Development Platform for Safe and Efficient Drive

Virtual Testing

Deliverable n.	D26.2– Specification of DESERVE platform optimised for virtual testing		
Sub Project	SP2	ADAS development platform	
Workpackage	WP2.6	Virtual testing	
Tasks	T2.6.3 T2.6.4	Identification of application requirements verifiable with virtual testing techniques and definition of the relative testing specifications Definition of DESERVE platform to enable virtual function testing	
Authors	G. DUNAND et al.	Intempora SA, INRIA	
File name	D26.2 - Specification of DESERVE platform_v1.0.docx		
Status	Draft		
Distribution	PU (Public)		
Issue date	30/11/2013	Creation date	01/10/2013
Project start and duration	1st of September, 2012 – 36 months		

REVISION CHART AND HISTORY LOG

Ver	DATE	AUTHOR	REASON
0.1	2013-10-01	G. DUNAND (Intempora)	Table of contents and document structure created
0.2	2013-10-25	G. DUNAND (Intempora)	Part 2 : definition of DESERVE platform to enable virtual testing
0.3	2013-12-02	J. Pérez (INRIA)	Contributions in section 1.
0.4	2014-01-23	J. Pérez (INRIA)	Other contributions in section 1.
0.5	2014-02-04	G. DUNAND	Executive summary + add precisions in part 2.
1.0	2014-02-28	G. DUNAND	Took into account reviews

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

ABBREVIATION	DESCRIPTION
ADAS	Advanced Driver Assistance Systems
INS	Inertial Navigation System
IMU	Inertial Measurement Unit

GPS	Global Positioning Systems
MIL	Model In the Loop
SIL	Software In the Loop
HIL	Hardware In the Loop
FOP	Frontal object perception
VRU	Vulnerable road users

EXECUTIVE SUMMARY

Virtual testing is a key point in the DESERVE project, mainly because it allows massive cost reduction. Indeed, the time spent to test scenarios in the real world could be quite important, especially if there are many cases to study. The main advantage of virtual testing is to test all configurations offline first. Dangerous situations, hard to reproduce scenarios, but also “classic” scenarios can be tested using virtual environment (virtual sensors, dynamics models…).

The goal of this document is to show how the DESERVE platform will enable virtual testing. The first part will identify what the sensors in play are (cameras, IMU) and the second part will show how the virtual environment interacts with the DESERVE platform and what the perception platform brings to virtual testing.

INTRODUCTION

During the last few years, many advanced driver assistance systems (ADAS) based on on-board vehicle sensors have been developed for automotive applications. These systems are becoming more and more promoted because several key functions permit to increase the level of vehicle safety. Most of the time, it is really a challenge to access to the equipment and sensors information on vehicles, making difficult to design and test these new algorithms. Some of the applications are based on perception sensors embarked on the vehicle, which interact with the vehicle, driver and environment through electronic control units. For this reasons, the simulations of the algorithms and the analysis of existing solutions for virtual testing are important tasks in DESERVE Project. This document describes the overall possibilities of virtual testing in development and testing, and also the characteristics of the DESERVE platform that will optimize the benefits of virtual testing.

Objective

One of the main difficulties for ADAS application is the complexity of the scenario (real traffic). Testing the application on real vehicles in real traffic scenarios is the approach followed, together with some recording feature to allow the capturing of the critical situations, where the solution fails for example, in order to reproduce them in some way later in laboratory.

The DESERVE platform will enable introduction/extension of the virtual testing of the application in the laboratory, to a level where real testing will be needed only for final confirmation, allowing complete debugging of the application.

This document will focus on identifying the possibilities of virtual testing in the development and testing of an ADAS application (costs and timing benefits), and the characteristics of the DESERVE platform that will make that possible.

Structure of the deliverable

The structure of this document reflects the different tasks associated with work package 2.6. The results of each task are described in a dedicated chapter.

