Robotics Architecture Frameworks, Available Tools and Further Requirements

Mostafa Pordel and Thomas Hellström
Department of Computing Science
Umeå University
Sweden

Feb 20, 2013
UMINF-13.02
ISSN-0348-05

Robotics Architecture Frameworks, Available Tools and Further Requirements

Mostafa Pordel, Thomas Hellström
Umeå University
Umeå, Sweden
{pordel,thomash@cs.umu.se}

***Abstract*—For every robotics project, choosing a suitable framework and middleware for software and hardware is a challenging task which may influence the entire project. Robotics applications typically are resource constrained when it comes to computations and memory usage. They are built on different hardware platforms and applied in different domains. Therefore it is hard to introduce a common framework for all types of projects. However, in recent years several new attempts have been made and received attention from both researchers and industry. These frameworks are still under development and need to be extended. This paper discusses the different features that are needed for robotics frameworks and compares some of the available middleware and standards.**

I. Introduction

Robotics industry has a long history and goes back to the 1930's [1]. Currently, robotics is being used in automobile, pharmaceutical, life science and many other fields [2]. However, having such a background has not yet resulted in development of a software framework that can be widely used in industry and research. Due to the emerging market of robotics in new domains like renewable energy, healthcare, entertaining and warfare, having a robotics platform that can speed up the development process is indeed needed [3]. Defining an architecture framework that provides simplicity in the development phase and enables several programmers to work on the same robot application concurrently would decrease the costs of development. Several new companies aim to create and support such frameworks. One example is Willow Garage (founded in 2006) which introduced Robotics Operating System (ROS) [4]. In addition, also existing top-notch software companies have started working in the same direction, such as Microsoft with Microsoft Robotics Development Studio (MRDS) released in 2006 [5] and Google with Android robot library [6] initiated in year 2008. For new robotics projects, the choice between using one of such already available frameworks and developing a new platform depends a lot on the application type. The more general frameworks support more features but usually require more resources, which may not be always available. For example, using MRDS requires Microsoft Windows which may not a possible option for some robot projects.

This paper describes the benefits of using some available tools, and describes state of the art of robotics frameworks. Section II describes some available standards and frameworks for robot development. Dependability is an important feature of the robotics middleware that is argued in Section III. A case study of how selecting different frameworks may change the project design and implementation is presented in Section IV. Finally, conclusions are given in Section V.

II. Available Frameworks and Standards

Several architecture frameworks have been developed for robotics applications. Some products, like MRDS, are commercial while others, like OROCOS and ROS [7], are open source products. Some standards like JAUS only aims to define what should be implemented for each robotics framework. Others, like OpenJAUS, follow these standards. In this section some of these projects are described in more detail.

A. JAUS

Joint Architecture for Unmanned Systems (JAUS) is a standard framework, initially created by Department of Defence (DoD) for military purposes. It has several commercial and open source implementations [8], [9] and focuses on interoperability of robotics systems separated into several subsystems. For reasons of national security, DoD stopped supporting JAUS publicly, and Society of Automotive Engineers (SAE), which is an organization for aerospace, automotive, and commercial vehicle industries, is now supporting JAUS in the form of SAE JAUS [10]. It includes two main parts; Domain Model (DM), which is a specification that shows how to integrate the system to military and commercial space, and Reference Model (RM), which holds an architecture framework with a set of message protocols. RM defines a hierarchical architecture with three layers in each system.

Figure 1 shows the high level architecture of JAUS. It simplifies the application by dividing the robotics application to smaller elements. JAUS describes each system as a collection of several subsystems which can be mapped to physical parts of a robot. Each subsystem is then divided into several nodes and finally each node is a collection of some components. JAUS defines a component as an element that cannot break into smaller elements. A component can be a software or hardware element. Regarding hierarchies, subsystems communicate with each other, and nodes communicate within each subsystem. Within each node there is a network by which components can reach each other. As shown in Figure 1, each subsystem has only one communication component and each node has one node manager, which is responsible for routing data within

