**Preprints of the 19th World Congress**
**The International Federation of Automatic Control**
**Cape Town, South Africa. August 24-29, 2014**

# A novel model-based approach to support development cycle of robotic arm applications

**E. Estévez, A. Sánchez García, J. Gámez García, J. Gómez Ortega**

*System Engineering and Automation Department, University of Jaen
Campus las Lagunillas, S/N Jaén, Spain.
e-mail:{eestevez, asgarcia, jggarcia, juango}@ujaen.es

Abstract: The robotic-manipulator has, as aim, the integration of robots into people's quotidian issues. To this purpose, there are a great number of physical devices, such as sensors, actuators, auxiliary elements, tools… which can be incorporated into a robot. That is why *integration, reuse, flexibility and adaptability* are crucial characteristics demanded by current robotic applications. Using Model Driven Engineering (MDE) software development methodology that promotes an intensive use of models- systems' abstractions-, the complexity inherent to the robotic application design can be reduced. This work explores the advantages of the use of (MDE) to provide support to development cycle of such type of applications. So, in this paper, a modeling approach, where the model is the key concept, is developed to generate automatically the target code. In addition, using this concept, the design of the robotic-arm application could be done independently of robotics' communication middleware.

*Keywords:* Component Software Engineering, Model Based Engineering, Robotics Software Middleware.

## 1. INTRODUCTION

The software development in specific fields such robotics is closer to art than a systematic discipline. Chella A., et. al. (2010) details the most important reasons: (1) *variability of applications types* and the hardware/software *components* used in such specific domain; (2) *reusability difficulties* due to lack of limitation of architectural elements (device manipulators, algorithms, communication middleware …) and (3) *interoperability lack* among tool involved in the development cycle phases. Over the last years, software architectures for robotics manipulators systems are getting more and more complex, demanding the development of more adaptable applications. As a consequence, these software infrastructures should allow developers face up to the complexity imposed by different issues such as hardware, software, real time, and distributed computing environments.

In order to reduce significantly the effort to develop new software applications, a strong move toward the application of software engineering principles is taking place in robotics (Iborra, A., et. al., 2009) (Friedrich M. Wahl and Torsten Kroger, 2009). In fact, the use of CBSE – Component Based Software Engineering- (George T. Heineman and William T. Councill, 2001), (Ian Sommerville, 2007) and MDE- Model Driven Engineering- (Schmidt, D., 2006) disciplines and middleware platforms are very helpful for achieving requirements such as: system scalability, composition, reusability and flexibility, which are much related with previous commented drawbacks.

The CBSE, to face up the complexity -as a consequence of the variability of hardware/software components and applications-, offers mechanisms that increase the abstraction level of modelling elements. Hence, systems can be developed with independent modules that just interact each other by means of their interfaces. The use of CBSE is very spread in Robotics field (Brooks, A., et. al., 2005), (Gamez, J., et. al., 2008), (Brugali, D. and Scandurra, P., 2009), (Brugali, D. and Shakhimardanov, A., 2010). These approaches offer high rates of reusability and ease of use but a little flexibility with respect to the platform where those components are deployed and run. In this sense, there is a robotic platform's dependency.

The Model-Based paradigm increases the abstraction layer and allows describing applications independently of software platform (Selic, B., 2003). This approach relies on the use of models to represent the system elements from an specific domain viewpoint and their relationships. These models act as the input and the output at all stages of the development cycle until the final system is itself generated (Balasubramanian, K., et. al., 2003). Actually, some attempts have been done by different authors towards the use of modelling concepts in robotics field (Brugali, D. and Shakhimardanov, A., 2010).

Michael Geisinger et., al. (2009) purposes a two models based programming tool that generates target source code for Robotino® mobile robot platform. This tool allows users to define the functionality of the task (application logic model) and the required hardware resources. V$^3$CMM- 3 View Component Meta-Model (Alonso D., et. al., 2010) allows modelling robotics software developments from 3 views: (1) *structural view* describes application's static components structure; (2) *coordination view* models event-driven behaviour of each component, and (3) *algorithmic view*, collects the information that describes the algorithm executed by each component. Those views are performed via UML- Unified Modelling Language- subset of diagrams: the component, state-machine and activity respectively (Booch, Grady, et. al., 2005). The target is ADA code generation for CORBA- Common Object Request Broker Architecture- platform (communication middleware). More recently, the SmartSoft MDSD –Model Driven Software Development - Toolchain (2013) allows modelling robotic applications with SmartSoft Component concept (Schlegel, C., et. al., 2012a, 2012b). The target code runs over CORBA platform. BRICS- Best Practice in Robotics- European project (Bischoff, R., et. al., 2010) has as main goal to structure and to formalize the robot development process. A set of models, functional libraries have been developed. Target code runs over OROCOS - Open RObot COntrol Software- platform.