1. Identification of application requirements verifiable with virtual testing techniques and definition of the relative testing specifications

Based on the requirement presented in D12.1 [1], an identification of the overall possibilities of virtual testing is presented in this section. Table 1 shows the requirements for the perception modules, using the ID, interface and format for each requirement.

Table 1. Requirements for the virtual testing in the perception modules.

Requirements (ID)	***Interface used***	***Format registered***	***Description and Synchronization***
4.4.1 3D Reconstruction (3D-R)	DAIMLER: frontal Camera	.avi, JPG (3D point lists)	N.A.
4.4.2 ADASIS Horizon (ADA)	CRF: GPS, Maps Daimler: tbd	map data (**NMEA from GPSs**)	Digital map information. GPS, Gyroscope, Odometer
4.4.3 Driver monitoring automotive (DMA)	CRF: Interior Camera (Cont). Electrical bioimpedance.	.avi, JPG	Driver face analysis (10 Hz) Direct drowsiness (20 Hz) Biological drowsiness (1Hz)
4.4.4 Driver monitoring motorcycle (DMM)	Ramboll: GPS, camera.	(**NMEA from GPSs**). .avi	Camera should provide 720p color image. It should be synchronized with the GPS
4.4.5 Frontal object perception (FOP)	CRF: MRR and Stereocamera Volvo: tbd	2D radar point. .m4v, .mp4 y .mov	Area of observation. data fusion techniques will be implemented (radar and Cam).
4.4.6 Lane course (LC)	Daimler: front Camera, Radar	.avi, JPG, .m4v??. 2D radar points.	Trajectory of the lane in front. Information from ADASIS (4.4.2) will be used. Fusing of camera and radar data.
4.4.7 Scene labelling (SL)	Daimler: Front Camera	3D point list	Information from Lane Course module (4.4.6) will be used. Class Membership Image: road, vehicle, pedestrian,etc.
4.4.8 Self Calibration (SC)	Daimler: Front Camera	3D point list	Odometric sensor and camera information.
4.4.9 Vehicle trajectory calculation (VTC)	CRF: Volvo:	Positioning and CAN bus information.	Information from Lane recognition. Sensor data fusion. It uses future ego-vehicle positions and speed profile.
4.4.10 Vulnerable road users (VRU)	CRF:stereo-camera	m4v, .mp4 y .mov, .avi	Object position. VRU classification. Observation zone.

Other Perception requirements to be defined are:

- Assignment of objects to lanes (AOL)
- Detection of the free space (DFS)
- Lane recognition (LR)
- Moving object classification (MOC)
- Parking lot detector (PLT)
- Recognition of unavoidable crash situations (RUCS)
- Relative positioning to the road of the ego vehicle (RPR)
- Road data fusion (RDF)
- Side/rear object perception (SROP)
- Traffic sign detector (TSD)
- Vehicle filter/state (VFS)
- Vehicle trajectory calculation (VTC)

However, since most of them are dependent on each manufactures and demonstrators specification, there is no format proposed in this document. In the Specification of virtual test lab for DESERVE development platform (D.26.3) these information will be provided.

Table 2. Requirements for the virtual testing in the application modules

Requirements (ID)	***Interface used***	***Format registered***	***Description and Synchronization***
5.3.1 ACC control (ACC-C)	set speed/time gap	tbd	position and velocity of target vehicle
5.3.2 Driver Intention Detection (DID)	*Input*: Interior and external camera. *Output*:probability for each manoeuvre	CAN data	E.g.: lane change, overtaking, car following.
5.3.3 IWI manager (IWI)	Acceleration requests and warnings information	tbd	Arbitration between the driver and the control of the vehicle
5.3.4 Target Selection (TS)	Perception Horizon, position and velocity	tbd	Relevant target for each applications.

