Adaptive Robot Design and Applications in Flexible Manufacturing Environments

Ning Gui *, Vincenzo De Florio*, Gabriella Caporaletti** , Chris Blondia*
PATS group, University of Antwerp, Belgium,
*and IBBT, Ghent-Ledeberg, Belgium**
*EICAS Automazione S.p.A.***
***{ning.gui, vincenzo.deflorio, chris.blondia}@ua.ac.be** **
***gabry@eicas.it** ***

Abstract: Robots have played a very important role in the growing popular flexible manufacturing environments. However, state-of-art industrial robots with high accuracy are rather costly and static.

Our works aims at providing a low-cost fast integrating platform with advanced middleware support to seamlessly integrate off-the-shelf or future robot sensors, robots, and actuators as well as industrial IT system. To support such approach, a component-based reconfigurable middleware system is designed. A system runtime service is employed to manage the dependence and whole lifecycle of realtime components by reasoning from component's contract-based service description. A continues deployment mechanism is also designed The software architecture was implemented by so called – Hybrid component model. The evaluation shows that the ARFLEX system achieve the goal of enhance in accuracy, flexibility while provide good real-time characteristics.

1. INTRODUCTION

Flexible manufacturing system (FMS) is a new concept of manufacturing system which generally is characterized by small production batches and big diversity of products. Its fast change-over capability may give the company additional opportunities to react to market developments and receptivity. For these reasons, it is regarded as a trend for future industry manufacturing.

However, robotics systems, as promising candidates to bring about such changes, failed to play such a role so far. In order to guarantee the required accuracy and efficiency, current manufacturing robots are normally designed case-by-case and optimized to perform just one or limited tasks(Stewart et al., 1997). As a consequence, robot automation technologies have mainly been confined to those capital-intensive large-volume manufacturing environments. Those resulting costly and complex robot systems are too static to be used in FMS environments. Although there are some off-the-shelf robots for general applications, however, due to cost consideration, their platform is normally characterized by limited power and extensibility and cannot easily integrate COTS sensors & actuator technologies.

All these facts show that in order to come up with a truly Adaptive Robot for Flexible manufacturing environments (ARFLEX), a totally different design methodology is needed. Rather than designing an industrial robot for each given application, ARFLEX aims at providing a low-cost fast integrating platform with advanced middleware support for automation modules. By seamlessly connecting off-the-shelf or future robot sensors, robots, and actuators as well as industrial IT systems, existing industrial robot's accuracy and adaptability can be enhanced at the same time.

This document describes the design strategy and structure of ARFLEX platform. After analyzing the multi-sensor real-time integrations requirements in different aspects, we focus on the design of a software platform that makes such kind of fast integration possible. This platform is designed to support system modules (component) reconfiguration and adaptation while still providing the real-time guarantee. System Modules adaptation includes adding and removing sensors, changing configuration of modules and even manufacturing process mutation. In this framework, a declarative real-time component model is proposed. In order to enhance the composability and reconfigurability, the component is designed by using contract based component description. A Declarative Real-time Component Runtime (DRCR) is implemented to support the component dynamicity, resolving the component dependence and manage installed component's real-time contracts. In this platform, DRCR resolves the component constraints, composes the compatible components and performs correspondent initialization processes and creates (destroys) newly (un)satisfied components. A continuous deployment mechanism is also supported in this platform which enables the system to install, update, and uninstall the bundles without requiring the whole system to be restarted.

The rest of this paper is structured as follows: in Section 2 we give a general overview of our ARFLEX platform design methodology. Then in Section 3 the adaptive component based middleware is introduced to support the real-time,

dynamicity and reconfiguration capability. Then in section 4, the middleware's implementation methods are introduced. Section 5 provides a general evaluation to the ARFLEX platform in three different aspects. Future work and our conclusion are given in the final section.

