A Dependable Real-Time Platform for Industrial Robotics

Goran Mustapic, Johan Andersson, Christer Norstrom

ABB Robotics, Västerås, Sweden

goran.mustapic, johan.x.andersson, christer.e.norstrom @se.abb.com

Abstract

Industrial Robots are complex systems with hard real time, high safety, reliability and availability requirements. Robot Controllers are part of these systems and they are complex hard real time computers, which control a robot's mechanical parts. To be useful, Robot Controllers must be programmable by end customers. This is typically done through a domain and vendor specific programming languages.

In this position paper, we will describe some of the architectural challenges we are facing and work we have done, in the process of turning the Robot Controller from an application platform into a dependable platform whose base functionality can be extended by a third party which is not necessarily the end customer.

1. Introduction

Industry demands for safety at work and 60.000 hours of mean time between failures put high demands on the quality of hardware and software of industrial robots. Industrial robots are systems, which consist of a mechanical unit (robot arms that can carry different tools), electrical motors, Robot Controller (computer hardware and software) and clients. Clients are used for on-line and off-line programming of the Robot Controller.

The focus of this article will be the open software architecture of the Robot Controller. According to Issarny [6], in open systems, components do not depend on a single administrative domain and are not known at design time. In this article, we describe a domain specific platform, which faces significant new challenges on the way to become an open platform.

The reason for opening up the controller for third parties is to increase the possibility for partners to provide functionality that ABB Robotics do not either find prioritized or do not have resources for. In a closed platform, the development organization responsible for the platform is the limiting factor. To increase the development speed in the future we can either increase the size of the development organization or open up the system for third party. We believe in the latter, that the number of new types of usages of the robot will increase if we let niche companies adapt the robot for a specific type of applications and customers. The challenges can be divided into:

- developing an appropriate business model,
- determining what type of extensions that should be supported,
- defining an open and dependable architecture,
- defining the certification process and technical details

In this paper we will focus on the technical challenges.

Since we have the system responsibility towards our customers we have to ensure our customers that extensions made by a third party do not have negative side effect on the delivered system. This is a big difference compare to the desktop based systems, where a company that builds a system on top of for example Microsoft Windows© is responsible towards the end customer of the system quality.

The contributions presented in the remainder of this paper are the following:

- We make a short analysis of an industrial robot system and analyze the relevance of the individual dependability attributes for the industrial robot domain.
- We present some initial thoughts on the architectural level reasoning about the open dependable platform for the Robot Controller.
- We present the results of the work to model the platform and enable early reasoning of the architectural level choices.

The outline of the paper is as follows. Section 2 presents a short background about ABB Robot Controller. Section 3 explains the relevance of the individual dependability attributes for our system. In Section 4 we discuss similarities and differences of an open Robot Controller and open desktop systems. Section 5 describes the initial ideas about an open architecture and also the results of an initial case study to model our control system. Finally, in Section 6 we make some conclusions and discuss future work.

2. Background about ABB Robot Controller

The ABB Robot Controller was initially designed in the beginning of the 1990. The requirement was that the same controller should be used for all different types of robots, and thus the architecture is a product line architecture. In essence, the controller has an object-oriented architecture and the implementation consists of approximately 2500 KLOC of C language source code divided on 400-500 classes organized in 15 subsystems. The system consists of three computers that are tightly connected: a main computer that basically generates the path to follow, the axis computer, which controls each axis of the robot, and finally the I/O computer, which interacts with external sensors and actuators.

Only one of those three mentioned computer nodes is open to the end users, and that is the main computer. End users write their programming logic in the form of an imperative language called RAPID. This can be done through off-line programming on a PC, or on-line programming on a client called Teach Pendant Unit (custom hardware).

The system was originally designed to support easy porting to new HW-architectures and new operating systems. There were no initial requirements to have an open architecture. Further, the system was not initially designed to support temporal analysis, because of its closed nature and limited amounts of changes that could only be done by the internal development groups.

3. Discussion of dependability attributes in the context of industrial robots

The analysis of the dependability attributes in this section is done using the terminology presented by Avizienis, Randell and Laprie in [2]. Dependability is described by the following attributes: Reliability, Availability, Safety, Integrity, Confidentiality, and Maintainability.

