACRE requires a lot of resources. SACRE's adaptations performance is based on the correctness of the data mining results. In turn, the quality of the data mining results for the JRip algorithm depend on the amount of data it has in order to find patterns, as much data as better results it provides. Taking into account the dependencies mentioned above, a really reliable and optimized way to persist the data in SACRE should be used.

In order to tackle this challenge, SACRE uses the Java ME API: RMS (Record Management System). RMS provides the functionality of store application data that persist across invocations. It is a very simple record store that can be seen as a simple database which consists on key-value style data storage. The greatest problem is that the keys are automatically assigned reducing its utility.

If more complex data should be stored, for example involving more than one column or even worse, if complex calculations like aggregation, join, etc. should be executed, they have to be managed by code from scratch. Moreover, the data stored in each value of a record has a limited size (maximum 32 characters). Even if RMS seems highly restrictive, it allowed executing successfully the testing phase of SACRE. For sure, in real world applications the amount and complexity of data is much more higher which will obligate the developers to use other techniques of data persistence, totally out of the scope of the proof-of-concept implementation shown in this work.

The SACRE's architecture presented in section 3.2 is now provided for clarification purposes, in Figures 29, 30 and 31, at lower level of abstraction exposing the places occupied by the technologies discussed in this implementation section.

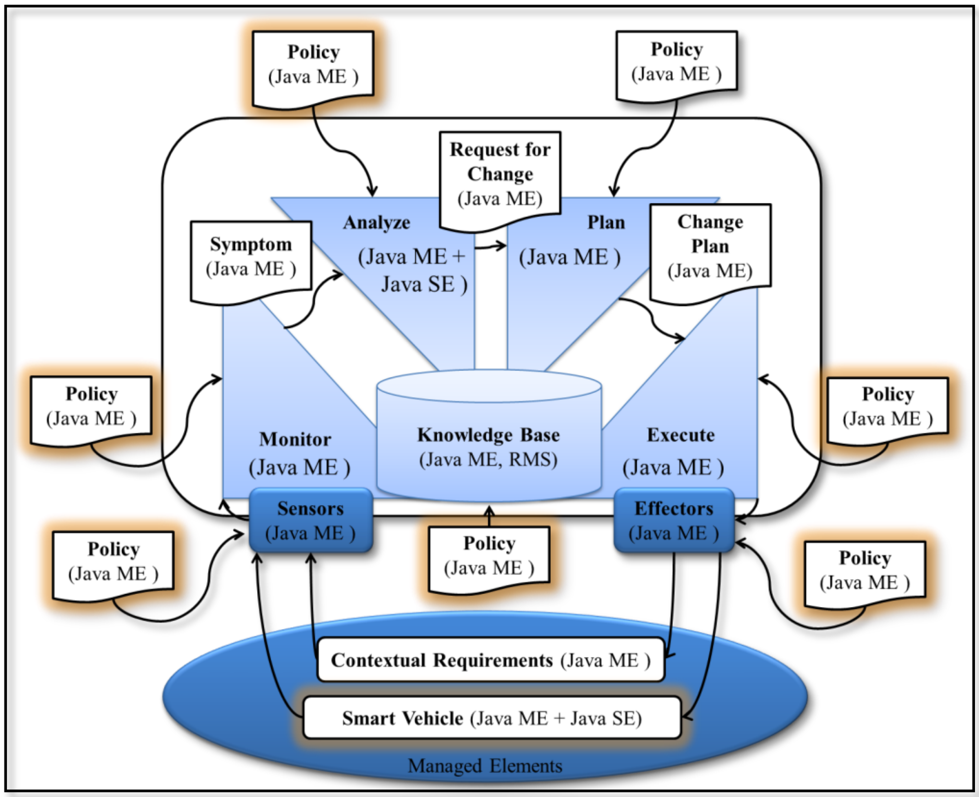


Fig. 29: Adaptation feedback loop in SACRE (technologies)

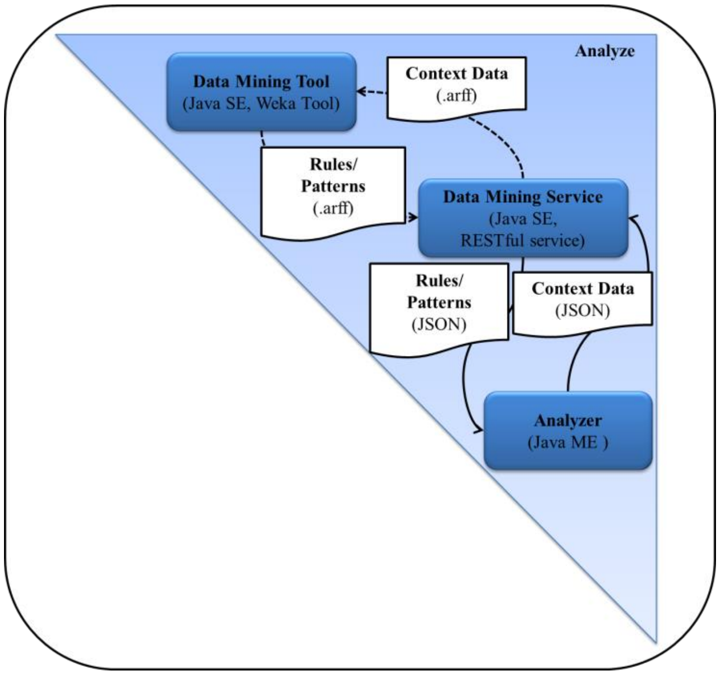


Fig. 30: Expanded analyzer module (technologies)

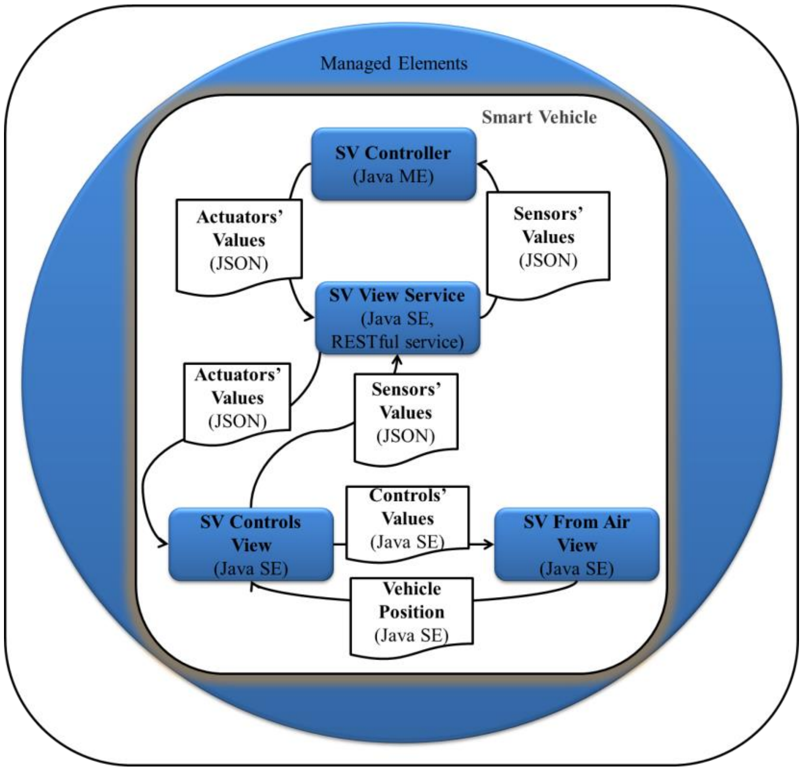

Managed Elements
Smart Vehicle
SV Controller (Java ME)
Actuators' Values (JSON)
Sensors' Values (JSON)
SV View Service (Java SE, RESTful service)
Actuators' Values (JSON)
Sensors' Values (JSON)
SV Controls View (Java SE)
Controls' Values (Java SE)
SV From Air View (Java SE)
Vehicle Position (Java SE)

Fig. 31: Expanded application domain module (technologies)

3.5.2 Challenges

From the previous sections some challenges could be already foreseen. Apart of those challenges, for example the device's resources limitation, some others have emerged along the SACRE's design and implementation. In this work the challenges have been divided in two kinds: technical and domain-related. Even if both kind of challenges depend on the specific domain application selected, the first ones refers to the technologies and techniques used for implement SACRE; meanwhile, the second ones regard to the implications of simulate SACRE's domain because hardware absence, not strictly related with code.

3.5.2.1 Technical

As it has been mentioned before, SACRE's implementation results in a mixture of Java technologies. This situation carries the following technical challenges:

- Combining Java ME and Java SE without affecting the performance of the whole application. The combination is effectuated through the communication channels established by the RESTful service which sometimes could experience a de-synchronization due to the unsynchronized calls by both parts. Issue already fixed, but that could add delays of data freshness or time response.

Moreover, these services are highly loaded with calls because the cycles of the system are short, for example: the smart vehicle controller asks every second to the view (through the corresponding service) the sensors' values; and, the monitor element requests the context to the sensors module every 2 seconds which later trigger a data mining service call by the data mining module embedded in the analyzer element.

- Finding the common characteristics between Java ME and Java SE environments and exploit them in order to be transparent for the user. For example, the information interchange between the modules, through the RESTful services, is done in a common structure of data: JSON. This decision facilitates the avoidance of data misunderstanding, data incompleteness, etc. Problems usually experienced when transform data from one structure to another.

- Maintaining as much modules as possible in the Java ME environment in order to provide a lightweight application. A great effort has been done in order to maintain as much modules as possible. The only modules out of the Java ME are the views of the specific domain application, which don't should affect the cycle performances, the service that communicates them with the cycle, which would not exist in case the views would be removed. And, the data mining service convenient for its substitution or even for have more than one data mining provider serving the adaptation feedback loop.

-

3.5.2.2 Domain

SACRE implements a part of the functionalities of a smart vehicle. The tasks in charge of the sensors and actuators should be simulated in some way, as well as the scaled view from the air of the vehicle in the path. The challenges carried by the implementation of these simulations are following explained.

- *Simulating sensors.* The sensors mentioned in section 3.1.2 should provide realistic measures in order to simulate the sensor hardware behaviour in real-world. For the sensors related to measure the driver state, the measures' values were based in previous works for each of the specialized areas: for example, drowsy people eyes behaviour studies, heart beats behaviour experimental studies when people is asleep and awake, etc. and the sensors currently available in the market for these purposes in order to increase SACRE’s applicability in real world. The challenge with these sensors is to simulate a long period driver’s behaviour according to the states considered by this demonstration (awake, tired, dangerously tires and asleep) without overlap and create confusion of the driver’s state that is intend to be simulated. This task is challenging because even in real-world these states are subjective and not well delimited.

Regarding the sensors aimed to report the vehicle position: lateral and frontal distances and the deviation angle; the challenges emerged from the measures simulation and their calculation at any time taking into account user input (or, simulated user input) and the scaled view. In real-world the sensors would report the measures to SACRE which could delimit its scope to only process these measures and propose the adaptations; but, for this demonstration an approximation has been calculated based in the smart vehicle and path view through trigonometry theory principles, as shown in Figure 32, 33 and 34. The approximation of the measures calculations is necessary because the inter repaint time and all the communication this displays has to managed.

- *Simulating view from air of the vehicle in the path.* This simulation is totally out of the main purpose of SACRE. Nevertheless, it is a really useful complementary evidence of its performance, particularly when the support lane keeping is activated. This simulation requires the study of graphics theory and a little incursion in video games principles. Moreover, the study of automotive physics has been needed in order to maintain a minimum degree of realism. This part of SACRE has been developed using the Java 2D APIs which also implies some time for its study and testing before reach the final product. Because this view plays a complementary role it has been kept simple.

- *Simulating actuators.* While the vibration and sound-light alarms are easy to simulate (only indicated by the activation or deactivation of a button) the support lane keeping actuator represents more work. The support lane keeping is a trend in the smart vehicles and really attractive for the community. Adding the challenges to calculate vehicle position and simulate it in an aerial view, an algorithm for the support lane keeping actuator should be designed. This algorithm has been kept simple since also is not in the main purpose of this work. The algorithm should take into account soft-requirements such as: safety and realism. The pseudo code of the algorithm used by the support lane keeping is shown in Figure 35.

