01 Jan 2006

A Methodology for Deriving System Requirements Using Agent Based System Modeling

Karthik Gopalakrishnan

Sreeram Ramakrishnan
Missouri University of Science and Technology

Cihan H. Dagli
Missouri University of Science and Technology, dagli@mst.edu

Recommended Citation

K. Gopalakrishnan et al., "A Methodology for Deriving System Requirements Using Agent Based System Modeling," *Conference on Systems Engineering Research*, Institute of Electrical and Electronics Engineers (IEEE), Jan 2006.

Paper 144.

A Methodology for Deriving System Requirements Using Agent Based Modeling

Karthik Gopalakrishnan, Sreeram Ramakrishnan, Cihan H. Dagli
Smart Engineering Systems Lab (SESL)
Department of Systems Engineering
University of Missouri – Rolla
Rolla Missouri 65401
kg6g7@umr.edu, sreeram@umr.edu, dagli@umr.edu

Abstract

In this paper, we discuss a method to derive the requirements for developing an Industrial Automation and Control System (IACS). An IACS has software components and associated hardware, which together implement the required monitoring, supervision and control of operations in a production plant. The requirements of such a system are multi-dimensional and may require multiple layers of abstraction. For this domain, we propose an agent-based modeling approach to capture information regarding end user's requirements. One of the motivations for adopting an agent-based modeling approach is the implicit flexibility afforded by agents and the negotiation techniques that can be implemented to streamline the change management process associated with requirement modeling and analysis. This paper utilizes modeling constructs from UML/SysML to model and visualize the interactions among the agents. The types of agents and their roles are discussed in detail.

Introduction

System requirements can be broadly classified into three categories - business, functional and non-functional. This paper discusses a means to capture functional requirements of a real-time industrial control system. The definition of requirements is an important part in the role of a systems engineer (INCOSE 1993). Requirements identify the goal of the system and the context in which it will be used – critical information for a system architect. But every new project presents a new challenge to the designer, since requirements are often unique to a project. Research is this direction has spawned the development of various requirements management software tools such as DOORS®.

Current industrial automation systems are implemented in two forms – either as Distributed Control Systems (DCS) or as Programmable Logic Controllers (PLC). In both cases, the control architecture is often monolithic and not easy to modify or to integrate with other systems (Brennan et al, 2002). Due to influx of new software technologies, the domain of control systems has made ad-hoc efforts to adapt itself constantly, so that it can make best use of the available functionality. This has resulted in multiple custom-made proprietary solutions from different suppliers and absence of a standard. As a consequence of this, difficulties have arisen in technical and semantic integration of these software tools.

The recently introduced standard of using function blocks as the main building blocks for development of industrial control systems has addressed this issue to a certain extent. However, function block architecture is still not mature and has some unresolved issues such as, lack of clarity in defining interfaces among blocks, lack of code reuse capability (inheritance) etc. Due to these differences, the function block architecture has not been able to fully exploit the benefits of object-oriented architecture, which is the standard for software development (Thramboulidis, 2001). This presents a requirements modeling problem, due to the differences in the two architectures.

Agent-based systems are continuing to gain acceptance in modeling, as increased number of problems in industrial, commercial and networking applications are being solved using agent-based solutions. The concept is extensively used in software development, where applications are written as agents. Agents have also been adopted in the manufacturing community in the form of holonic manufacturing techniques, where holons or agents are used for upper-level planning and scheduling of the manufacturing process. Even though agents have been used extensively for such higher levels of control, very little effort has been done on applying them to lower real-time control implementation. (Brennan et al, 2002). In order to apply them to lower levels of control, the real-time constraints have to be met to achieve system safety and reliability. In this paper, our effort is focussed on using agents at lower levels of the control hierarchy.

The concept of agent based modeling has also been used in the realm of requirements engineering as well in the form of goal oriented requirements engineering (Choudhury et al 2005). Goal-based methods offer us a clean view of the high-level requirements of the system, by identifying goals of the stakeholders. Popularly used goal-based requirements engineering methods include KAOS (Van Lamsweerde, 2001) and NFR (Chung et al, 2000). Research oriented toward using COTS components for capturing requirements for building complex systems has succeeded in combining ideas of goal oriented and agent oriented approaches and have developed COTS – Aware Requirements Engineering (CARE) (Chung et al, 2002).