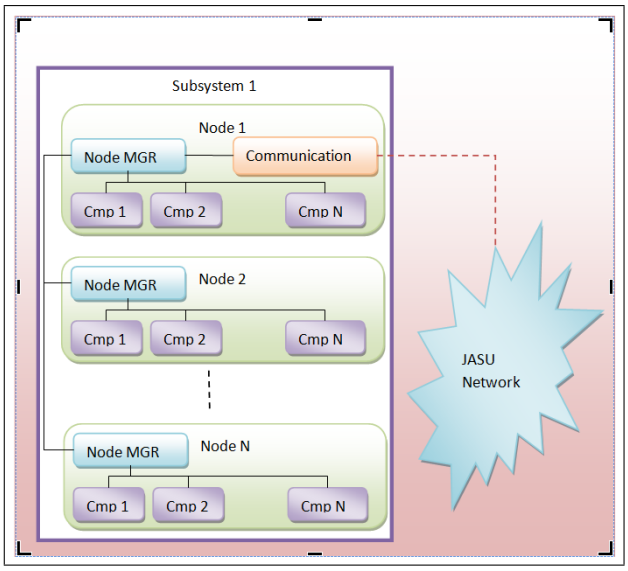

Fig. 1. JAUS system hierarchy includes three layers

the application. More specifically, in JAUS the communication component routes messages to other subsystems and the node manager routes messages to the destination node within the subsystem. Similar to the three levels of communication, there are three levels of compliance for JAUS applications; subsystem level, node level and components level. A JAUS system should at least implement JAUS subsystem specifications which are mainly about communication methods between subsystems. The other two compliances are related to the lower levels, node and component respectively. JAUS specifies the three compliances without references to any specific implementation or methodology for implementation. Therefore, the implementation language and targeted hardware of JAUS systems could be anything, as long as it satisfies the JAUS specifications. JAUS defines messaging as a means for communication. Each message has a fixed header that describes the body of the message (similar to UDP header that describes the body of UDP message). Messages might be Command, Query, Inform, Event Setup, Event Notification, Node Management or Experimental [8] types. These different categories of messages cover all common communication protocols, and provide a structure (Experimental Message) for new protocol definitions. The interoperability feature of JAUS is realized by defining message protocols as the only means to connect two subsystems. Therefore, each subsystem only needs to interpret the messages, and does not depend on the structure or underlying layers of other subsystems. JAUS can be implemented using different programming methods including modular, Service Oriented Architecture (SOA), Object Oriented Programming (OOP) or any combination.

JAUS supports component based development in order to increase reusability and incremental development in robotics projects. There are a set of properties that JAUS components should have (see [11]). JAUS components should not make any assumptions on the underlying system and platform. JAUS components should be applicable for any other JAUS system. They should be technology independent which means they may be used in architectures with different technical solutions [11]. However, in a general shift, a new JAUS standard extension from SAE holds different movements toward Service Oriented Architecture (SOA) by JAUS Service Interface Definition Language (JSIDL) [9] which is a methodology for robotics application development.

1) OpenJAUS to support JAUS: OpenJAUS is an open source Software Development Kit (SKD) that implements JAUS [9]. It was founded by a researcher at Florida University in 2006 and in 2009 it covered features from SAE JAUS. Currently, it is one of the biggest open source implementations of JAUS. Most of the definitions in OpenJAUS are done through JAUS Service Interface Definition Language (JSIDL) which is based on XML language and provides structures for service interfaces in JAUS. As declared in the official website of OpenJAUS, new implementations of OpenJAUS will cover Model Driven Engineering (MDE). This would add an interest to JAUS by reducing complexity and implementation costs [12]. While JAUS only contains definitions that can easily be changed in a new version, OpenJAUS is a real implementation that requires adaptations to newer versions, something that may not be easily done for some existing libraries.