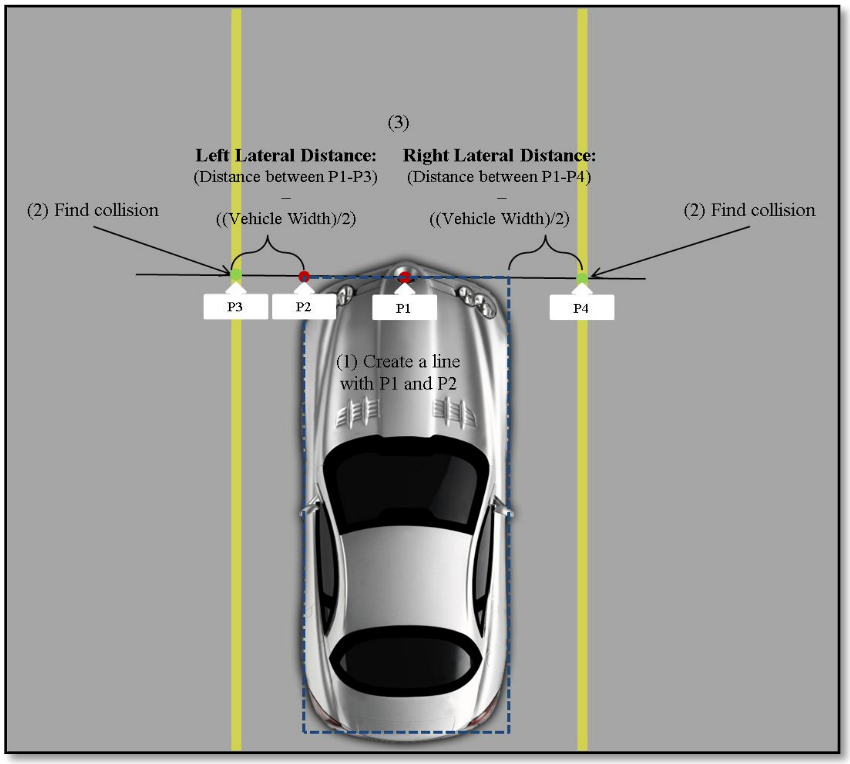


Fig. 32: Lateral distances calculation

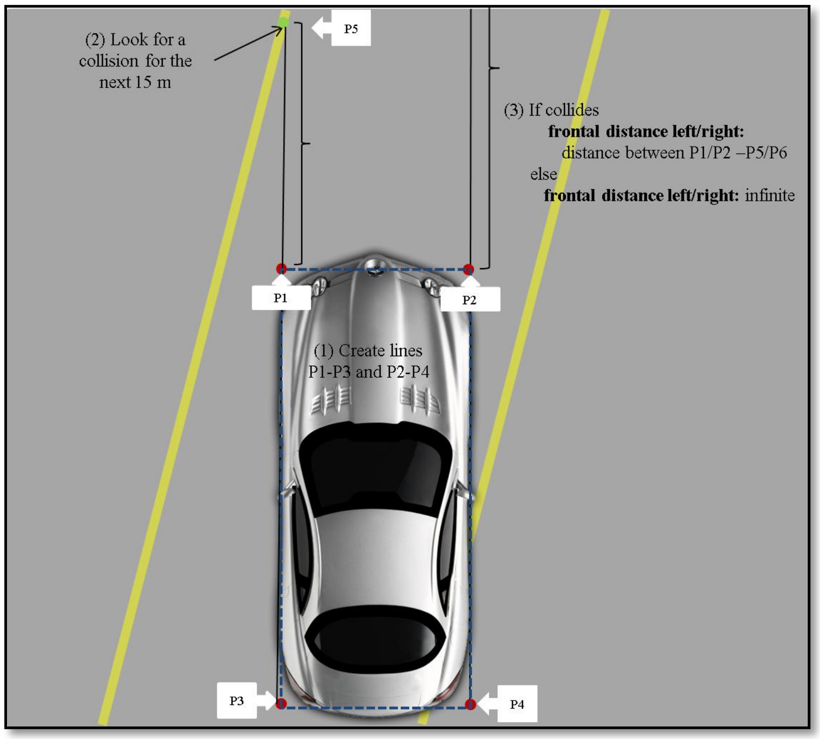

(2) Look for a collision for the next 15 m
P5
(3) If collides
frontal distance left/right:
distance between P1/P2 –P5/P6
else
frontal distance left/right: infinite
P1
P2
(1) Create lines P1-P3 and P2-P4
P3
P4

Fig. 33: Frontal distances calculation

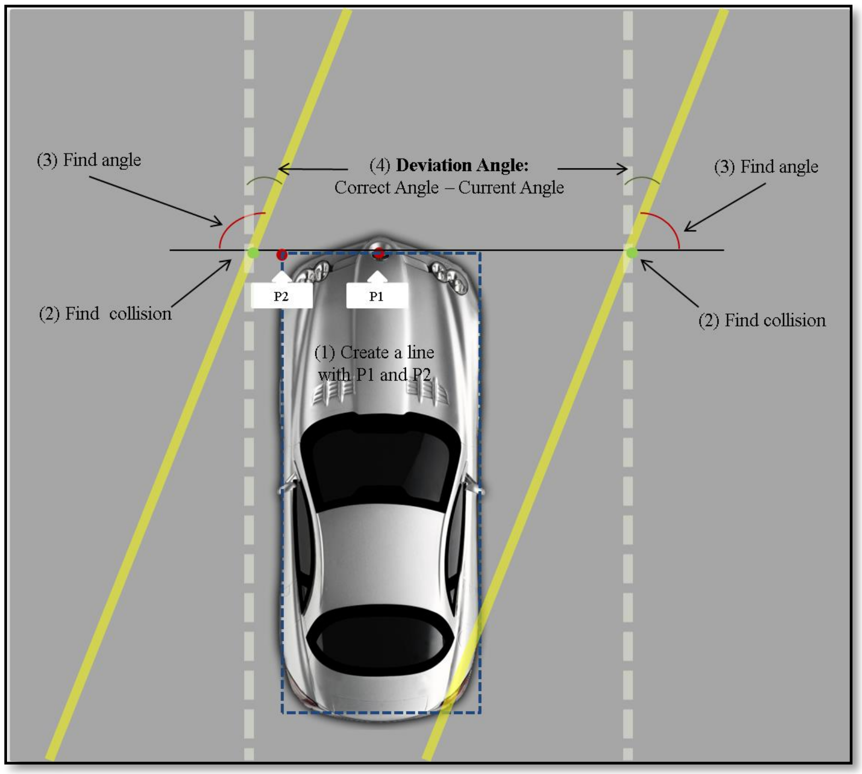


Fig. 34: Deviation angle calculation

```
if Support Lane Keeping is active
    if Deviation Angle != 0
        if Support Lane Keeping Sensors are not OK
            if Vehicle is moving
                stop vehicle
            else
                do nothing
        else
            if Speed > targetSpeedDev
                decelerate at decRateDev km/h^2
            else
                do nothing
        steering wheel rotation = (current steering wheel rotation + deviation angle) * -1
    else
        if Speed > targetSpeedNonDev
            decelerate at decRateNonDev km/h^2
        if Speed < targetSpeedNonDev
            accelerate at accRateNonDev km/h^2
        else
            do nothing
        if Steering wheel rotation != 0
            steering wheel rotation = current steering wheel rotation * -1
        else
            do nothing
else
    do nothing
```

Fig. 35: Support lane keeping algorithm.

Chapter 4

Evaluation

4.1 Experiments

In this chapter a set of experiments executed over SACRE is presented. These experiments not only will be useful to evaluate SACRE's performance, but also to validate the ACon approach implemented by SACRE. Because the contextual requirements designed are oriented to detect and support drowsy drivers, the scenarios of the experiments, in order to test SACRE's adaptations' accuracy, are based on the driver's state changes. The path (curve or straight) and kind of setting (automatic or manual) selected in the scenarios are parameters that do not affect the adaptations but since they are some of the SACRE's functionalities, their performance should be evaluated as well. Finally, the support lane keeping actuator's decisions' correctness, when activated, is neither affecting the adaptations as it has been explained in previous sections; nevertheless, it is evaluated in the scenarios as a complementary functionality.

The scenarios considered by this work have the goal of evaluating the contextual requirements' adaptations at runtime and validate the approach behind these adaptations (ACon). The configuration of the scenarios is presented in Table 3. The table's columns are the possible input parameters of SACRE and the execution duration time of each scenario; while the rows are the initial values for each parameter considered in each scenario. A brief explanation of the scenarios is provided below.

Table 3: Experiments' scenarios

Scenario	Setting	Path	Driver's state	Execution time
1	Manual	Curved	Awake	10 minutes
2	Automatic	Curved	Tired	15 minutes
3	Manual	Straight	Dangerously Tired	10 minutes
4	Automatic	Straight	Asleep	15 minutes
5	Automatic	Straight	Awake*	15 minutes

- *Scenario 1:* The first scenario corresponds to a manual setting which means that the accelerator, decelerator and steering wheel rotation controls' manipulation is in charge of the user. As it has been mentioned before these controls can be manipulated using the keyword (arrows). The path selected is a path with curves that the user must follow with the controls. The driver's state selected is *awake*. This implies that any actuator should be activated by the system since the actuators have been designed for support drowsy drivers. The sensors' values related with the driver's information are read from a file and correspond to values which a person in normal conditions would show if s/he is awake.

  This scenario has been designed in order to demonstrate a normal flow of SACRE's behaviour in which the driver does not interact with the actuators. This means that no adaptation is expected; instead what it is expected is to have all the contextual requirements satisfied during the whole execution. Because of the simplicity of this scenario and the expected absence of adaptations, the execution time has been left to 10 minutes.

- *Scenario 2:* The second scenario corresponds to an automatic setting which means that not only the sensors values defining the driver's state are read from a file but also the values of the controls: accelerator, decelerator and steering wheel rotation. In case the support lane keeping actuator is turned on, by the system or by the user, the controls' values are going to be read from the support lane keeping system and not from the file. Once the support lane keeping is turned off, the controls' values are going to be read again from the file, in this automatic setting.

  The path selected has curves and the driver's state selected is *tired*. This driver's state, according to the initial contextual requirements' operationalization shown in Section 3.3, triggers the vibration alarm actuator. But, in order to incentive an adaptation and evaluate the actuator performance, the support lane keeping actuator is manually turned on (using the mouse) triggering a symptomatic state and letting the system in charge of the controls. It is expected, apart from the adaptation, that the vehicle is maintained in the lane without abrupt changes and procuring driver's and vehicle's safety.

The activation of the support lane keeping would indicate that the driver wants that the system in the future triggers this actuator when s/he experiences the current context and not necessarily when the system detects her/him asleep, as the initial contextual requirements' operationalization indicates. Or also could be understood as that the current context is actually describing a slept driver and not a tired driver from the user's point of view.

In SACRE, if the support lane keeping actuator is activated, any other alarm could not be activated. Particularly, in the case of manual activation of the support lane keeping actuator the contradiction of having it activated instead of other alarms (in case they are triggered by the system and unable to be activated) the system will show symptoms. These symptoms correspond to false *IAFactors* (variable based on [12] defining if a behaviour is true or false) which means that the system will not adapt since some patterns describing when they are true are provided. This situation could be fixed turning off the support lane keeping and letting the alarms turn on when system requires (storing context's patterns where they are on); however, this has not been considered for this experiment.

This scenario has been previously trained with the first scenario, where no actuator is activated. The reason of training the system is to provide to the data mining algorithm more meaningful data to contrast and in consequence facilitate the task of finding patterns. Because of the complexity of this scenario, a period of 15 minutes has been selected for its execution time.

- *Scenario 3:* The third scenario corresponds to a manual setting in a straight path with a dangerously tired driver. This experiment is intended to demonstrate a SACRE's scenario in which the alarms are turned on by the system because driver's tiredness, provoking an ephemeral awake state in the driver who later shows dangerously tiredness symptoms again. This driver's states cycle will be repeated during the whole execution. The alarms will be turned on and off only by the system and the driver will not interact with them. This means that any adaptation is expected to be triggered since the driver considers the system behaviour accurate. The alarms fulfil their aim, wake the driver up, and the requirements should be satisfied all the time.

