Co-Design Architecture for Reconfigurable Assembly Platforms

JOSE L. MARTINEZ LASTRA & REIJO TUOKKO
Institute of Production Engineering, Assembly Automation Lab.
Tampere University of Technology
P.O. Box 589. Tampere. FIN-33101
FINLAND
jose.lastra@tut.fi http://www.pe.tut.fi/aal

Abstract: - This paper proposes a new architecture for distributed assembly platforms that is well suited to hardware-software codesign. The key qualitative attribute is the reusability of her atomic architectural units called *"assembly actors"*(software/hardware devices) due the correlation between actors' goals and primitive assembly operations. We describe the main components of the codesign architecture and focus on the interfaces between components. The collaborative approach is illustrated using two-robotic axis iteration.

Key-Words: - Hardware/Software Reusability; Architecture-based System Development; Service-based Collaboration; Intelligent Physical Agents; Agent UML; Assembly Automation

1 Introduction

The use of architecture-centric development for large and complex systems helps to produce better quality software with a shorter time-to-market [1, 2]. However those architecture are developed from a software perspective and do not discuss the hardware related issues. Concurrent engineering in general and software/hardware codesign in particular have been active research fields in the last few years. These trends light us for discussing in terms of mechatronic architecture.

Single universal manipulators (6 DOF robots) can execute many assembly tasks. However, in some cases like in the electronics production, multi-robot systems with less DOFs can accomplish these tasks using simpler and less expensive mechanisms.

In this paper, a reference mechatronic architecture focus software/hardware codesign is proposed. Using resource-based collaboration between atomic units (Assembly Actors), the system executed the required assembly tasks. Our approach is not only targeting a co-design at the lower level, it is also, and using an assembly taxonomy, to concurrent design of assembly systems. The result is an architecture that allows reconfigurability of her units due the correlation between individual unit's goals and assembly system needs in the form of primitive assembly operations.

1.1 Organization of the paper

Section 2 describes Actor-based Assembly Systems from an individual perspective emphasizing the linkage between primitive assembly operations and individual goals of the architectural units. It also presents system's approach by describing the proposed architecture from a mechatronic point of view.

Section 3 discusses the interaction issues between atomic units and presents an approach to collaboration based on services though the individual resources. At the end of this section an illustrative example is presented for the case of two robotic axes.

Section 4 is dedicated to our initial conclusions and future research topics.

2 Actor-based Assembly Systems

The Fig.1 shows the Assembly as a specific manufacturing process. The assembly tasks are accomplishing by different assembly processes. Finally each of the assembly processes is composed of a number of primitive assembly operations

Assembly actors are the atomic units of the architecture, as described above they form the basis for generating actor-vectors corresponding to assembly processes.

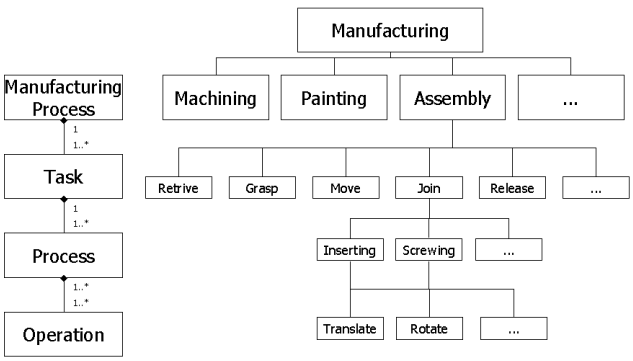

Fig.1. Assembly Taxonomy.

The actor device is modeled by combination of resources including: computational, communication, actuator and sensorial as Fig.2 shows. The resulting mechatronic device is capable of achieve by use of these resources an individual-goal (assembly operation), however is capable of provide services to other members of the society in order to achieve a cluster- or society-goal (assembly processes).