2) Model Driven Engineering: The goal of Model Driven Engineering (MDE) is to encourage developers to develop models instead of code [12]. MDE sees the world as a composition of different models and the mission within this environment is to describe different models and automatically transfer some of these models to the other formats. Model Driven Architecture (MDA) is a methodology of modeling in MDE that is founded by Object Management Group (OMG) in year 2000. It focuses on Platform-Independent Model (PIM) and Platform-Specific Model (PSM) [12]. The transformations between PIMs and PSMs are done using Atlas Transformation Language (ATL). PIM focuses on models that are independent of hardware platforms. This is similar to the way JAUS specifies without any assumptions of the hardware platform. However, MDA supports the hardware platforms by introducing PSM. For robotics applications, MDE integration can be done through MDA. The design is then described with some PIMs that should be developed for robotics applications. The next step is to target a hardware platform. The hardware platform processing unit can be a microcontroller, FPGA [13], CPU etc. For a specific hardware, there may be several languages that can be used for programming such as C++, VHDL, Verilog and VB. Once the PIMs are set for a robotics application, the MDA solution should automatically support code generation from PIMs to PSMs with the desired languages. The transformations between PIMs and PSMs are done with ATL language. The package of PIMs and ATL code that is generated for one robotics application can be used in other projects too. MDA supports developers with incremental improvements of new generated code that is edited in several projects. By shifting the focus to PIMs rather than coding,

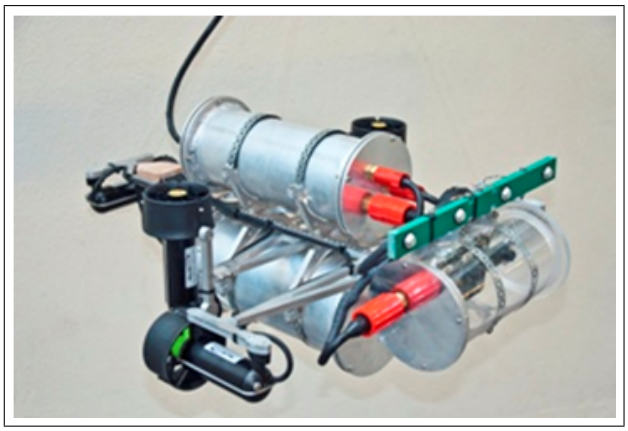

Fig. 2. Ralf II in an underwater robot with real-time vision system

design of robotics applications is optimized and distributed easily. For supporting new hardware for robotics applications, it is only needed to add a new transformation from PIM to the targeted PSM. In most software projects, including robotics projects, there is a gap between what is designed and what is implemented later on based on the design. One of the main benefits of using MDE is to provide a developing environment in which the design is synchronized with the executable code (PSMs). If MDE will be successfully integrated with OpenJAUS (this project has been delayed since May 2009), then JAUS specification models will be modeled with PIM and then transferred to the PSM. This will help robotics developers to share common model specifications and generate different code (models in MDE) for different robot hardware platforms.

B. ROS

Robot Operating System (ROS) [4],[4] is another framework for robot development that has been developed since 2007. It is an open source framework and aims to attract robot developers to share and contribute to its development. If you describe the software functionality of a robot as a grid, then each node represents a service, and ROS messages are sent over the edges to other nodes. Currently several hardware platforms use ROS as software framework. Care-O-bot 3, iRobot Create and Aldebaran Nao are among the most famous hardware platforms using ROS [14]. ROS has a repository in which developers can use code that has been already written for robot components like motion planners and vision systems. As ROS is Linux based software, it cannot guarantee the hard real-time properties of a system. However, ROS supports soft real-time applications. In addition, it can be integrated easily with OpenCV which is an open source library written in C++ for image processing.