This scenario is useful for evaluating the initial SACRE's contextual requirements' operationalization performance in normal conditions (without driver disagreements). Because the scenario's simplicity it has been left a period of 10 minutes for the execution time.

Scenario 4: For the fourth scenario an automatic setting and a straight path have been selected. This scenario has a slept driver in the vehicle which should cause the system to eventually turn on the support lane keeping actuator. However, in this scenario, the driver turns off all the alarms at the beginning of the execution and never turns them on again. This could be translated into the situation in which the driver wants to drive the vehicle even if s/he is tired and does not want to experience any alarm; or, that the current context actually is describing an awake driver from this driver's point of view (because the concept of tiredness maybe different from one drive to another).

The driver's actions should trigger some adaptations. The file of asleep driver contains some intervals where the driver is not totally slept which could provoke that from time to time all the requirements are satisfied. But, in general they should be not. This scenario is expected to generate some adaptations in the long term; therefore a 15-minute period of time has been stated for its execution.

- Scenario 5: This fifth scenario corresponds to an automatic setting in a straight path with an awake driver. The peculiarity of this scenario is that the file that reports the sensors' values in charge of monitoring the driver's state contains some wrong values. The errors in the values simulate that the driver's vigilance level sensor is damaged during those periods of time. The facility to select the presence of damaged sensors is not provided to the user in this SACRE's version, but has been developed for testing purposes. Since the file has been designed to report these wrong numbers soon in the execution, the adaptions should not take much time to appear.

  The damaged sensor is expected to be removed from the variables. This will trigger some symptoms that in consequence are expected to provoke the removal of the affected variables from the contextual requirements' operationalizations. After some time, the variables are recovered (the file has normal sensors' values again) and the sensors module and analyzer module include them again in their operations. This eventually should trigger an adaptation that re-integrates them in the contextual requirements' operationalization. A period of 15 minutes has been assigned to this execution.

4.2 Results

This section provides the results of the execution of the scenarios detailed in the previous section. A set of screenshots per experiment is provided in order to show the parameters selected and decisions taken by the user and the system in each scenario. Also a brief description of the results obtained in each scenario is provided. Later, in the *Discussion* section some insights resulting from the results of the experiments are presented.

- Results of Scenario 1. Manual – Curved – Awake

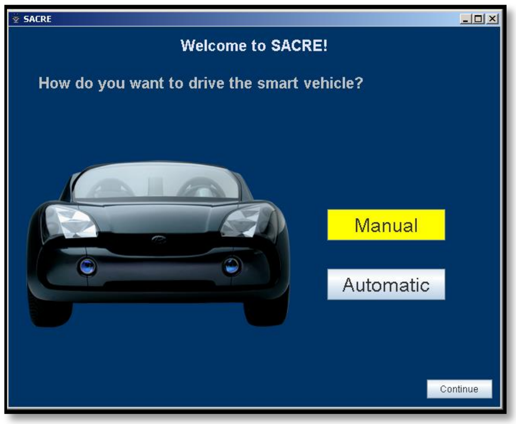


Fig. 36: Manual Setting

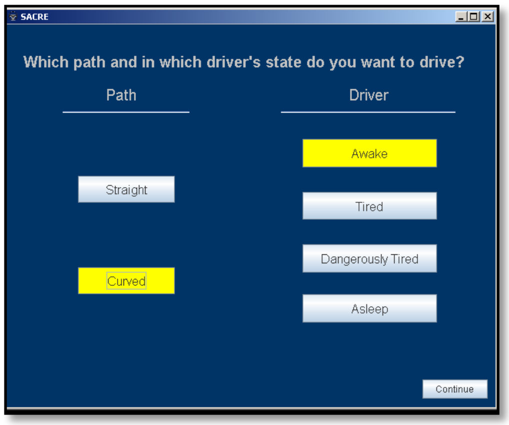
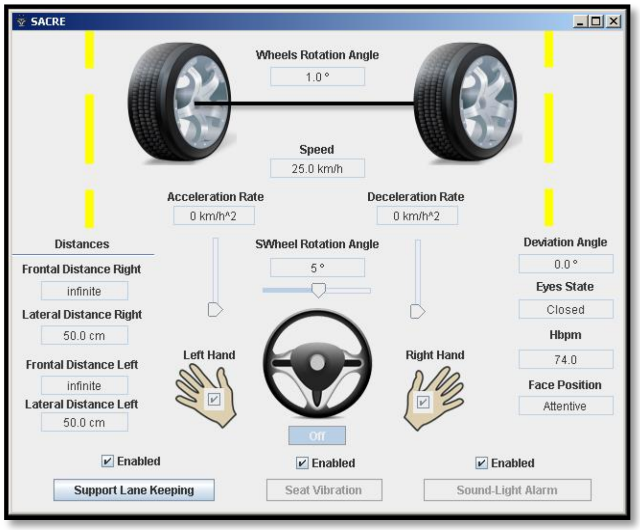


Fig. 37: Curved path and awake driver

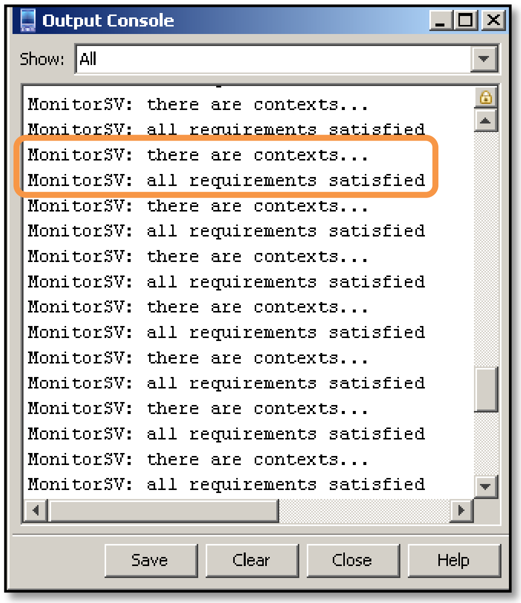
Figures 36 and 37 show the initial parameters selected, for this scenario, in the configuration menus provided by SACRE.



Wheels Rotation Angle: 1.0 °
Speed: 25.0 km/h
Acceleration Rate: 0 km/h^2
Deceleration Rate: 0 km/h^2
Distances
Frontal Distance Right: infinite
Lateral Distance Right: 50.0 cm
Frontal Distance Left: infinite
Lateral Distance Left: 50.0 cm
SWheel Rotation Angle: 5 °
Deviation Angle: 0.0 °
Eyes State: Closed
Hbpm: 74.0
Face Position: Attentive
✓ Enabled ✓ Enabled ✓ Enabled

Fig. 38: SACRE with actuators deactivated

After some minutes of execution SACRE has not experienced any actuators activation as shown in Figure 38. This is because the driver is awake and the actuators have been designed, as mentioned before, for drowsy drivers. In consequence in Figure 39 it can be seen that any adaption or symptom has been detected during this experiment. This is a scenario exemplifying a typical day of use of a smart vehicle, assumed as the most used to be experienced in real world.



```
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
```

Fig. 39: SACRE all requirements satisfied

- Results of Scenario 2. Automatic – Curved – Tired

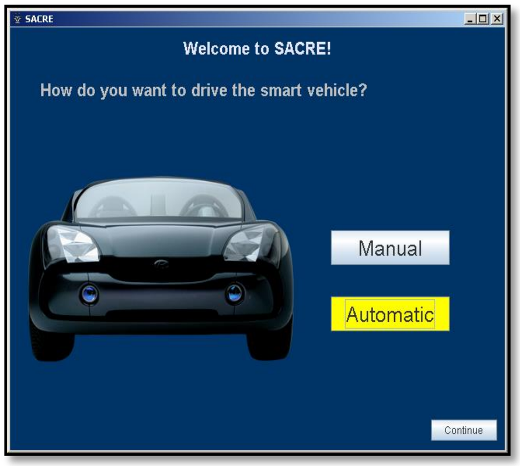


Fig. 40: Automatic Setting

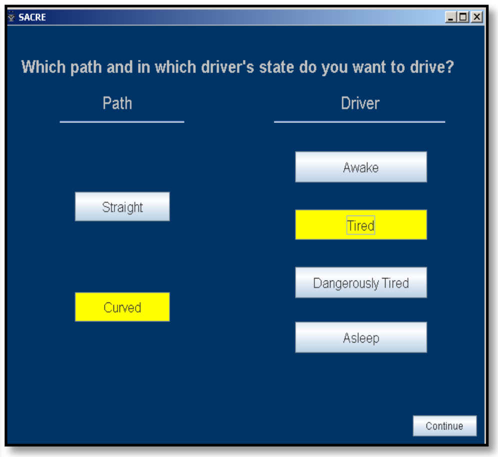
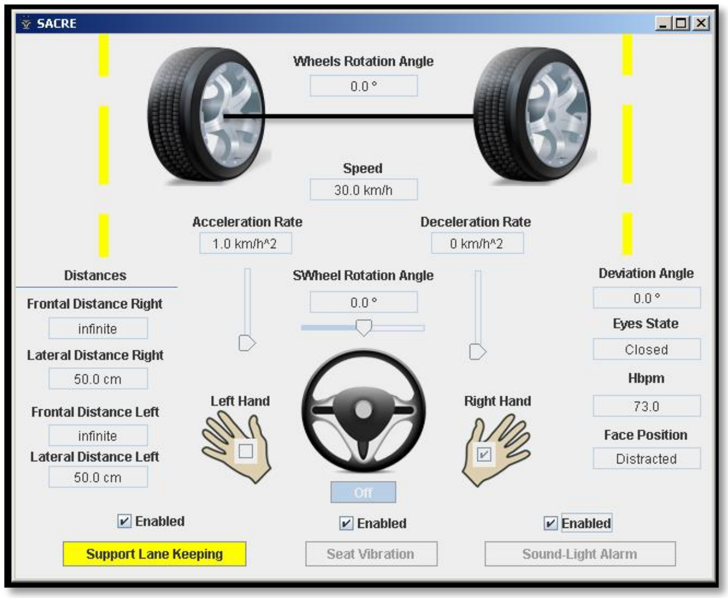


Fig. 41: Curved path and tired driver

Figures 40 and 41 show the initial parameters selected, for this scenario, in the configuration menus provided by SACRE, after the training phase (1 minute). In this experiment the driver's state is tired, and even s/he could well manipulate the controls (accelerator, decelerator and steering wheel) it has been assumed that s/he has decided not to do it and turn on the support lane keeping actuator as shown in Figure 42.