2. SYSTEM HARDWARE STRUCTURE DESIGN

The ARFLEX concept is to get a significant improvement of accuracy, flexibility and adaptability of industrial robots by providing the field with state-of-the-art and future technologies, such as advanced control theory, new sensor devices, and electronic embedded systems. In this paper, we focus on ARFLEX software platform design because it is the key point in achieve system flexibility, details can be found in the ARFLEX reports (ARFLEX, 2008).

2.1 Dual-level control loop

ARFLEX, without requiring any change of what exists, aims at introducing a new control loop that, starting from the measurements given by a new sensor system (ARFLEX sensors), generates a correcting command to be added to the present reference signal coming from the trajectory module. Through this correction of the present reference signal we obtain the precision improvement of the robot performance.

The ARFLEX control is conceived following a hierarchical approach that allows keeping the existing robot and the relevant control algorithms essentially unchanged; moreover, ARFLEX controller, working as a integration platform, greatly expand the typical robot system expansibility by making the system modular and distributed at the same time. Vision sensors, force sensors and different actuator modules can be managed, with help of middleware based software platform, be easily tailor to a large set of application requirements.

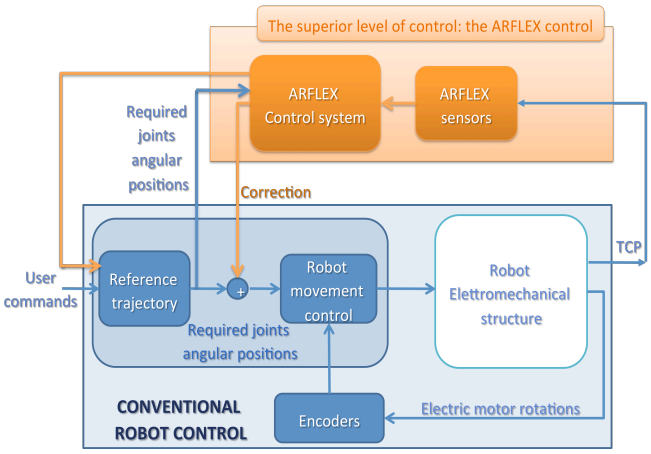


Fig. 1. Functional scheme of the interaction between ARFLEX and conventional control

2.2 Enhanced Accuracy Approach

The robot considered is assumed to be equipped with a conventional position control loop (based on the electric motor angular position measurement and control). It can only offer a performance satisfactory from the point of view of the "repetitive position accuracy", however, in manufacturing environments, the "absolute position accuracy" is generally demanded(Wall et al., 2002).

ARFLEX platform achieves the "absolute position accuracy" by using computer video sensor technologies as well as its two level control loop. The vision sensors are installed around the working area of the robot and are deployed so that the robot can be seen in the best way in all working conditions. A set of markers are distributed on the robot surface and a suitable illumination system makes visible the mechanical structure movements to the vision system. At the higher level, the tool pose control rebuilds the TCP position by means of vision sensors and advanced 3D vision algorithms, and generates the corrections to be provided to the conventional robot control, at the lower level. The smart camera is calibrated with the marks pre-placed on the working places. The 3D processing algorithm can be found in our previous work (Caporaletti).

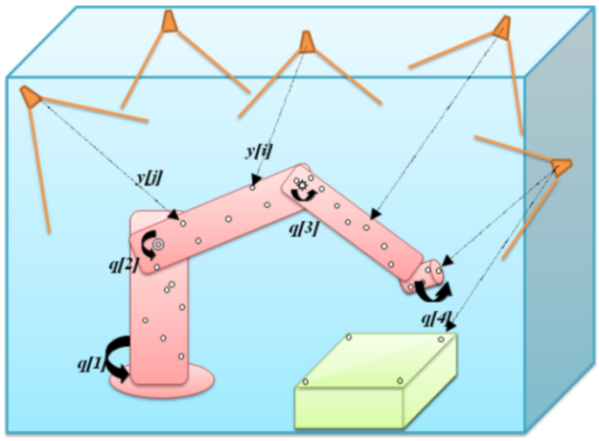


Fig. 2. ARFLEX measuring system based on markers placed on the robot

2.3 Force sensor aided control

Force sensors are also integrated in the ARFLEX platform to promote the system security. They are mounted on the joints of the robot arm. When the force is above certain threshold, it will notify the ARFLEX controller to enter the safe mode. The power output of different servo motor is limited to certain range. It can also be used to teach the robot the trajectory that the robot should follow rather than hand- code the whole trajectory.

