Specification of a Device Interface for Service-Oriented Automation Control Components

José Barbosa *, J. Marco Mendes ** and Paulo Leitão *

** Polytechnic Institute of Bragança, Campus Sta Apolónia, Apartado 1134, 5301-857 Bragança Portugal (Tel: +351-273303003; e-mail: {jbarbosa, pleitao}@ipb.pt).*
*** Faculty of Engineering of University of Porto, Rua Dr. Roberto Frias, 4200-465 Porto Portugal, (e-mail: marco.mendes@fe.up.pt)*

Abstract: Service-oriented paradigm is being used to develop distributed and reconfigurable control solutions for factory automation environments. Since service-oriented automation control components are logical entities that provide and consume services, they may have an interface that maps the logical processes into the actions of the representative physical mechatronics. The inter-connection with the physical hardware devices, namely accessing I/Os, is a crucial issue to achieve the vertical IT-enterprise integration in these service-oriented systems, covering the shop floor device control level. This paper describes a device interface approach, in the context of a service-oriented control architecture, in which High-level Petri nets are used as the control description to access to the physical device. The outgoing features of the solution allow integrating the physical behavior into the control of automation components and consequently thereby incorporate it in the modular service-oriented control architecture.

Keywords: Service-oriented Architectures, Re-configurable Automation, Device Interface, High-level Petri nets.

1. INTRODUCTION

Programmable Logic Controllers (PLCs) still remains the key of industrial automation, presenting nowadays advanced features, such as networking and high-level programming environments, to support the development of distributed systems. However, last years have witnessed a demand for reconfigurable, modular and cost-effective solutions for industrial automation. In fact, cost, quality and responsiveness are the three main foundations on which every manufacturing company stands on, nowadays, to be competitive in the current global economy (ElMaraghy, 2006).

Several approaches have been proposed to solve the challenge for re-configurability, from Multi-agent Systems (MAS) (Wooldridge, 2002; Leitão and Restivo, 2006) to Service-oriented Architectures (SoA) (Jammes and Smit, 2005). SoA seems suitable to overcome some unanswered questions related to MAS, such as interoperability and modularity, and can support the development of distributed, ubiquitous, networking, modular, interoperable and reconfigurable control systems, based on autonomous, reusable and loosely-coupled distributed components. In this context and aiming to cover the modularity and re-configurability requirements, a service-oriented control architecture for industrial automation is proposed in (Mendes et al., 2008a). The suggested control architecture is built upon modular control components, namely mechatronic control, process control and intelligent support components, each one providing a set of services that represent its internal functionalities. The control of such systems is mainly related to the coordination of the services provided by these distributed components.

The approach considers several levels of control based on a bottom-up perspective, from the device level to the enterprise level, providing a loused-coupled inheritance that enhances the autonomy and reconfiguration capabilities. This organization is also characterized by aggregation capabilities, i.e. simpler units might be aggregated in order to generate more complex structures. For more details about the service-oriented control architecture, see Mendes et al. (2008a).

An important aspect to be considered in the proposed service-oriented control approach for industrial automation is the connection to the hardware automation devices, i.e. the I/O connectivity. In this case, since service-orientation doesn't give any direct access to the "real world", the participants of the system must have a sort of physical interaction to what they stand for. For example, a logical service-oriented conveyor should have an interface to the physical equipment that it represents. Then it makes sense the existence of a device interface as a software module of the logical control component that provides the interface to the physical component. At the end, this allows fulfilling the vertical IT-enterprise integration ranging from enterprise control levels to the hardware control level.

Some approaches to device interface are available on literature, namely the IEC 61499 Function Blocks (Lewis, 2001). However, the IEC 61499 has a close resemblance with object-oriented (Hussain and Frey, 2005), and this work is

related to service-oriented systems. In fact, services are not distributed objects (Vogels, 2003) or even function blocks or entities, but functionalities provided by entities. For this purpose, this paper points out the requirements for specifying interfaces to support the integration of automation hardware devices into the modular framework of a service-oriented automation control component. The focus is the definition of a device interface that provides a way in which the logic controller, e.g. using High-Level Petri Nets (HLPN), accesses the physical device that it represents. Aiming to proof the concept, a particular implementation is developed to a simple case study, as the way to embed new HLPN behavioral models to face re-configurability.