Security related attributes (confidentiality and integrity), tend to be of less importance for industrial robots as robots tend to be physically isolated, or only connected to a control network together with other industrial devices. Integrity of data which is not security related is very important, as it is unacceptable that e.g. one task in the system causes a hazard situation by damaging the safety subsystem. All other dependability attributes are very relevant.

Even though the contact of humans and robots in industrial environments is restricted (robots work in their cells, which are physically isolated by a fence), safety can never be underestimated because a lack of safety can cause substantial physical damages to the robot equipment and its environment. For example, larger types of robots are powerful machines capable of manipulating a weight of 500 kg. Industrial robots belong to the category of safety-critical systems, which do have a safe state.

Because of the nature of the application, it is crucial to have very high availability and reliability. Unreliability leads to un-availability, which means production stop and huge costs. Because of the complex setup of e.g. car production line, a stop of a single robot leads most often to unavailability of a whole production line. In a complex case, a stop of a single robot can cause up to one day production stop.

Maintainability is important in the sense that it is related to availability. The shorter maintenance time the higher is availability of the system. Ideally, the system should be upgradeable without stopping the production. System upgrades are complicated even for a closed platform, but get much more complicated in a platform which is extendable by a third party, because of the compatibility issues.

When it comes to the dependability threats – fault, error and failures, both hardware and software faults need to be considered. Robot Controller software has both roles of sending control sequences to the hardware as well as predicting preventive hardware maintenance.

There are many different fault-tolerance methods that can be applicable for industrial robots. Error recovery with temporary graceful degradation of performance is not acceptable. A robot either works or it does not work; it cannot make its tasks by working slower depending either on the task (such as arc welding) or because it is a link in the production flow.

4. Towards an open architecture

In the introduction of this paper, we presented some of the motivations for opening up the system. In this section we will try to describe similarities and differences between industrial robots and open desktop platforms where openness is taken for granted. We will conclude this section by a short analysis about which dependability attributes are most threatened by opening up the system.

4.1. A comparison to Windows© platform

Good examples of open platforms are Microsoft Windows© and perhaps even more Linux operating systems. We shall take the example of Microsoft Windows©, which is closer to our case. It is possible to extend the base platform on three basic different levels: device driver level, win32 programs and .Net applications. This is illustrated in the Figure 1.

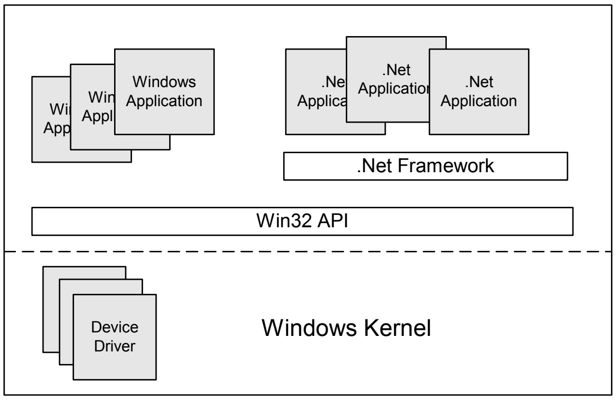

Figure 1: *Different ways to extend Windows© platform*

The architecture of the system guarantees that each of the different extensibility mechanism only can make certain amount of harm to the system, where device drivers can do the most harm and .Net application least harm. Apart from the basic or native ways to extend the platform, many of the applications define their own extensibility mechanisms, e.g. Internet Explorer and SQL Server.

The current way of adding functionality to the Robot Controller corresponds to adding .Net applications to Windows©. As previously mentioned, this is by adding RAPID programs. This is shown in the Figure 2.