Here, we can see that by integration of new sensors technologies, traditional robots can fit with many different domains. The vision and force sensor integrations are just two typical applications of ARFLEX platform. ARFLEX system can also be extended to support reconfigurable actuator modules. It supports easily changing the actuators fixed on the robot arm to perform different tasks.

3. SOFTWARE PLATFORM DESIGN

ARFLEX system integrates with distributed and heterogeneous hardware and software modules, such as the vision sensors, force sensors, actuators and controller modules. To cope with this increasing complexity, the component-based software engineering has been used in our software system design. By dividing software systems into

manageable parts, these components can be developed largely independently and reused many times in different application contexts.

However, using the component-based approach itself could not satisfy the adaptive and reconfigurable requirements in FMS. Most real-time component-based software systems use glue-code to connect different components. When updating or maintaining these systems, the glue code need to be regenerated. Thus, the whole system need to be shut down for recompilation (Wang et al., 2004). Reconfiguring a system on-line without the need to generate glue code is desirable for complex real-time system that requires continuous hardware and software upgrades during system operation (Stewart et al., 1997). Moreover, in such a complex real-time system, the challenges of implementing real-time behavior include not only decoupling and modularizing the real-time behavior, but also the ability to deal with the adaptation requirements as component deployment, life cycle management, component dependence resolving, etc. All those requirements call for a robotics programming environment which supports reconfigurable robots, able to integrate multiple device modules and support modules dynamicity. At the same time, the hard real-time characteristics should be preserved.

3.1 System design consideration

In order to perform such kind of adaptation and support continuous deployment mechanisms, several issues remain:

Component characteristics Reification: In order to perform the run-time adaptation effectively and correctly, the detailed information about the component's characteristics should be available. However, most of component implementations use simple binary codes as deployment entity. The dependence between components is hidden between the lines of source codes. This makes the software system very hard to manage. In order to tackle this problem, an explicit contract-based component description is designed in our component model together with the deployment codes.

Runtime-based Adaptation: the common way to achieve adaption, to component changes, is to manually write codes which monitor component status changes, such as component departure or arrival and perform the specific adaptation logics. However, this low level programming model is complex and error-prone. This approach could not be easily used by the robot operators who normally are not programming experts.

Component dynamicity support: In order to support the component dynamicity availability and composition, the full control of the component's lifecycle is required. Failing to do so, the system would miss the accurate picture of the installed components. That means a mechanism should be designed to take control of all requests from creation, configuration, reconfiguration and un-initialization (the whole of the components' lifecycle) from underlying OS.

Real-time support: As ARFLEX system operates in high frequency and has very stringent time requirements, the software platform should be simple and introduce minimal overhead to the real-time tasks. We designed a hybrid real-time component implementation that provides a light-weighted solution.

3.2 Contract-based component description

Component-based software development is becoming the mainstream for conventional applications. However, earlier efforts to manually integrate adaptation capabilities from different system layers proved ineffective in coping with the dynamic changing requirements. In our platform, we use meta-data level, contract-based component description to explicitly describe the component descriptions rather than hiding such information inside the codes. In our real-time component model, each component is with an XML file describing the component contract. This contract includes both the functional part and the QoS requirements – the non-functional part.

The functional part includes the component's name, implementation code, the required and provided ports, the port format and communication methods. The properties are used to perform the bindings between input and output ports. Compatibility between input and output ports is used to create the communication links between components.

