23rd ABCM International Congress of Mechanical Engineering
December 6-11, 2015, Rio de Janeiro, RJ, Brazil

IMPLEMENTING A RECONFIGURABLE AND DISTRIBUTED MANUFACTURING CONTROL SYSTEM

Robson Marinho da Silva
Universidade Estadual de Santa Cruz, Ilhéus, BA, Brasil
rmsilva@uesc.br
Fabrício Junqueira
Diolino José dos Santos Filho
Paulo Eigi Miyagi
Escola Politécnica da Universidade de São Paulo, São Paulo, SP, Brasil
diolinos@usp.br,pemiyagi@usp.br

Abstract. *This paper presents a step-by-step method to implement a Reconfigurable and Distributed Manufacturing Control System (RDMCS), which must allow the integration of value-added activities, information, resources and knowledge among heterogeneous business workflow of manufacturing subsystems in different geographical locations. These subsystems need to be aware of each other to attain the global objective of the system. Due the involved complexity, a modeling technique with well-defined formalism is used to ensure a systematic application of the design specifications and an effective degree of collaboration. Tools for edition, simulation and implementation should be considered to reduce involved costs and dispensed time. There is not a unique technique that allows develop a RDMCS that is, the proposed method is based on Petri net, holonic and multi-agent system and service-oriented architecture techniques, which have also associated software for generation quasi automatically of controller languages and for debug, deployment and simulation. Application example demonstrates the effectiveness of the resultant control system, such as in autonomy, flexibility, collaboration, reuse and robustness.*

Keywords: *reconfigurable manufacturing system, distributed system, Petri net, holonic and multi-agent system, service-oriented architecture.*

1. Introduction

Manufacturing systems have been designed to meet demands of goods production, transforming materials and processing information based on control functions. These functions were organized into hierarchical levels: *enterprise control* — where decisions and control are made in long-term management; *factory control* — where decisions and control are related to ordination of requests and supervision of productive processes; and *shop-floor control* — where decisions and control are machines operations, which are automatically executed through programmable controllers connected to sensors and actuators (da Silva *et al.*, 2015a).

The evolution of the manufacturing systems is characterized by a gradual migration of paradigms: *mass manufacturing* is concerned to cheaper products; *lean manufacturing* is concerned to continuous quality improvement; *flexible manufacturing* is concerned to products diversity; and *reconfigurable manufacturing* is concerned to be adjustable to business and market interests. The future paradigm is called industry 4.0 which is considered the fourth industrial revolution. In industry 4.0, new technologies (such as cyber physical systems) should integrate machines and humans to compose value chains of entities (such as industrial plants) in different geographical locations (disperse), which should provide services and products in an autonomous manner (Kolberg and Zühlke, 2015; da Silva *et al.*, 2014).

Despite advantages of the new paradigm, its implementation depends of a control system that ensures the integration of the technologies. In this context, the control systems must consider new aspects, such as: (i) reconfiguration flexibility (da Silva *et al.*, 2015a) to increase safety and allow creating, updating or replacing the new and/ or legacy systems in case of new demands, (ii) autonomous and intelligent components for adequate use of the resources, and (iii) supervision of the system goal combined with the autonomous components. These challenges, associated with the lack of quantitative performance data to evaluate these aspects, restrict the expansion of new paradigms (Vrba *et al.*, 2011; Kolberg and Zühlke, 2015). A breach in traditional methods must be considered to model and implement this control system, here named Reconfigurable and Distributed Manufacturing Control System (RDMCS).

In da Silva *et al.* (2015a), a general description of a method to design manufacturing control system considering the reconfiguration flexibility is proposed. da Silva *et al.* (2015b) focused on modeling new mechanisms to allow this reconfiguration flexibility. However, according to Caire *et al.* (2008), workflow is understandable by domain experts as well as by developers, because a workflow reduces the need for programming skills. Therefore, a new challenge to process reconfiguration flexibility is to bring that method and its modeling approach from the control levels to the system internal

logics.

For adequate use of resources, a method to model control system based on formalisms must also avoid repetition of tasks and overlapping of project scope through reuse models, ensuring the implementation of the design specifications, considering interoperability and collaboration among its entities (Hasegawa *et al.*, 1999).

Holonic and Multi-Agent System (HMAS) allows conception of an intelligent control component (agent) for RDMCS. Moreover, the agent-based control software, which is part of the resource, can be endowed with communications and information processing capabilities, transforming it into a self-reconfiguring, intelligent element, i.e., a holon. The result is a distributed intelligent automation system associating hierarchical and heterarchical control that can combine global supervision with the autonomous components (Quintanilla *et al.*, 2015; Vrba *et al.*, 2011; da Silva *et al.*, 2014).

To consider collaboration activities among distributed subsystems and their staff, the service oriented architecture (SOA) is an effective solution. In SOA, an integration level is introduced between the *factory control* and *shop-floor control* levels. The integration level is a middleware which has available application tools and interfaces to communicate with other applications in other levels. Some studies (Quintanilla *et al.*, 2015; Caire *et al.*, 2008) present new perspectives to combine the SOA and HMAS concepts, however there are few publications considering a method based on formal technique (formal method). da Silva *et al.* (2014) propose a architecture to apply SOA concepts in HMAS which can be extended with appropriate use of tools to the agents request web services.