Previous work of authors identifies and models the minimal features of each component that appears in robotic-arm tasks (sensors, robots, terminal elements, control algorithms…). As example, a common minimal interface for each element has been proposed in (Sanchez Garcia, A., et. al., 2013). The idea is that these standard interfaces can provide to designers common methods for managing the elements, being not necessary knowledge about vendor specific drivers.

The main contributions of this paper are: (1) an UML based approach for modelling Robotic arm tasks. This modelling defines the functionality of the application independently of communication middleware over it will be run; (2) identification of Model-to-Text transformation rules to generate target source code that runs over a certain communication middleware.

The remainder of this work is as follows: Section 2 describes a methodology for modelling robotic-arm applications. Section 3 presents the main features of most spread communication middleware in robotic field. The main rules for achieving the automatic code generation are detailed in Section 4. An industrial case study is illustrated in Section 5 showing experimental results obtained from the robotic platform developed under this proposal. Finally, Section 6 introduces the conclusions of this work.

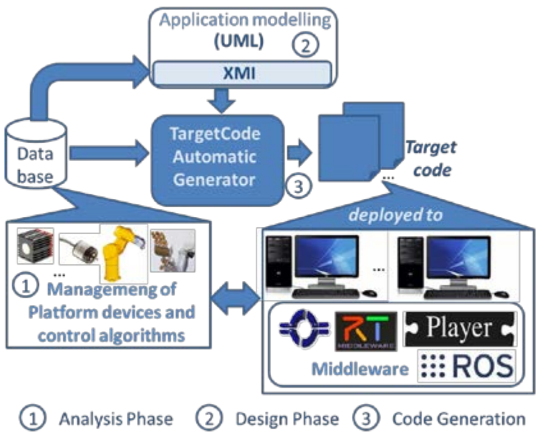


Fig. 1: General scenario of the proposed framework

## 2. MODELLING OF ROBOTIC-MANIPULATOR APPLICATIONS

In order to provide the development cycle support, first, the codification of the device's management in an isolated form, i.e. without taking into account the application, and software algorithms, must be done. After, the design phase is the responsible for the definition of the functionality of a robotic-arm based application and deals with the selection of the specific hardware platform. Finally, the coding phase deals with the automatic generation of the target code. Fig. 1 illustrates the general scenario of the proposed framework that explores the advantages of model-based techniques in order to automate the design and development of robotic arm-based platforms.

Sanchez Garcia, A., et. al., (2013) identifies and characterizes those components that take part in an execution of robotic manipulation tasks: Sensors to get the information from the environment; Robot, and-or its terminal elements (e.g. gripper, hand, hook) and other algorithms, which during the task execution, requires environment's information provide by sensors, check the state of robot and send the corresponding command. All these elements are stored in a database (See Fig. 1).

This section details a methodology for modelling robotic arm applications during the design phase of the development cycle. This consists of three main steps: (1) import from the repository the minimal code structure; (2) definition of the functionality with UML component diagram and (3) deploy this functionality to the platform by means of deployment diagram. Following sub-sections detail each proposed step.

### 2.1 Import the repository templates to UML model

The Object Management Group (OMG) proposed the XML Meta-data Interchange (XMI) standard in order to provide interoperability among UML tools. This specification has an eXtensible Markup Language (XML) notation and contains all information about the UML elements and diagrams in a Markup Language (ML) format. Those UML tools, that support XMI, can interchange projects because a XMI file stores all information, i.e. UML elements and printing information of diagrams. Therefore, authors have selected this format to import the templates into UML project before the definition of the robotic application. As UML supports OO paradigm, the meaning of UML class is the same as C++ or java classes.