In addition to the functional descriptions, the non-functional requirements such as CPU, memory, network etc are also specified in the component's contract. This is especially important in complex real-time systems, as one real-time task may have certain impacts on other tasks' real-time performance while competing for the system resources. The description of non-functional part enables the system to perform more fine-grained control in managing the system performance and keeping the system in accordance. Details in the component functional and non-functional meta-data format and the extension design can be found in our previous works (Gui et al., 2008a, Gui et al., 2008b).

3.3 DRCR supported runtime adaptation

The DRCR execution environment works at the core of the software platform. When a component is installed, it will use a parsing service to get the structured component's properties. Then, the component dependence will be analyzed in both the functional and non-functional aspects. Then, it manages installed component instances and performs the actions reasoned by resolving the dependences. It also takes the responsibilities to monitoring the change events sent by Resource Management Service and underlying system. A component instance registry is designed to keep the installed real-time component instances' information and it is maintained by DRCR.

Based on the decisions made by the constraint resolving service, DRCR takes control of the component lifecycle. By taking into account each component's real-time contracts and the current system configuration, it provides the basis to support the component's dynamicity without impairing the deployed component's real-time performance. In current prototype, the only action the DRCR can take is creating and stopping the component instance. In this framework, a complete component's lifecycle model and a corresponding real-time component management interface are designed. The adaptation process will be triggered by component state changes such as component installation, stop or update. It can also be triggered by the notification from Resource Manager for the changes of resource availability. In the following

sections, we provide a more in-depth description of key elements and processes of this architecture.

3.4 Component Lifecycle Control

A real-time component's lifecycle is under full control of DRCR. In doing so, DRCR can keep a complete and accurate global view of current system context. Doing otherwise, i.e. allowing each component to be created or destroyed by its own proprietary interfaces/methods, the system would lose track of the deployed components' state and system resource utilization status.

In our software component model, each component has at least four states managed by the DRCR: *Installed*, *Unsatisfied*, *Active*, and *Destroyed*. Parts of lifecycle changes are driven by external events such as component deployment and destruction (which still need to go through DRCR). Some state changes are automatically managed by DRCR, such as the transition from Unsatisfied to Active. During execution, the DRCR receives notifications from the underlying framework for component state changes as well as notifications from Resource management service for resource changes. These notifications can trigger another round of reconfiguration activities.

3.5 Simplified deployment design

Our platform provides support for continuous component deployments for code updates and configuration. Each component's meta-data contain an entry for this components' update URL. When a user initiates an *update* command for certain components, this framework will initiate a thread to pull the code and configuration information from this URL. After successfully downloading these codes, the framework will automatically install the component into the system. After a component is installed into the system, the DRCR will check and manage the dependence between those related components and perform necessary initiation job.

All these processes are done in the runtime without the need to restart the whole platform. No glue code is needed in system configuration changes. The complexity of component deployment is hidden from the target users. This is very important for the viewpoint of system usage.

4. Hybrid REALTIME COMPONENT

In implementing this framework, there are several issues that need to be considered. Firstly, we want to design a simple adaptive component model which is also largely compatible with one of currently most popular middleware systems, so we can easily integrate many existing useful services, such as logging, device management etc. Our challenges are to do so, with as less impact as possible on the hard real-time tasks.

4.1 Implementation structure

In order to achieve these two different requirements, we designed the Hybrid Real-time component (HRC) implementation**.** Our implementation is based on the OSGi component model (OSGi Alliance, 2008). It provides a very flexible .and extensive component framework. The real-time characteristics are guaranteed by Real-Time Application Interface (RTAI) –an open source real-time Linux kernel extension(E. Bianchi, 2006).