SACRE

Wheels Rotation Angle: 0.0 °
Speed: 30.0 km/h
Acceleration Rate: 1.0 km/h^2
Deceleration Rate: 0 km/h^2
SWheel Rotation Angle: 0.0 °

Distances
Frontal Distance Right: infinite
Lateral Distance Right: 50.0 cm
Frontal Distance Left: infinite
Lateral Distance Left: 50.0 cm

Deviation Angle: 0.0 °
Eyes State: Closed
Hbpm: 73.0
Face Position: Distracted

Left Hand ☐
Right Hand ☑
Off

☑ Enabled Support Lane Keeping
☑ Enabled Seat Vibration
☑ Enabled Sound-Light Alarm

Fig. 42: SACRE with support lane keeping activated

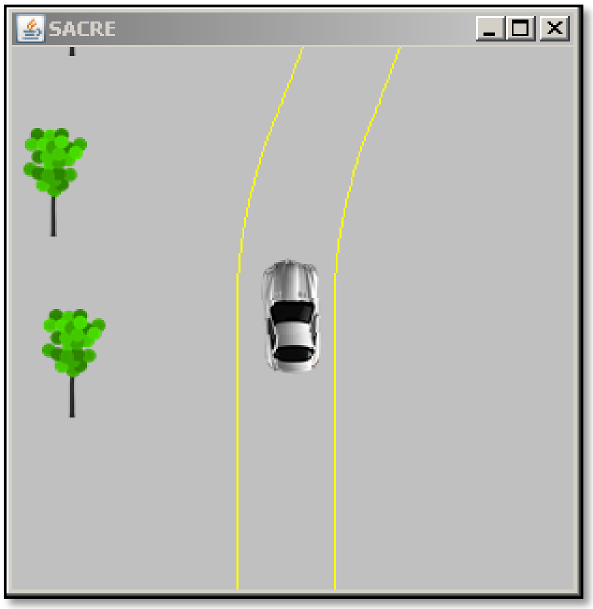
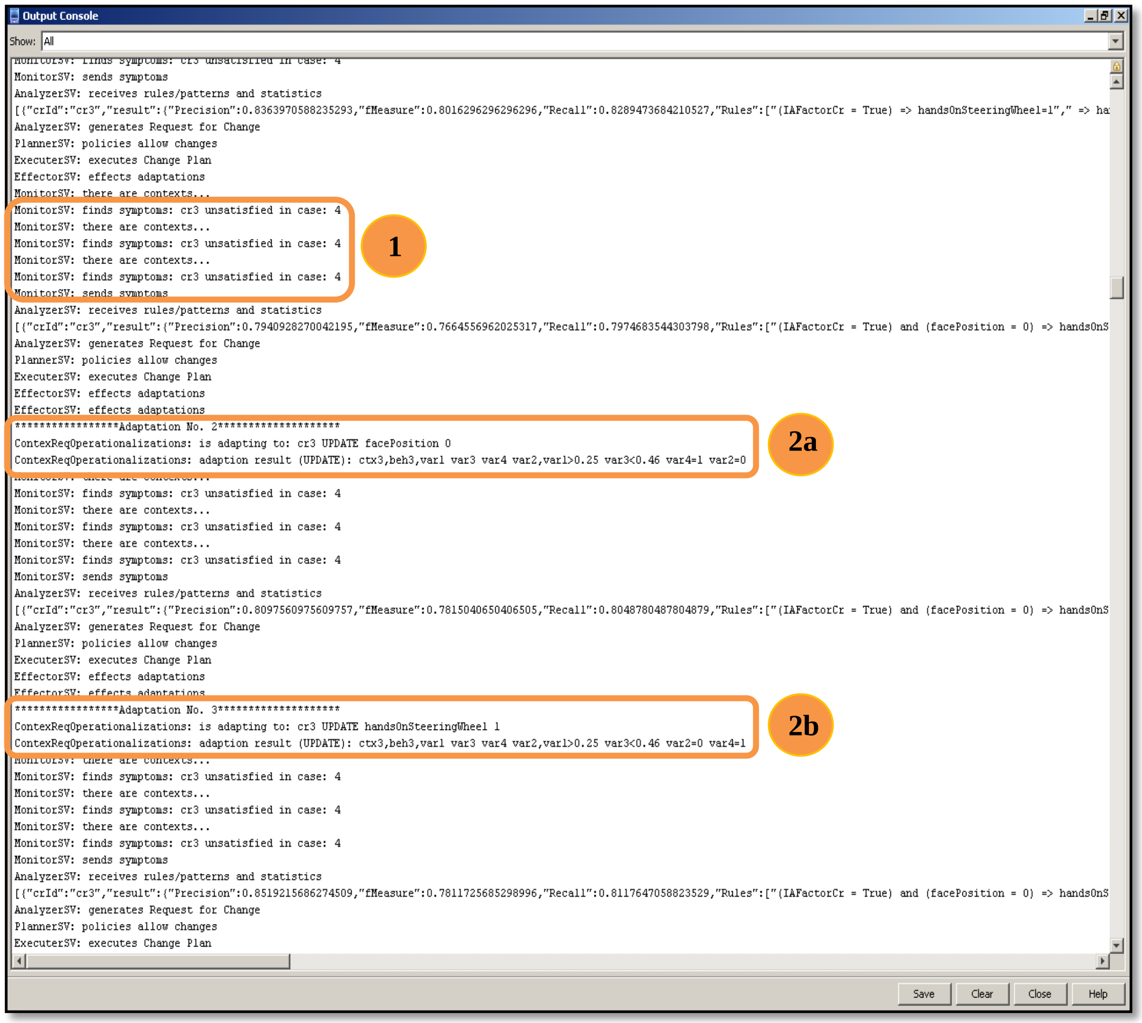


Fig. 43: SACRE view from air

The support lane keeping is in charge of maintaining the vehicle in the lane. During this experiment the vehicle has never gone out of the lane even considering the difficulty that a path with curves implies and the simple algorithm used for this task. In this demonstration, when the support lane keeping actuator is activated the speed for a straight stretch is maintained around 30 km/h; if a curve appears in the path the speed is decreased and maintained around 20 km/h. A view from the air of the vehicle in the path during this execution is shown in Figure 43.

The driver's decision of turning on the support lane keeping triggers a set of symptoms in the monitor element as expected: from time to time a symptom about the contextual requirement 1 in case 3 (context true, behaviour false); and most of the time, another one about contextual requirement 3 in case 4 (context false and behaviour true). An example of symptom is shown in Figure 44, area number 1. Meanwhile, examples of the adaptions effectuated over SACRE's contextual requirements' operationalization in this scenario are indicated in the areas 2a and 2b.

About the adaptations, the keys var1, var2, var3 and var4 present in Figure 44 (areas 2a and 2b) correspond to the variables: perclosed, face position, heart beats per minute and hands on steering wheel respectively. This legend is used for simplification in all the console outputs.



```
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":0.8363970588235293,"fMeasure":0.8016296296296296,"Recall":0.8289473684210527,"Rules":["(IAFactorCr = True) => handsOnSteeringWheel=1"," => ha
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":0.7940928270042195,"fMeasure":0.7664556962025317,"Recall":0.7974683544303798,"Rules":["(IAFactorCr = True) and (facePosition = 0) => handsOnS
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
EffectorSV: effects adaptations
*****************Adaptation No. 2********************
ContexReqOperationalizations: is adapting to: cr3 UPDATE facePosition 0
ContexReqOperationalizations: adaption result (UPDATE): ctx3,beh3,var1 var3 var4 var2,var1>0.25 var3<0.46 var4=1 var2=0
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":0.8097560975609757,"fMeasure":0.7815040650406505,"Recall":0.8048780487804879,"Rules":["(IAFactorCr = True) and (facePosition = 0) => handsOnS
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
EffectorSV: effects adaptations
*****************Adaptation No. 3********************
ContexReqOperationalizations: is adapting to: cr3 UPDATE handsOnSteeringWheel 1
ContexReqOperationalizations: adaption result (UPDATE): ctx3,beh3,var1 var3 var4 var2,var1>0.25 var3<0.46 var2=0 var4=1
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 4
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":0.8519215686274509,"fMeasure":0.7811725685298996,"Recall":0.8117647058823529,"Rules":["(IAFactorCr = True) and (facePosition = 0) => handsOnS
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
```

Fig. 44: SACRE with symptoms and adaptations

- Results of Scenario 3. Manual – Straight – Dangerously Tired

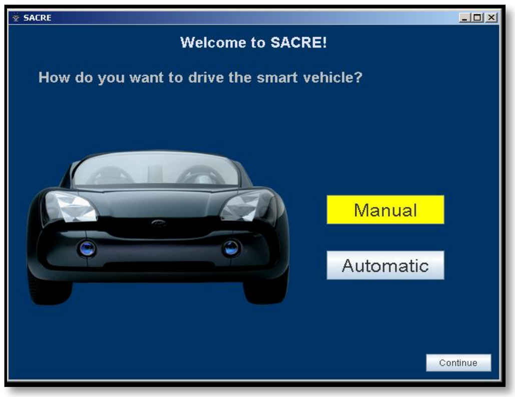


Fig. 45: Manual Setting

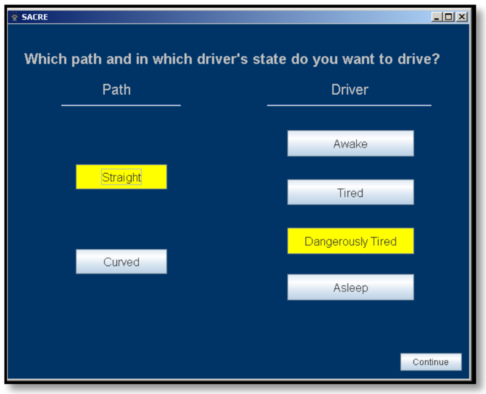
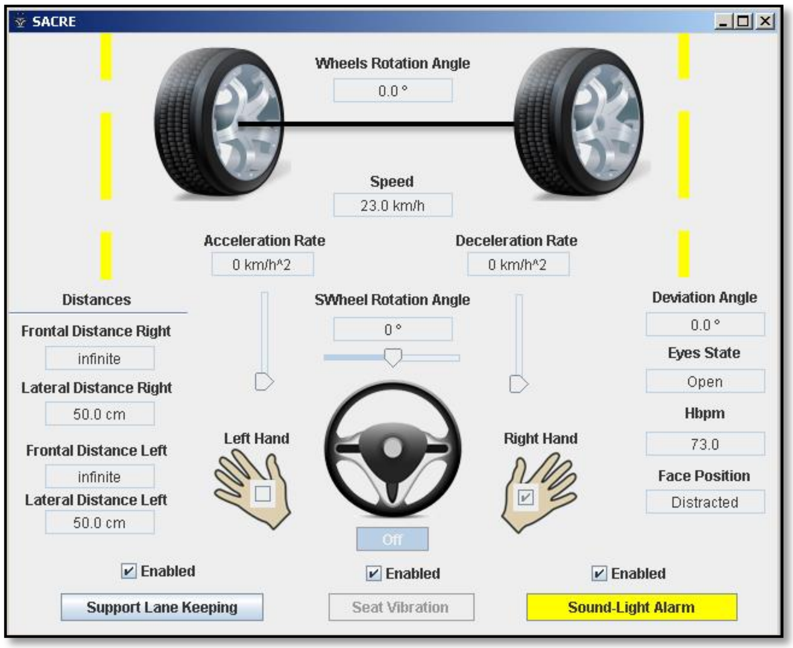


Fig. 46: Straight path and dangerously tired driver

Figures 45 and 46 show the initial parameters selected, for this scenario, in the configuration menus provided by SACRE.

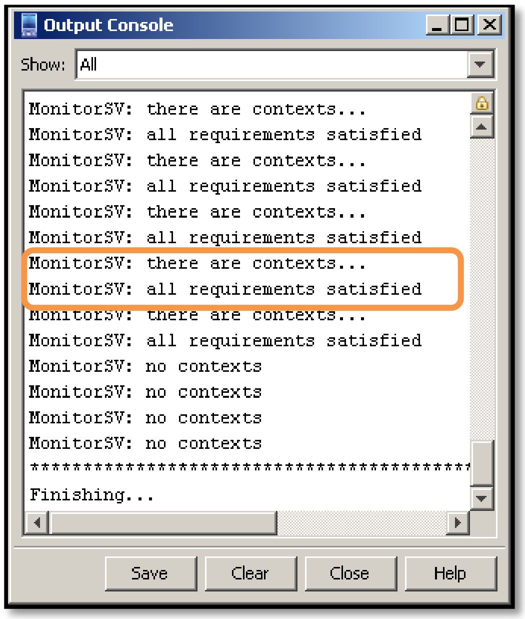


Wheels Rotation Angle: 0.0 °
Speed: 23.0 km/h
Acceleration Rate: 0 km/h^2
Deceleration Rate: 0 km/h^2
SWheel Rotation Angle: 0 °
Distances
Frontal Distance Right: infinite
Lateral Distance Right: 50.0 cm
Frontal Distance Left: infinite
Lateral Distance Left: 50.0 cm
Deviation Angle: 0.0 °
Eyes State: Open
Hbpm: 73.0
Face Position: Distracted
Off
✓ Enabled ✓ Enabled ✓ Enabled

Fig. 47: SACRE with sound-light alarm activated

The driver's state in this scenario is *dangerously tired*; the actuator related to this scenario is the sound-light alarm. Figure 47 shows the moment when the context of the contextual requirement aimed to detect dangerously tired drivers holds (because its operationalization is fulfilled) and the alarm is activated by the system.