In order to build the "right" system, both the user and the developer need to have the same "visualization" of the system. Our objective in this paper is to develop a formal method by which the requirements of the control system can be formally conveyed to the software engineer. In addition to that, the method must also provide a clear means of validating these requirements. Since UML is by far the industry standard for development of complex systems, we are interested in representing these requirements in UML. The rest of the paper is focussed in developing the means for converting functional requirements for developing a control system into UML constructs using agent based modeling technique. In this paper, the process of requirements development is viewed as an interaction between three persons. For the record, we define them as: (1) the control engineer, who is the person responsible for the operation of the plant, (2) the software engineer, who is responsible for developing the control software and (3) the systems engineer who is the liaison between the two.

Background

Industrial Automation and Control System

An industrial automation and control system (IACS) is a system that monitors and controls multiple integrated industrial systems, allows human interaction with these systems, and provides information about the system parameters being controlled. The architecture of an industrial control system is hierarchical (Kopetz, 1999). It can be described as a four-layered pyramid, with the sensors and actuators at the lowest level, associated PLC and DCS logics at the next level, operator and supervisory nodes with graphical interfaces at the next level, and MIS computers that extract data from the operator nodes at the highest level. The architecture is described by Figure 1 below:

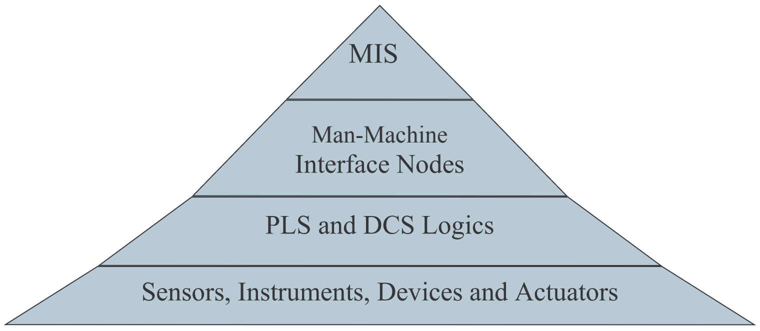
MIS
Man-Machine Interface Nodes
PLS and DCS Logics
Sensors, Instruments, Devices and Actuators

Figure 1. Industrial Automation System Architecture

In present day industrial control systems, different control platforms have been developed by different suppliers using their own control strategies, influenced by the nature of applications over the years (Vogel-Heuser et al, 2002). This has resulted in different types of software and tools to support the development of IACS. To formalize this process, the IEC 61499 standard was introduced by the International Electrotechnical Consortium (IEC) which made function blocks as a building block for industrial control systems. The standard defines a function block as "a functional unit of software comprising an individual, named copy of the data structure specified by the function block type, which persists from one invocation to the next. The functionality is provided by means of algorithms, which process inputs and internal data to generate output data" (Thramboulidis, 2001). Currently, development of industrial control systems takes place according to this standard.

However, there is an inherent break in communication between the user and developer in this process of development. The requirements for the proposed control system are developed by the control engineer in the plant. These requirements are often generated in accordance with the process and instrumentation diagrams (P&IDs) that the control engineer is familiar with. A control engineer's perspective of the system is in terms of devices and flow of data and control between them. Accordingly, the requirements generated are also of a format reflecting plant devices, their attributes and their interconnections. From a software engineer's point of view, the function block paradigm appears to represent software as pieces of hardware. The ideal configuration of requirements for a software engineer is in the form of UML diagrams, where the software to be developed conforms to the object-oriented paradigm. Due to this difference in perspective between the software engineer and the control engineer, the requirements traceability becomes difficult. Therefore, there is a need for a transformation methodology that can represent function block architecture in the form of object-oriented models. This transformation must also act as a means of validating the development method, so that the developed control system is in conformance with the control engineer's needs.

The Function Block Architecture

According to IEC 61499 (first released in 1998), a function block is a modular, reusable unit consisting of 2 parts, one handling the event inputs and outputs and execution control, and the other handling the data inputs and outputs, related algorithms and internal data (R. W. Lewis, 2001). It offers a mechanism that allows industrial algorithms to be encapsulated in a form that can be understood by engineer who are not specialists in implementation of complex algorithms. The data and the events are related; so that the data of the block is associated with the event at occur. The only communication of a function block is with another function block thorough an interface. Multiple such blocks can be connected together to form an application, which again is in the form of a function block. The interface connections for a particular type of function block are destined only to a certain set of function blocks and not all.

Agent Based Systems

The concept of agent-based modeling of systems has invoked a widespread interest. An agent is an active entity that has autonomous properties. It has certain goal and can make decisions conforming to a set of constraints and resources. Based on the environment that an agent encounters and its constraints, a certain behavior is displayed by the agent.