Therefore, this paper proposes a method to model a RDMCS focusing on implementation phase description and its associated tools. The method combines Petri Net (PN), HMAS, SOA, and other good practices to describe the models, validate control mechanisms, analyse the resulting behavior, generate control programs for controllers, allow qualified staff to collaborate among them in a cloud engineering vision (da Silva *et al.*, 2015b). The considered control architecture, an overview how to obtain it and the main involved concepts are presented in Section 2. Section 3 presents the method to model a RDMCS and the adopted formalisms. Application examples are presented in Section 4. In Section 5 there are the main conclusions.

2. Proposed Architecture for Reconfigurable and Distributed Manufacturing Control System

Figure 1 illustrates an overview of the method to develop the holonic control architecture. To create, modify ou delete manufacturing systems processes, the systems engineering should abstract models of processes of certain domain. This must be made using methods and a control functions to define artifacts which are conceived by analysts, managers and developers.

Holonic systems abstraction (Quintanilla *et al.*, 2015; da Silva *et al.*, 2014) is used, in wich a holon can be composed by larger holons or being decomposed in atomic holons. They should have characteristics of autonomy, recursion and collaborating and interact with same level and with holons of higher or lower levels. The interactions vary from simple information exchanges through requests to perform a particular operation until complex negotiations.

The architecture considers the following holons: (i) *Product Holon (PrH)* which contains the necessary knowledge for the general operation of RDMCS and choosing the general strategy that attains the planned objectives. Each *PrH* has an internal process flow, the required input types and output types. The *PrHs* represent the products of the RDMCS which are treated as intermediate products if there is some manufacturing operation to get the final products; (ii) *Task Holon (TH)* which contains the knowledge to manage the execution of the strategies to attend the requests (market demands); (iii) *Supervisor Holon* (*SuH*) which contains all the knowledge to coordinate holons of lower hierarchical levels, registering the abilities of each component and providing services combined with other entities of the control system; and (iv)

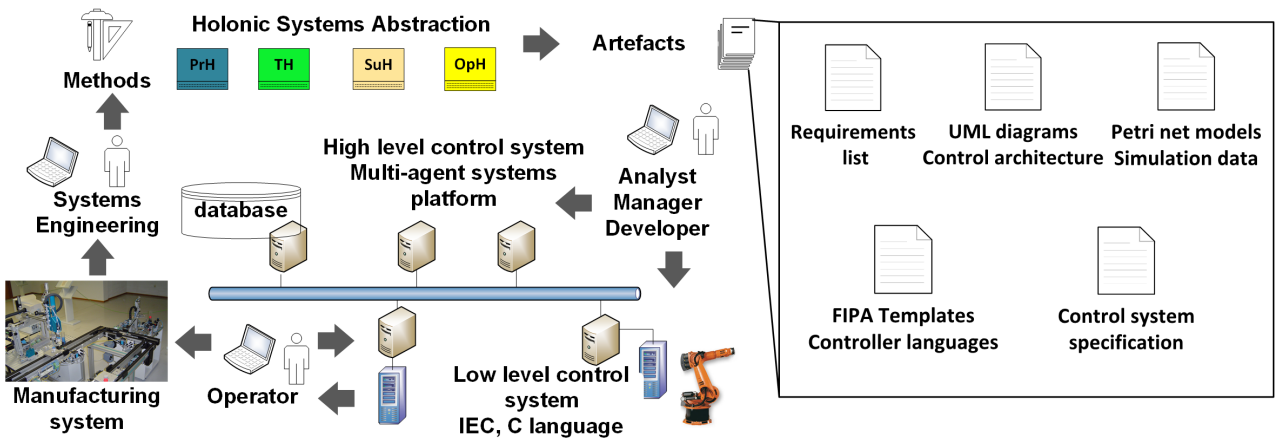


Figure 1. An overview of the control architecture of reconfigurable and distributed control system.

Operational Holon (*OpH*) which represents the RDMCS physical resources (operators or equipments) that have specific control devices for its automatic operation, and determines the behavior of these resources in accordance to its objectives and abilities. To process a production order, a holarchy represents a business processes. These holarchies are created based on collaboration of holons, providing a middleware for integration among the manufacturing control levels. A holon can belong different holarchies of different control levels. The control system considers mechanisms to determine the best holarchy formed to fulfill the production order based on the available resources and negotiation. Section 3 details the used and generated artifacts to define the control system specification, such as requirements list, UML diagrams, PN models and simulation data, as well as the adopted modeling technique.

A trend to implement systems based on holonic abstraction is the use of *Holonic and Multi-agent System* (HMAS) (Quintanilla *et al.*, 2015; da Silva *et al.*, 2014) technique, in which holon and agent concepts (autonomy, reactivity, cooperation, social capacity and fault tolerance mechanisms) are combined. To implement a multi-agent systems, certain functions such as messages transportation, codification, semantic analyses and ontology (Béhé *et al.*, 2014) are required. For this purpose, HMAS techniques adopts the concept of open source which comply to object oriented programming and standards of Foundation for Intelligent Physical Agents (FIPA) (FIPA, 2002).

In this control system, for high-level software is made in JAVA language using JADE (Java Agent Development Framework) (Bellifemine *et al.*, 1999) and its extension, the Workflow Agent Development Environment (WADE) (Caire *et al.*, 2008). They are open source middleware for the development of distributed applications based on the multi-agent system paradigm. JADE plus WADE are also adopted because they facilitate the reconfiguration and bring the workflow approach to the level of system internal logics, and, furthermore, provide advanced administration and fault tolerance mechanisms.

