Proceedings of the Twenty-Eighth International Florida Artificial Intelligence Research Society Conference

An Ontology-Based Domain Representation for Plan-Based Controllers in a Reconfigurable Manufacturing System

Stefano Borgo, Amedeo Cesta, Andrea Orlandini
CNR – National Research Council of Italy
Institute for Cognitive Science and Technology
{name.surname}@istc.cnr.it

Alessandro Umbrico
Roma TRE University
Department of Engineering
alessandro.umbrico@uniroma3.it

Abstract

The paper describes a knowledge-based control loop as the key feature of a generic control architecture for nodes in a Reconfigurable Transportation Systems (RTSs). In particular, two main aspects are presented: (i) the design of an ontology-based representation of information related to both internal configurations/capabilities of a node and the active connections with its neighbors in the plant; (ii) the definition of a relationship between the ontology and the abstract model exploited by a temporal planning and execution module to implement control strategies of a single node. The main contribution is in proposing a connection between ontology-based representation and planning and execution model. The goal is to enable dynamic inference of control models so as to adapt control strategies to mutating shop-floor scenarios.

Introduction

A key enabling factor for highly automated production systems to compete in evolving production environments is related to their capability to quickly adapt (or even anticipate) changes in the production requirements (Wiendahl et al. 2007). Traditional plant control systems based on centralized/hierarchical control structures exhibit good performance in terms of productivity over a restricted and specific product range. However, these systems often require major overhauls of the control code in case any sort of system adaptation and reconfiguration needs to be implemented. These systems are not very efficient to face the current requirements of dynamic manufacturing systems (i.e., flexibility, expansibility, agility and re-configurability). For this reason, an increasing attention is being dedicated to Reconfigurable Manufacturing Systems (RMSs) (Koren et al. 1999), that are equipped with a set of enablers for reconfiguration that can be related either to the single system component (e.g., the mechatronic device) or to the entire production cell and system layout. The role of each enabler is to implement the correct system reconfiguration in response to changes of the production demand. Not surprisingly, AI based approaches have been considered among local enablers nevertheless the design of offline control models remains a crucial element in these approaches. Hence, structural modifications in the shop floor configuration usually entail a re-design of the control strategies that can be hardly manageable on line. Indeed the fast adaptation capabilities strongly require the equipment to be physically modular and its software to be knowledge-based so as to support high reconfigurability both from the mechanical standpoint and at reasoning level.

This paper focuses on a production environment composed by a community of generic control modules that encapsulate the physical mechatronic equipment, and introduces a knowledge-based control loop for the nodes of such a reconfigurable plant. More specifically it concentrates on two aspects: (i) the design of an ontology-based representation of the information related to both internal configurations/capabilities of a node and the active connections with its plant neighbors; (ii) the definition of a relationship between the ontology and the abstract model exploited by the temporal planning and execution module to implement intelligent control strategies of a single node. The use of ontologies to represent knowledge in control architecture is being increasingly popular in Robotics (e.g., (Hartanto and Hertzberg 2008; Tenorth and Beetz 2009)) and Manufacturing (e.g., (Balakirsky 2015; Solano, Rosado, and Romero 2013)) systems to enable autonomous agents to reason about the environment in which they act.

The main contribution is a particular connection between the ontological representation and the planning and execution model. The goal is to enable dynamic inference of control models so as to adapt control strategies to mutating shop-floor scenarios. The paper shows this connection in operation during two phases when the node controller know-how is adapted: (1) the *set up phase* in which the ontology provides the controller with the key information about the initial physical configuration of the specific node, i.e., the internal equipment and the associated functional capabilities, and (2) the *reconfiguration phase* in which the ontology is exploited to capture dynamic changes occurring either in the internal configurations, e.g., due to a failure of a component, or in the connections with other nodes in the RTS, e.g., given some due maintenance activities. The generic control architecture then takes advantage of such knowledge dynamically inferring and adapting the planning and execution model and, thus, (re)establishing a suitable control configuration both at startup time and after disruptions.