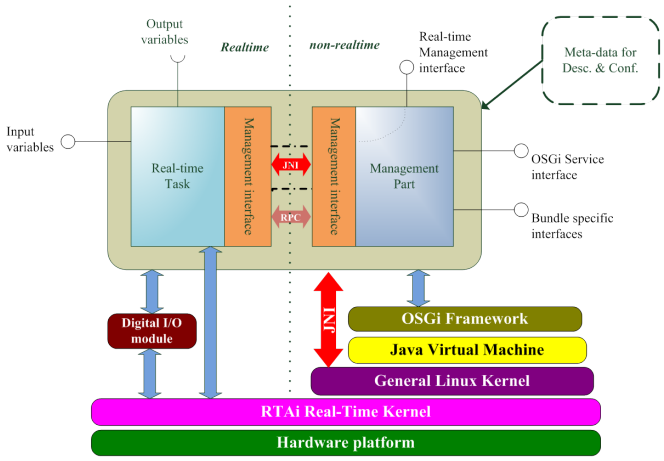


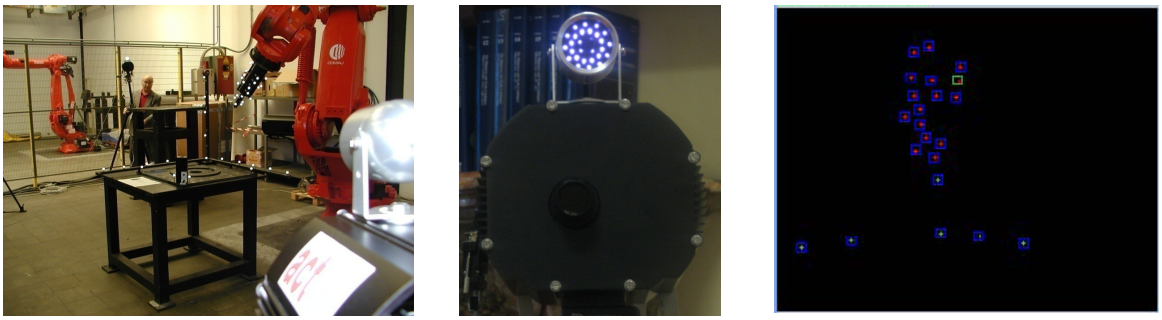
Fig. 3: split container architecture

The real-time part of HRC is implemented as an RTAI concurrent process. This part is implemented following a general template with a set of pre-structured functions. It enables the real-time task receiving data and commands from the non- realtime parts while responding inquiries from its non real-time partner. Each real-time task is initialized by the property values such as name, description, task type and task specific properties described in component description meta-data. Communication with other real-time modules is restricted to component input and output ports. The real-time task can also connect to sensors or actuators, via the digital I/O module. The details of accessing the hardware are encapsulated within the real-time task.

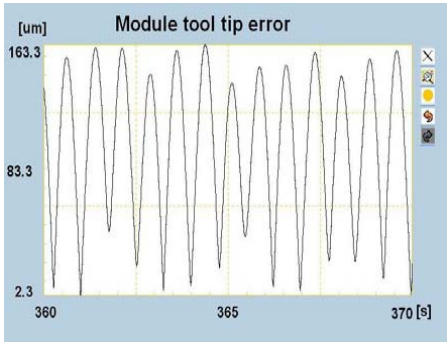
In the non real-time part, we implemented a general management interface containing methods for getting/setting component parameters or getting the component state. This management interface is registered as management service in the OSGi service registry and thus can be managed by external modules. We reused the OSGi framework service to realize the modules deployment, version control, etc.

A key concept in our approach is that we want to separate the actual adaptation logic as much as possible from the real time business logic constituting the components' code. The large and complex adaptation and management parts run in a classic non-realtime environment and only small, predictable parts in a real-time environment. This component model is rather concise and easy to implement compared to the pure real-time component model.