Figure 2 illustrates specific WADE components (highlighted in blue) (Caire *et al.*, 2008). Each instance running of the JADE plus WADE is called "Container" and a set of containers is called "Platform". In a platform a single special "Main Container" must always be active and the other containers register with it at start-up. One or more application agents can be started into a container. To perform its tasks an agent may need to communicate with other agents in the platform. The format for exchange messages is the Agent Communication Language (ACL) defined by Foundation for Intelligent Physical Agents (FIPA) (FIPA, 2002). WADE specific components are: (i) Boot Daemon: there is one for each host in a platform and it activates the containers in its local host; (ii) ConFiguration Agent (CFA): it runs in the main container and is responsible for interacting with the boot daemons and controlling the application life cycle; (iii) Controller Agent (CA): there is one for each container in the platform and they are responsible for supervising activities in the local container and for all the fault tolerance mechanisms.

Furthermore, WADE provides an extension called Workflow Engine Agent (WEA), that embeds a workflow engine of the basic JADE agent components (in yellow in Fig. 2). For this, an Eclipse plugin named WOLF that allows developers to work with a graphical view in synchrony with Java code view is used. Consequently, besides normal JADE behaviours, all active WEA is able to execute workflows according to specific formalism. Thus, workflows can be edited, re-factored, debugged and can include methods, inner classes, references to external classes and so on to implement the process details. Developers can also describe other purely Java patterns elsewhere.

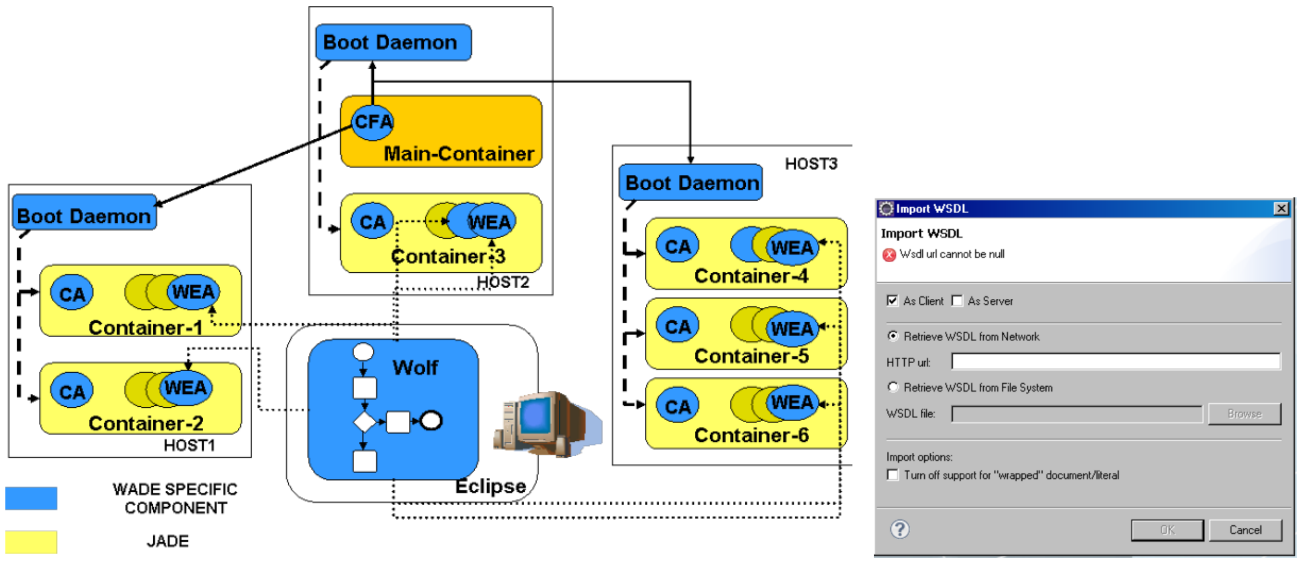


Figure 2. WADE plus JADE platform to implement multi-agent systems using workflow mechanisms (Caire *et al.*, 2008). WADE components (highlighted in blue) provides an extension that embeds a workflow engine of the basic JADE agent components (highlighted in yellow).

The SOA is adopted to ensure the interaction between heterogeneous systems, i.e., to allow inter-enterprise collaboration. In the other hand, Quintanilla *et al.* (2015) have considered another combination of the use of SOA concepts in HMAS. This advance is considered in WADE, where a Web Services Description Language (WSDL) routine can be performed by agents requested service available by agents of other platforms. On right in Fig. 2, a WADE interface where the developer can program an agent to request a service as client or server is illustrated.

For low-level control, there is few hardware that can be implemented based on SOA and HMAS concepts. Vrba *et al.* (2011) present a development and hardware in this sense, however their solution is proprietary. Therefore, as this study considers a practical solution, for low-level control applications the code generation must follow graphical language proposed in IEC61131 (John and Tiegelkamp, 2010) (through sequential flow chart due its similarities to PN models) or using C code (through the use of tools for automatic generation).

3. Modeling and Implementation of the Proposed Control Architecture

This Section presents the method (Fig. 3) considering the development since initial conception until operation phase of the proposed control architecture. A protocol for communication and integration among different design specification is proposed associating a derivation of channel/ agent PN, called Production Flow Schema (PFS) (Hasegawa *et al.*, 1999) with UML. The method is divided into *phases* and each one involves *actors*[1], the utilized and generated *artifacts*. The *phase 5 – integration* – ensures a closed-loop to manage the re-engineering of the other phases.