A Reference Domain

A pilot plant from an on-going research project called GECKO is used to elicit a case study over which we test the proposed Knowledge-Based Control Loop (in what follows KBCL). The plant is a Reconfigurable Manufacturing System for recycling Printed Circuit Boards (PCB). The pilot plant is composed by different automatic and manual machines devoted to perform loading/unloading, testing, repearing and shredding of PCBs and a conveyor system that connects them implemented through a Reconfigurable Transportation System (RTS). The RTS is composed of a set of reconfigurable mechatronic components, called *Transport Modules* (called TM in what follows). The goal of the system is to analyze defective PCBs, automatically diagnose their faults and, depending on the gravity of the malfunctions, attempt an automatic repair of the PCBs or send them to shredding. Since here the TM exemplifies our agent, now we briefly introduce the device. Each TM combines three Transportation Units (TUs). The units may be either unidirectional or bidirectional; specifically the bidirectional units enable the lateral movement (i.e., cross-transfers) between two TMs. Thus, each TM can support two main (straight) transfer services and one to many cross-transfer services. Different configurations can be deployed varying the number of cross-transfers components and enabling multiple I/O ports. TMs can be connected back to back to form a set of different conveyor layouts. The manufacturing process requires PCBs to be loaded on a fixturing system (pallet) in order to be transported and processed by the machines. The transportation system can move one or more pallets (i.e., a number of pallets can simultaneously traverse the system) and each pallet can be either empty or loaded with a PCB to be processed. At each point in time a pallet is associated to a given destination and the RTS allows for a number of possible routing solutions. The destination of a pallet carrying a PCB changes over time as the production process is carried out (e.g., the test station, the shredding station, the loading/unloading cell). The new destination is available only at execution time. Transport modules control system (Borgo et al. 2014) have to cooperate in order to define the paths the pallets have to follow to reach their destinations. These paths are to be computed at runtime, according to the actual status and the overall conditions of the shop floor, i.e., no static routes are used to move pallets. Moreover, each TM must be able to dynamically handle exogenous events by adapting its actions to the possible changing internal status and to that of its neighbors.

Knowledge-based Control Loop

In the software architecture used in the pilot plant, any component is wrapped with a software component that can be schematized with a layered control architecture (introduced in (Borgo et al. 2014)) and defined as the composition of three interacting layers. A *Coordination Layer* has been designed in order to constitute the architectural element responsible to make the single module able to interact with other modules as well as to participate in the distributed routing process for part routing dynamic policies management). A *Production Layer* is responsible for real-time continuous control by adapting the node's activities to the production needs. Given the requests provided by the Coordinantion Layer, an off-the-shelf temporal planning and execution system is deployed to synthesize and execute intelligent control strategies. Finally, a *Control Layer* is the composition of a *Control Software*, based on a distributed approach supported by an IEC61499 standard reference model, and a *Mechatronic Module* (in the specific case, either a Machine or a Transport Module).

The goal of this work is to enable dynamic inference of control models so as to adapt the intelligent control strategies while facing mutating shop-floor scenarios. In this regard, two main aspects have been investigated: (i) the design of an ontology-based representation of information that, beside the global shop floor information, can manage both the internal configurations/capabilities of a node and the active connections with its neighbors in the plant; (ii) the definition of a relationship between the ontology and the abstract model exploited by a temporal planning and execution module to implement intelligent control strategies of a single node. A conceptual schema of the integration of knowledge and control in the above generic architecture is depicted in Fig. 1. Specifically, it is the composition of three steps: (step 1) an *ontology-based abstraction* to represent the mechatronic device and to define an associated Knowledge Base (KB), (step 2) a *semantic mapping* applied to the KB to dynamically generate the control model of the system, (step 3) an off-the-shelf *planning and execution* system activated using such a model to synthesize a suitable set of actions (i.e., a timeline-based plan) that supports the production flow. Therefore, the *Knowledge-based Control Loop* is the process

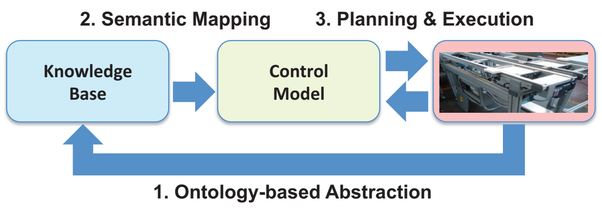

Figure 1: The Knowledge-based Control Loop

which allows the control architecture to dynamically represent the system's capabilities and automatically build the related timeline-based model description. It is then a continuous monitoring of the configuration of the plant dominating the usual closed-loop control implemented by the planning and execution system and keeping the KB up to date with respect to the real state of the system. Consequently, it also allows for the control model to be dynamically updated. In fact, whenever a reconfiguration of the mechatronic node is detected, the KB is updated and a new iteration of the loop starts over. Then, for each cycle, the planning and execution system can rely on a control model adherent to the actual operational conditions of the module.