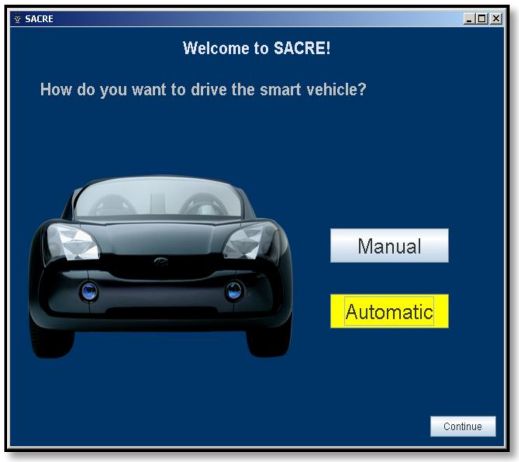
The driver accepts the correctness of the alarm occurrence and does nothing to change its activation. The driver ephemerally awakes when hears the alarm provoking that the system turns it off. Later s/he falls again in the same state of dangerously tired and the system turns on the alarm again, this cycle is repeated during the whole execution as expected. Because the driver never contradicts the initial contextual requirements configuration the requirements are satisfied during the whole execution and any adaption is effectuated as shown in Figure 48.



```
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: no contexts
MonitorSV: no contexts
MonitorSV: no contexts
MonitorSV: no contexts
**********************************************
Finishing...
```

Fig. 48: SACRE all requirements satisfied

- Results of Scenario 4. Automatic – Straight – Asleep



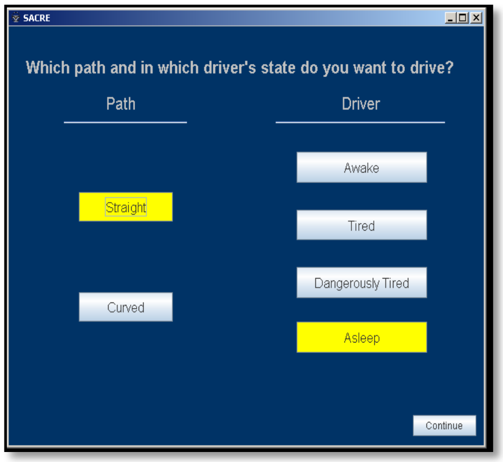
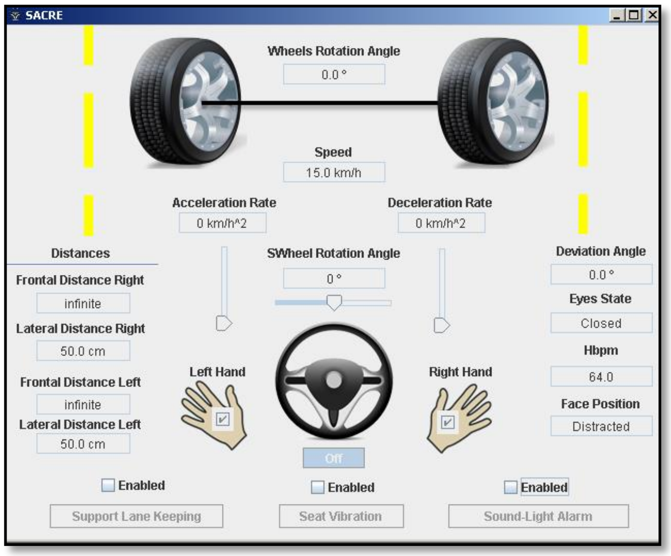


Fig. 49: Automatic Setting Fig. 50: Straight path and asleep driver

Figures 49 and 50 show the initial parameters selected, for this scenario, in the configuration menus provided by SACRE.


SACRE

Wheels Rotation Angle: 0.0 °

Speed: 15.0 km/h

Acceleration Rate: 0 km/h^2

Deceleration Rate: 0 km/h^2

Distances

Frontal Distance Right: infinite

Lateral Distance Right: 50.0 cm

Frontal Distance Left: infinite

Lateral Distance Left: 50.0 cm

SWheel Rotation Angle: 0 °

Left Hand

Right Hand

Off

Deviation Angle: 0.0 °

Eyes State: Closed

Hbpm: 64.0

Face Position: Distracted

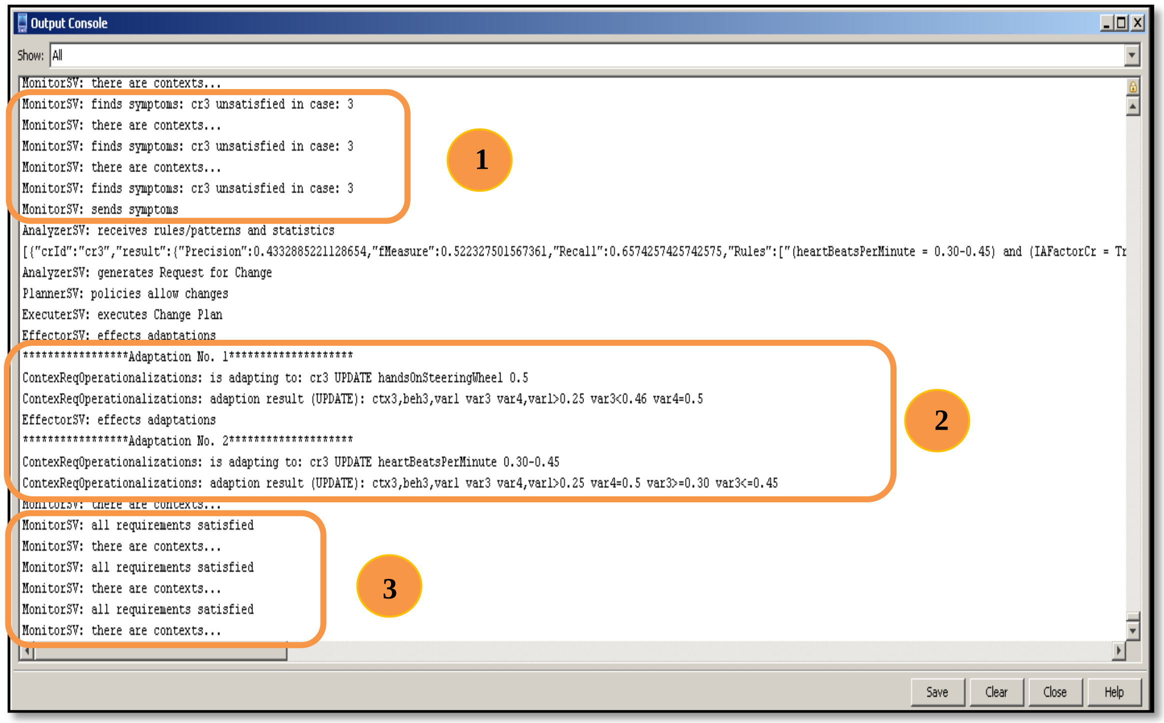
☐ Enabled — Support Lane Keeping

☐ Enabled — Seat Vibration

☐ Enabled — Sound-Light Alarm

Fig. 51: SACRE with all actuators disabled

In this scenario, for trigger an adaption, it has been simulated that even if the driver shows drowsiness symptoms s/he decides to deactivate all the actuators at the beginning of the execution and never enable them again. This context is shown in Figure 51.


Output Console

Show: All

```
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 3
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 3
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 3
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":0.4332885221128654,"fMeasure":0.522327501567361,"Recall":0.6574257425742575,"Rules":["(heartBeatsPerMinute = 0.30-0.45) and (IAFactorCr = Tr
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
*****************Adaptation No. 1********************
ContexReqOperationalizations: is adapting to: cr3 UPDATE handsOnSteeringWheel 0.5
ContexReqOperationalizations: adaption result (UPDATE): ctx3,beh3,var1 var3 var4,var1>0.25 var3<0.46 var4=0.5
EffectorSV: effects adaptations
*****************Adaptation No. 2********************
ContexReqOperationalizations: is adapting to: cr3 UPDATE heartBeatsPerMinute 0.30-0.45
ContexReqOperationalizations: adaption result (UPDATE): ctx3,beh3,var1 var3 var4,var1>0.25 var4=0.5 var3>=0.30 var3<=0.45
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
MonitorSV: all requirements satisfied
MonitorSV: there are contexts...
```

Fig. 52: SACRE with symptoms and adaptations

Even if this scenario has a slept driver in the vehicle, in the real world a driver is not expected to be totally asleep, from time to time s/he would show some symptoms of being awake. So, the file with the sensor's values describing a slept person has being designed following this idea. In the intervals where the driver is awake, the support lane keeping is supposed to be off and all requirements should be satisfied but for the other time intervals (when the driver is asleep) the system triggers symptoms as shown in Figure 52, area number 1.

After some iterations showing symptoms and calling the data mining service, SACRE has adapted the contextual requirement's operationalization related with the support lane keeping activation (contextual requirement 3). This adaptation is shown in Figure 52, in the area indicated with number 2. Meanwhile, in area number 3 it can be noticed that after the adaptation, all the contextual requirements are satisfied. This means, that the third contextual requirement is now also satisfied because its context's operationalization has been adapted.

- Results of Scenario 5. Automatic – Straight – Awake

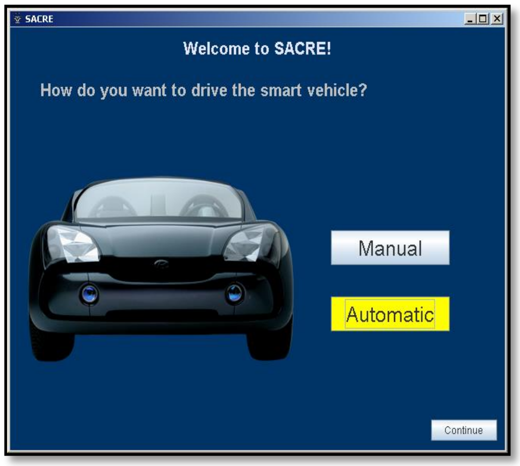


Fig. 53: Automatic Setting

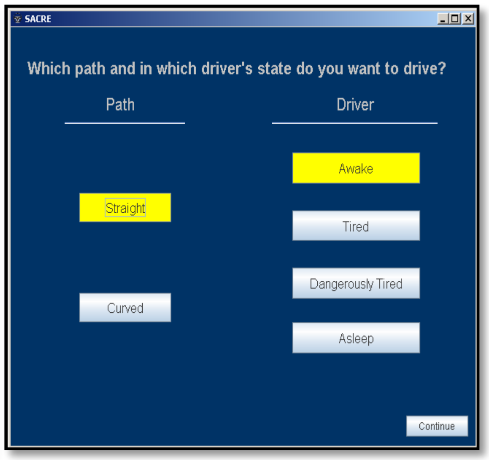
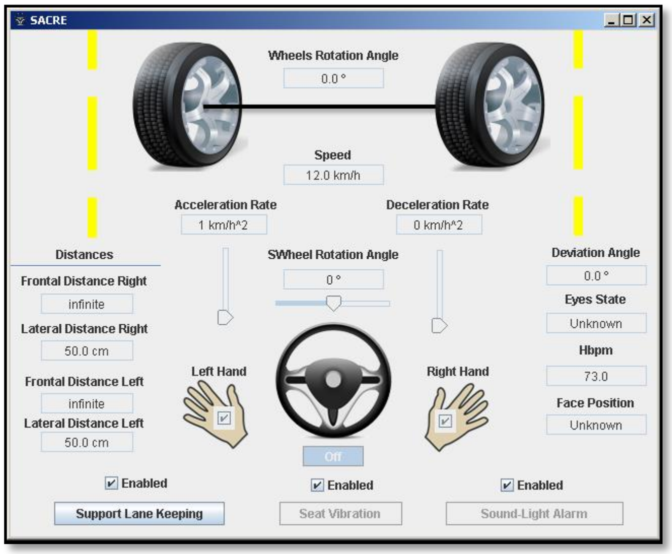


Fig. 54: Straight path and awake driver

Figures 53 and 54 show the initial parameters selected, for this scenario, in the configuration menus provided by SACRE. The source file containing driver's information has been modified, but the file corresponding to the automatic setting is the same as in previous scenarios.