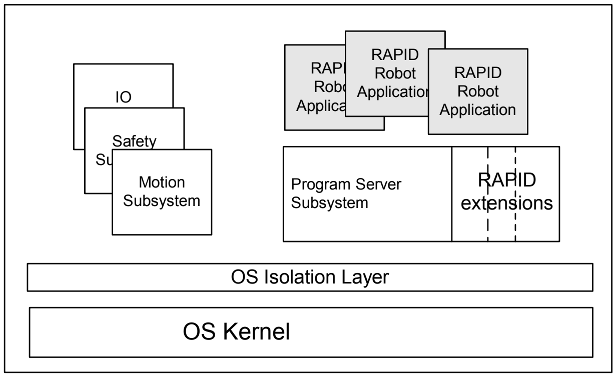

Figure 2: *Current ways to extend the Robot Controller software*

We are considering the following additional ways of extending the Robot Controller:

- Extensions to the robot programming language RAPID
- New subsystems
- Extension logic to the existing subsystems (e.g. fine-tuning of the robot path, new type of IO card, new types of sensors for fine tuning of the robot path etc.)

Let us consider the first type of extensions – RAPID instructions. Basic commands in programming of a robot are "motion" instructions instructing a robot to move its arms to different positions. Some of the basic motion commands are implemented as RAPID extensions and perform their tasks by communicating to the motion subsystem. The basic part of the Program Server contains the engine for executing RAPID programs, and has features like single stepping and executing the program backwards (that is, running the robot backwards). There is a limited set of commands that make the robot programming language. New instructions can be added to enable easier programming and facilitate e.g. very special kinds of tools that a robot is using. This has traditionally been restricted for in-house development because of the harm these extensions can do to the system and prohibitive costs of verifying the correctness of extensions. This harm is equivalent to the harm that unmanaged (native) code can do to a .Net application. This extension code can bring the .Net application down and .Net Framework has no ways to prevent it from doing this. In the Robot Controller case, situation gets more complicated when timing requirements of the Robot Controller are considered.

Other types of extensions mentioned are potentially even more dangerous to the overall system because they most likely require more open access to the lower level services in the system.

4.2. Revisiting the dependability attributes

If we shortly revisit the dependability attributes, which we have analyzed in the section 3, we will see that opening up the architecture will have some significant consequences for several of the attributes. In particular: reliability, availability and safety of the system are threatened by a third party code. Maintainability of the system gets much harder because of the need to handle versioning problems between the platform and the extensions.

Thus we have to find a dependable architecture to support extending the platform in a predictable way.

5. Initial architecture reasoning about open Robot Controller platform

We see the following as the most important architectural goals and also biggest challenges we are facing:

- Defining the dependable platform architecture
- Good support for the development of extensions
- Support for the predictable assembly of extensions and the platform

The platform will provide a Software Development Kit (SDK) for developing the platform extensions. SDK functionality may be grouped with focus on different types of extensions. Support for modeling and simulation will be a part of the SDK, as well as a framework for certification of extensions.

Because timing aspects are crucial in the Real-Time environment, we have already done some initial studies on modeling/simulation of the system, to be able to verify the architectural design in early stages. The rest of the work mentioned below is in an initial phase.

5.1. Platform architecture

The work on architecture for dependable systems is relevant in the context of defining the architecture of an open dependable platform. According to [3] Non-Functional Requirements (NFR) can be divided to: Separation, Additive (a subset of the Separation) and Integral NFR. The classification is based on the way NFR can be taken into account in the system architecture. Additive non-functional requirements (AFNR) are pure add-on components to the architecture, while integral non-functional requirements (INFR) can affect the components of the entire system.

In the case of an open dependable platform, we will need to use means for dependability to create a framework for adding extensions. The choice of the architecture and implementation of the dependability requirements, will lead to this framework. Means for dependability can be transparent, with a different degree of transparency, to extension developers. We believe that for a hard-real time system, extensions of the platform will have to be aware of the platform dependability requirements. An example is the timing requirements.

Some of the existing frameworks for implementing fault-tolerant software present interesting ideas that we may benefit from. An example is a framework for implementing complex fault-tolerant software presented by Xu, Randel and Romanovsky in [13].

It is also important for the software architecture to support good testability, which is a contradictory requirement to fault-tolerance. Example of the work in this field that is relevant for us, is the work done by Voas [9,10], and especially some of the work in the area of distributed real time systems [8].