C. MRDS

Microsoft Robotics Developer Studio (MRDS) is a recent initiative of Microsoft in robotics framework technology. It runs on .NET framework which itself runs on Microsoft Windows operating systems. Therefore, MRDS inherits features from these platforms. The goal of MRDS is to integrate Windows platform with robotics development. It uses a Service Oriented Architecture (SOR) as the method to increase interoperability and flexibility. MRDS considers a robotics application as a composition of different services which can be loosely coupled. These services communicate with each other through a new developed protocol from Microsoft that is named Decentralized Software Services Protocol (DSSP) [15]. DSSP optimizes message passing between services, compared with traditional message passing protocols used in Microsoft Windows like Simple Object Access Protocol (SOAP). DSSP provides different message passing methods for services that may run on different CPUs. The physical distances between the CPUs in which the services are running on, determine the communication method within the DSSP. Therefore, message passing in MRDS is especially beneficial for robots with CPU cores that run several services on each core and involve lots of data transmission. Another interesting feature of MRDS is Concurrency and Coordination Runtime (CCR) which is a library for asynchronous programming. This is beneficial when several concurrent operations share the same data. CCR provides an abstraction level for memory locking and communication. Several universities and companies, for instance Siemens and Tyco, use these features also for non-robotics applications [5]. Although the performance of MRDS with CCR and DSSP is impressive, MRDS still cannot guarantee hard real-time performance since it runs on Microsoft Windows platform. Besides, MRDS source code is not publicly available which makes it less interesting within communities aiming to enhance open source robotics operating systems.

III. DEPENDABILITY

Dependability is a metric that shows how good a system is in terms of being depended on by other systems. Another definition of dependability is ”Trustworthiness of a computing system that allows reliance to be justifiably placed on the services it delivers”, where reliance is ”contextually subjective and depends on the particular stakeholders’ needs” [16]. It includes different sub metrics like reliability, availability, maintainability, testability, integrity, and safety. Many robotics applications are real-time applications that are mission critical or safety-critical systems. When it comes to developing robots that interact with human beings or when the system failure may be harmful for the environment, studying dependability is needed during development. One of the main features that can increase dependability of robotics frameworks is support of hard-real time properties. This means that execution time can be analyzed and Worst Case Execution Time (WCET) and Best Case Execution Time (BCET) can be calculated. It is however noticeable that also for frameworks with support for hard-real time properties, there may be complex and dynamic situations in which WCET and BCET cannot be calculated. Generally speaking, static architectures allow system evaluators to certify a high degree of dependability and dynamic architectures provide a good flexibility but less dependability [17].

A. Real-Time Framework

Real-time systems are divided into soft real-time and hard real-time systems. Robotics frameworks that run on top of an operating system should consider the real-time properties of the underlying operating system as well. Soft real-time applications try to execute tasks before the dead-line. However, it may happen that they miss their mission, but in that case it will not end in a system failure and can be tolerated. In most cases when robots are interacting with humans, it is required to guarantee robot actions and decisions with high dependability. So far, Linux based real-time systems are all soft real-time systems. In general this means that the timing delays are good enough for many applications but it is not possible to compute a WCET for their execution. It means that, given interrupts and events the operating system may face, the execution time will vary a bit and cannot be calculated and guaranteed for the applications. However, one interesting ongoing project, tries to build such a framework with Ada language that can be run on microcontrollers. This enables support for hard real-time systems and the robot applications running on top of it could be safety critical robots, with provided WCET [18] and BCET. This will provide predictable execution time for applications.

IV. Case Study

In order to illustrate some features of existing frameworks, a case study is presented in this section. A robotics application which has been developed earlier is presented [19] [20]. Based on different frameworks that the application could have used, some changes would have been required. These changes are presented and pros and cons of such adaptation are illustrated.

A. Underwater Robotics Project

Ralf II is an underwater robot which has been initiated in February 2010. It is executing several simple tasks like following a line in a pool, firing to a target, grabbing some equipment from the floor and finally communicating by radio for asking for the exit point. Figure 2 shows Ralf II that weighs about 15 Kg.

Ralf II is similar to many other robotics projects that need to perform their missions regardless of the software framework they are using. For such projects, developers have the option to use an existing software framework or develop a new framework from scratch.