SACRE

Wheels Rotation Angle: 0.0 °

Speed: 12.0 km/h

Acceleration Rate: 1 km/h^2

Deceleration Rate: 0 km/h^2

Distances

Frontal Distance Right: infinite

Lateral Distance Right: 50.0 cm

Frontal Distance Left: infinite

Lateral Distance Left: 50.0 cm

SWheel Rotation Angle: 0 °

Deviation Angle: 0.0 °

Eyes State: Unknown

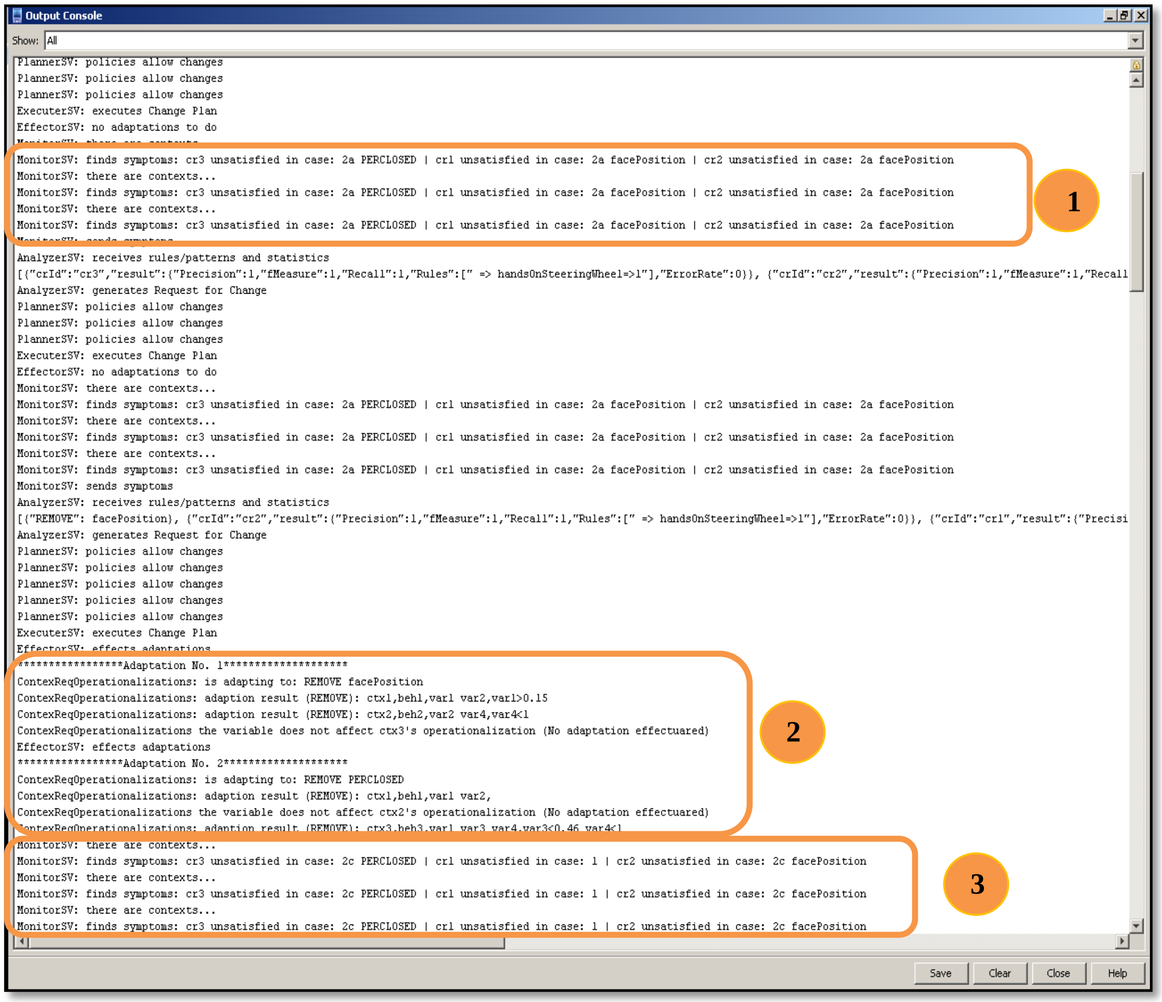
Hbpm: 73.0

Face Position: Unknown

Fig. 55: SACRE sensors failure

After some seconds, the driver's vigilance level sensor fails as shown in Figure 55 (showing unknown values for the variables linked with it). The system should adapt and re-operationalize the contextual requirements only with the sensors' values it has available. This could affect driver's satisfaction but is the only way SACRE's and the ACon approach have to deal with this uncertainty. Further adaptations would be provoked if the user would start to interact with the actuators but it is not the case of this scenario. In this scenario the driver agrees with the adaptations and smart vehicle's behaviour. Some adaptations of this scenario are shown in Figure 56.

First, the monitor finds symptoms, then the analyzer propose adaptations and the contextual requirements managed element is adapted. As it can be seen in area number 2, the first contextual requirement that depends on the two variables resulting from the driver's vigilance level sensor has been left without operationalization. In area number 3, this situation triggers a case 1 symptom (contextual requirement's context no operationalized), which will be fixed only when this requirement is operationalized again. Also in area number 3, it can be noticed that the driver's vigilance level sensor is up again, since case 2c symptoms (sensor up again) are experienced. This will eventually cause that the analyzer adds it again in the data mining requests.



```
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: no adaptations to do
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"crId":"cr3","result":{"Precision":1,"fMeasure":1,"Recall":1,"Rules":[" => handsOnSteeringWheel=>1"],"ErrorRate":0}}, {"crId":"cr2","result":{"Precision":1,"fMeasure":1,"Recall
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: no adaptations to do
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2a PERCLOSED | cr1 unsatisfied in case: 2a facePosition | cr2 unsatisfied in case: 2a facePosition
MonitorSV: sends symptoms
AnalyzerSV: receives rules/patterns and statistics
[{"REMOVE": facePosition}, {"crId":"cr2","result":{"Precision":1,"fMeasure":1,"Recall":1,"Rules":[" => handsOnSteeringWheel=>1"],"ErrorRate":0}}, {"crId":"cr1","result":{"Precisi
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: effects adaptations
****************Adaptation No. 1*******************
ContexReqOperationalizations: is adapting to: REMOVE facePosition
ContexReqOperationalizations: adaption result (REMOVE): ctx1,beh1,var1 var2,var1>0.15
ContexReqOperationalizations: adaption result (REMOVE): ctx2,beh2,var2 var4,var4<1
ContexReqOperationalizations the variable does not affect ctx3's operationalization (No adaptation effectuared)
EffectorSV: effects adaptations
****************Adaptation No. 2*******************
ContexReqOperationalizations: is adapting to: REMOVE PERCLOSED
ContexReqOperationalizations: adaption result (REMOVE): ctx1,beh1,var1 var2,
ContexReqOperationalizations the variable does not affect ctx2's operationalization (No adaptation effectuared)
ContexReqOperationalizations: adaption result (REMOVE): ctx3,beh3,var1 var3 var4,var3<0.46 var4<1
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
MonitorSV: there are contexts...
MonitorSV: finds symptoms: cr3 unsatisfied in case: 2c PERCLOSED | cr1 unsatisfied in case: 1 | cr2 unsatisfied in case: 2c facePosition
```

Fig. 56: SACRE with symptoms and adaptations

In future implementations, the facility to manipulate sensors' operation is a priority for testing purposes. The prototype shown in this work provides a display closer to a real world smart vehicle display than to a testing purpose display. Moreover, if hardware would be used, the sensors' operation would be in charge of them, as it has been mentioned before, and have to be not simulated by the tool.

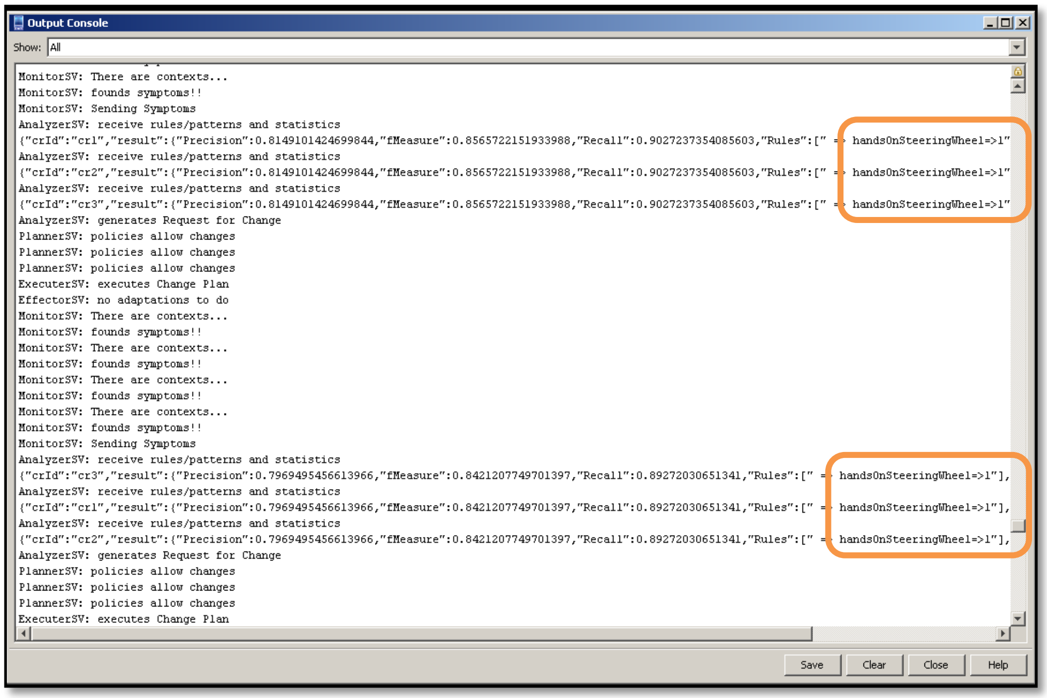
Discussion

In this work the proof-of-concept implementation, SACRE, of the novel ACon approach has been presented. The SACRE tool has used the data mining techniques as well as the architecture proposed by ACon [12] at its core for dealing with uncertainty in contextual requirements at runtime. The integration of the end-user in the adaption feedback loop is an important factor that increases the user satisfaction in any application. The contextual requirements as shown in the *experiments* section have experienced an adaptation process along the system execution time thanks to the requirements engineering tasks effectuated by SACRE, and proposed by ACon, at runtime.

The aim of SACRE, as well as any other implementation of ACon, is to minimize the number of unsatisfied contextual requirements at every moment. In order to accomplish this main goal, the system adapts the contextual requirements' operationalization continuously at runtime. The experiments run for the SACRE's evaluation and ACon validation shown in the previous section expose these adaptations in different situations.

For this demonstration, SACRE relies in the data mining algorithm proposed in [12]. Nevertheless, during the experiments some difficulties to obtain an acceptable adaptation have been experienced. These difficulties could be due to the few amounts of data provided to the algorithm which is a well-known factor that may affect its performance. In the future, some other techniques or algorithms should be investigated and applied in situations where the application has this few amounts of data. However, this fact is not affecting the evaluation of SACRE and validation of ACon since in real world the amount of data available in the applications' environments used to be huge. Actually, nowadays for some applications the amount of data grows exponentially, for example, in smart city applications with a lot of users interacting with them.