Ontology abstraction

The agents must utilize a language rich enough to share information about the evolving environment. Two aspects are particularly important in this perspective. The first amounts to ensure the understanding of the types of information relevant for their (actual and expected) functioning in the system. This was achieved by applying ontological analysis to discover the Modules' information needs and setting a coherent classification of data into information types. The second aspect is the adoption of an articulated information structure according to which received information can be understood (depending on the agent's type) and new information distributed. In this regard, the use of an ontological approach ensures a reliable mechanism to dynamically generate a high-level description of the Module. Thus, an *Ontology-based Layer* controls also this language for *building* the KB and all the structural, performance and production information concerning the mechatronic device. From the control perspective, the *Production Layer* exploits the KB's information to dynamically infer the Module actual status and to build a model capturing what is required to safely/effectively control the Module. Therefore, every time the KB is updated, the control model is updated as well.

The GECKO Contexts

Since the KB collects information on several topics, like machine capacities, product classification and transportation time, the information classification system must be well articulated. To maintain maximal flexibility and adaptability, we relied on a general and domain-independent approach based on a set of contexts that we derived from the ontological analysis of the content of the information.

Contexts are devised according to ontological principles and are exploited to *model* the factory along two perspectives: classification of entity and data following the ontology, and possible roles of information types within a context. The ontological analysis of the shop floor's agents and processes, based on works like DOLCE (Borgo and Masolo 2009) and (Guarino and Welty 2009), led to identify relevant information flows and types of information potentially needed. The analysis, carried out from the viewpoint of a Module, led to separate three types of context: global, local and internal.

Global context. This context is about information the Module cannot control nor modify. In fact, since the Modules act in an integrated and coordinated shop floor, a common language allows to exchange organizational and production information. In this regard, we assume that a communication channel and shared protocol(s) to exchange data already exist at the shop floor. On top of this, the agents in the system must agree on a vocabulary (terms and relations) suitable for representing knowledge and for reasoning at the agent and shop floor levels. This language ensures they correctly understand how to recognize an identifier, what it means to receive a request or to perform an operation, etc. This context is then composed by three sub-contexts.

SUB-CONTEXT 1: FACTORY LANGUAGE. This context collects the language that all the Modules of the factory must use for public communications: vocabulary, rules and semantic constraints to describe functionalities; temporal and topological information for coordination; requests of actions and committed actions like shared plans, agenda etc. For example, the context provides the list of the capabilities in the shop floor, e.g., moving (an item), joining (two or more items), testing (input-output parameters), as well as their classification, e.g., carrying is a specialization of moving and welding of joining.

SUB-CONTEXT 2: FACTORY SHOP FLOOR. This context collects information on the elements (agents and products, including their identifiers) presently at the shop floor and the information exchanged for the production like requests for action and information on agents' availability. E.g., it lists pairs of an item and the requested action, e.g., (item #123, resistor impedance test), and pairs of a machine and its capability, e.g., (impedance test, machine #098). Note that the relationship between, e.g., resistor impedance test, impedance test and test is part of the factory language sub-context.

SUB-CONTEXT 3: FACTORY REGULATION AND PERFORMANCE. This context collects information related to general constraints, e.g., due to physical or technological requirements, production policies and safety regulations, as well as to the performance of cells or of the factory as a whole (productivity, consumption, throughput). Here we find data on average electricity usage and operational constraints like "impedance testing must be anticipated by voltage testing".

Local information context. This context is tuned to the agent type. Focusing on our Modules, each one is directly connected to other Modules or machines, these form its neighborhood. The information about active connections (local topology), coordination activities and commitments within the neighborhood form this context. In particular, this context keeps an updated list of the pairing (port$_i$, agent$_j$), for each port of the Module, and the agreed plan for each pair, e.g.: "receive item #123 via port$_1$ at time t_5", "deliver item #123 via port$_B$ at time t_8", possibly including time constraints and tolerance.

Module internal context. This context is dedicated to the information that an agent recovers about itself or generates by, e.g., self-diagnosis: identifier and status, components, actual capabilities (what it *can* possibly do) and, for each capacity, performance time and related information (e.g., it can continuously perform action a for at most 5 min; it can perform the action on pieces of size x, etc.), the possible change-overs, time needed to actuate a change, possible limitations in the changes, maintenance schedule, information on quality, partial/total malfunctioning of components, etc. For example when booting a Module will build a list that includes: identifier #123, ports port$_1$,...,port$_m$, engines (for cross-transfer, conveyor etc.), sensors and their tasks like (s_i, engine$_j$ left stopper) and (s_k, position cross-transfer$_l$) etc. In this way it can generate a list of capacities, e.g., (port$_1$, delivering) while the lack of (port$_1$, receiving), due by the fact that the related engine does not reverse the movement direction, indicates a partial malfunctioning on that port. By reasoning on these data and by updating them over time, the Module will be able to establish and make public at the shop floor which capacities it makes currently available.