Table 1 illustrates how the main concepts of the OO paradigm are expressed in XMI. A *Class* is a packagedElement characterized by three attributes: name, id and type. A class needs to have *uml:Class* value in xmi:type attribute. The identifier's value must be unique in overall XMI file. The isAbstract attribute only appears to indicate that the class is abstract. The OO properties are expressed as ownedAttribute fixing in this case the xmi:type attribute with *uml:Property* value. Every property is characterized by its *name*, *identifier* and *visibility*. This, latter, is used to indicate if the property is public, protected or private. OO methods are

**Table 1. Mapping rules for the generation of XMI file**

| OO concept | XMI notation |
|---|---|
| Class | <packagedElement xmi:type="uml:Class" xmi:id="id" name="ClassName" isAbstract="true"/> |
| Property | <ownedAttribute xmi:type="uml:Property" xmi:id="id" name="PropertyName" visibility="public \| protected \| private"/> |
| operation | <ownedOperation xmi:type="uml:Operation" xmi:id="id" name="MethodName" visibility="public \| protected \| private"/> |
| parameter | <ownedParameter xmi:type="uml:Parameter" xmi:id="id" name="paramName"/> |
| inheritance | <generalization xmi:type="uml:Generalization" xmi:id="id" general="Class_id"/> |

expressed as ownedOperation fixing the xmi:type attribute with *uml:Operation* value.

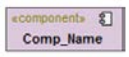
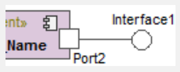
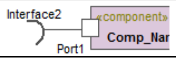
The inheritance is expressed with generalization element. The value of general named attribute indicates the class from which the information is inherited.

## 2.2 Definition of a robotic arm application

The functionality of the robotic arm application is a component based application where each component encapsulates the code of the class and provides the logic of an application, i.e. the communication and data interchange among the components that take part. Authors propose the use of Component UML 2.x diagram to perform this functionality. Table 2 illustrates the UML elements that take part for modelling the robotic arm applications. UML Component is used to represent each application component. The encapsulated code is expressed with a UML class, imported from the data-base. An UML component encapsulates code only if an UML class realizes it with *ComponentRealization* UML concept. UML port concept is used to provide external accessibility to a protected property. UML Interface is used for managing properties.

The components data flow interchange requires an UML interface element, linked to the corresponding UML ports.

**Table 2. UML component diagram elements in robotic-arm application modelling**

| UML concept | Graphical notation | Role in Robotic arm application modelling |
|---|---|---|
| Component |  | Robotic application component.. |
| Port | «component»<br>Comp_Name<br>Port1<br>Port2 | Provide external accessibility to protected properties. |
| Interface | | |
| Provided (*InterfaceRealization*) |  | Read- only external access. |
| Required (*Usage*) |  | Write-only external access. |

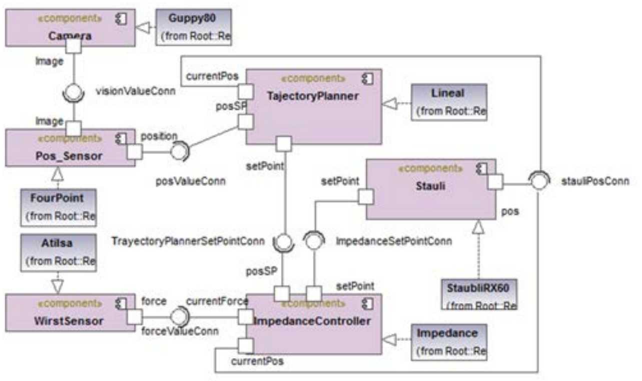
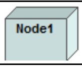
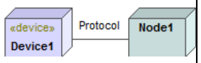
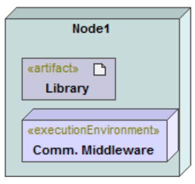


Fig. 2. Example of robotic application

The external read-only permission access has been provided to UML interface adding as many *getPropertyName* methods as data that the application component publishes with read-only permission. The external accessibility is achieved with UML *InterfaceRealization*. The external write-only permission access is attained with UML usage concept. A UML port uses an UML Interface to update the property's value.