5.3.5 Threat Assessment (TA)	receive position and velocity of target vehicle.	CAN data.	it provides specific TTC and TLC measures. Longitudinal and Lateral risks.
5.3.6 Vehicle Motion Control (VMC)	- Acceleration requests from continuous control - Braking intervention	CAN data.	It provides the longitudinal acceleration and steering wheel torques requests

Other application modules (not considered in this document) are:

- Trajectory control (TC)
- Trajectory planning (TP)
- Vehicle model (VM)
- Reference Manoeuvre (RM)

These modules will be tested in simulations, based on the development reached in WP42 –Control Functions- and WP44 –Automated Functions-. First, simulation will be implemented in Matlab-Simulink, based on different vehicle models [2]. Then, other simulations will be tested with RTMaps and ProSivic, as were described in [3].

Finally, requirements for the virtual testing in the IWI modules will be consider in the final deliverable. These requirements depend on the specifications of each demonstrator.

2. Definition of DESERVE platform to enable virtual function testing

Virtual testing is a very important task within the DESERVE project, because it allows reduction costs and benefits in development time. As a consequence, the DESERVE platform should enable virtual testing as smooth as possible. As discussed in D26.1, virtual testing will be done in MIL and SIL. This is in this development phase that you change and tune algorithms in no time. Most of the interaction of the simulator with the DESERVE platform will be with the perception platform. So we will focus on the **perception platform** and see what it brings for virtual testing. In this project, at least two fast-prototyping platforms are used: RTMaps and ADTF. This document will focus on the first one because the author knows it far better. But no doubt that most of the stuff presented here could apply to others as well.

2.1. Graphical Environment : fast and easy to use

The perception platform is a user friendly platform with a graphical user interface: the studio. The studio allows to build an application with components (seen as blocks) connected to each other. Creating a simple multisensor application takes only a few minutes because all the user need is to focus on tasks to be done by the application.

Virtual sensors from the virtual testing environment are components too and can integrate easily in any diagram. They are seen as ‘sensors’.

Besides, using the studio enables the user to visualize the whole DESERVE application. The global application can be virtually splitted into several “blocks” represented by components and macro-components. The macro-components allow to simplify the diagram by regrouping components into only one, with one single click.

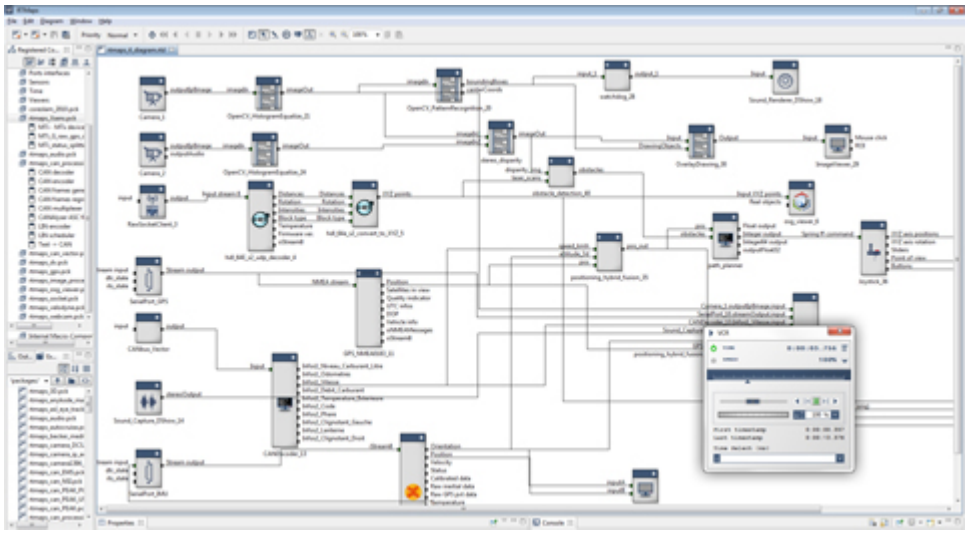

Figure 1 : RTMaps studio

Using a graphical user interface has a lot of advantages. Here are listed a few.

