Available online at www.sciencedirect.com

ScienceDirect

Procedia Manufacturing 3 (2015) 1903 – 1909

Procedia MANUFACTURING

6th International Conference on Applied Human Factors and Ergonomics (AHFE 2015) and the Affiliated Conferences, AHFE 2015

Creating rich human-machine interfaces with Rational Rhapsody and Qt for industrial multi-core real-time applications

Liher Granado*, Oskar Berreteaga

ULMA Embedded Solutions, Garagaltza auzoa 51-77, Oñati, 20560, Basque Country, Spain

Abstract

Industrial real-time applications and modern rich and friendly user interfaces have been separated worlds for a long time. But today, customers demand advanced user interface solutions for their real-time products, in order to gain competitive advantage in their markets. This creates to the development teams the big challenge of integrating those two worlds. This paper presents a real-life experience of the development of a system and software architecture for a real-time application with an advanced user interface. Starting with the analysis of the existing needs and requirements, the paper presents the followed steps for the selection of the most appropriate hardware platform, operating systems, Model driven development and other working tools for each architectural requirement, the process of adaptation and integration of the different environments and development tools, and finishes with the developed system results. In addition to the presented results, an analysis of the possibilities of reusing the created platform for other products is presented, following thesame process and making the required adaptations, in order to reproduce the obtained benefits in other real-time industrial solutions with advanced user interfaces.

Peer-review under responsibility of AHFE Conference

*Keywords:*Industrial real-time applications; Human-machine interfaces; Model driven development; Rational Rhapsody; Qt

* Corresponding author. Tel.: +34-943-250-300.
*E-mail address:*lgranado@ulmaembedded.com

Peer-review under responsibility of AHFE Conference
doi:10.1016/j.promfg.2015.07.233

1. Introduction

The software applications have been present in the industrial appliances sector since early 60's, and its presence has grownincrementally,reaching today's situation where most of the industrial appliances present in the market have some kind of software embedded. Most of those software applications are usually involved on the device control, and because of that, the use of real-time software is a commonly adopted natural choice. Due to its nature, real-time software has been traditionally associated with simple user interfaces, in many cases with the goal of avoid possible problems when interacting with the control algorithms, but other times also because economical reasons. Knowing that a rich user interface usually demands more system resources than real-time control tasks, the increase on the system resources demand carries a subsequent increase on the platform costs.

For many years the relation between embedded real-time software and simple user interfaces in industrial appliances has been accepted by the industrial appliances market. But now, the perception about user interfaces in the society is rapidly changing [1], and the perception of the industrial markets is changing with it. Today, the non-industrial world is being taken over by smart phones and similar last-generation devices, with very high capacities and system resources. Nowadays, the users are used to and really appreciate the high usability provided by touch-screens and last generation devices. Even if general appliances market and industrial appliances market are slightly different, this change of perception affects also the industrial appliances market, so an evolution has to happen and is happening in that sector.

The industrial appliance companies are aware of the new userperception regarding usability, and that new generation user interfaces are very positively valued. For that reason, obtaining the key nicely integrating industrial real-time applications and advanced user interfaces, without incurring on raising the platform costs, can be the best path to gain competitive advantage against their competitors.

2. Architecture definition

2.1. Preliminary analysis

When starting the design of a new platform architecture, the first step is to analyze the system requirements. In this case, the main high level requirements of the platform are principally defined by the needs of the industrial appliances market. The designed platform should be able to perform the real-time control of an industrial application; it should also support a powerful user-interface, including animations and video reproduction; it should provide a clear abstraction between the user-interface application and the control software, to avoid that a problem in the user-interface affects the real-time control algorithms; and because the system has to succeed in the industrial market, it has to be economically competitive.

2.2. Hardware platform selection

Looking at the architectural design from a software point of view, most of the details of the hardware platform are really not very relevant. But the control unit and its capacity are directly related with the possibilities of the developed software, so it has to be considered at a very early stage of the software architecture design.

A typical approach to solve the problem of having two clear necessity types in a single system is the combination of two process and control elements, such as an application MPU and a real-time MCU. But that increasesthe hardware complexity, and needs a special work in the integration of both systems and communications between them. Another approach to solve the problem can be covering both necessities with only one process unit. However, that solution increases the software architecture complexity, because each application may require a different architecture due to their different characteristics.

Nowadays, and thanks to Moore's law and the integration of multiple processor cores in a single chip, it is possible to look to a solution with different type of processors in the same controller chip, avoiding the hardware and integration complexity, and also the problems related to architectural complexity.