A complex dynamic system can be modeled as group of interacting agents (Wooldridge and Jennings 1995). In such multi-agent systems, system behavior can be realized using the behavior of the agents that make the system. Agent-based models can be decomposed in two types, functionally or physically. Functional decomposition encapsulates modules assigned to functions such as planning, scheduling, handling etc. Physical decomposition is used to represent agents as physical entities such as machines, devices, etc. In our approach, we use both types of decomposition. We use physical decomposition to represent the hardware devices that are required for the control system. These include sensors, actuators and other plant devices that interact with the plant. We use a functional decomposition to represent the control actions that the system is required to undertake. The functional decomposition provides us with a model where every control requirement is represented by the goal of an agent.

The advantage of agent based modeling is that the system model evolves by modeling lower level agents. This makes the task of modeling more tangible. Due to the nature of this modeling strategy, large and complex systems can also be modeled. However large the system is, the model will be built by a process of diffusion, composed of sub-systems and components. At this level, if we attempt to state the purpose and characteristics of each component, we are essentially drafting the agent's functionality. This process adds the coherency required in the modeling process, and at the same time performs the required modeling.

Object Oriented Principles

Even though, the modeling of the system will use agent based concepts, object oriented concepts have to be used for developing the software architecture. Agents are driven by task oriented behavior while object oriented models are driven by structure oriented behavior. Agents possess all properties of objects like passive structure, inheritance, data encapsulation. Beyond these, they have additional properties which are discussed in the sections below.

Establishing this fact is important for our methodology since hardware and software structure of the control system must be object oriented, so that they can be transformed to UML constructs. Field devices like sensors, motors, actuators can be represented in an object oriented fashion. For example, pressure sensors and flow sensors can be treated as objects of the class "sensors". Other hardware components like actuators, motors etc can also be described in a similar fashion. By this decomposition, we are transforming the physical decomposition at the plant hardware level into a functional decomposition at the software level. This ensures the validity of our approach and sets the stage for the use of modeling languages like UML or SysML for modeling of such automation components. UML unifies and formalizes the system development lifecycle and thereby helps developers to visualize the system clearly. The recently developed Systems modeling language (SysML) can also be used since our model uses the templates that are common to both.

Proposed Framework

Requirements Gathering

In this paper, we are interested in deriving the functional requirements of the system by modeling the plant instruments, control system logics, and the interaction between these two. Referring back to the IACS architecture presented in Figure (1), these are represented by the last 2 layers of the pyramid. Following the function block architecture to model process control systems proposed by IEC 61499, we can represent the control system as a hierarchical organization of function blocks. These blocks are arranged in the increasing level of granularity as, system model, device model, resource model and application model (Brennan et al, 2002).

In our approach, we represent hardware and software entities as agents, in accordance with the physical and functional decomposition of the system. We use function blocks to represent the hardware devices and the control software. Group of software function blocks can be treated as the next level application. Hardware agents are modelled as weak agents with limited abilities. Strong agents are software applications that spans over weak agents, forming a network of interacting devices. (Wooldridge and Jennings 1995). The architecture looks as shown in figure (2).