B. Initial Architecture of Ralf II

As shown in Figure 3, Ralf II has six subsystems, Vision, Sonar, Real-Time Communication (RT Communication), Motor Controller, .NET Laboratory (.NET LAB) and the Brain. It implements a hard real-time vision system in FPGA with VHDL and Verilog languages. The Vision subsystem communicates with the Brain and the .NET LAB. In order to enable real-time communication, an Ethernet communication system has been developed in the FPGA with VHDL and Verilog languages [20]. The Brain is the unit that controls all other subsystems. It is written in Ada programming language and runs on Linux Ubuntu operating system. The Sonar subsystem

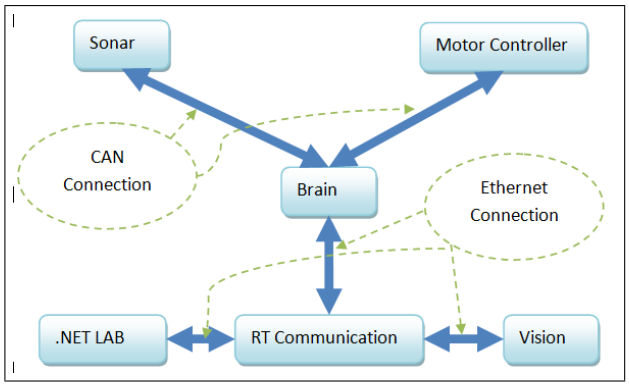

Fig. 3. Ralf II has six subsystems which are communicating through CAN and Ethernet protocols

is used for underwater communication. It is written in C++ and runs on a micro-controller. The simulation and testing of different algorithms were first implemented in the .NET LAB which is based on .NET libraries, written in C-Sharp language and connected to MATLAB and OpenCV for using available algorithms and components. All the .NET LAB concept has been implemented in order to save time by early testing with available code mostly in MATLAB, and later transferring the successful code to the Vision component which is based on the FPGA [19]. All subsystems are communicating through the well-known communication protocols Ethernet and CAN. The provided architecture supports independent development by defining communication interfaces between different subsystems. Languages that have been used are C++, Ada, C-Sharp, MATLAB, VHDL and Verilog combined with Windows and Linux underlying operating systems. In the initial design, the robot has not implemented any particular robotics framework. It is based on several cores that each has its own architecture. The rest of this section compares features of the robot with the cases that it would have been developed using other frameworks. These comparisons enable understanding of the benefits of using existing tools. It also helps with finding out what has been missed so far that should be implemented in as new features to the developing tools and standards.

C. Design with JAUS

Ralf II has split the application to six divisions named subsystems. Generally, theses subsystems can be mapped to what JAUS define as subsystems. As mentioned in II-A, applications can be JAUS compliant in three levels. The minimum JAUS compliant application requirement is to divide a system to several subsystems and communicate within standards that are supported in JAUS. Both CAN and Ethernet communications are JAUS supported protocols. However, according to JAUS, a communication node and a node manager should be defined in each subsystem. For Ralf II, the Vision subsystem has been connected to the Communication component in the same FPGA. So if you consider the Communication component as a node within the vision subsystem, it could be mapped to the communication node in JAUS for the Vision Subsystem.

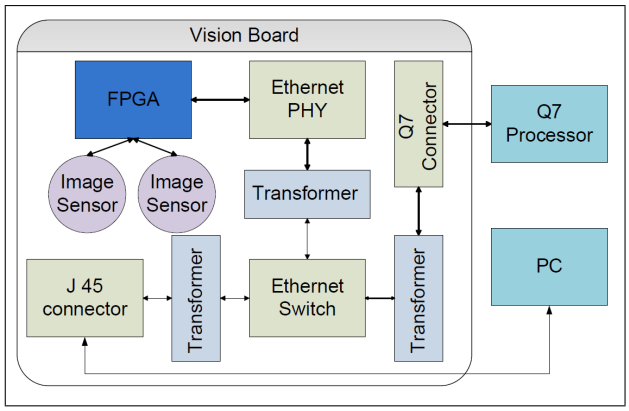

Fig. 4. Block diagram of the Vision board of Ralf II underwater project[20]