A RDMCS can be approached as a class of discrete event system. Therefore, PN and its extensions are adequate for modeling its behavior. According to Hasegawa *et al.* (1999), this combined approach allows modeling system structures as well as system behaviors in a more simplified and transparent manner than ordinary Petri net representations. The PN technique is also used in the description of workflow of itself method.

According to Murata (1989), the place transition PN is a 5-tuple (P, T, A, W, M) where: (i) $P = P_1, P_2, P_3, ..., P_r$ is a finite set ($r \in \mathbb{N}^*$) of `places`[2]; (ii) $T = T_1, T_2, T_3, ..., T_s$ is a finite set ($s \in \mathbb{N}^*$) de `transitions`; (iii) $A = PXT \bigcup TXP$ is the `oriented arcs` set; (iv) $W : A, I, H \rightarrow \mathbb{N}^*$ is a function `weight` associated to `arcs`, in which the absence of indication indicates an unitary `weight`; and (v) $M = M_o, M_1, M_2, ..., M_r$ is the `marking`, in which $M = m_1, m_2, ..., m_i$ ($i \in \mathbb{N}^*$) is defined by vector of these `markings` involving the number of `marking` m_i in the `place` P_i. M_0 is the `initial marking`.

Specifically, this work adopts the PN extensions: (i) a channel/agent PN type called Production Flow Schema (PFS) (Miyagi *et al.*, 1988), and (ii) a non-autonomous PN model named Input Output Place Transition (IOPT) (Gomes *et al.*, 2007).

The PFS is a technique for workflow specification at abstraction level and it is effective to obtain a PN according to procedures base on "good properties" to describe productive systems.

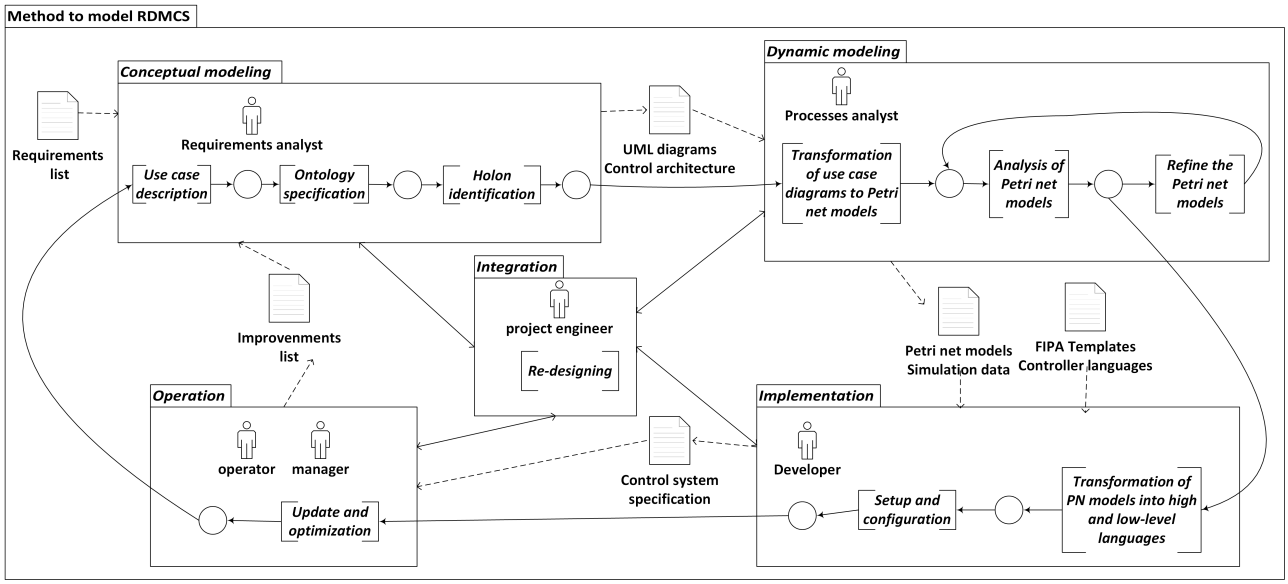


Figure 3. Production flow schema of the method to design reconfigurable and distributed manufacturing control system, using and holon, agent and UML techniques. This method considers the development since initial conception until operation phase of the proposed control architecture.

[1]These *underlined terms are related to SOA technique.*.
[2]The terms related to PN are highlighted `this form`.

The IOPT extends the class of place-transition PN, it is a 12-tuple $(P, T, A, M, W, TA, WTest, priority, isg, ie, oe, osc)$ where: (i) the definition of P, T, A, M and W is the same of the place transition PN; (ii) TA is a set of `Test Arcs`, such that $TA \subseteq (PxT)$; (iii) $WTest$ is a `weight function` associated to `test arcs`, $WTest : TA \rightarrow \mathbb{N}^*$; (iv) $priority$ is a partial function applying non-negative integers to `transitions`, $priority : T \rightarrow \mathbb{N}^*$; (v) isg is an `input signal guard` partial function applying Boolean Expressions (BE) composed only by `input signals` (IS) to `transitions` , $isg : T \rightarrow BE$, where $\forall BE \in isg(T), Var(BE) \subseteq IS$; (vi) ie is an `input event` partial function applying `input events` (IE) to `transitions`, $ie : T \rightarrow IE$; (vii) oe is an `output event` partial function applying `output events` (OE) to `transitions`, $oe : T \rightarrow OE$; and (viii) osc is an output signal condition function from `places` into sets of rules, $osc : P \rightarrow P(RULES)$, where $RULES \subseteq (BESxOSxIN_o)$, where $BES \subseteq BE$ and $\forall e \in BES, Var(e) \subseteq ML$, with ML being the set of identifiers for each `place marking` after a given execution step. Each `place marking` has an associated identifier, which is used when executing the generated code.