2.1 Actor as an Intelligent Physical Agent

As it was mentioned in the previous chapter, one of the resources used for modeling individual actors is the computational. In our approach we used an approach to agent technology for this particular resource. It has been very much research done in the field of agent technology in the software and AI communities. Some of these research results have been used in manufacturing applications. However, the use of agent technology was limited to the middle and up levels of enterprise operations, been the scheduling the main target for the manufacturing arena. We explore the limits of agent technology in terms of real-time execution and communication, incorporating to the assembly actors with an extra layer of reasoning that the one provided by traditional logic technology, many times executed by Programmer Logic Controllers – PLCs.

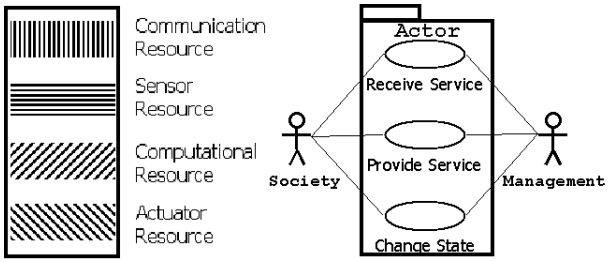

Fig. 2. a) Resource-based Actor Model and b) Use Case UML Diagram representing the Actor Model

The technology in the field of industrial communications is changing driven by advances in other fields like the office automation and multimedia entertainment. In our case we make an extensive use of protocols initially developed for other applications like IEEE-1394 (also known as FireWire) for the communication of the axis controllers, Ethernet for the collection of data or even dedicated Ethernet for commanding I/O modules. This allows having a good real-time information exchange.

Probably one of the most exciting behavior experimented by these mechatronic units is showed using that interaction between individual and society as presented by the Fig.2 and explained later in a dedicated chapter.

2.2 Architecture Description

The development of actor-based assembly systems uses an architecture-based approach. Thus the creation and specification of the system architecture is the main research effort presented in this paper. In this context specification means prescription of what the pieces (Actors) of the architecture are (discussed in the previous chapter) and how are they connected and how they interact.

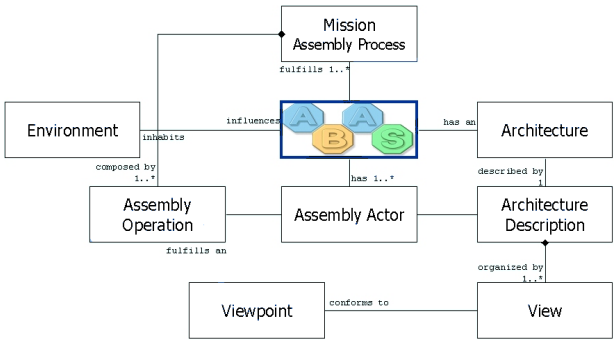

Fig.3. Conceptual Model of the ABAS Reference Architecture following the IEEE Std. 1471-2000 guidelines where the stakeholder is the atomic unit

Unified Modeling Language is used as representation tool for the architecture description and the atomic units models. As noted in [1], one of the main concerns was that UML emerged from object-oriented design, so it is most commonly used to describe things at the detailed design level.

The conceptual model of the architecture is presented in Fig.3. The model follows the recommendations published by the Institute of Electrical and Electronics Engineers under the standard IEEE Std. 1471-2000 [3N]. The model supports the architectural description organized by different views.

The use of views is widely accepted within the software community for describing software architectures. Probably the most representing cases are:

- 4 Views Architecture as appeared in [1]
- 4 + 1 View Model of Architecture by Kruchten [4N]
- 6 View Organization of Models which represent a systems architectures [5N]

Currently we concentrate in two of the views as described by Krunchten 1) Logical View and 2) Physical View. In addition to these views, and also proposed by Krunchten, we provide with scenarios. In the studied domain are representative situations that should be defined as scenarios by the architect such as System-Start-Up, System-Shut-Down or System-Recovery-Fault.

The Fig.3. represents the dissemination of assembly process within the architectural units. It must be noted the use of hierarchical communication networks. This approach is similar to the one standardized by the

International Electrotechnical Commission under IEC-61499 [6N].