Although the Communication component is located in the same physical part of the robot as the Vision subsystem (JAUS subsystem compliance requirements), there is still no definition of the Node Manager for different components (nodes) in the Vision subsystem. All Vision system components are communicating with external components and internal components through one component, and it is really hard to implement JAUS Node Manager for internal communication. Figure 4 shows how the Vision system connects to the other subsystems. If the Vision system would be considered a node, it would not need any Node manager. The similar case exists when each block code (component) in VHDL at top level is considered as a node in JAUS. In such case, the top component in the VHDL code in the FPGA is the JUAS Node Manager as it passes the parameter it receives from other subsystems to the internal VHDL blocks. So in principle, JAUS level one compliant is more or less implemented in the Vision system. However, some details at level one, like the node manager and the communication node, seem to be more of an obstacle than a help for this project. Therefore, for the minimum compliance requirements, defining the interfaces and protocols of what a subsystem should expect from other subsystems and moving toward more flexible communication protocols would be more interesting rather than moving toward specification within each subsystem. As it is clear, Ralf II is not following JAUS level II and III compliance at all. Because JAUS does not make any assumption on underlying hardware or software and does not define any implementation, it can cover a wide range of different solutions. However, for some hardware like FPGA, it may be a bit challenging to completely implement JAUS components. One interesting feature of JAUS that is Reference Architecture (RA) is enabling the robot to integrate the existing JAUS component to the application. Nevertheless, for Ralf II, it has not been the case as it not using any JAUS component. In principle, what is argued so far for the Vision system and JAUS could be extended for other subsystems.

D. Design with ROS

If Ralf II had used ROS, it should have implemented it on each subsystem. For the Brain component, instead of running Ada with Ubuntu, it should have had C++ based ROS and Ubuntu. The performance then would be the difference between the application written in Ada and the equivalent one in C++. For the Vision system, unfortunately, the replacement is not effortless. It is almost impossible to guarantee hard real-time properties as described in section III-A while relying on current distributions of Linux. Even if we solve this problem, the performance of system with ROS would drop dramatically. This is due to the fact that the FPGA technology is a hardware-based solution that runs very fast and does not accept any interrupts. These are all features that ROS does not support while they may be required in some applications. For instance, in a safety critical system like a surgery robot, the behaviour of the robot has to be fully predictable. The vision subsystem of Ralf II initial design is based on FPGA technology that provides hard real-time vision system. However, it could benefit from ROS integration with OpenCV library which is an open source C++ based library for image processing. This would accelerate the implementation and avoid low level programming in VHDL. Nevertheless, for the .NET LAB subsystem, Ralf II would have missed a lot building on low level C++ programming instead of a set of available tools and libraries including MATLAB and C-Sharp that provide testing and simulation environments. .NET LAB provides a fast developing environment and high level programming that is not possible in ROS. For the Motor controller and the Sonar system, if Ralf II would have run ROS; it would need a CPU. It would add cost to the system without adding any performance. Low level programming with Ada guarantees real-time properties on the microcontroller and this could not be achieved with ROS as mentioned earlier.

V. Conclusion and Future Work

Robotics middleware is a new and emerging field that requires some efforts for being widely accepted among developers. For the time being there are some open source frameworks mostly written in C++ that can run on Windows and Linux operating systems such as OROCOS [7] and ROS [4] and some commercial frameworks like MRDS [5]. However, none of them are widely accepted in industry or research communities as the implementation of robotics application on top of such middleware may introduce some new costs similar to what is presented in Section IV. On the other hand, the main challenge of making such a general framework is to support different hardware platforms and components and at the same time provide good interoperability. Also most of the existing frameworks like ROS and MRDS do not guarantee hard-real time features and are not applicable for safety critical robotics systems. On the other hand, the other frameworks which guarantee the mentioned properties like the combination of Ada language and microcontroller or VHDL and FPGA are difficult and time consuming to be used and debugging and development tools are not mature enough. With a case study of Ralf II [19] project, some other pros and cons of the robotics frameworks are illustrated.