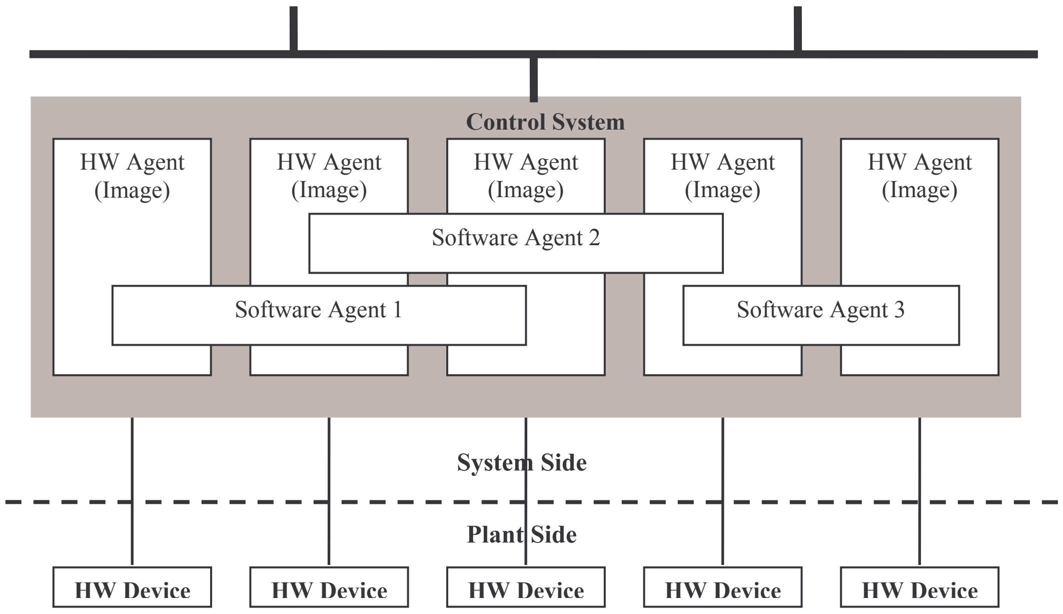

Figure 2. System architecture for IACS

The properties of hardware agents are:

1. Autonomy – Agents have the ability to operate independently, without intervention.
2. Reactivity – Agents respond to external stimulus provided by the control system or by the environment.
3. Communication – Agents can communicate their status to the control system.

Human operator is also modelled as a hardware agent. Software agents have the properties of hardware agents, and also the following properties:

1. Goals – Each software agent has a certain objective.
2. Collaboration – Software agents have the ability to collaborate with other software agents.

Specification of requirements

The first step of the proposed method is to understand the process and identifying the hardware devices that are required to control it. This step provides us with the hardware agents and their roles in controlling the system. Once these agents are identified, they can be structured into a hierarchical decomposition as shown below in figure (3). In addition, the unique behaviour of each of the class of agents is identified. In this way, sensors, actuators, and other hardware devices are aggregated for easy identification. The level of abstraction depends on the application.

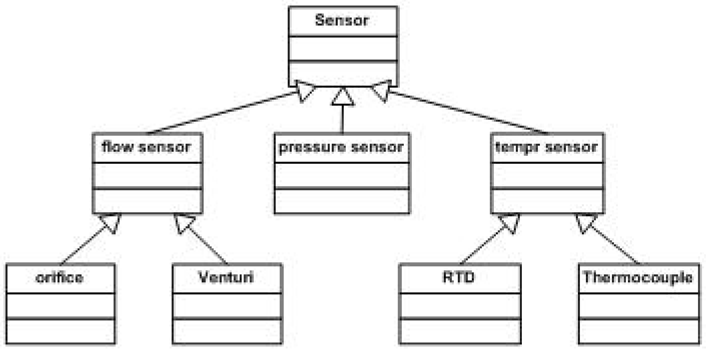

Figure 3. Class diagram for Hardware agents

The next step involves identification of the software agents that span over these hardware agents. There can be any number of software agents over a given set of hardware agents. In addition, software agents can be encapsulated into one another. For example, a temperature controller software agent could be a part of a steam boiler controlling agent. Each of these agents is to be identified. These two steps are implemented by the control engineer.

The next stage of this process of requirements development is executed by the systems engineer, who can now perceive the hardware agents and software agents clearly. First, goal of each software agent is used to represent each software agent as a use-case. This process provides us with the set of use cases that fully represent the process to be controlled. From these use cases, use case diagrams are developed. These diagrams show the operator, the hardware agents that are involved, use cases and their interaction with the hardware agents. The use case diagrams completely capture the main actors and the functionalities of the control system to be developed. In addition to use case diagrams, collaboration and sequence diagrams can be developed.

From these diagrams, the control software is developed. Primarily, for every hardware agent, there has to be an image of the hardware in the software. This image is indeed an object that is developed as belonging to that particular class of devices. Each class of hardware agents will be implemented in software as a function block. A key aspect of function block development of hardware devices is that a certain amount of inheritance can be exploited. Since field devices are abstracted to required levels, field devices at lower levels inherit the capabilities of higher devices. From example, a flow sensor can be an orifice-type flowmeter or a venturi-type flowmeter. In such cases, both these sensors inherit the capabilities of the PRESSURE SENSOR object. Software agents will be implemented in the form of function blocks, or as a network of function blocks. The discretion is left to the software engineer; though it is better to have a smaller **basis** set of function blocks than a large one, from the point of view of modularity.

Since a function block has certain unique properties, this information must be conveyed to the software developer as well. Particularly, this information must be acquired from the 3 agent models for hardware and software agents described below. According to (Burmeister, 1996), an agent can be represented completely using 3 models. For our purpose, we attempt to represent the hardware and software agents based on these three models. The models are:

1. The Agent model – contains the information about the internal structure of the agent and its environment.
2. The Organization model – it specifies the relationships among agents based on roles of agents according to their type
3. The Communication model – contains information regarding the interaction protocol between agents.