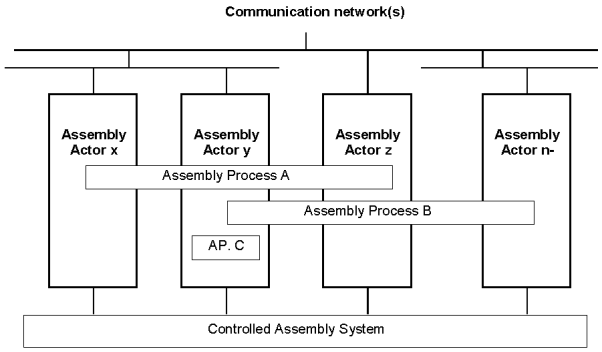

Fig.3. Assembly Process Distribution through the different Actors. The figure also shows the deployment of different communication networks according to the real-time transmission requirements.

3 Service-based Collaboration

Ferber evaluates in [7] different types of situations for agent interactions according the goals, resources and skills of the members of multi-agent systems. This analysis ends in three main categories 1) indifference, 2) cooperation and 3) antagonism. Our approach belongs to first and partially to the second categories, what we call "collaboration". The Fig.4. shows the taxonomy for agent interactions.

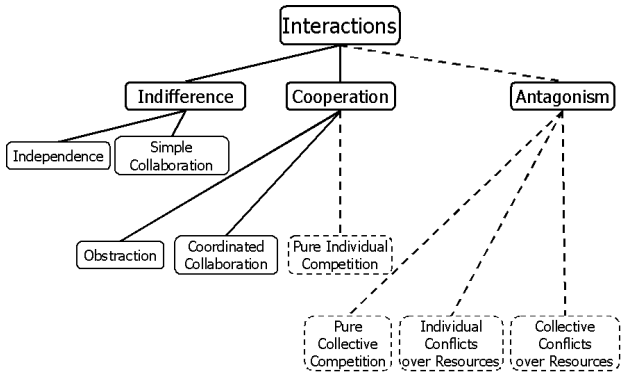

Fig.4. Taxonomy of ways in which Agents can interact. The paths in discontinued lines are not contemplated in the proposed Architecture

The interaction between atomic units is based on services. It is in this situation where the advantages of using agent technology in instance of object-oriented technology are noticed. An object is defined by a certain number of services (its methods), which it cannot refuse to carry out if another object asks it to, and the messages are thus necessarily invocations of methods. However, agents can receive messages, which are not confined, to execution requests but can also consist of information or request for information. The main difference is an agent can refuse to carry out a given job. This special feature is extensively used in the request protocol standardized by the Federation of Intelligent Physical Agents [8] and showed in Fig.5.

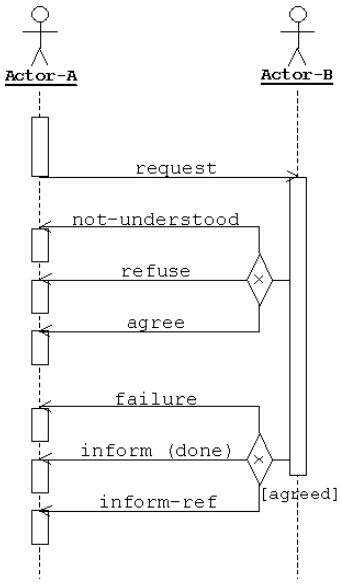

Fig.5. Request Protocol used during the interaction between Actors following FIPA standard

Practically the request protocol starts its execution by the *Initiator,* which can be any member of the actor society including the product information traveling in a bar code label or RF devices. A new software component is dynamically created called *Recruiter*. The mission of *Recruiter* is to secure the services provided by actors and actor clusters. The Recruiter will poll the request for service to all those member of the society which are potentially capable of provide it. This polling action is better semi-constructive showed in the Fig.8 representing the example in the next chapter. The *Recruiter* forms an entity called :Cluster that have the goal of keeping together in a collaborative approach those actors involved in a particular process to be executed by the society, the :Cluster is also a dynamic component that will be destroyed once the society is not demanding any more that process. The Fig.7 provides a static view of the mechanism.