The Freescale Vybrid™ controller presents an asymmetric dual-core solution, combining a Cortex-A5 application processor and a Cortex-M4 for real-time control. The main benefit of this multi-core architecture is that

a single chip is able to concurrently perform both activities required by the system, running a user interface application with high system requirements on the Cortex-A5 processor, and a real-time deterministic application on the Cortex-M4 processor. An API for the communication between the two cores, called Multi-Core Communication (MCC),and tools to debug both cores and their communication are also provided. With the integration happening on-chip, low power consumption is achieved, helping to lower the overall system cost. Considering all these advantages, it was the selected hardware platform.

2.3. Operating systems

After the selection of the hardware platform, the next step is the decision of using operating systems or not, and if that was the case, selecting which ones. The dual core nature of Vybrid micro-controller offers the possibility of having different operating system on each core, a high-level operating system in the Cortex-A5 core and a real-time operating system in the Cortex-M4 core.

Linux and MQX were chosen for the Cortex-A5 core and the Cortex-M4 core respectively, to support the system requirements of the applications to be executed in each of them. Manufacturer recommendation and support for that combination was also an important point on the operating system selection.

2.4. Model Driven Development

Regarding the development methodology, Model Driven Development (MDD) was selected as the method for the design and development of the applications. Several studies [2,3,4]show the benefits of using Model Driven Development, including productivity increase, development process simplification, and development time reduction.

IBM Rational Rhapsody was selected as the development tool, being a well known, proved and mature Model Driven Development tool. This means that application models were created in Rhapsody, and the tool generated the code automatically from the designed models. The Rhapsody's automatically generated code has some particularities, one of them being that it should be deployed to the target using some specific Rhapsody frameworks. IBM offers different deployment frameworks, specifically adapted to different execution environments and programming languages.

For our case, we selected a different execution framework for each core, due to the particularities of each core and operating system. For the Cortex-A5 Linux core, we selected Standard Object Execution Framework (OXF). It is the most complete execution framework, provides all the functionalitiesrequired by the Cortex-A5 application, and supports C++ code, the language selected for that core application. It doesn't provide the real-time functions, like periodic scheduling and deterministic control, but they are not required in that core application. For the Cortex-M4 MQX core, real-time services are required, and the core capacities are also more limited than the Cortex-A5 core, so based on that we selected MicroC Execution Framework (MXF), which also constrains the programming language C, which was selected for that core control application.

2.5. User interface

For the graphical user interface, we choose Qt framework for support the application development. Currently being developed both by the Qt Company and the open-source Qt Project, it is a cross-platform framework, well known for its features to develop high performance user interfaces. Offering the same power and speed of native applications, Qt has multimedia, 2D and 3D graphics support, including image animations, and other functionalities to help the work with SQL, XML, unit testing and multilingual support. It also provides features for reproducing audio and video files.

It can be said that the selection of Qt was a smart choice, because apart from providing such a large amount of functionalities and graphics support, being cross-platform and supporting many compilers it was very helpful for the development process. It was possible to run and debug the application on the development environment before doing it on the target platform, allowing for early validation of the user interface, probably the hardest part of the system to validate, due to human factors and personal tastes. The early validation of the user interface also greatly

improved the development time.

2.6. Overall system architecture

After the analysis and selection of the different parts of the system, the architecture for the solution was specified, making a whole system from the different parts.The Fig. 1 shows the solution architecture block diagram from a high level perspective.

The resulting architecture is able to run a real-time control application together with a high performance human-machine interface application in a single chip. Both applications run using the corresponding Rhapsody framework, and the communication between them can be easily implemented using the Multi-Core Communication API provided by Freescale.

3. Building the platform

With the system architecture specified, the platform itself was built and the development environment was prepared for the development of the applications. For building the platform, some steps have to be followed, like selecting the packages and installing the operating systems, preparing the system scripts for the launch of the applications and opening the multi-core communication channels among others. Only one of those steps is considered crucial and isgoing to be explained in detail, the process of adaptation of the Rhapsody frameworks to run over the different cores. Regarding the development environment, some integrations where done between environments and tools, to create the most integrated and practical environment for the developers. Two interesting points are going to be described in detail, the use of Rhapsody configurations and the Qt profile for Rhapsody.

3.1. Rhapsody frameworks adaptation

The real-time control application and the human-machine interface application were developed using a model driven development tool, Rational Rhapsody. This tool automatically generates the code of the applications from the designed models, and that code has to be run over one of the Rhapsody frameworks. It is important to notice that the frameworks need to be prepared before using, in order to adapt them to the different environments. It is possible to perform that adaptation using the source files of the frameworks provided by IBM.