The consequence, in SACRE, of not having great amounts of data available is the emergence of useless patterns in some iterations, for example the ones shown in Figure 57. These patterns are useless because they are no related to a specific IAFactor which defines if a behaviour is true or false. So, even if the patterns would be translated to an operationalization, the system would not know to which contextual requirement corresponds or which context this new operationalization describes. More experiments during larger periods of time or training phases to persist data over executions, should be necessary in order to deeply test the algorithm, as done in [23]. Nevertheless, this task has been never considered in the scope of this work because it has been not foreseen.



```
MonitorSV: There are contexts...
MonitorSV: founds symptoms!!
MonitorSV: Sending Symptoms
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr1","result":{"Precision":0.8149101424699844,"fMeasure":0.8565722151933988,"Recall":0.9027237354085603,"Rules":[" => handsOnSteeringWheel=>1"
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr2","result":{"Precision":0.8149101424699844,"fMeasure":0.8565722151933988,"Recall":0.9027237354085603,"Rules":[" => handsOnSteeringWheel=>1"
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr3","result":{"Precision":0.8149101424699844,"fMeasure":0.8565722151933988,"Recall":0.9027237354085603,"Rules":[" => handsOnSteeringWheel=>1"
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
EffectorSV: no adaptations to do
MonitorSV: There are contexts...
MonitorSV: founds symptoms!!
MonitorSV: There are contexts...
MonitorSV: founds symptoms!!
MonitorSV: There are contexts...
MonitorSV: founds symptoms!!
MonitorSV: There are contexts...
MonitorSV: founds symptoms!!
MonitorSV: Sending Symptoms
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr3","result":{"Precision":0.7969495456613966,"fMeasure":0.8421207749701397,"Recall":0.89272030651341,"Rules":[" => handsOnSteeringWheel=>1"],
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr1","result":{"Precision":0.7969495456613966,"fMeasure":0.8421207749701397,"Recall":0.89272030651341,"Rules":[" => handsOnSteeringWheel=>1"],
AnalyzerSV: receive rules/patterns and statistics
{"crId":"cr2","result":{"Precision":0.7969495456613966,"fMeasure":0.8421207749701397,"Recall":0.89272030651341,"Rules":[" => handsOnSteeringWheel=>1"],
AnalyzerSV: generates Request for Change
PlannerSV: policies allow changes
PlannerSV: policies allow changes
PlannerSV: policies allow changes
ExecuterSV: executes Change Plan
```

Fig. 57: Useless patterns

In order to minimize this issue, SACRE offers the opportunity to accumulate data from different smart vehicle's usage providing an ON-OFF button. Moreover, the possibility of re-start the view, after turning off the vehicle, without affecting the adaptation feedback loop execution (running in an emulated device) is provided. The data has not been designed to be accumulated over loops' executions in order to avoid possible noise, mainly in the testing process. In SACRE, each adaptation feedback loop's execution simulates a different vehicle, in consequence different historical data. The ON-OFF button as its name suggest turns on and off the smart vehicle, simulating for example that the driver has arrived to her/his destination. The vehicle could be though to be used by the same driver or by different drivers (i.e. family members) each time it is turned on.

The facility to change the parameters selected in configuration menus during a smart vehicle's view execution (even the vehicle is turned off) is not provided for simplicity. But the parameters could be changed if the user turns off the vehicle and re-starts the application without restarting the emulated device. The variables' values that use accumulators in the adaptation feedback loop are also restarted every time the vehicle is turned on (i.e. the perclosed variable value).

In analysing SACRE, it has shown good results implementing the ACon approach. The adjustments to the adaptation feedback loop in [12] have represented good improvements to take into account for future implementations. Moreover, the adaptations resulting from the SACRE's executions, taking into account the amount of data limitation, have been the ones expected by the user. SACRE's results reaffirm the ACon approach applicability in real world implementations. The specific domain application developed for this demonstration has made evident the power of ACon and SACRE to deal with extremely demanding applications with human interaction and running in real time.

Thanks to uncertainty, SACRE has an innumerable variety of possible scenarios in which it could be tested. Since, the tool adapts and responds to every possible user interaction (infinite possibilities because of the human factor) each SACRE's execution will be different. Extensions of SACRE as well as other implementations of the novel ACon approach are expected for the future. The goal is to further validate the ACon approach in other to support its acceptance. In the case of SACRE, the extensions are aimed to improve its adaptations' accuracy (improve statistics) and timeliness and the whole system performance i.e. time response, user experience, etc.

Conclusions and future work

SACRE is a first step towards the validation of the novel ACon approach presented in [12]. This validation has been developed in an extremely demanding application domain executed in real-time. The evaluation and validation in charge of this work corroborate that this novel approach is applicable in real-world domains, where self-adaptive systems are increasingly present. Specifically, in the smart vehicles domain, a currently trending topic with opportunities in both communities: industry and research.

This proof-of-concept implementation also works as a complementary tool for the ACon validation effectuated in [12] automatizing the tasks that have been manually done in that process and also executing in real-time, unlike the validation performed in [12] which was a post-mortem analysis. Out of the ACon approach validation, this tool is useful for researchers interested in the requirements engineering field, particularly the evolution of contextual requirements at runtime in uncertain environments and the modelling of this kind of requirements. Moreover, it is interesting for the community developing self-adaptive software as a possible solution for their historical experienced challenges.

As it has been mentioned before in future work improved versions of this prototype have been considered. These improved versions could be thought to be integrated in real-world contexts with simulation purposes, and could become the seed of more sophisticated tools that could eventually be integrated in real-time systems. More investigations about machine learning and other techniques must be done in order to improve the performance of this kind of applications. Probably, a great classification of these techniques by domain and performance could result helpful for the interested community's support.

Appendix A

A set of UML diagrams is provided in this Appendix as complementary evidence to support this work. The diagrams describe the system's domain, architecture's component, process flow and functionalities.