We also designed the formalized intra communication interfaces working as the bridge between real-time code and its non real-time count part. The communication ports between the real-time tasks are designed and explicitly expressed and configured in a specific component's meta-data file. In current stage, our implementation support RTAI shared memory and RTAI mailbox: two simple inter-process communication methods.



(a) (b) (c)



(e) (f)

Fig. 4. (a) The accuracy approach via ARFLEX smart camera (b) The smart camera with IR light (c) the Captured Image from one of the Smart Camera – four in total (d) the static tool tip error without ARFLEX platform control (e) The static tool tip error with ARFLEX platform control.

4.2 Coding

The structure of a HRC is designed so that a system developer can simply define module-specific code, and not be concerned with any of the details of creating a real-time process. A template is created for typical component structure, both in real-time part and non-realtime peers. As the complex and error prone management work is done by the system runtime. The HRC's real-time structure is rather concise and similar to existing real-time system development practice. The only revision is that the real-time developer should and can only communicate according to predefined interfaces and response to very simple commands sent from non-realtime peers. Thus, existing code is easy to migrate to the HRC model, which is very important for ARFLEX platform as many real-time legacy codes already exist.

For the OSGi parts, the component develops the management interface for the DRCR to control its dependence and lifecycle. A template is also provided, and a set of helper classes are provided to assist the communication between real-time codes and the OSGi peer. Component developer should also state the component's communication interfaces (out-ports, in-ports), properties and the resource requirements in the meta-data. As it is written in XML, it is easy to understand and write compared to programming the code. A visual tool is developed in helping system developer & user to write and change the component's configuration.

5. EVALUATION

The above platform has been deployed in the ARFLEX project. In order to evaluate the enhancement of this platform with respect to accuracy, flexibility and the real-time performance, we designed a set of experiments. Due to page limit, here we can only provide a brief introduction.

5.1 Vision sensor aid accuracy enhancement

The general concept of vision sensor aided accuracy enhancement is discussed in Section 2.2. Fig. 4 shows the field image of this approach as well as the preliminary results for the improved accuracy.

Fig. 4a shows Ageneral robot (COMAU NJ 370-2.7) in use with ARFLEX systems, the robot arm is placed with passive marks which are sensitive to the Infrared Radiation. They are placed at predefined distances. The robot arm's actual position can be computed by the reference of the 3D frame. Fig. 4b shows the ARFLEX smart camera with IR source. The smart camera can identify the marks through its embedded computing FPGA and transfer those data back to ARFLEX platform. Fig. 4c shows the image presented by one of the four cameras. Some marks are on the robot arm while some from the reference frame.

The accuracy improvement can be seen from the Tool Center Point (TCP) pose errors of Robot (Fig. 4e and Fig. 4f). By compensating much of the impacts of the errors in the kinematic model and reducing the manipulator deflection under gravity, the ARFLEX vision enhanced robot achieves much better accuracy than the general industrial robots.

5. 2 Application Structure adaptation

Normally, an adaptation is expressed within the component. However, this process is tedious and error-prone and could

not be done at run-time. In our framework, the application's structure can be easily changed without interrupting the other modules' execution.

In this simulating platform, the controller component is designed with an interface that exposes the internal parameters and flag status to the external world. However, in the normal application, this information may not be needed by the end users. When certain adjustment is required, a system debugging component can be remotely installed so as to monitor certain key system properties. When the adjustment process is finished, this debug component can be uninstalled during the runtime. As our component implementation complies with OSGi model, we easily integrated this modules with OSGi's Rich Client Programming (RAP) approach. By doing so, we can present the data through web applications. Remote debug can be achieved without having to implement the whole system from scratch.

Certain adaptation can be automatically performed by the system runtime. When certain component's constraints cannot be satisfied, DRCR will automatically stop this component and all the components depending on it. The reverse process can also be done by DRCR. This mechanism greatly simplifies the system management tasks.