2.1.1. Focus on interfaces

Graphical components have inputs, outputs and properties. During the application conception, focusing on interfaces is generally a good idea because it allows and forces team work. Indeed, once inputs and outputs types are defined; **changing an algorithm for another** is not a problem anymore, provided the interface is fixed and does not change. In that, the user has just to change the component for another and the work is done! In big projects like the DESERVE project, this is undoubtedly a big advantage due to the number of partners. Strict definitions about interfaces are necessary to ensure the compatibility of the work between partners.

The use of macro-components can definitely simplify the diagram by splitting the global problem into sub-problems (sub-diagram). For example, the next figure deals with face detection. All the implementation is hidden in first appearance to simplify the reading, but of course looking under the mask is still possible.

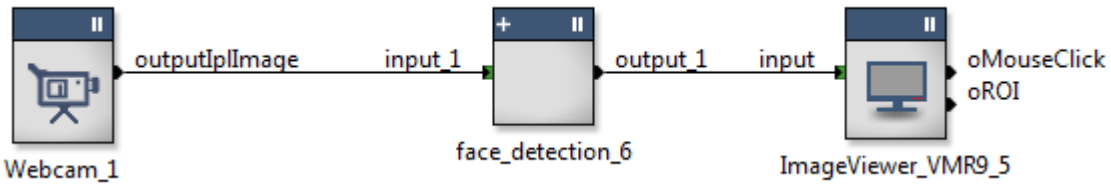

Figure 2 : Example of a face detection algorithm as a RTMaps macro-component

2.1.2. Beyond virtual testing

In the application diagram, each component runs in its own thread and exchanges informations with other connected components. So every component is independent of others, and the diagram can be easily updated.

As a consequence, changing an algorithm or even going from virtual testing to real testing is not a problem. Indeed, we only need to replace virtual sensors components with real sensors components and we keep algorithms. This allows a great timing benefit.

2.2. Many existing tools

The perception platform has many tools to accelerate and ease application development. All those tools come natively with any fresh installation.

2.2.1. Debugging utilities

In very large project like the DESERVE project, it is important to have informations about any part of the application, during the execution. For instance, RTMaps can display a lot of information during the execution of the diagram. As an example, the DataViewer can display generic informations (timestamps, size, etc.) and specific ones (width and height of an image if current data is an image) as a tree. This is very useful to inspect data along a processing chain and check that such component behaves correctly.