For instance, in Fig. 2, *Camera* UML component encapsulates the GUPPY 80 class code. In order to provide an external accessibility to the sensor measurement value, the *image* UML port has been defined. The external accessibility of captured image is illustrated in UML by *visionValueConn* UML interface. The read-only permission is accomplished with UML InterfaceRealization, i.e. *image* UML port realizes *visionValueConn* UML interface. On the other hand, *Pos_Sensor* UML component encapsulates the *FourPoint* class code. This component has two UML ports to provide external accessibility to *image* and *position* properties. The *image* property has write-only external access, so the corresponding UML port uses *visionValueConn* interface to update the value. The *position* has read-only external access, accomplished realizing the PosValueConn UML interface.

**Table 3. UML deployment diagram elements in robotic-arm application modelling**

| UML concept | Graphical notation | Role in Robotic arm application modelling |
|---|---|---|
| Device | «device»<br>Device1 | Device of robotic arm platform. |
| Node |  | Node (e.g. PC) of robotic arm platform. |
| Communica-tionPath |  | Communication protocol between nodes, devices and node-device(s). |
| Artefact |   Node1<br>«artifact»<br>Library<br>«executionEnvironment»<br>Comm. Middleware | Library with device's isolated management code. |
| Execution Environment | | Communication middleware. |

### 2.3 Deployment to platform

Table 3 illustrates the UML elements that take part for modelling the deployment platform. The UML deployment diagram is formed by as many Device UML elements as devices of the robotic-arm platform. Besides, at least, one node is required, where to deploy the software components that form the application. The communication protocols between devices and nodes are expressed with *CommunicationPath* UML elements. The *artefact* UML element is used to indicate the necessity of other code or libraries in the Node; so in this work it is used to provide information about the code for managing every device in an isolated way. Finally the *ExecutionEnvironment* UML element is used to indicate the communication middleware.

## 3. AUTOMATIC GENERATION OF THE ROBOTIC-ARM APPLICATIONS TARGET CODE

This section is centred in the code generation phase of robotic arm applications' development cycle (see point 3 of Fig. 1). The MDE discipline has been followed, which relies on the model and model transformation concepts in order to automate the software development process (Schmidt, D., 2006). Two kinds of transformations can be distinguished: Model to Model (M2M) and Model to Text (M2T) transformations. Both have as input a model and they generate a new model conforming to a meta-model or source code, respectively. This work is centred in the second type of transformation, M2T, having as input the functional model -in XMI standard notation-, obtained after modelling the robotic-arm application. The definition of the M2T transformation rules implies having knowledge about both the input model and the structure of the target code. Following sub-sections detail the main characteristics of the input model, the main features of the most spread communication middleware in robotic field and, finally, the transformation rules.

### 3.1 Structure of the input model

The input model, from which the code generator gets information to automatically generate the target code, is obtained exporting the resulting UML model to XMI standard notation. Table 4 summarizes the information to be processed from Component and Deployment UML diagram in XMI notation in order to generate target code.
Every UML component that forms part of a robotic application (see Fig. 2) is expressed in XMI as first raw of Table 4 illustrates. This is characterized by as many number of ownedAttribute as ports it contains. In order to know if the port is an input or an output data port, the *Usage* UML element must be checked (see third raw of Table 4). Those port identifiers, that appear in xmi:idref attribute of client element will be considered input data ports (i.e. write-only external accessibility); the rest will be considered output data ports (i.e. read-only external accessibility). The code encapsulated in the component can be located by the realization element. In concrete, the class name is referred in realizingClassifier attribute.