The IOPT technique is adopted due its formalism based on input and output signals allows the description of logics to control devices and its web tool that allows edition and simulation of a control object in a clean and synthetic manner. Furthermore, the IOPT web tool (Gomes *et al.*, 2007) allows quasi-automatically the conversion of IOPT models in C code for programs of industrial controllers.

Modeling holons starts using PFS and successive refinements are applied to generate the models in IOPT. In this approach, a `place` represents a state while a `transition` represents an event or operation that conducts the process from one state to another. The register of messages are represented through PFS `distributors`. Each control object model includes normal and fault states (represented by `places`) and conditions (represented by `transitions`) for the change of states; these conditions are represented by logical of `guard expressions`. Thus, the *OpHs* models consider real component behaviour including influence of the environment and faults. The synchronization of the holons can be made with `auxiliary places` or by `input event`, `output event`, `guard expression` and `asynchronous channel`.

The IOPT tools allow three automatic code generation to allow transform IOPT models into programs in C, VHDL or JavaScript codes. The C code generator produces codes following ANSI C syntax rules into compressed archive containing the following source code files: (i) *net_types_h* that is the data-type definition and function declarations. It includes data-types for `places marking, input and output signals` and `events`; (ii) *net_main.c* that is main execution loop (main function); (iii) *net_io.c* that is the input and output code used do read the values of input signals from physical hardware inputs and write output signals to hardware; (iv) *net functions.c* that has the functions implementing all IOPT semantics and rules: `transition firing`, `guards, events`, `output actions`, etc.; (v) *net_exec_step.c* that has the function that executes one entire IOPT execution step, using the various functions defined in *net functions.c*; and (vi) *makefile* that is an optional Unix project makefile with instructions to build the project. All code produced should compile directly on most systems and should not need any changes, except the *net_io.c* file that must be adapted to each target application.

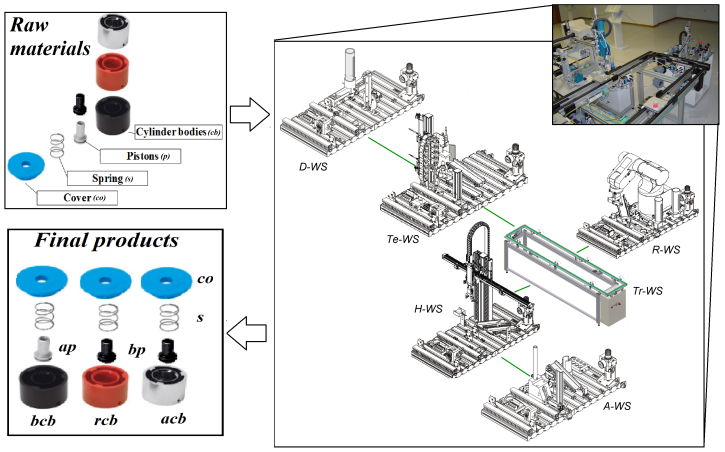
4. Application Example

In the following explanation, an application example of benchmark of reconfigurable manufactuting system is presented (Fig. 4). It is composed by six workstations (WSs) that represent autonomous subsystems: distributing workstation ($D-WS$), testing workstation ($Te-WS$), transporting workstation ($Tr-WS$), handling workstation ($H-WS$), assembly workstation ($A-WS$) and robot workstation ($R-WS$). Work orders (wos) can be executed on each WS which has its respective controller and operates independently as a stand-alone industrial plant. Each controller is connected to the internet allowing operators and clients interact. The aim of benchmark is automatically assembling products composed of workpieces (wps): a cylinder body (black [bcb], red [rcb] or aluminum [acb]), a piston (black [bp] or aluminum [ap]), a spring [s] and a cover [co]. The aluminum piston is assembled only on black cylinder body while black piston is assembled on red or aluminum cylinder bodies. Springs and covers are the same in all assemblies. Thus, the final products are: [bcb + ap + s + co]; [rcb + bp + s + co]; and [acb + bp + s + co]. Control devices, their control functions, commands and signals for actuation and detection are identified. The identification is made according to DIN/ISO 1219-2:1996-11 and IEC 61346-2:2000-12 standards.

In the center of Fig. 5 is the `[production plan]` holarchy that represents the *PrHs*. Each *PrH* represents a product and has an internal process flow, the required input types and output types. There are the intermediate *PrHs:* [bcb], [rcb], [acb], [ap], [bp], [s], [co], [bcb + ap], [rcb + bp], [acb + bp], [bcb + ap + s], [acb + bp + s], [rcb + bp + s] and there are three final *PrHs*: [bcb + ap + s + co], [rcb + bp + s + co] and [acb + bp + s + co].

The control specification in Fig. 5 is based on models of the holarchy `[business process]` that represents all the processes performed by the benchmark example. The `[supplier selection]` holarchy represents a customer looking for suppliers to provide products at an acceptable price and delivery time. If more than one supplier provides an offer, then the customer chooses one based on some criteria such as the lowest price. This figure also illustrates a control strategy named *[Request provides wp of D-WS to Te-WS]* which is proposed to control the D-WS service.