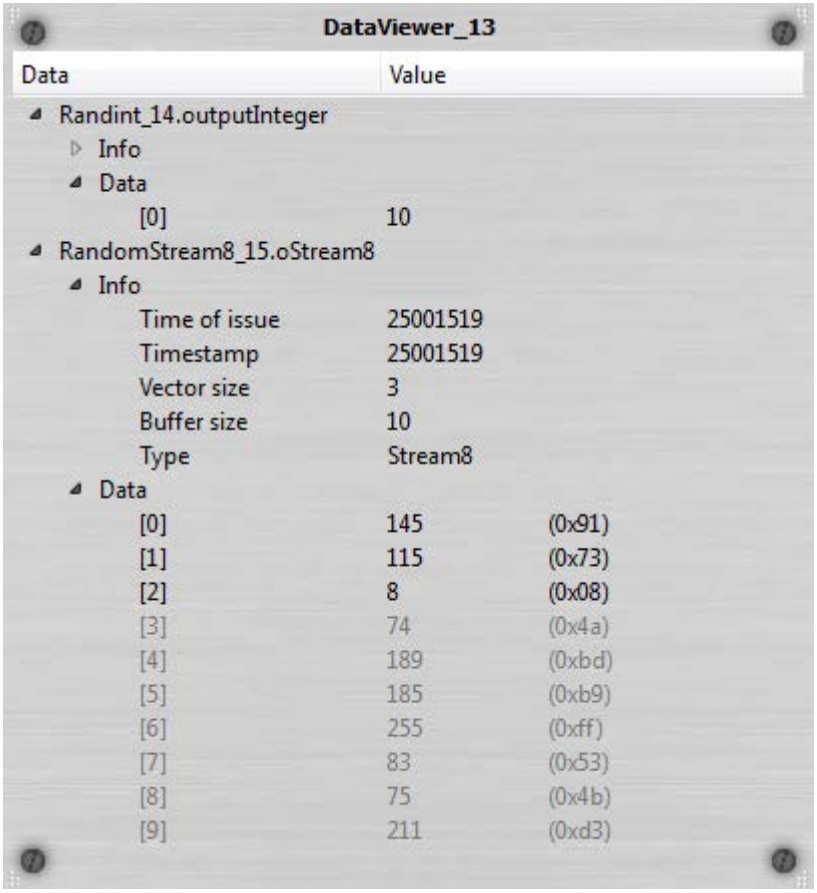

Figure 3 : Data viewer widget

This ability to display images and information during the execution enables users to monitor the application execution and see what is wrong quickly.

2.2.2. Performance indicators

As a matter of fact, RTMaps timestamps all the data acquired, so the **latency** can be calculated at any time. This is useful to estimate the computational time of an algorithm, so we can directly compare the speed of two algorithms or invalidate an algorithm because it takes too much time to compute. RTMaps also estimates the **CPU load of** each component to monitor the local and global performance of the diagram.

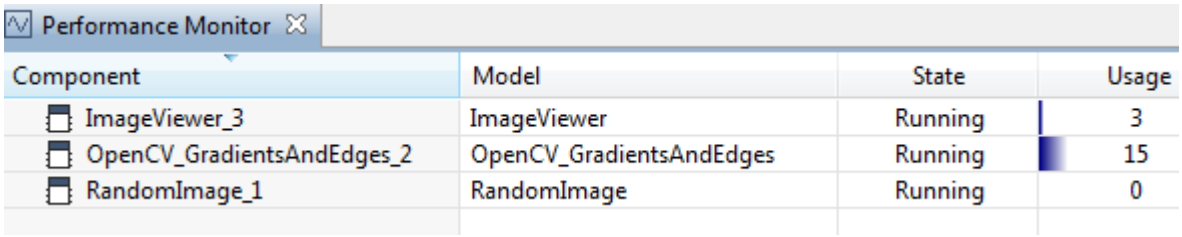

Figure 4 : CPU consumption of components on the diagram

2.2.3. A lot of standard components

RTMaps comes with a lot of standard components for all-round use :

- **Audio Video** : RTSP server, RTSP client, Video stream encoder / decoder, audio stream encoder / decoder.
- **Communication :** Sockets, Serial Port, SMS
- **Data conversion** : Matrix, Image, Text, Stream, Vector...
- **Data processing** : Calculator, condition, Text Scanner, switcher, vectorizer / devectorizer...
- **Generators** : Button, potentiometer, float constants generator
- **Player - Recorder**
- **Viewers** : 3D viewer, CAN frame viewer, data viewer, gauge, led, oscilloscope...

This is not an exhaustive list. Another example for automotive, some components such as the CAN decoder takes directly .dbc files and generate automatically outputs corresponding with the description.

All these components are part of the general library. This library is a powerful tool that enables the user to save a lot of time for acquisition, standard processing and display. In the end, the only task of the user is to focus on his personal expertise: algorithm and control functions.

2.3. Test and validation

2.3.1. Record – Replay ability

2.3.1.1 Record

RTMaps has the capability to record any sources asynchronously and so respects the natural sampling of every data source. Whereas most synchronous systems have to choose a global sampling frequency which depends on the sampling rates of the data sources used, RTMaps does not need any configuration at all. Indeed, the user has just to select the proper recording method. Of course, the recording format differs if you are recording images or raw data. If different methods exist for storing data, the user can select the one he prefers. For example, videos can be stored as a JSEQ, Video File, etc. (see figure xx)