The rest of the paper is organized as follows: first, Section 2 overviews the modular structure of the service-oriented control components and points out the importance of interfaces to support the integration of hardware devices. Section 3 specifies the generic device interface and describes the integration within HLPN controllers. Section 4 describes the implementation of the proposed concepts into a case study. Finally, Section 5 presents the conclusions.

2. MODULAR SERVICE-ORIENTED CONTROL COMPONENTS

In service-oriented systems, the functionalities of automation devices can be abstracted and encapsulated as services. In this section, the generic modular structure for the control components will be briefly described, with special attention to the device interface that provides functions to interconnect physical devices, namely accessing to I/Os.

2.1 Modular Structure of a Control Component

A service-oriented control component is structured in a puzzle of pluggable and reusable modules (Mendes et al., 2008a), as illustrated in Fig. 1, namely the Logical Controller, Communication, Decision and Exception Handler, Device Interface and Event Router-Scheduler. The heart of the puzzle structure is the Event Router-Scheduler module that acts as coordinator of the several asynchronous control components' modules.

The logic controller module regulates the behavior of the component by coordinating the execution of atomic services according to its behavior, described by a logical model. A possible solution to implement this module is to use a kind of HLPN tailored for service-oriented systems (Mendes et al., 2008b), taking the advantage of its powerful mathematical foundation to represent discrete, dynamic and distributed systems. These logic controllers have to interpret and execute the process model expressed in Petri nets. Note that the HLPN formalism is not only used in the design phase (edition, analysis, validation and simulation) but it is also used to implement real-time control systems through engines that execute Petri nets models.

The communication among the control components is done via the invocation of component's services hosted and provided by the communication module. A suitable technological solution to implement the service-oriented communication module is to use Web technology, and most specifically Web services.

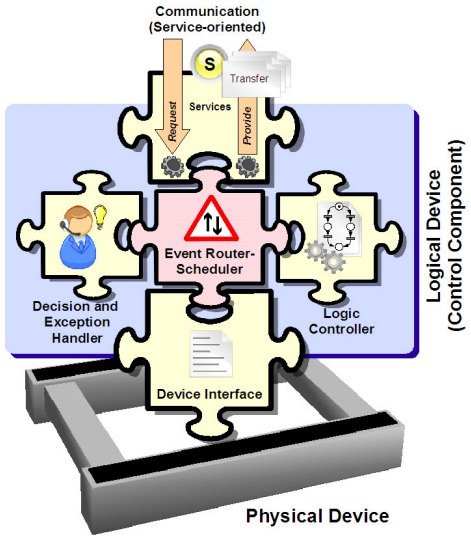

Fig. 1. Modular Structure of a Service-oriented Control Component.

The decision and exception handler module provides, at runtime, decision-making and conflict resolution services, to support the coordination of the services' execution, since the system logic model does not describe a fixed sequence of actions, but rather all possible combinations thereof. Multi-agent systems are technologically suitable to support decision and conflict resolution.

The device interface module provides the mechanisms to integrate the physical device, such as robots or digital sensors, within the logic controller module of the control component.

2.2 Device Interface Module

Having in mind to cover the integration of device control, it is necessary to consider an interface that provides, to the logic controller module, functions to access the real automation devices. The use of standard and decoupled interfaces provides benefits by reducing the development costs, especially in heterogeneous scenarios, as industrial automation or production systems are.

This interface should provide mechanisms for the exchange of real-time data and supervisory control information between the logic controller module and the physical device based on the principles of connectivity and independency. The connectivity aims to support the access to the real-time information provided by the device. The independency allows achieving interoperability and is related to the decoupling of the development of the control solution from the particularities of the device integration. This allows that the integration details are hidden for the control development and the details of control solution are hidden for the integration developer.

For this purpose, the device interface module supplies generic functions to be invoked by the logic controller module. They should be generic enough to be appropriated for a wide variety of devices and applications. For instance, a read function should allow the control component to read a variable from a device, whether the device is a PLC or a robot. The invocation of the function is always the same, from the point of view of the client part, being its implementation customized according to each physical device particularities. Similarly, the functions should be independent of the type of application, as diverse as material handling, assembly systems and chemical systems.

3. SPECIFICATION OF THE DEVICE INTERFACE MODULE

Considering the importance of achieving a transparent and re-usable connection to the physical automation devices, this section devotes a special attention to the specification of the device interface module.

3.1 Interface Implementation Approaches

The implementation of the device interface module is dependent of the controller platform where the logic control engine is embedded: PC-based solution and embedded system solution.