Figure 6 presents the detail in IOPT for the control strategy of *D-WS* illustrated in Fig. 5 using PFS. Each `transition` has signals represented by `guard expression` (G) which are related to devices listed in Fig. 4. For instance, the "Arm-



Ident.	Description
1A	Cylinder that removes the a cylinder body of the buffer
2A	Vacuum generator that takes the cylinder body
3A	Rotary actuator that transports a cylinder body to Te-WS
1B1	Magazine in front position
1B2	Magazine in back position
1B4	ON=no workpieces in the feeder
1Y1	Solenoid of the magazine cylinder
2B1	Sensor vacuum is on
2Y1	Solenoid switch on the vacuum
2Y2	Solenoid switch off the vacuum
3S1	Swivel drive at magazine
3S2	Swivel drive at the next station
3Y1	Solenoid swivel drive to magazine
3Y2	Solenoid swivel drive to next station
IP_FL	Light barrier to the following station

Figure 4. Manufacturing system, its raw materials, final products and example of devices list.

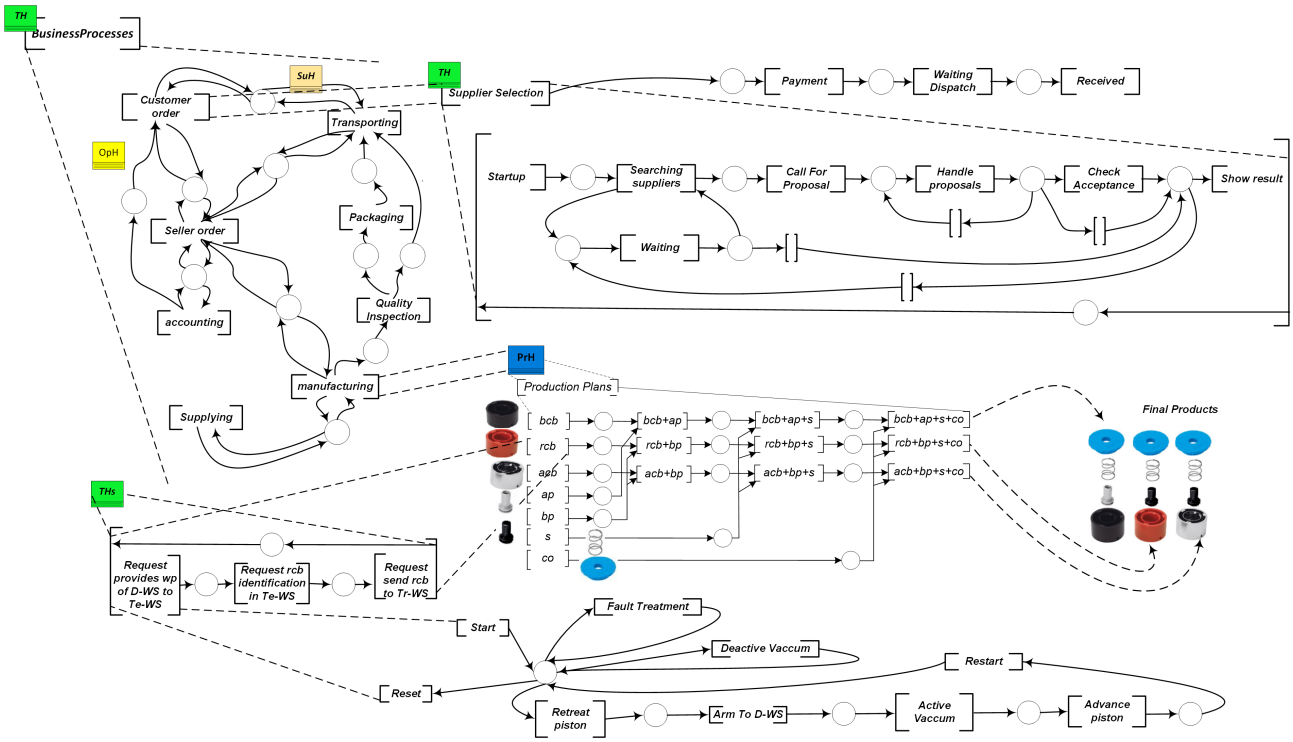


Figure 5. Business processes, the refinement of the customer order `activity`, production plan and a control strategy.

ToDWS" `transition` has associated the "G: 1B1=1 AND 1B2=0" that means "magazine is front position" and the "1B1 sensor" is sending signal on. Thus, a `token` can be fired for "ArmInDWS" `place` sending a `output signal` "3Y1=1" to modify the state of this solenoid.

Using the IOPT web tool, the model in the Fig. 6 is used to generate the programs to control D-WS and a snippet of the file net_exec_step.c for the `transition` StartDWS is as follows:

```
/* Transition 394 - StartDWS */
int t_394_enabled( empty_NetMarking* prev,
                   empty_NetMarking* avail )
{    return ( avail->p_185 >= 1 );
}
int t_394_events( empty_InputSignalEvents* events )
{    return ( events->StartOn );
}
int t_394_guards( empty_NetMarking* marking,
                  empty_InputSignals* inputs,
                  empty_PlaceOutputSignals* place_out,
                  empty_EventOutputSignals* ev_out )
{    return ( ev_out->_1B1 == 0 && ev_out->_1B2 == 1 && ev_out->_1B4 == 0 && ev_out->_2B1 == 0 && ev_out->
    _3S1 == 1 && ev_out->_3S2 == 0 && inputs->_4B2 == 1 );
}
void t_394_remove_marks( empty_NetMarking* marking )
{    marking->p_185--;
```

```
}
void t_394_add_marks( empty_NetMarking* marking )
{    marking->p_11++;
}
void t_394_generate_output_events( empty_OutputSignalEvents* ev_out )
{
}
```