**Table 4. Information to be processed**

| *UML Concept* | *XMI notation* |
|---|---|
| Component | <packagedElement ***xmi:type="uml:Component"*** ***xmi:id***="id" name="Comp_Name"> <ownedAttribute ***xmi:type="*****uml:Port**" xmi:id="id" name="Port_Name" visibility="protected"/> <realization xmi:type="uml:ComponentRealization" xmi:id="id" ***realizingClassifier***="Class_Id"> ..</realization> </packagedElement> |
| Interface | <packagedElement ***xmi:type="uml:Interface***" xmi:id="id" name="interface_Name"> <ownedOperation ***xmi:type="uml:Operation"*** xmi:id="id" name="op_Name" visibility="public"/> </packagedElement> |
| Usage | <packagedElement ***xmi:type="uml:Usage"*** xmi:id="id"> <***supplier*** xmi:idref="interface_id_ref"/> <***client*** xmi:idref="port_id_ref"/> </packagedElement> |
| Node | <packagedElement xmi:type="***uml:Node***" xmi:id="Node_id" name="Node_name"> <deployment xmi:type="uml:Deployment" xmi:id="id" deployedArtifact="Artifact_id"> <client xmi:idref="Node_id"/> <supplier xmi:idref="Artifact_id"/> </deployment> </packagedElement> |
| Device | <packagedElement xmi:type="***uml:Device***" xmi:id="Device_id" name="Device_name"/> |
| Comm. Path | <packagedElement xmi:type="***uml:Association***" xmi:id="Association_id" name="Protocol_name"> <ownedEnd xmi:type="uml:Property" xmi:id="Property1_id" visibility="protected" type="***Node_id***"> <association xmi:idref="Association_id/></ownedEnd> <ownedEnd xmi:type="uml:Property" xmi:id="Property2_id" visibility="protected" type="***Device_id***"> <association xmi:idref="Association_id"/></ownedEnd> ***<memberEnd xmi:idref="Property1_id"/>*** ***<memberEnd xmi:idref="Property2_id"/>*** </packagedElement> |
| Artefact | <packagedElement xmi:type="***uml:Artifact***" xmi:id="Artifact_id" name="RequiredLibrary"/> |
| Execution Env. | <packagedElement xmi:type="***uml:ExecutionEnvironment***" xmi:id="EE_id" name="CommunicationMiddleware"/> |

Regarding to deployment information, the hardware platform is defined by a set of UML nodes and devices. The communication protocol between those elements implies the use of *communicationPath* UML element. Artefact and Execution Environment are packagedElements.

### 3.2 Communication middleware features

The M2T generator also needs knowledge about the target code structure and requirements. Section 2.1 of (Shakhimardanov, A., et al., 2010) summarizes the main characteristics of the most spread robotic specific middleware. Since, all these communication frameworks are component based; they characterize their components with the same concepts but with different lexicon. Table 5 synthetises these concepts in terms of component's interface and timing information. Every one offers an interface for achieving synchronous and asynchronous operations. If the goal is to interchange a data, they offer other type of interface.

**Table 5. Interface the state of robotic specific Middleware' Components**

| Concept | Component Based Middleware | | |
|---|---|---|---|
| | Orocos | OpenRTM | Player |
| Commands (Asynchr.) | Operation | --- | Command |
| Methods (Synchr.) | | Service port | --- |
| Buffered/shared Data (Asynchr.) | Data Port | Data Port | Data Interface |
| Runtime mod. param. | Property | Configuration interface | Service Port |
| States of the component | Preoperational, stop, running | Created, inactive, active | --- |

Two of them, which offer an interface type, allow modifying parameter's value at runtime. Besides, the framework's execution engine controls the state in which component is (configure, start or stop).

Once the structure of input model and the structure of component-based communication frameworks for robotics are known, the main 5 common transformation rules can be identified: **Rule 1** generates a middleware application component for each UML component; **Rule 2** (R2) consists of providing external accessibility to the information. Hence, the UML ports of the UML components with R2 are transformed to buffered or shared data. The R2 also processes UML interfaces to give the execution engine the communication dialog between middleware's application components. **Rule 3** is applied to every protected method of the UML class that realizes the UML component. As a result of this transformation rule a synchronous method for each *getPropertyName* method and an asynchronous command for each *setPropertyName* method are added to the middleware's application component. **Rule 4** is applied to every public property of the UML class that realizes the UML component, having as result runtime modifiable parameters for the middleware's application component. Finally, **Rule 5** indicates middleware's execution engine how the component is setup. To do this, it processes the value of the sample property. If this value differs from zero, a periodic execution thread is generated with this period. Otherwise, a non-periodic execution thread is generated.

## 4. CASE STUDY

Nowadays, a vehicle headlamp is a high-sophisticated device that has to pass exhaustive quality inspections, normally demanded by the car manufactures (Satorres-Martínez et al, 2009). One of the stages of the production process of headlamps is the assembly of the components where, the main operation consists, basically, of the positioning and fixing of the lens-made of polycarbonate-, over a black housing made of polypropylene (Fig. 3). The rest of the components: reflector, lighting system and bezel are placed into the housing.