The PC-based solution is characterized by having the logic controller module running in a PC which interacts to the physical automation devices using e.g. data acquisition boards. This solution is more appropriate for upper control levels (e.g. a manufacturing cell controller), where the coordination can be performed using web technologies, and not for lower hardware control level due to the lack of compacted and robustness solutions and to the expensive cost of the solution (cost of the PC plus the data acquisition board).

The embedded system approach aims to achieve a more compact, robust and economic solution by embedding the logic controller module in a microcontroller or a PLC. Usually, the control programs embedded in a PLC are coded using the IEC 61131-3 languages, commonly using Ladder Logic diagrams, allowing an easy access to I/Os but very difficult to debug and modify (Zurawski and MengChu, 1994). In this work, instead of the traditional IEC 61131-3 languages, the implementation of logic controller engines uses HLPN for the control description, taking advantages provided by the mathematical foundation of the Petri nets. In fact, Petri net based controllers are easy to design, implement, maintain and re-configure.

One of the first documented implementations of Petri Net based sequence controllers were developed in the early 80's by Hitachi Ltd. (Murata et al., 1986). It was successfully used in real applications to control parts on an assembly system and to control an industrial robot. The use of Petri nets, as reported, substantially reduced the development time compared with the traditional approaches (Murata et al., 1986).

The major difficulties in implementing embedded Petri net based controllers are due to the lack of proper programming environments and real-time resource constrains. In this case, the software-development is much more complicated and expensive when compared with the PC-based solution, and the software solution has physical restrictions, such as the memory limitation.

3.2 Definition of the Generic Device Interface

The type of embedded solution and the particularities of the controller brand impose specific requirements in the development of the device interface module.

The generic device interface module comprises a set of functions that allow accessing the physical automation device. The pertinent question associated to the definition of such generic interface is related to the identification and specification of the set of functions provided by the interface. In this work, the identification of the functions provided by the device interface module is done taking in consideration the MMS (Manufacturing Message Specification) protocol (ISO/IEC9506-1, 1992). The use of MMS specifications in the definition of interface functions is important in order to achieve standardization, but due to their complexity and restricted scope of the current interface, only a sub-group of functions were defined.

In this way, the functions supplied by the device interface module are aggregated in reusable libraries, providing the interaction with the industrial automation devices, such as I/O systems, PLCs and robots. These three libraries, as represented in Fig. 2, are the Variable Handling, Program Handling and Event Management.

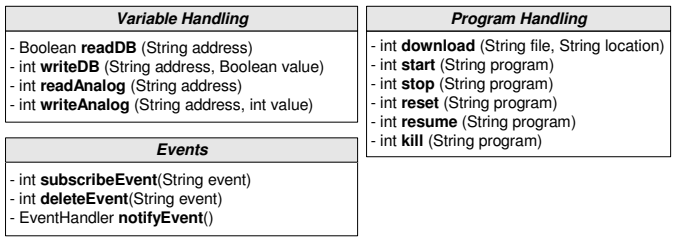

Fig. 2. Libraries of Functions Provided by the Device Interface Module.

The functions associated to the variable handling group intend to access the contents of remote variables hosted in hardware devices, namely reading or changing their values. The variables can be from different types, namely I/O variables or analog variables. The functions included in this group are:

- *readDB* and *readAnalog*, used to obtain the values of I/O and analog variables, respectively.
- *writeDB* and *writeAnalog*, used to change the values of I/O and analog variables, respectively.

The program handling group provides a set of functions that allow handling the execution of programs in the device, namely:

- *download*, used to transfer a program, e.g. a PLC ladder program or a robot program, from a specific location to the device controller.
- *start*, *stop*, *reset*, *resume* and *kill* are used to control the execution of a program, causing a change in the program status.

The functions provided by the device interface module, especially those related to the previous groups, are normally triggered by the logic controller module that has the initiative to execute an action over the real world. However, the event management group provides services for notifying dynamically the logic controller module of event condition transitions, e.g. an emergence signal, allowing the event-driven operation mode. The functions defined in the event management group are:

- *subscribeEvent* and *deleteEvent*, used to subscribe and delete event condition objects that when occurred will trigger a notification event.
- *notifyEvent*, issued by the device interface to notify the logic controller module upon the occurrence of a change in the state of the event condition that were specified in the subscribed list of events; for instance, an alarm is generated when a process variable, e.g. temperature, exceeds a certain preset threshold.