The JAVA codes are generated using JADE plus WADE platform and a snippet for the negotiation among customers and suppliers illustrated in Fig. 5 using PFS is as follows:

```
private void defineTransitions() {
                        registerTransition(new Transition(), STARTUP_ACTIVITY,
                                SEARCHINGSUPPLIERS_ACTIVITY);
                        registerTransition(new Transition(),SEARCHINGSUPPLIERS_ACTIVITY,
                                CALLFORPROPOSAL_ACTIVITY);
                        registerTransition(new Transition(), CALLFORPROPOSAL_ACTIVITY,
                                HANDLEPROPOSALS_ACTIVITY);
                        registerTransition(new Transition(), HANDLEPROPOSALS_ACTIVITY,
                                HANDLEPROPOSALS_ACTIVITY);
                        registerTransition(new Transition (ALLPROPOSALSRECEIVED_CONDITION, this),
                            HANDLEPROPOSALS_ACTIVITY, CHECKACCEPTANCE_ACTIVITY);
                        registerTransition(new Transition (NOSUPPLIERAVAILABLE_CONDITION, this),
                            SEARCHINGSUPPLIERS_ACTIVITY, WAITING_ACTIVITY);
                        registerTransition(new Transition(), WAITING_ACTIVITY,
                                SEARCHINGSUPPLIERS_ACTIVITY);
                        registerTransition(new Transition(SUPPLIERCONTRACTED_CONDITION, this),
                            CHECKACCEPTANCE_ACTIVITY, SHOWRESULT_ACTIVITY);
                        registerTransition(new Transition(DEADLINEEXPIRED_CONDITION, this), WAITING_ACTIVITY
                            , SHOWRESULT_ACTIVITY);
                        registerTransition(new Transition(DEADLINEEXPIRED_CONDITION, this),
                            HANDLEPROPOSALS_ACTIVITY, SHOWRESULT_ACTIVITY);
                        registerTransition(new Transition(), CHECKACCEPTANCE_ACTIVITY,
                                WAITING_ACTIVITY);      }
```

Other scenarios was identified and models were built for each case. These models met the restrictions and achieve the objectives of the work. The implemented control system allowed the negotiation among holons and reconfiguration of different productive scenarios with safety and correctness during operation. The resulting control system also facilitated a systematic location of fault, comparing conditions and actions that need to be met in each service, with `transitions` and `places` of the models. Real-time monitoring system for supervision and control can be accomplished by synchronizing operation of IOPT models with sensor signals that represent the devices state. Therefore, operational advantages were demonstrated such as, better and more efficient use of manufacturing resources, speed of production and ability to deliver products faster.

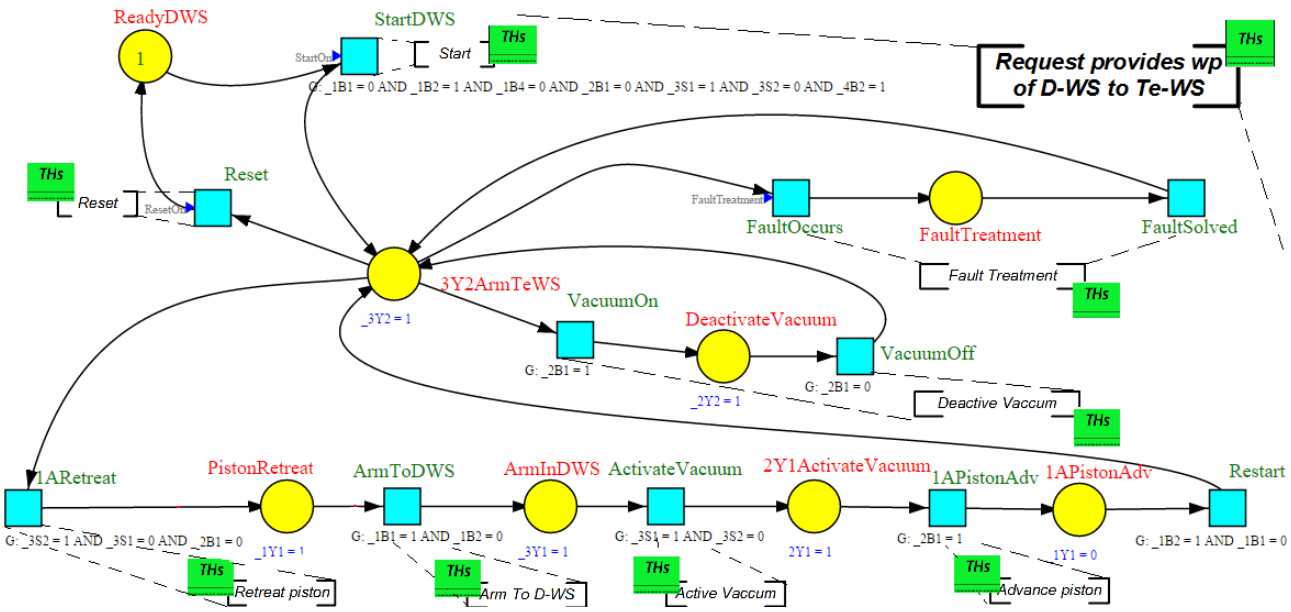


Figure 6. The IOPT model of the control strategy [Request provide wp of D-WS to Te-WS] presented in Fig. 5.