Hardware Agent

a. Agent model: Hardware agent model describes attributes of the agent, algorithms and internal data so that it can be implemented as a function block. These attributes are unique to each class of the hardware device. The execution control part of the function block is a finite state automaton representation, indicating the possible states of the block and the transition conditions. This represents the behavior model of the agent. The data input, data output, event input and event output which are common to all hardware agents are represented as beliefs of the agent.

b. Organization model: The organization model identifies the role of each hardware agent. This model is actually a representation of the hierarchy of the devices in the plant. The properties that the hardware agent must possess to be implemented in a certain setting are described using the organizational model. This model is a passive structure of the hardware agents required for the system, describing the inheritance hierarchy as shown in figure 3.

c. Communication model: A function block can be interfaced to another function block only. Hardware agents communicate their operational status and holding value to software agents they are linked to. For example, the actuator communicates its mode of operation (Auto/Manual/Maintenance/Fault) and the percentage the actuator valve, is open. With respect to a function block, the communication is either an event or data. The IEC 61499 standard defines stereotypes for data and event communication. The set of messages that can be sent or received for each type of function block must be clearly defined in this model. Standard protocols for informing, querying, proposing etc must be developed. These are common to the Human operator as well. There is no concept of cooperation in hardware agents, since they are just hardware devices and have no decision making ability.

Software Agent

a. Agent model: Software agents have fixed goals. For example, a software agent's goal is to develop actuation signals based on the error between set value and process value. The agent has settings like P, I and D values, range, etc. There are also algorithms for calculation of the actuation value. These are the attributes of the function block. Software agents can be in different states. Typical states for a controller are Manual, Auto and Cascade. The execution control part of the function block indicates the possible states of the block and the transition conditions representing the behaviour of the software agent. As in a hardware agent, the beliefs of the software agent are the inputs and outputs that the block can have. The properties are similar for function block networks too.

b. Organization model: By inspecting goals of the software agents, a basis set of simple goals can be identified which are to be implemented as unique function blocks. For more complex goals, two or more function blocks are organized in the form of a network to accomplish the goal. There can be systems and subsystems in this arrangement of function blocks, with each subsystem satisfying a certain goal.

c. Communication model: The protocol for inter-function block communication in software agents is same as that of hardware agents. Even in the case of function block networks, the interfaces are defined only for the function blocks in it. Cooperation is achieved in software agents in terms of negotiation between agents. In a batch operation scenario, a common set of hardware agents are used for different production batches. Agents communicate with the hardware agents being used by them in the current batch, so that another process that uses the same hardware cannot be initiated.

UML Constructs Used in IACS – An example

Let us use the above method to derive the requirements of a simple control system in a plant environment, using the example of a steam boiler control system. The objective of the control system is to maintain the pressure of the steam from the boiler to turbine at a set value. Let us use a simplified model, where, there is a temperature controller, a flow controller for the water flowing in to the boiler and the pressure controller that determines the set values for these two. We start by developing the block diagram of the required control system from plant P&IDs. It is as shown in figure (3). The block diagram can be used to display the agents involved in the control system.

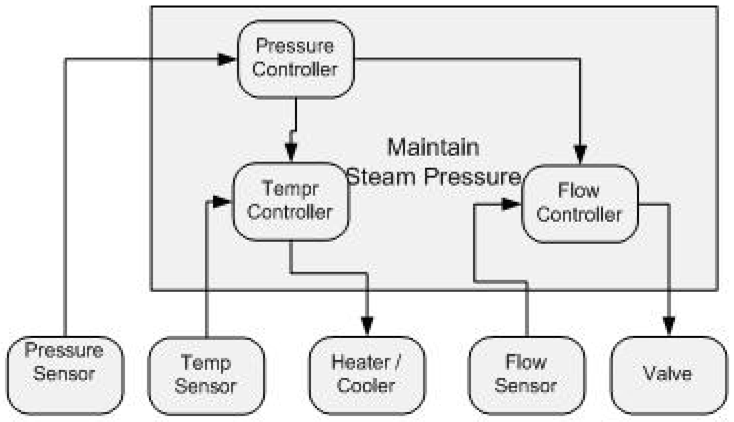

Figure 4. Block Diagram for a plant control process

In the above diagram, we can identify 5 hardware agents and 4 software agents. The "maintain steam pressure" is also a software agent of a network of function blocks. From this diagram, along with the agent models for hardware and software agents, we can derive the use case diagram. The use case diagram for this block diagram is given as:

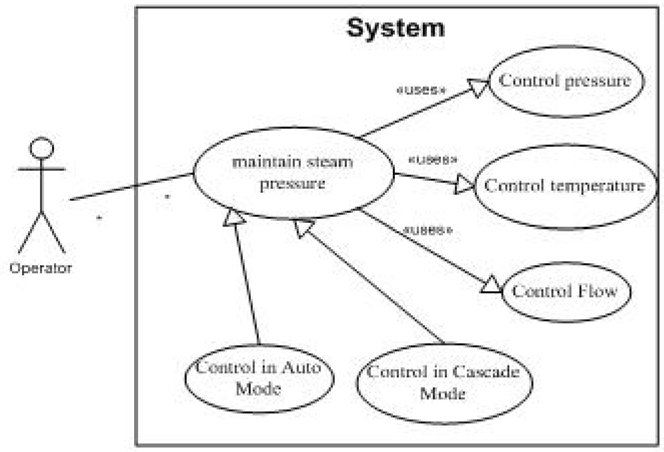

Figure 5. Use case Diagram for a plant control process

In addition to the use case diagram, sequence diagrams and activity diagrams must be provided to depict the steps of action to be taken inside each use case. The agent models reveal this information as well. Collaboration diagrams can be used to represent function block networks. This will reveal the hardware agents associated with each use case. These will also show the data and event messages passed between agents.

Summary

We can see that the agent model captures the functional requirements of an industrial control system both in terms of hardware and software. In addition, the agent models also facilitate the transformation of control system requirements in process control terms into control system requirements in software terms. The process of requirements elucidation is driven by modeling of the agents that are present is the system, and their interactions.

This gives an abstract view of the system functional model. A real time implementation of this method will require several levels of abstraction before we can be satisfied that all the requirements have been identified. By combining multiple goal models, we can construct a model, which starts with the highest-level goal, which could be "continuous generation of power by the turbine" to the lowest goal such as "maintain boiler temperature at a certain temperature". This method also checks for the completeness of the requirements, since at every level, we can identify the hardware and software agents that form the model. The advantage of this method is that UML use case diagrams can be directly evolved from agent models. The validity of this method can be verified from the fact that goal-oriented and agent-oriented methods are well-established methods used in the industry.

Conclusions

This paper proposes a method by which the IEC 61499 standard can be exploited to represent system requirements to the software developer. This paper addresses the issue of requirements management and suggests a method to capture the function block paradigm in UML, thus improving the quality of the development lifecycle. The methodology is conceptually simple and is based on existing proven methods of handling requirements. By this method, the requirements definition part of the system engineering process can be simplified by a large extent. Areas of future work include representation of all types of IACS requirements using this method and implementation of the proposed method using SysML requirements diagrams. In a different front, future work can be focussed on analysis tools for agent based systems and their implementation in systems engineering processes.

References

Brennan et al, An agent based approach to reconfiguration of real time distributed control systems, IEEE transactions on robotics and automation, August 2002

Burmeister, Models and methodologies for agent-oriented analysis and design. In Klaus Fischer, editor, Working Notes of the Kl'96 Workshop on Agent-Oriented Programming and Distributed Systems. 1996.

Choudhury et al, quantifying the evolution of goals in requirements engineering: A study on the quality assurance review assistant tool, INCOSE international symposium, 2005

Chung et al, Non-Functional Requirements in Software Engineering, 2000

Chung et al, A COTS aware requirements engineering Process: a goal and agent oriented approach, INCOSE conference 2002

Kopetz H, Do Current Technology Trends Enforce a Paradigm Shift in the Industrial Automation Market?, IEEE 1999

R. W. Lewis, Modeling Industrial Control Systems using the IEC 61499 Function Block Standard, www.searcheng.co.uk, 2001

Thramboulidis, Using UML for the development of Distributed Industrial process measurement and control systems, IEEE Int. conf on Control Applications, September 2001

Van Lamsweerde, Goal-oriented requirements engineering: a guided tour Proceedings of Fifth IEEE International Symposium on Requirements Engineering, 2001.

Vogel-Heuser et al, Requirements of a process control description language for distributed control systems (DCS) in process industry, Industrial Electronics Society, IEEE 2002

Woolridge and Jennings, Agent theories, Architectures and Languages, First International conference on Multi-Agent Systems, 1995