Previously was explained one of the possible scenarios in such actor interaction, in addition the actor cannot refuse to provide the service (e.g. if the execution of the service will end in a negative state for his optimization goals like the case of energy consumption) or can simply not understand for the service that is asked (e.g. in case that is not capable of provide a particular service, this will not be in his list of potentials and obviously will not understand if is asked)

3.1 Two Robotic Axes Illustration

Precise motion is a necessary feature in the field of assembly automation. Many assembly processes require

the control of movement either rotational or transactional.

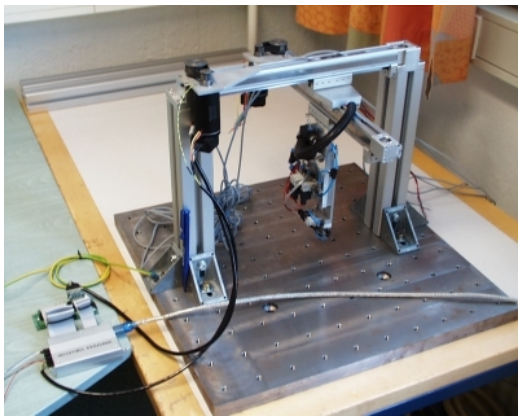

Fig.6. Actor-based Manipulator

For illustrating the collaboration of two actors we use components for the developed actor-based manipulator in Fig.6. This actor society is capable of performing one assembly task: Joining using two different assembly processes: 1) Inserting and 2) Screwing according to the product needs. For both of the processes we need to have at least 3 axes (the second process will need an extra actor for providing one more DOF). The example of collaboration is related to the axes X and Y of such society.

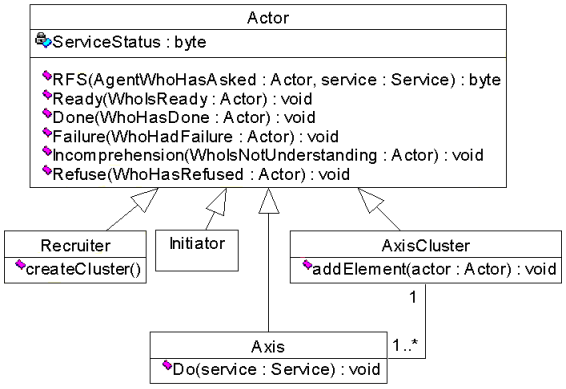

Fig.7. Two Robotic Axes Collaboration UML Class Diagram

The interaction protocol is requesting by the *Initiator* the service move-to a new location. Automatically *Recruiter* is created, which will poll the service to the member involved in the society. Some of these members will not understand the request for service. Those members that will understand the request will evaluate internally if they want (or are capable) of provide the service. In our case axes Xn, Yn and Xn+1 refuse to provide the service motivated by different reasons (in this case Xn, Yn and Xn+1 represent other axes in the society out of our test platform of Fig.6 such as other devices for manipulating in another stage the product to be assembled. *Recruiter* will create *Axis Cluster,* which will distribute the correct attributed for performing the requested services by the actors that were agreed on do it. *Axis Cluster* is also a dynamic entity and is created and destroyed as many times as needed. The Fig.8 illustrates the full sequence.