5.3 Performance test

The measurements are based on steady state observations - in order to eliminate the transitory effect of cold starts. The target component includes a computing task with priority 1. It runs at rate 1000Hz. We test 4000 observations from for the for this periodic task scheduling latency, which is the difference of actual schedule time and the prescribed time, negative value means the task is scheduled ahead of prescribed time. Due to page limits, we could not show all the system configurations. Please refer to (Gui et al., 2008a) for more detailed simulation descriptions.

Table 1 Latency Test (light & Stress) mode

Context/ Latency	AVERAGE (ns)	AVEDEV(ns)	MIN (ns)	MAX (ns)
HRC (light)	1334.9	3760.03	-24125	25846
RTAI(light)	1533.8	4033.52	-25436	28988
HRC (stress)	-19083.74	378.89	-23454	-18739
RTAI(stress)	-20184.52	405.41	-25531	-16984

We evaluate the scheduling latency of our hybrid implementation with the pure RTAI based implementation. The results are in nano-seconds, which shows that our approach has similar performance with respect to pure RTAI applications in different system configuration - stress mode (several CPU and disk normal Linux applications running along with this demonstration application) or light work mode.

6. CONCLUSION

ARFLEX is the first step towards the full-fledged robot architecture able to comply with requirements of FMS. It provides the designer with architecture able to fulfill the vast range of design goals that we consider as essential to match the dynamicity of market today: cost effectiveness, precision, and flexibility:

Cost effectiveness is achieved by the open architecture which is designed to use as much as possible off-the-shelf components – robot, actuator, sensors and even the controller algorithms.

Precision is enhanced by a higher level control loop that makes use of sensors. Currently, this is achieve through ARFLEX smart camera, which has been designed and successfully deployed.

Flexibility is a key consideration for ARFLEX project. The ARFLEX platform provides software platform that makes such kind of fast integration possible. The system runtime takes the burden from the component developer to the dependence identification and resolution. A continuous deployment mechanism is also implemented to greatly simplify the system evolution without the need to restart the whole system.

Realtime timeliness is guaranteed by the hybrid component implementation. This approach put the most complex and error prone adaption work in the non-realtime part while the real-time part is kept rather concise and easy to develop.

Our ongoing work focuses on designing more complex real-time component model into existing hybrid component model. The run-time monitoring interface will be designed and implemented in future work. A more fine-grained control interface will be studied rather than just stop and start the components.

Reference

OSGi ALLIANCE, (2008) OSGi Service Platform, v4.1.

CAPORALETTI, G. (2007) The ARFLEX project: Adaptive robots for flexible manufacturing systems. *8th IFAC International Workshop on Intelligent Manufacturing Systems*. Alicante, Spain.

E. BIANCHI, L. D., P. MANTEGAZZA (2006) RTAI Programming Guide.

GUI, N., DE FLORIO, V., SUN, H. & BLONDIA, C. (2008a) A framework for adaptive real-time applications: the declarative real-time OSGi component model. *The 7th Workshop on Adaptive and Reflective Middleware(ARM)*. Leuven,Belgium.

GUI, N., DE FLORIO, V., SUN, H. & BLONDIA, C. (2008b) A Hybrid real-time component model for reconfigurable embedded systems. *ACM symposium on Applied computing*. Fortaleza, Ceara, Brazil.

STEWART, D. B., VOLPE, R. A. & KHOSLA, P. K. (1997) Design of dynamically reconfigurable real-time software using port-based objects. *Ieee Transactions on Software Engineering*, 23**,** 759-776.

WALL, A., LARSSON, M. & NORSTROM, C. (2002) Towards an impact analysis for component based real-time product line architectures. *Proceedings of the 28th Euromicro Conference***,** 81-88

WANG, N. B., GILL, C., SCHMIDT, D. C. & SUBRAMONIAN, V. (2004) Configuring real-time aspects in component middleware. *On the Move to Meaningful Internet Systems 2004: Coopls, Doa, and Odbase, Pt 2, Proceedings*, 3291**,** 1520-1537.