The functions provided by the device interface module were completely characterized in terms of scope, implementation structure and protocol used. In general, the input parameters and return values of each available function are always the same from the point of view of the logic controller module, making transparent the development of the control applications from the particular details of each physical device, improving the ability to support the heterogeneity. As an example, consider the *writeDB* function that is used to write digital outputs when invoked, e.g. *writeDB(O5.3,1)*. The function comprises the address parameter that represents the address of the bit to be written and the value parameter that contains the value to be written. The function returns a value that will identify the success of its execution: -1 if an error had occurred during the execution of the function and 0 if the function was successfully executed.

3.3 Integration with HLPN-based Controllers

As previously referred, the use of the HLPN formalism to implement the control of service-oriented based automation systems allows a modular, flexible and reconfigurable solution. It supports the complete life cycle of these systems, namely the design, analysis, validation, operation and re-configuration. Additionally, logic controller modules based on HLPN formalism allows covering different control levels from the high-strategic control level to the physical control level, as illustrated in Fig. 3.

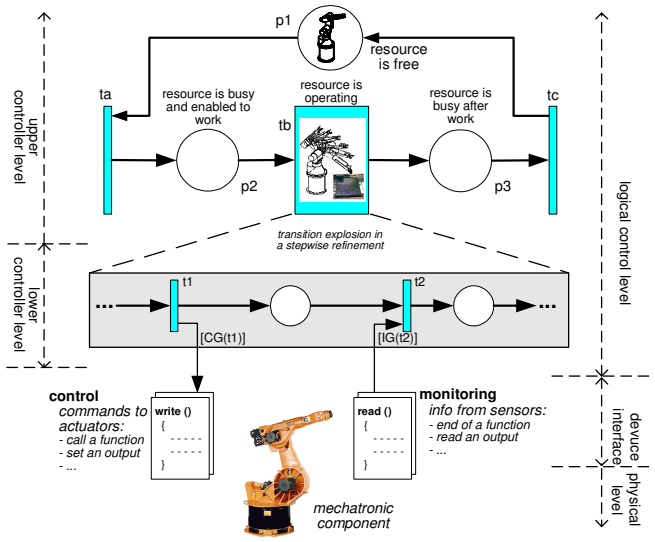

Fig. 3. Real-time control using High-level Petri nets.

This is possible by the successive refinement of the HLPN models, by exploding the three-phase (timed) transitions (Colombo, 1998), allowing to achieve a close interaction with the real devices (Colombo, 1998; Murata, 1989). The operation *"resource is operating"* represented by the transition *tb* of Fig. 3 may be refined into a detailed sub- Petri net that can invoke production automation operations, by calling the functions provided by the device interface, namely the functions included in the variable handling, program handling and events groups. Accessing the physical device interface functions, the evolution of the Petri net model can be synchronized with the real signals from the automation devices, both to send commands (e.g. calling a function or set an output) or to get information (e.g. end of a function or read an output). The pertinent question is related to how the interface functions provided by the device interface module are used in the HLPN models.

Guards, also called transition firing-modes, are associated to the transitions. They represent restrictions to the type of data value, i.e., colored marks, which a transition can move during its firing. Typically, different functions can be modeled:

i) [G&(t)], that is a Boolean function of parameters;

ii) [IG(t)], that is a Boolean function of external events;

iii) [CG(t)], that is a Boolean function that issues external events.

These guards' functions associated to the transitions use interface functions according to the specification of the operation to be executed. On the other hand, the arcs have an attached function (expression) that describes how the state of the HLPN changes when the transitions are fired, i.e. defining the relations between the firing-modes of transitions and the marking of places.

4. IMPLEMENTATION INTO AN EXPERIMENTAL CASE STUDY

An experimental test has been done to validate the proposed concepts, namely the device interface specification for the modular framework of the component, as part of the service-oriented control architecture for industrial automation.

4.1 Device Interface Implementation

In this experimental validation, the interface addresses the PC-based solution, i.e. the HLPN controller is running in a PC and the interface uses a data acquisition board to connect the I/O hardware. For this purpose, it was used the National Instruments USB-6216 data acquisition board, equipped with 16 analog inputs, 2 analog outputs and 32 digital I/O's and 2 counters. The analog inputs can be sampled as much as 400kS/s with a resolution of 16-bit and the counters have a resolution of 32-bit.