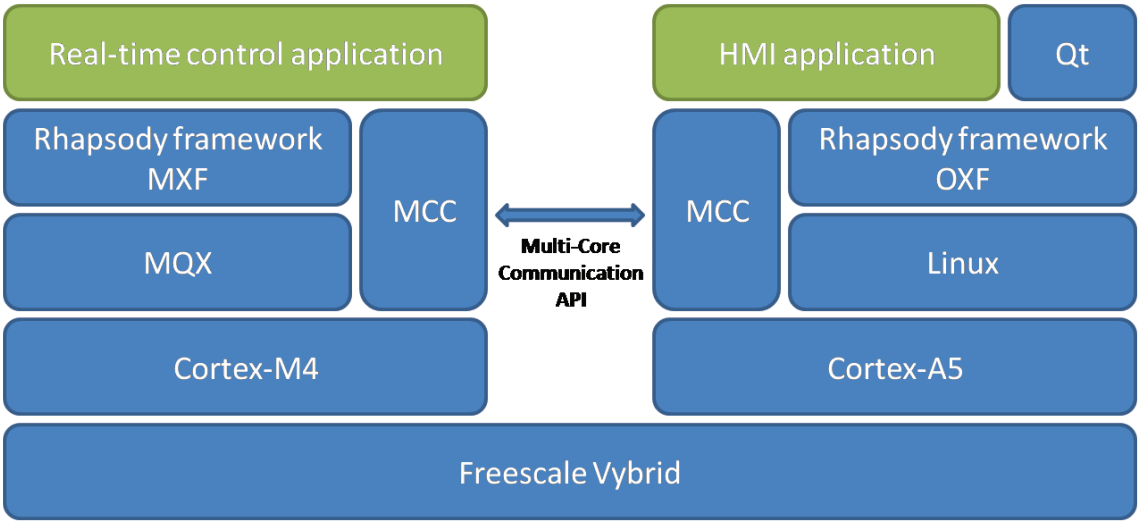

Fig. 1.Overall system architecture.

The OXF framework for the Cortex-A5 core was built cross-compiling the source code for the correct processor architecture. Once it was compiled in form of a set of libraries, it was included as part of the human-machine interface application when compiled and deployed. In the case of the MXF framework, another strategy was followed to compile and deploy it, as it requires an adaptor that receives the task information from the operating system and sequences the framework's internal tasks scheduling. An adaptor of that type was created, receiving the task information from MQX and scheduling the real-time control application tasks. The adaptor and framework source files were not built as separated libraries. Instead, they were cross-compiled as a separated package together with the real-time control application.

Regardless the framework preparation strategy, in both cases the application and the corresponding framework were compiled in a unique binary file, and when those binaries are invoked, the framework code takes control of the execution thread and runs its application, scheduling or generating the required threads.

3.2. Use of Rhapsody components and configurations

When developing large applications with diverse functionalities, especially when a complex user interface is involved, a traditional approach in software engineering proposes that the user of the application defines a set of requirements before starting the development, and then gives them to the application designers and developers for the design and implementation process.

But with that procedure sometimes problems appear when once the application is implemented and the user tries it, realizes that the defined requirements don't fit well with the real needs of the application, or all the requirements where not defined. Many times that happens because the user did not have all the necessary information before starting the project implementation, about how the application should work or should respond in specific situations, becausehaving never seen the application working before, the user has never imagined that situations could happen.

Early validation techniques, proposed on agile software development methods, help to avoid those types of problems [5]. The techniques propose to involve the end user in an agile manner within the development process, validating as much as possible of the functionalities, in order to get early feedback and adapt the existing requirements to the real needs, or specify new requirements that otherwise are difficult to anticipate.

Also, due to the actual fast time-to-market of this type of products, hardware and software are usually developed in parallel, and therefore it is possible for the software development team to be without a working hardware platform until the software application is completely developed. To be able to perform an early validation of the applications, the software must be able to run in a simulation environment, preferably in the host development environment.

Rational Rhapsody helps achieving that goal. It is possible to create and use different Rhapsody Components and Configurations, in order to define different execution environments for the same application. For example, it is possible to define two working contexts, a debug context for the development environment, and a release context for the real hardware environment. With that situation, it is possible to select the debug context and compile, run, debug and animate the application in the development environment, and perform the proposed early validation with the user. After that, simply switching the configuration, it is possible to generate and build the code for the target hardware.