REFERENCES

[1] Y. Endo and R. Arkin, "Implementing tolman's schematic sowbug: behavior-based robotics in the 1930's," in *Robotics and Automation, 2001. Proceedings 2001 ICRA. IEEE International Conference on*, 2001.

[2] M. Eibeck, A. Risch, and M. Steiner, "Range of robotic applications at magna steyr and areas for further advancement in automotive operations," in *Robot Sensing, 2004. ROSE 2004. International Workshop on*, May 2004, pp. 65 – 68.

[3] L. Ross, "Global investment activities in robotics, nanotechnology and energy," in *Robotics and Biomimetics (ROBIO). 2005 IEEE International Conference on*, 0 2005.

[4] S. Cousins, "Ros on the pr2 [ros topics]," *Robotics Automation Magazine, IEEE*, vol. 17, no. 3, pp. 23 –25, sept. 2010.

[5] A. Cesetti, C. P. Scotti, G. Di Buo, and S. Longhi, "A service oriented architecture supporting an autonomous mobile robot for industrial applications," in *Control Automation (MED), 2010 18th Mediterranean Conference on*, june 2010, pp. 604 –609.

[6] T. Hashimoto, M. Suzuki, and H. Kobayashi, "Development of remote class support system and field trial using an android robot," in *Mechatronics and Automation, 2009. ICMA 2009. International Conference on*, aug. 2009, pp. 2042 –2047.

[7] H. Bruyninckx, P. Soetens, and B. Koninckx, "The real-time motion control core of the orocos project," in *Robotics and Automation, 2003. Proceedings. ICRA '03. IEEE International Conference on*, vol. 2, 2003, pp. 2766 – 2771 vol.2.

[8] C. R. W. Steve Rowe, "An introduction to the joint architecture for unmanned systems (jaus)," 2008. [Online]. Available: http://www.openskies.net/papers/

[9] D. K. Thomas Galluzzo, "The openjaus approach to designing and implementing the new sae jaus standards," Auguest 2010.

[10] A. 4c Information Modeling and D. Committee, "Jaus mobility service set," 2010. [Online]. Available: http://standards.sae.org/as6009/

[11] J. Standard, "The joint architecture for unmanned systems,architecture framework," vol. II, 2007.

[12] M. Pordel and F. Yekeh, *Model-Based Approach for Concurrent Development in MDE*, August 2010.

[13] S. Hauck, "The roles of fpga's in reprogrammable systems," *Proceedings of the IEEE*, vol. 86, no. 4, pp. 615–638, 1998, cited By (since 1996) 141. [Online]. Available: https://www.scopus.com/inward/record.url?eid=2-s2.0-0000950606&partnerID=40&md5=8b770f52b2cc1bb602a248799740f492

[14] S. Cousins, B. Gerkey, K. Conley, and W. Garage, "Sharing software with ros [ros topics]," *Robotics Automation Magazine, IEEE*, vol. 17, no. 2, pp. 12 –14, 2010.

[15] H. F. Nielsen and G. C. from Microsoft, "Decentralized software services protocol dssp/1.0."

[16] P. Donzelli, D. Hirschbach, and V. Basili, "Using visualization to understand dependability: A tool support for requirements analysis," in *Software Engineering Workshop, 2005. 29th Annual IEEE/NASA*, 2005, pp. 315 –324.

[17] A. G. L. G. I. P. Brian Ford, Peter Bull, "Adaptive architectures for future highly dependable, real-time systems," 2009.

[18] K. Gregertsen and A. Skavhaug, "A real-time framework for ada 2005 and the ravenscar profile," in *Software Engineering and Advanced Applications, 2009. SEAA '09. 35th Euromicro Conference on*, 2009, pp. 515 –522.

[19] F. Y. L. A. Mostafa Pordel, Nima Moghaddami Khalilzad, "A component based architecture to improve testability, targeted fpga-based vision system," in *2011 International Conference on System Modeling and Optimization*, January 2011.

[20] F. Y. L. A. Nima Moghaddami Khalilzad, Mostafa Pordel, "Fpga implementation of real-time ethernet communication using rmii interface," January 2011.