The current interface was developed by implementing two basic functions in the data acquisition board, namely the *writeDB* and the *readDB*. These functions were developed in the C programming language, as all the other software pieces (e.g. the HLPN engine). One of the main goals is to develop a solution that is independent from the manufacturer or device that is to be targeted. For this purpose, the developed functions were the most generic and yet simple. Fig. 4 illustrates the structure of the code implementation and the integration of the interface functions into the HLPN-based controller.

The device interface module (resulting after the compilation into a software library) implements the function description of the corresponding header file, i.e. the functions *writeDB* and the *readDB*. It also makes the interface to the data acquisition board, accessing the necessary values to setup the equipment.

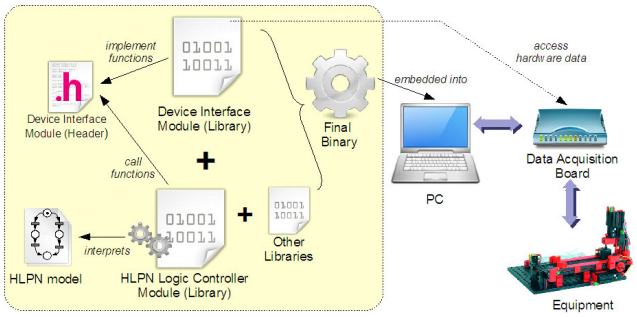

Fig. 4. Development of Integrated Control Solution.

The development of the HLPN logic controller, besides its internal procedures to handle HLPN models and setting up the main loop to run the applications, also requires the inclusion of the header of the device interface (i.e. *interface.h*) in order to call the functions *writeDB* and the *readDB*. Linking the several libraries together results in a binary kernel that can run in the test controller, in this case a PC, connected to the acquisition board.

4.2 Experimental Test Validation

Having the HLPN controller running in a PC and the device interface developed to be used by a data acquisition board, a HLPN model representing the behavior of the system is required to complete the control system for the case study.

The first experience was related to a transport system comprising two interconnected conveyors, from FisherTechniks™, to transfer parts from the input to the output of the system. Each conveyor has one motor (*m1* and *m2*) and two infrared sensors (*s1_i*, *s1_o*, *s2_i* and *s2_o*) to indicate that the part is in the input and output positions. The behavior of the system is the following: when a part is positioned in front of the input sensor, the conveyor starts moving until the part arrives to the output position sensor.

This behavior is represented with the Petri net model illustrated in Fig. 5. This model aggregated to the HLPN kernel and the device interface module, constitutes the HLPN controller.

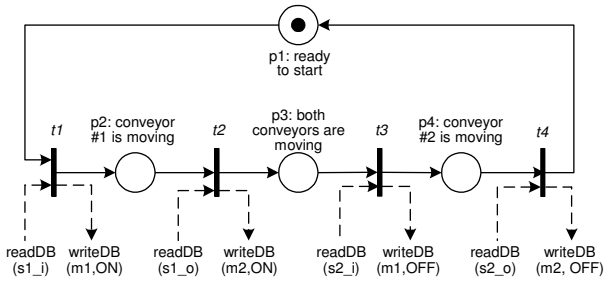

Fig. 5. Behavioral Model for the Transport System.

Each transition has associated the invocation of functions over physical I/Os, in order to monitor and change the system status. The access to sensors and motors from the logic controller module allows the real-time evolution of the system, according to the current status of the system. The operation of the HLPN controller allows verifying that it controls the system as expected during the specification phase, with significant advantages in terms of development of the application.

In order to proof the powerfulness of the proposed approach, supporting the easy and fast re-configuration of the system, a second case study is used, in this case a drilling cell from FisherTechniks. The drilling cell comprises a conveyor and a drilling machine, aiming to drill the parts that arrive to the input position of the conveyor. The system has two infrared sensors to indicate that the part is in the input position (*s1*) and in the machine position (*s2*). The conveyor is moved in two directions with one motor and the drill is moved up and down with another motor. The behavior of the system is simple: when a part is positioned in the input sensor the conveyor starts running till the machine position sensor be reached; then the machine starts drilling during a specific amount of time and then the part is conveyed out the cell.

In this situation, the HLPN engine and the device interface remains the same, being only necessary to change the Petri net behavioral model, which is represented in Fig. 6.