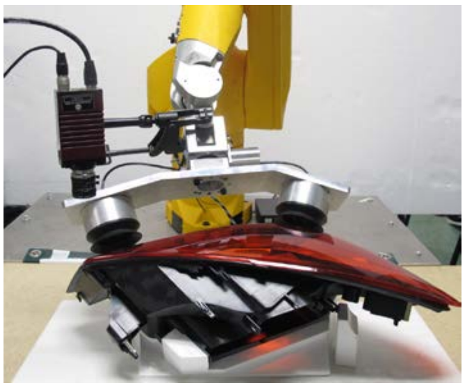


Fig. 3. Photograph of the assembly carried out by the robot.

Because of the nature of the production process, the dimensional variability of both housing and lens are relatively high if it is compared with the position requirements demanded by the car manufacturers. Obviously, this dimensional variation supposes a problem during the assembly.

The experimental setup, implemented using the methodology developed in this paper, is the assembly of both components using an industrial manipulator (Gomez Ortega et al. 2011). The idea is, considering the housing as a fix element and, using its gum channel, to move the lens inside the channel with the robot- whose width is around 2 mm--- in order to minimise the contact forces.

For this assembly task two sensors were used; a wrist force sensor attached to robot tip, which can determine the forces and torques generated by the manipulator and its contact point; and a vision sensor whose mission is to determine the position of the gum channel. So, the assembly procedure, illustrated in Fig. 2, is a follows: once the vision sensor identifies the gum channel, the manipulator moves the lens to the housing channel; then, using the wrist sensor, the contact point is determined, together to the forces and torques exerted by the lens over the housing. Finally, the robot goes to the position that minimizes the forces and torques (Fig. 4). As force controller an impedance algorithm is applied.

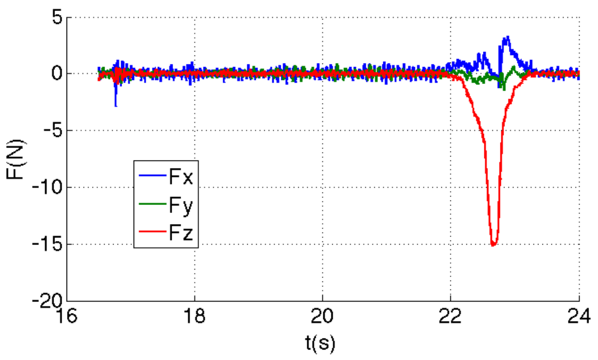


Fig. 4. Contact force exerted during the assembly process.

## 5. CONCLUSIONS

This paper presents a framework that provides support of development cycle to robotic arm tasks. The Model Driven

Engineering approach has been followed to provide support to design and coding phases of the development cycle. The model concept has been used for describing the functionality of robotic arm task, which takes place during the design phase. Also, M2T transformation has been identified to generate the target code for component-based communication framework.

Authors' proposed framework uses UML as modelling language. In concrete UML 2.x component and deployment diagrams have been used. The functional model's XMI standard notation is the input model of M2T transformer that generates the target code for communication middleware. As XMI is a standard notation, the proposed framework is valid for any UML modelling tool that supports this import/export option.

## ACKNOLEDGEMENTS

This work was partially supported by the projects DPI2011-27284, TEP2009-5363 and AGR-6429.

## REFERENCES

Alonso D., Vicente-Chicote, C., Ortiz, F., Pastor, J. and Álvarez, B., (2010), *V3CMM: a 3-View Component Meta-Model for Model-Driven Robotic Software Development*, Journal of Software Engineering for Robotics, Vol:1, Issue:1, pp. 3–17.

Balasubramanian, K., Gokhale, A., Karsai, G., Sztipanovits, J. and Neema S., (2006), *Developing applications using model-driven design environments*, Computer, Vol. 39, Issue: 2, pp: 33–40.

Bischoff, R, Guhl, T., Prassler, E., Nowak, W., Kraetzschmar, G., Bruyninckx, H., Soetens, P., M. Haegele, P., Pott, A., Breedveld, P., Broenink, J., Brugali, D., and Tomatis, N., (2010), *BRICS - best practice in robotics*, Proc. of the 41st International Symposium on Robotics, pp. 1 –8.

Booch, Grady, Rumbaugh, James and Jacobson, Ivar, (2005), *The Unified Modeling Language User Guide*, 2nd Edition, Addison-Wesley Professional.