5.2. Research in Component Based Software Engineering (CBSE)

The extensions of our platform could also be called components. Component-Based Software Engineering (CBSE) and Software Architecture research are much related [4] and experiences from this field can help us in architecting our platform. It is recognized that current Component Technologies handle only syntactic aspects of component compositions, while semantic and especially extra-functional (non-functional) aspects of component specification are open areas of research [4]. One of the biggest challenges of the CBSE is predictable assembly of components and an example of a technology in this research area is called PECT. We believe our ideas are quite inline with the PECT framework presented by the research group at SEI Carnegie Mellon University in the article “Packaging Predictable Assembly” [5]. In this article, prediction-enabled component technology (PECT) is presented, as both a technology and a method for producing the instances of the technology. A PECT instance is created by integrating a software component technology with one or more analysis models. However, focus of their work seems to be more on integration of the existing technology and models, while our focus will be more towards defining a dependable platform and a simple custom component model with good dependability characteristics.

Besides the predictable assembly, research experiences in CBSE can be very useful for handling problems of maintainability and compatibilities between platform and extensions.

5.3. Certification of extensions

An example of a certification process for platform extensions is Microsoft’s WHQL (Windows Hardware Quality Lab [7]) certification program for the device drivers. Some ideas from this process are definitely applicable to our case. We would also need to act as a certification authority. In our case, the situation is more complicated because it is not only the certification of a single component we are concerned about, but also the already mentioned predictable-assembly. One of the possible approaches to certification of COTS software is presented by Voas in [11].

5.4. Model Checking and Simulation

In an open real-time system, with third party components we need a method for extending the base system in a safe way. Since the robot system is very sensitive for timing errors, we have developed a method for analyzing the impact of adding third party components. In a large system such as the robot controller, this temporal side-effect is hard to predict without models, due to the size and complexity of the system.

Testing and debugging of a complex real-time system is already difficult and when introducing a low-level interface for extending the system with new components makes it even harder. Now, not only the base system has to be verified, but all combinations of extending components that are to be used must be verified as well.

A component might work perfectly, but when it is combined with another component it might introduce an overload situation in some scenarios. This is dangerous since an error caused by this side-effect can be hard to find by testing and affects not only the current task, but may cause a global overload situation, possible delaying several tasks, causing multiple task deadlines to be missed, and finally that the robot fails doing its task properly.

Often the manufacturer of the base system wants to focus on the performance and features of the base system, not integration of special third party components. Letting the component manufacturer be responsible for the extended system is not better and in the industrial robot business, big customers do not accept this. The third party developers will not have access to the source code of the base system, except perhaps an SDK, and will not have the same expertise in the internal structure of the system. Also, they are probably not able to achieve the same quality on their system verification as the base system manufacturer.

A better solution is to let the component developer create a model of their component and let the base system manufacturer certify the component. Their component can through the process be certified for use with the system as well as together with other certified components. This is good for the component developer since it provides a quality label for their software and the base system manufacturer can sell more base systems.

We have developed a method and a prototype tool for describing and analyzing these models. The approach is developed for a robot controller, but the method can be used for other systems as well.

In earlier work, the language-based simulation tool-suite called ART-ML [1,12] was developed, a model creation process has been developed and using it we created a rough model of the controller. A specialized query-language for powerful analysis simulation results as well as data measured on the target system has been developed [12].

So our approach consists of one base step were we develop an initial model of the system and validate that the model represents the modeled system correct. When we have a model of the base system we can add component models to the base model and analyze the consequences of adding a particular set of components. Currently this analysis will be performed off-line but in the future we could do this even on-line.

When creating an initial model $M_{0,}$ of an existing system S_0, several distinct activities are required. These activities are depicted in Figure 4. First the structure has to be identified and modeled, i.e. the tasks in the system and synchronization and communication among them. In the next step, we measure the system and populate the structural model with data about the temporal behavior. Moreover, information needed in the validation phase is collected, e.g. response times. When tuning the model, the initial model M_0 is compared with S_0 by simulating the model and comparing the results with the validation data collected in the previous step. In this step it is possible to introduce more details about the tasks behavior in order to capture the system's behavior accurately.