5. Conclusions

This paper presented a control architecture and a method to implement a Reconfigurable and Distributed Manufacturing Control System (RDMCS) combining Petri Net (PN), Service-Oriented Architecture (SOA), Holonic and Multi-Agent

System (HMAS) techniques and other good practices. The method and the control architecture combine bottom–up and top–down approaches using extensions of Petri Net (PN): the Input Output Place Transition (IOPT) PN for functional description and Production Flow Schema (PFS) for conceptual representation. This proposal presents special characteristics such as: (i) presenting all the components with certain autonomy and intelligence; (ii) representing the manufacturing control level in a flexible manner by holarchies concept; (iii) combining different modeling vision divided in modules based on product, processes or components; (iv) considering the cloud engineering allowing the collaboration of different design staff, using the Production Flow Schema technique and a IOPT web tool; (v) facilitating the implementation using HMAS platforms and IOPT web tool; and (vi) applying reconfiguration not only in the case of occurrence of faults but also to attend new demands, such as increasing production gain or number of final products. Further works involve the application in case studies of other industrial plants.

6. Acknowledgments

The authors would like to thank the partial financial support of the governmental agencies: CNPq, CAPES, FAPESP.

7. References

Béhé, F., Galland, S., Gaud, N., Nicolle, C. and Koukam, A., 2014. "An ontology-based metamodel for multiagent-based simulations". *Simulation Modelling Practice and Theory*, Vol. 40, pp. 64–85.

Bellifemine, F., Poggi, A. and Rimassa, G., 1999. "JADE–A FIPA-compliant agent framework". In *Proc. of PAAM 4th International Conference on Practical Application of Intelligent Agents and Multi-Agent Technology*. Vol. 99, pp. 97–108.

Caire, G., Gotta, D. and Banzi, M., 2008. "Wade: a software platform to develop mission critical applications exploiting agents and workflows". In *Proceedings of the 7th international joint conference on Autonomous agents and multiagent systems: industrial track*. International Foundation for Autonomous Agents and Multiagent Systems, pp. 29–36.

da Silva, R.M., Benítez-Pina, I.F., Blos, M.F., Santos Filho, D.J. and Miyagi, P.E., 2015a. "Modeling of reconfigurable distributed manufacturing control systems". In *Information Control Problems in Manufacturing*. Vol. 15, pp. 1348–1353.

da Silva, R.M., Blos, M.F., Junqueira, F., Santos Filho, D.J. and Miyagi, P.E., 2014. "A service-oriented and holonic control architecture to the reconfiguration of dispersed manufacturing systems". In *Technological Innovation for Collective Awareness Systems*, Springer Berlin Heidelberg, Vol. 423 of *IFIP Advances in Information and Communication Technology*, pp. 111–118.

da Silva, R.M., Watanabe, E.H., Blos, M.F., Junqueira, F., Santos Filho, D.J. and Miyagi, P.E., 2015b. "Modeling of mechanisms for reconfigurable and distributed manufacturing control system". In *Technological Innovation for Cloud-Based Engineering Systems*, Springer, pp. 93–100.

FIPA, A., 2002. "Foundation for intelligent physical agents". *http://www.fipa.org*.

Gomes, L., Barros, J., Costa, A. and Nunes, R., 2007. "The input-output place-transition Petri net class and associated tools". In *Industrial Informatics, 2007. INDIN'07. 2007 5th IEEE International Conference on*. IEEE, pp. 27–32.

Hasegawa, K., Miyagi, P.E., Santos Filho, D.J., Takahashi, K., Ma, L. and Sugisawa, M., 1999. "On resource arc for Petri net modelling of complex resource sharing system". *Journal of Intelligent and Robotic Systems*, Vol. 26, No. 3-4, pp. 423–437.

John, K.H. and Tiegelkamp, M., 2010. *IEC 61131-3: programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision-making aids*. Springer Science & Business Media.

Kolberg, D. and Zühlke, D., 2015. "Lean automation enabled by industry 4.0 technologies". In *Information Control Problems in Manufacturing*. Vol. 15, pp. 1919–1924.

Miyagi, P.E., Hasegawa, K. and Takahashi, K., 1988. "A programming language for discrete event production systems based on production flow schema and mark flow graph". *Trans. of the Society of Intrument and Control Engineers (SICE Japan)*, Vol. 24, No. 2, February, pp. 77–84.

Murata, T., 1989. "Petri nets: Properties, analysis and applications". *Proceedings of IEEE*, Vol. 77, No. 4, pp. 541–580.

Quintanilla, F.G., Cardin, O., L'Anton, A. and Castagna, P., 2015. "Implementation of a process orchestration model in a service oriented holonic manufacturing system". In *Information Control Problems in Manufacturing*. Vol. 15, pp. 1167–1172.

Vrba, P., Tichý, P., Mařík, V., Hall, K.H., Staron, R.J., Maturana, F.P. and Kadera, P., 2011. "Rockwell automation's holonic and multiagent control systems compendium". *Systems, Man, and Cybernetics, Part C: Applications and Reviews, IEEE Transactions on*, Vol. 41, No. 1, pp. 14–30.

8. Responsability Notice

The authors are the only responsible for the printed material included in this paper.