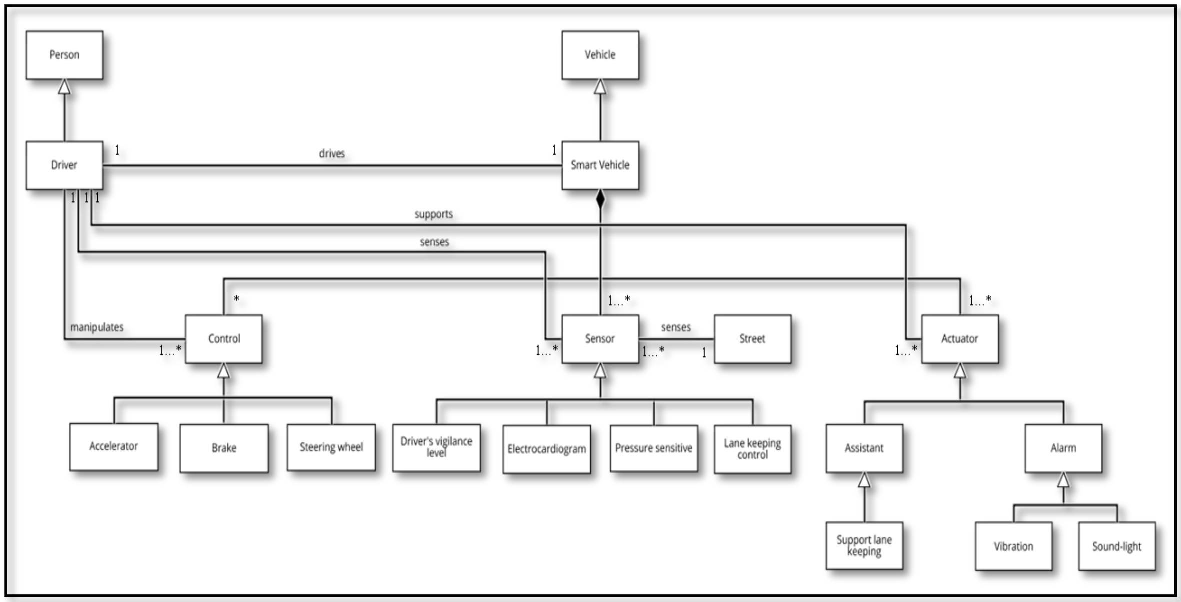


Fig. 58: SACRE's domain - Conceptual Model

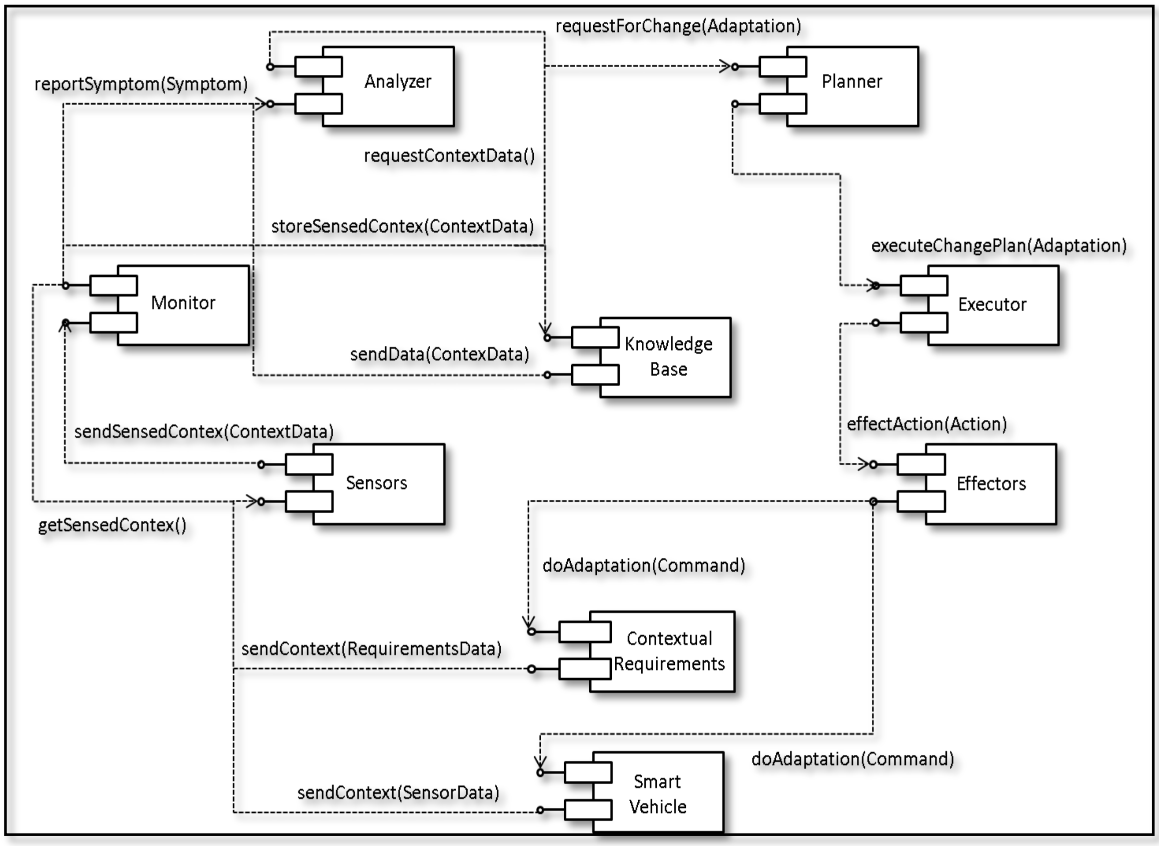


Fig. 59: SACRE’s architecture - Component Diagram

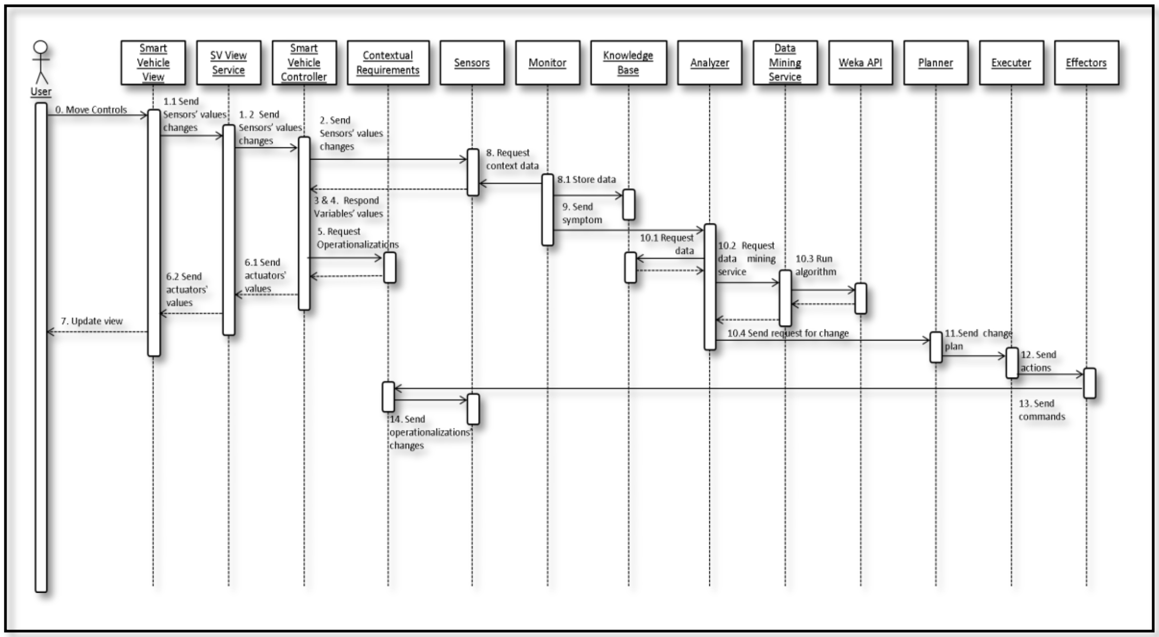


Fig. 60: SACRE’s process flow – Sequence Diagram

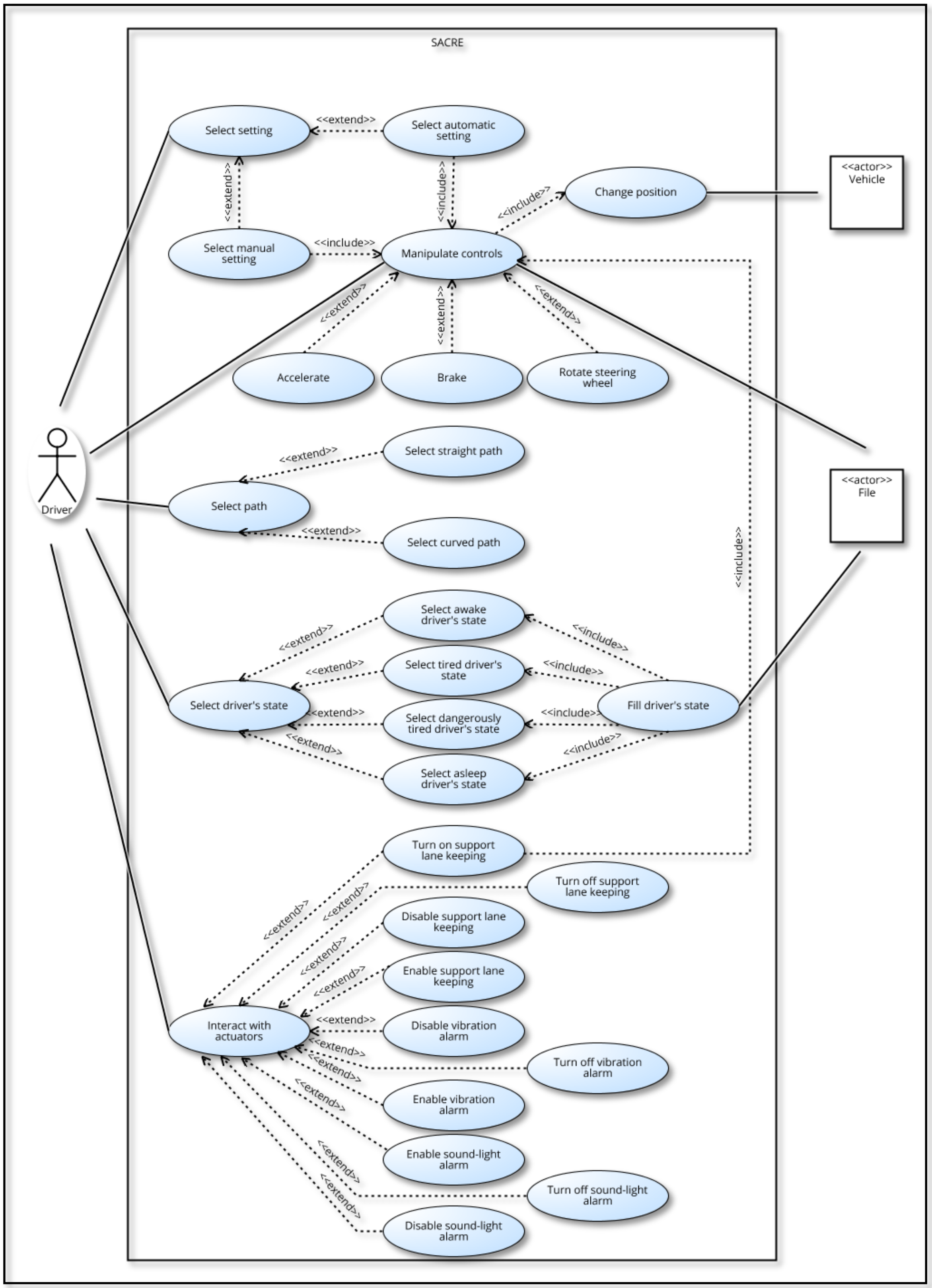


Fig. 61: SACRE's functionalities – Use Case Diagram

Appendix B

The data mining tool that has been used in this work is Weka. In order to send a request to the Weka Java API an Attribute-Relation File Format (ARFF) document should be submitted. The specification of the structure of this document is provided in: http://www.cs.waikato.ac.nz/ml/weka/arff.html.

In SACRE, the training (one for each contextual requirement) and execution time .arff documents are composed as follows:

```
@relation sensorData

@attribute IAFactorCr {True,False}
@attribute perclosed {-1,<0.05,0.05-0.14,0.15-0.20,0.21-0.30,>0.30}
@attribute facePosition {-1,<0,0,1,>1}
@attribute heartBeatsPerMinute {-1,<0.30,0.30-0.45,0.46-0.55,0.56-0.75,>0.75}
@attribute handsOnSteeringWheel {-1,<0,0,0.5,1,>1}

@data
False,>0.30,0,0.56-0.75,1
False,>0.30,0,0.56-0.75,0
…
```

Bibliography

[1] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T.F. Knight Jr., R. Nagpal, E. Rauch, G.J. Sussman, R. Weiss. “Amorphous computing,” Communications of the ACM 43(5), 74–82, 2000.

[2] Y. Diao, J.L. Hellerstein, S. Parekh, R. Griffith, G. Kaiser, D. Phung. “Control theory foundation for self-managing computing systems,” IEEE Journal on Selected Areas in Communications 23(12), 2213–2222, 2005.

[3] G. Di Marzo-Serugendo, M. Gleizes, A. Karageorgos. “Self-organisation in MAS,” Knowledge Engineering Review 20(2), 165–189, “2005”

[4] Y. Brun and N. Medvidovic. “Fault and adversary tolerance as an emergent property of distributed systems’ software architectures,” 2nd ACM International Workshop on Engineering Fault Tolerant Systems (EFTS 2007), Dubrovnik, Croatia, pp. 38–43, 2007.

[5] W.B. Cannon. “The wisdom of the body,” W.W. Norton & Company, Inc., New York, 1932.

[6] N. Wiener. “Kybernetiks,” Econ-Verlag, Düsseldorf, 1963.

[7] J.A. Martín, J. de Lope, D. Maravall. “Adaptation, anticipation and rationality in natural and artificial systems: computational paradigms mimicking nature,” Springer Science+Business Media B.V, 2008.

[8] S.H. Siadat and M. Song. “Understanding Requirement Engineering for Context-Aware Service-Based Applications,” Journal of Software Engineering and Applications, 5, 536-544, 2012.

[9] R. Ali. “Modeling and Reasoning about Contextual Requirements: Goal-based Framework,” PhD thesis, Univ. of Trento, 2010.

[10] N.U. Bhaskar and P. Govindarajulu. "Context Exploration for Requirements Elicitation in Mobile Learning Application Development," IJCSNS International Journal of Computer Science and Network Security, 8(8):292–299, 2008.

[11] P. Inverardi and M. Mori. "Requirements Models at Run-time to Support Consistent System Evolutions," IEEE Int. Works. On Requirements@Run.Time, pages 1–8, 2011.

[12] A. Knauss, D. Damian, X. Franch, A. Rook, H.A. Müller, A. Thomo. "ACon: A Learning-Based Approach to Deal with Uncertainty in Contextual Requirements at Runtime," submitted.

[13] C. Thiemich and F. Puhlmann. "An Agile BPM Project Methodology," Master thesis, Germany, 2012.

[14] K. Tian and K. Cooper. "Agile and Software Product Line Methods: Are They So Different?," 1st International Workshop on Agile Product Line Engineering, Maryland, USA, 2006.

[15] K. Schwaber and M. Beedle. "Agile Software Development with Scrum," Prentice Hall PTR, 2001.

[16] P. Feiler, R.P. Gabriel, J. Goodenough, R. Linger, T. Longstaff, R. Kazman, M. Klein, L. Northrop, D. Schmidt, K. Sullivan, K. Wallnau. "Ultra-large-scale systems: The software challenge of the future," Technical report, Software Engineering Institute (http://www.sei.cmu.edu/uls/), 2006.

[17] B.H.C. Cheng, R. de Lemos, H. Giese, P. Inverardi, J. Magee. "Software Engineering for Self-Adaptive Systems: A Research Roadmap," B.H.C. Cheng et al. (Eds.): Self-Adaptive Systems, LNCS 5525, pp. 1–26, Springer-Verlag Berlin Heidelberg, 2009.

[18] J.O. Kephart and D.M. Chess. "The Vision of Autonomic Computing," IEEE Computer 36(1), 41–50, 2003.

[19] IBM Corporation. "An architectural blueprint for autonomic computing," White Paper, 4th edn., IBM Corporation, 2006.

[20] Y. Brun, G. Di Marzo Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, H. Müller, M. Pezzè, M. Shaw. "Engineering Self-Adaptive Systems through Feedback Loops," B.H.C. Cheng et al. (Eds.): Self-Adaptive Systems, LNCS 5525, pp. 48–70, Springer-Verlag Berlin Heidelberg, 2009.

[21] IBM Corporation. "Symptoms Reference Specification V2.0," (http://download.boulder.ibm.com/ibmdl/pub/software/dw/library/autonomic/books/fpu3mst.pdf), 2006.

[22] H. Müller and N. Villegas. "Runtime Evolution of Highly Dynamic Software," Evolving Software Systems, pages 229–264, Springer, 2014.

[23] A. Rook. "On the Feasibility of Integrating Data Mining Algorithms into Self Adaptive Systems for Context Awareness and Requirements Evolution." Master thesis, Univ. of Victoria, 2014.

[24] A. J. Ramirez, A. C. Jensen, B. H. C. Cheng. “A Taxonomy of Uncertainty for Dynamically Adaptive Systems,” IEEE Software Engineering for Adaptive and Self-Managing Systems (SEAMS’12), pages 99–108, 2012.

[25] L.M. Bergasa, J. Nuevo, M.A. Sotelo, R. Barea, M.E. Lopez. “Real-Time System for Monitoring Driver Vigilance,” IEEE Transactions on Intelligent Transportation Systems, Vol. 7, No. 1, 2006.

[26] J.H. Coote. “Respiratory and Circulatory Control during Sleep,” J. exp. Biol., 100, 223-244, 1982.

[27] W. Karlen, C. Mattiussi, D. Floreano. “Sleep and Wake Classification With ECG and Respiratory Effort Signals,” IEEE Transactions on Biomedical Circuits and Systems, 2009.

[28] S. Suryanarayanan and M. Tomizuka. “Appropriate Sensor Placement for Fault-Tolerant Lane-Keeping Control of Automated Vehicles,” IEEE. IEEE/ASME Transactions on Mechatronics, Vol. 12, No. 4, 2007.

[29] T. Gordon, A. Blankespoor, M. Barnes, D. Blower, P. Green, L. Kostyniuk. “Yaw Rate Error – A Dynamic Measure of Lane Keeping Control Performance for the Retrospective Analysis of Naturalistic Driving Data,” Univ. of Michigan Transportation Research Institute, Paper Number 09-0326 (http://www-nrd.nhtsa.dot.gov/pdf/esv/esv21/09-0326.pdf), 2015.

[30] L. Castañeda. “A Reference Architecture for Component-Based Self-Adaptive Software Systems,” Master thesis, Univ. ICESI, 2012.

[31] Oracle® Java SE Embedded. “Developer's Guide Release 8,” E28300-05 (https://docs.oracle.com/javase/8/embedded/JEMAG.pdf), 2014.

[32] Software Reliability Enhancement Center, Technology Headquarters, Information-technology Promotion Agency. “ESCR Embedded System development Coding Reference guide [C Language Edition], Ver. 2.0,” IPA/SEC, 2014.

Links

[33] http://agilemanifesto.org/

[34] http://www.cs.waikato.ac.nz/ml/weka

[35] http://www.toyota-global.com/innovation/safety_technology/safety_technology/technology_file/active/lka.html

[36] https://www.youtube.com/watch?v=xHV3dXRCM7A