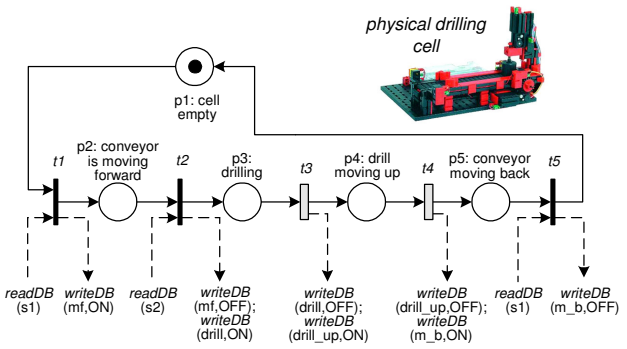

Fig. 6. Behavioral Model for the Drilling Cell.

The new model is simply aggregated to the HLPN engine and the device interface, and starts controlling the system immediately with low effort in the design phase. This allows proving the achievement of easy and fast re-configurability.

6. CONCLUSIONS

New generation of industrial automation and production systems has a strong presence of IT and artificial intelligence techniques. A promising approach is the use of service-oriented principles in automation domain aiming to achieve modularity and re-configurability. In these scenarios, and aiming to achieve vertical integration of service-oriented components from the factory's shop-floor into the IT-enterprise, an important issue is the interconnection of control system to the hardware devices.

This paper discussed the need for generic device interfaces to support the interconnection of the logic control with the physical devices. For this purpose a generic interface was specified based on a set of interface functions inherited from the ideas of MMS protocol. In order to validate the proposed concepts a preliminary implementation was done considering two automation case studies.

Future work is related to the further specification of interface functions, with special attention devoted to embed HLPN controllers in μC and PLCs platforms.

REFERENCES

Colombo, A. W. (1998). Development and Implementation of Hierarchical Control Structures of Flexible Production Systems using High-level Petri nets. PhD Thesis, Fertigungstechnik, Erlangen.

ElMaraghy, H. (2006). Flexible and Reconfigurable Manufacturing Systems Paradigms. *International Journal of Flexible Manufacturing Systems*, 17, pp. 261-271.

Hussain, T., and Frey, G. (2005). Migration of a PLC Controller to an IEC 61499 Compliant Distributed Control System: Hands-on Experiences. Proceedings of the IEEE International Conference on Robotics and Automation, pp. 3983-3989.

ISO/IEC9506-1 (1992). Industrial Automation Systems - Manufacturing Message Specification, Part 1 - Service Definition.

Jammes, F., and Smit, H. (2005). Service-oriented Architectures for Devices - the SIRENA View". Proceedings of the 3rd IEEE International Conference on Industrial Informatics, pp. 140-147.

Leitão, P., and Restivo, F. (2006). ADACOR: A Holonic Architecture for Agile and Adaptive Manufacturing Control. *Computers in Industry*, 57(2), pp. 121-130.

Lewis, R. (2001). *Modeling Distributed Control Systems Using IEC 61499 – Applying Function Blocks to Distributed Systems*. IEEE.

Mendes, J. M., Leitão, P., Colombo, A. W. and Restivo, F. (2008a). Service-oriented Process Control using Modular High-Level Petri Net. Proceedings of the 6th IEEE International Conference on Industrial Informatics, pp. 744-749.

Mendes, J. M., Leitão, P., Colombo, A. W., and Restivo, F. (2008b). Service-oriented Control Architecture for Reconfigurable Production Systems. Proceedings of the 6th IEEE International Conference on Industrial Informatics, pp. 750-755.

Murata, T. (1989). Petri nets: Properties, Analysis and Applications, *IEEE*, vol. 77, pp. 541-580.

Murata, T., Komoda, N., Matsumoto, K. and Haruna, K. (1986). A Petri net based Controller for Flexible and Maintainable Sequence Control and its Application in Factory Automation. *IEEE Transactions on Industrial Electronics*, 33(1), pp. 1-8.

Vogels, W. (2003). Web Services are not Distributed Objects, *IEEE Internet Computing*, 7(6), pp. 59-66.

Wooldridge, M. (2002). *An Introduction to Multi-Agent Systems*. John Wiley & Sons.

Zurawski, R. and MengChu, Z. (1994). Petri nets and Industrial Applications: A Tutorial. *IEEE Transactions on Industrial Electronics*, 41 (6), pp. 567-583.