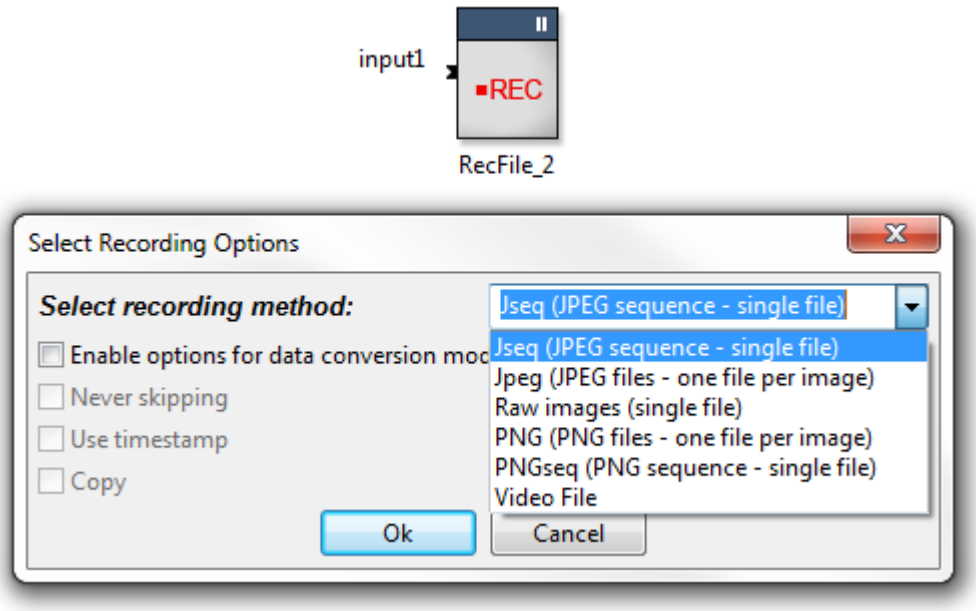

Figure 5 : Select record replay method

To summarize, the recorder stores data when it receives a fresh new data, whatever sampling rate this particular data source has. This allows recording many heterogeneous data with no overhead.

2.3.1.2 Replay

Replaying previously recorded data in RTMaps is a very easy task. Indeed, you just have to put a player component onto the diagram with the corresponding .rec file in the dedicated property and the player will create automatically outputs corresponding to the prior recording.

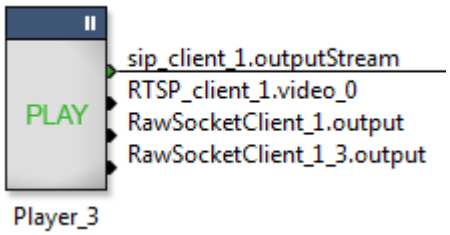

Figure 6: A player component

Then, you can connect to your other components, and that's all! All the recorded data sources will be replayed **exactly** as they were recorded, with the same latency and the same sampling rate.

In the DESERVE project, this ability is very important because it allows to replay real data in the application and even to mix real data and virtual data.

2.3.2. Batch processing

RTMaps has the ability to run in batch mode, which enables the user to test variations of a parameter automatically. To perform that, RTMaps enters a loop until all values have been tested. Here is an example of the influence of blurring in the face detection (figure 7).

Figure 7: RTMaps in batch mode with a custom GUI

The batch mode is surely a killer feature in the DESERVE project and can be applied to real and virtual testing.

2.4. Custom extensions

2.4.1. Developing a custom component

The "Software Development Kit" allows the design of your own components and include them in your RTMaps Library. The programming is done in C++ and is facilitated by the code skeletons generated automatically by the RTMaps SDK Wizard. Moreover, a complete API (Application Programming Interface) allows users to reach all the engine functions and to remain independent from the operating system (file system or real time programming for example).

As a matter of fact, unless otherwise specified, each component runs in its own thread. The developer is freed from the problems of data protection and inherent concurrent access of multithread applications.