Brooks, A., Kaupp, T., Makarenko, A., Williams, S. and Oreback, A. (2005), *Towards component-based robotics*, Proc. of the IEEE International Conference Intelligent Robots and Systems (IROS), pp: 163-168.

Brugali, D. and Scandurra, P., (2009), *Component-based robotic engineering (Part I) Reusable Building Blocks*, IEEE Robotics Automation Magazine, Vol: 16, Issue: 4, pp: 84-96.

Brugali, D. and Shakhimardanov, A., (2010), *Component-Based Robotic Engineering (Part II) Systems and models*, IEEE Robotics Automation Magazine, Vol: 17, Issue:1, pp:100-112.

Chella, A., Cossentino, M., Gaglio, S., Sabatucci, L., Seidita, V., (2010). *Agentoriented software patterns for rapid and affordable robot programming*. Journal of Systems and Software 83, Issue:4, pp:557 – 573.

Friedrich M. Wahl and Torsten Kroger (2009), *Advances in Robotics Research: Theory, Implementation, Application*, Springer-Verlag Berlin and Heidelberg GmbH & Co. K.

Gamez, J., Robertsson, A., Gomez Ortega, J., and Johansson, R. (2008), *Sensor fusion for compliant robot motion control*, IEEE Trans. Robot., Vol: 24, Issue: 2, pp. 430–441.

Gomez Ortega, J., Gamez Garcia, J., Satorres-Martínez, S., Sanchez Garcia, A. (2011). Industrial assembly of parts with dimensional variations. case study: assembling vehicle headlamps. Robotics and Computer-Integrated Manufacturing. Vol: 27, Issue: 6, pp. 1001–1010.

George T. Heineman and William T. Councill (2001), *Component-based software engineering: putting the pieces together*, Addison-Wesley.

Ian Sommerville, (2007), *Software Engineering*, eight Edition, Pearson Education.

Iborra, A., Caceres, D.A., Ortiz, F.J., Franco, J.P., Palma, P.S. and Alvarez, B. (2009), *Design of Service Robots, experiences using software engineering*, IEEE Robotics and Automation Magazine, Vol:16, Issue: 1, pp: 24-33.

Michael Geisinger, Simon Barner, Martin Wojtczyk, and Alois Knoll (2009), *A software architecture for model-based programming of robot systems*, Lecture Notes on Computer Science, Advances in Robotics Research, pp. 135–146.

Sanchez Garcia, A., Estevez, E., Gomez Ortega, J., Gamez Garcia, J. (2013), *"Component-based modelling for generating robotic arm applications running under OROCOS middleware"*, Proc. of the IEEE International Conference on Systems, Man, and Cybernetics, pp: 3633-3638.

Satorres-Martínez, S., Gomez Ortega, J., Gamez Garcia, J., Sanchez Garcia, A. (2009). *"A dynamic lighting system for automated visual inspection of headlamp lenses"*. IEEE Conference on Emerging Technologies and Factory Automation.

Selic, B., (2003), *The pragmatics of model-driven development*, Software, IEEE, Vol: 20, Issue: 5, pp. 19 – 25.

Schmidt, D. ,(2006), *Guest editor's introduction: Model-Driven Engineering,* Computer, Vol:39 (2), pp. 25–31.

Schlegel, C., Steck, A., Lotz., A. (2012a), *Robotic Software Systems: From Code-Driven to Model-Driven Software Development*, Robotics and Automation, Robotics Systems – Applications, Control and Programming, Intechopen, pp:473-502

Schlegel, C., Steck, A., Lotz., A. (2012b) *Model-Driven Software Development in Robotics: Communication Patterns as Key for a Robotics Component Model. In Introduction to Modern Robotics.* iConcept Press.

Shakhimardanov, A., Paulus J., Hochgeschwender, N., Reckhaus, R. and Kraetzschmar, G. (2010)v"Best Practice Assessment of Software Technologies for Robotics". Available at: http://www.best-of-robotics.org/pages/publications/BRICS_Deliverable_D2.1.pdf

SmartSoft MDSD Toolchain, (2013), SmartSoft Model Driven Development Software Design Toolchain, Available at: http://smart-robotics.sourceforge.net/index.php