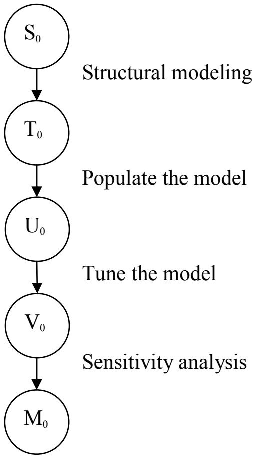

Figure 4: *The model creation process*

To validate the usefulness of the model it is necessary to perform a sensitivity analysis. The sensitivity analysis should be based on foreseen potential extensions or changes in the particular system. The extensions are introduced in the model as well as in the system and the new systems are compared. Any divergence between the behavior of the simulated model and the system indicates that more details must be introduced in the model. This increases the confidence in the created model.

In the robot controller we have studied, the following typical changes were identified:

- Change existing behavior of a task which results in changes in the execution time distribution.
- Add a task to the system.
- Change the priority of an existing task.

When a third party adds a component to the system a model of the added component has to be provided. This model is composed with all other added components' models and the basic system model. Based on this composed model, we can verify if the defined properties still hold for that particular combination of components, and if so draw the conclusion that the added components do not affect the system behavior badly.

6. Future Work and Conclusions

We intend to continue our work in the direction of defining architecture of a reliable, safe and maintainable platform. We will also continue working on the modeling and simulation to support the predictable assembly of the platform and extension components.

From the initial analysis, which we have presented in this paper, it can be seen that we will need to use experiences from multiple areas of research and to combine them to create an optimal domain specific solution.

Besides the technical challenges, there are also significant business challenges around the certification process. Without proper architecture, tool support etc, costs of certification may turn out to be larger than the benefits of having an open platform.

7. References

[1] Andersson J., Neander J., Timing Analysis of a Robot Controller, Master Thesis, Mälardalen University, Sweden, http://www.mdh.se/, 2002.

[2] Avizienis A., Randell B., and Laprie J.C., "Fundamental Concepts of Computer System Dependability", IARP/IEEE RAS Workshop On Robot Dependability, 2001.

[3] Brandozzi M. and Perry E.D., "Architectural Prescriptions for Dependable Systems", ICSE 2002 Workshop on Architecting Dependable Systems, 2002.

[4] Crnkovic I., Hnich B., Jonsson T., and Kiziltan Z., Specification, Implementation and Deployment of Components, *Communications of the ACM*, volume 45, issue 10, 2002.

[5] Hissam S., Stafford J., Wallnau K., and Moreno G., "Packaging Predictable Assembly", Proceedings of the First IFIP/ACM Working Conference on Component Deployment, Berlin, Germany, 2002.

[6] Issarny V., "Software Architectures of Dependable Systems: From Closed To Open Systems", ICSE 2002 Workshop on Architecting Dependable Systems, 2002.

[7] Microsoft Corporation, *Windows Hardware Quality Lab homepage*, http://www.microsoft.com/hwdq/hwtest/, 2003.

[8] Thane H., "Monitoring, Testing and Debugging of Distributed Real-Time Systems", Doctoral Thesis, Royal Institute of Technology, KTH, Mechatronics Laboratory, TRITA-MMK 2000:16, Sweden, 2000

[9] Voas J., "Factors That Affect Software Testability", Pacific Northwest Software Quality Conference, Inc., 1991.

[10] Voas J., *Software Assessment - Reliability, Safety, Testability*, John Wiley & Sons, Inc, 1995.

[11] Voas J., A Defensive Approach to Certifying COTS Software, report RSTR-002-97-002.01, Reliable Software Technologies Corporation, 1997.

[12] Wall A., Andersson J, and Norstrom C., "Probabilistic Simulation-based Analysis of Complex Real-Time Systems", will appear in the 6th IEEE International Symposium on Object-Oriented Real-time Distributed Computing, Hakodate Hoikado, Japan, 2003.

[13] Xu J., Randell B., and Romanovsky A., "A Generic Approach to Structuring and Implementing Complex Fault-Tolerant Software", nr 5th ISORC'02, IEEE Computer Society, 2002.