3.3. Qt profile for Rhapsody

For the development of the graphical user interface using Qt, the Qt framework itself offers a development environment called Qt Creator, where it is possible to graphically create the application screens, and later define the dynamic behavior and user interface general relations by code. But in this architecture, the applications were developed using model driven development in Rhapsody, and the code automatically generated from it, so some kind of integration was required between Qt user interface objects and the Rhapsody behavioral models.

To perform the integration between Qt and Rhapsody, and experimental profile for Rhapsody was used. The Rhapsody Qt profile was integrated on an existing Rhapsody project, and provided all the functionalities required to work with Qt framework in Rhapsody. The Qt screens were graphically designed with Qt Creator, and then the files imported to Rhapsody, automatically generating a class that defines the layout and elements for each Qt screen.

After that, another QtObject class had to be manually created to instantiate the screen class. That is where the dynamic behavior of the screen is defined, using the Qt libraries if necessary. Finally, it is possible to define the navigation between screens in Rhapsody. With all in place, Rhapsody generated all the source files required to build the application.

4. Results

The designed system and software architecture has proved to be a very valuable solution when facing the challenge of integrating real-time control and high performance user interface. The platform is able to provide both functionalities in a low power consumption and reasonably priced CPU, giving an important competitive advantage in the actual industrial market.

Regarding the software development process, the integration of Qt with Rhapsody and the simulation of the application in the development environment have allowed performing an early validation with the application user, resulting in an early feedback and improvement of development time and effort.

5. Conclusions and further work

Although the designed architecture was created for a particular application, the architecture design was done with reusability purpose in mind. It is possible to reuse it to create other real-time systems with high performance user interfaces, probably with slight modifications to adapt to the particularities of each new system, but keeping all the advantages the architecture provides. Moreover, the prepared tool integration and environment adaptation allows the developers to ask for early validation from the user, avoiding later dissatisfactions due to unknown customer expectations.

There are some interesting areas where the designed platform architecture may be evolved, most notably, the safety-related aspects of the platform and the product usability areas.

5.1. Support for safety-related functions

Even if it was not required for the practical solution implemented, safety considerations were slightly thought about. The main conclusion is that it would be possible to adapt the platform in order to run safety software in the Cortex-M4 core.

Two alternatives were identified to support safety functionalities. One is the use of a safety oriented operating system in the Cortex-M4 core, instead of the current one (MQX). Some work would be required to adapt and install, but it would be possible to use a safety oriented operating system on that core. The other possibility is the use of a Rhapsody framework with safety orientation. Actually IBM offers the so called “simplified versions” of the execution frameworks, prepared to run with less system resources and without an operating system. In the Cortex-M4 core, the MXF framework could be replaced by its simplified version, SMXF. SMXF is prepared to run safety-critical applications that use MISRA-C compliant C language, allowing running safety applications.

A deeper analysis would be required to explore the details of these and other different alternatives, though.

5.2. Product usability

Usability factors are key factors in actual software engineering as they are directly related with the quality of use of the products [6]. It is especially important when working with high performance user interfaces, were the user interacts in many forms with the software. System and software engineers have to put effort on improving their products usability, involving the user in the process to get valuable feedback, in order to have the most usable and intuitive software.

Product usability has not been the main driving requirement in this work; there have been other requirements with more priority. But knowing the importance of the usability factors, the product usability is marked as one of the main areas for further work. Different usability improvement methods are desired to be analyzed in near future, and

also a GUI test automation tool called Squish is desired to be used in order to improve GUI tests and overall product usability and quality.

Acknowledgements

We would like to acknowledge Giulio Santolifor its invaluable work on the Rhapsody Qt profile and his predisposition to share that work and all his knowledge about Rhapsody and Qt with us.

References

[1] S.Kratz, F. Hemmert, M.Rohs, Natural User Interfaces in Mobile Phone Interaction, TU Berlin, 2010.
[2] M. Broy, S. Kirstan, H. Krcmar, B. Schältz, J. Zimmermann, What is the benefit of a model-based design of embedded software systems in the car industry?, TU Munich, Altran, 2012.
[3] F. Fleurey, E. Breton, B. Baudry, A. Nicolas, J-M. Jézéquel, Model-Driven Engineering for Software Migration in a Large Industrial Context, IRISA, Sodifrance, 2007.
[4] M. Azoff, The Benefits of Model Driven Development, Butler Group, 2008.
[5] P. Abrahamsson, O. Salo, J. Rankainen, J. Warsta, Agile software development methods – Review and analysis, VTT Electronics, 2002.
[6] N. Bevan, Usability is Quality of Use, NPL Usability Services, 1995.