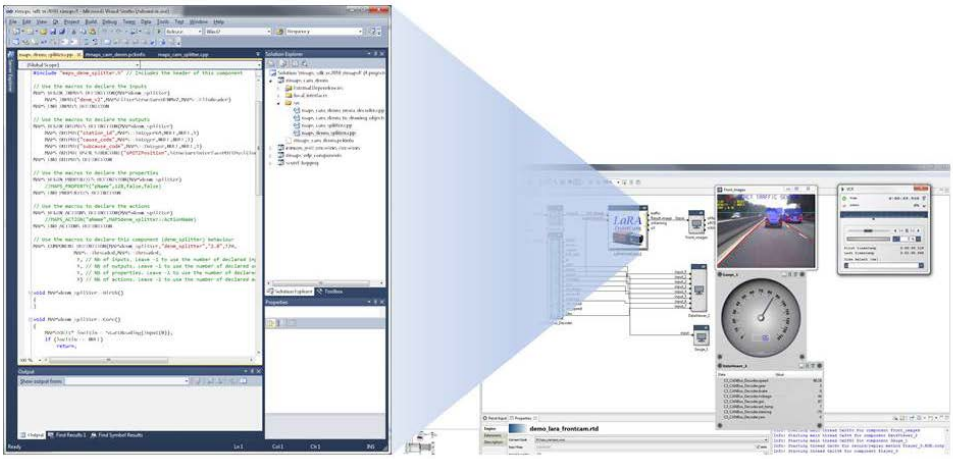

Figure 8: An RTMaps component is a C++ object.

What is particularly interesting for the DESERVE project is that the intellectual property remains to the partner who wrote the component. Indeed, the newly developed component can be shared to the DESERVE project without any restrictions, as only the binary code lies inside the component. So there is no risk of stealing the code, unless we talk about reverse engineering, which is another story.

2.4.2. Quick HMI for demos

Building an HMI is often a very important part because this is what people really see. An old fashioned ugly HMI will definitely lower the project's prestige, even if the technical part is brilliant. But, as always, the time allocated for HMI is somehow limited, so it would be great if the HMI could be built fast and looks fine.

There are many ways to do an HMI with RTMaps: C# with .NET, C/C++ with Qt, Qt/QML. They all have pros/cons, summarized in the table below.

Table 3 : Comparison of different langages for building HMI

Language	C++ & Qt	C#	Java	QML
Ease of development	*	*** Code simpler that C++/Qt	**	*** Extremely simple
Graphical editor available	Yes	Yes	Yes	Yes
IDE used for development	QtCreator	Visual Studio	Eclipse, Netbeans, etc.	QtCreator
Portablilty of code in terms of platforms	Yes	Windows only ; Linux under Mono (non official)	Yes	Yes
Performance and availibility	***	***	***	**
Possibility to create an application using RTMaps	Yes	Yes	Yes	Yes
Possibility to create an RTMaps component	Yes	No	No	Yes (without compilation)
Suited for heavy load applications : Database, etc...	***	***	**	*

As a matter of fact, the QML has a huge advantage, it can be integrated as an RTMaps component, and the design and animations of the HMI can be done directly in QML, without any C++ code. This allows separating design and functionalities, which suits well to modern work teams.

Figure 9: QML integration in RTMaps.

CONCLUSION

Many different sensors are used in the DESERVE project: cameras, radar… Most of them can be emulated in a virtual environment. Thanks to its open design, the perception platform allows to welcome any virtual sensor in the application design. So scenarios can be tested virtually before there are tested on real situation. This is time saving and of course cost saving.

Last but not least, the perception platform has many features that enable the engineer to accelerate his development: graphical interface, useful already available components… An HMI can even be built easily on top of the application.

REFERENCES

[1] D12.1 -Development Platform Requirements-, SP1, DESERVE project, 2013.

[2] D42.1 -Control functions solution design-, SP4, DESERVE project, 2013.

[3] D44.1 -Automated functions solution design-, SP4, DESERVE project, 2013.

ANNEX