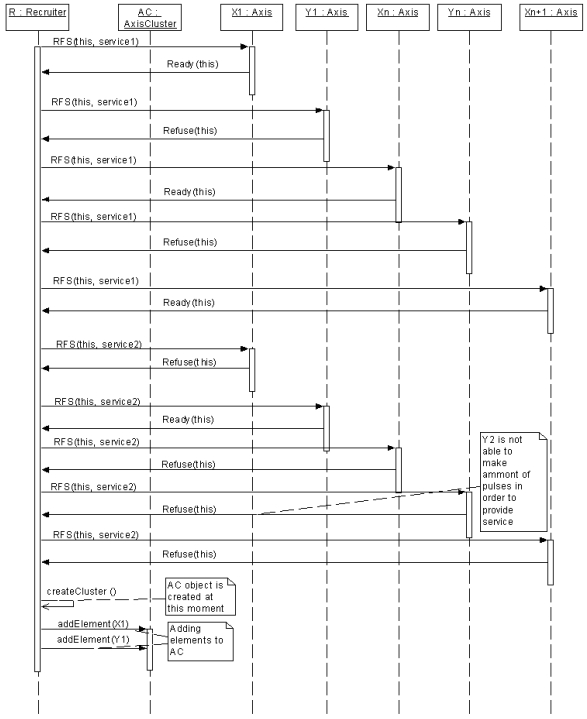

Fig.8. Sequence UML Diagram for a Generic Service-based Collaboration for ABAS with redundant Actors

4 Conclusions

A mechatronic architecture has been presented as an approach to hardware/software co-design of reconfigurable assembly platforms.

Once the application domain is clarified in advance, the architecture description defines hardware/software elements in this context called assembly actors and how they interact using a service-based approach. The mapping of functionality or assembly operations to architecture elements is also enunciated. These three arguments provide the "reference" attribute to our architectural approach.

The type of reconfigurability is statically since the assembly actors have not extra mobility features out of those needed for their assembly goals. The reusability is also an important qualitative attribute achieved by the architecture.

The interaction between actors has been explained and enunciated with a robotic example. Our approach uses the agent style defined by the FIPA protocol. We implement the agent concept using object oriented programming languages. The model is introduced using

Advanced UML diagrams as proposed by the AUML initiative.

Actor uniformity will result in significant manufacturing costs savings. Since there are few actor types to choose from, one might expect this architecture to frequently install excess capability, resulting in higher costs. The efficiencies inherent in producing a greater volume of a much simpler product actually result in considerable cost reduction.

Acknowledgment:
Work supported by the e2Manufacturing industrial consortium[1]; the Finnish National Agency of Technology[2]; and the Tampere University of Technology.
An extended version of this paper will appear as [9]

References:

[1] Hofmeister, C.; Nord, R. and Soni, D. *Applied Software Architecture*, Addison Wesley, 2000
[2] Bass, L. and Paulish, D. J. *Architecture Centric Software Project Managemenet: A Practical Guide*, Addison Wesley, 2001
[3] IEEE, *Recommended Practice for Architecture Description of Software-Intensive Systems*, Institute of Electrical and Electronics Engineers, 2000.
[4] Kruchten, B. The 4+1 View Model of Architecture, *IEEE Software Magazine*, Vol.12, No.6, 1995, pp. 42-50.
[5] Maier, M. Developments in System Architecting, *Proceedings of the 2nd IEEE International Conference on Engineering of Complex Computer Systems*, 1996, pp. 139-142.
[6] IEC, *Function blocks for industrial-process measurement and control systems - Part 1: Architecture*, International Electrotechnical Commission, 2000.
[7] Ferber, J., *Multi-Agent Systems –An Introduction to Distributed Artificial Intelligence*, Addison Wesley, 1999
[8] FIPA, *Contract Net Interaction Protocol Specification*, Foundation for Intelligent Physical Agents, 2001.
[9] Lastra, Jose LM, *A Mechatronic Reference Architecture for Reconfigurable Actor-based Assembly Systems*, Tampere University of Technology, Doctoral Thesis, 2002.

---

[1] e2Manufacturing Industrial Consortium is formed by: ABB Research Center Oy.; FlexLink Automation Oy.; GWS Systems Oy.; JOT Automation Oyj.; Nokia Mobile Phones Oyj.; PMJ Automec Oyj. http://e2manufacturing.net

[2] The Finnish National Technology Agency –TEKES funds the e2Manufacturing research project under the Technology Programme entitle "ÄLY: Intelligent Automation Systems" http://www.tekes.fi