The Ontology of Functions

An agent is itself a flexible and changing element of the environment; to correctly and reliably manage the changing functional capacities of the agents in the shop floor, we introduce an ontology of functions which, inspired by the foundational ontology DOLCE, gives an independent system for function classification. Here we primarily work with the functional taxonomy since this suffices to validate the system when the goal is restricted to match desired states with function plans and scheduling.

There is a large literature on function and function classification in engineering design and our ontology builds on three engineering approaches that have received much attention in the last ten to fifteen years, namely: the FOCUS/TX (Kitamura et al. 2011), the Functional Basis (FB) (Pahl et al. 2007), and further developed at NIST (Hirtz et al. 2001), and the Function Representation (FR) (Chandrasekaran and Josephson 2000). Following the vision of the GECKO project, we decided to build an ontology of functions based on the notion of "function as effect". That is, we analyze functions focusing on the effects they have on the operands (the entities they act upon) and independently of the actual implementation of the function. This level of generality allows us to distinguish "weld", "melt", "glue" etc. as ways to perform the (ontologically grounded) "join" function as suggested, e.g., in (Kitamura et al. 2011). Thus, our function classification has two components: the ontological component that contains functions like "join", "move" and "communicate", see Figure 2; and the implementation component (not discussed in this paper) which, for each ontological function, assigns a set of possible ways to execute it. Clearly, the ontology of functions should remain stable over time while the implementation component, which depends on the available technology, can be updated in real time (e.g. after the introduction of a new machine).

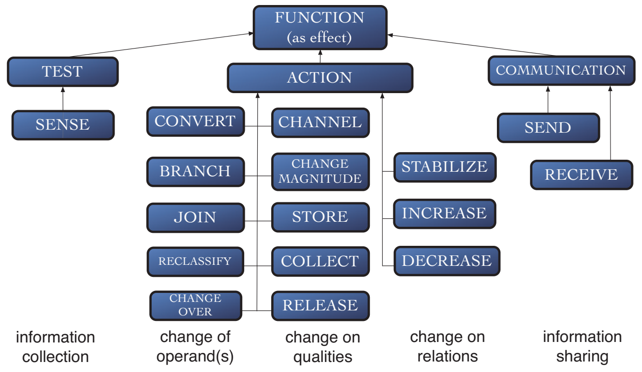

Figure 2: The ontological taxonomy of functions in GECKO and its rationale (bottom).

Figure 2 lists the ontological functions we are using in GECKO, among these "reclassify" stands for the function "to change the classification of an operand" as when changing the status of a PCB from malfunctioning to repaired after performing a test; "change-over" for the function "to change its own parameters" which occurs, e.g., when a Module acts on itself to activate/deactivate some component; "channel" for "to move an operand", that is, to change its location; "stabilize" for the function "to maintain relational parameters" like when tuning electronic components to regulate the input-output relationship; "sense" stands for the function "to test an operand", i.e., to acquire information without altering the status or qualities of the operand; finally, "send" stands for "to output information" like a signal that a PCB is going to be transferred to another Module or that a failure occurred.

The Semantic Mapping

From the control perspective, the objective is to exploit P&S technology in order to control complex systems acting in real world environments. In particular for planning and execution we rely on the timeline-based approach (Fratini, Pecora, and Cesta 2008) that allows to deal with the temporal aspects of the control problem (see also (Py, Rajan, and McGann 2010)). This technique aims at controlling the temporal evolution of the system in order to obtain desired behaviors. In Figure 1, the *Semantic Mapping* step aims at *bridging* the KB with timeline-based P&S model by exploiting semantics of KB data in order to extract relevant information and connection for the control model and automatically carrying out the modeling approach described above. Broadly speaking the *bridging process* consists of two phases: (i) a *first* phase allows to build the components of the control module by reading information from different contexts of the KB; (ii) a *second* phase defines the operating and temporal constraints (i.e. *synchronization rule*s in the control model) by combining data from different contexts of the KB. In general, the modeling approach follows a structured domain decomposition modeling approach as discussed for example in (Fratini, Pecora, and Cesta 2008). That decomposition is to identify a set of "relevant" features (system's components) that independently evolve over time. Then, a generic component is described by a set of values representing activities/states the system can perform/assume over time together with duration and transition constraints. Here, considering the contexts presented in the previous section, three classes of components have been identified: (i) *Functional*, (ii) *Primitive* and (iii) *External* components.

The *Internal Context* which describes the internal composition of the agent is suited to generate the *Primitive Components* of the timeline-based specification. Indeed, *Primitive Component*s provide a logical view of the elements composing the system to be controlled. Considering a Transport Module agent, the *Internal Context* contains information about the cross transfer engines, main and cross conveyor engines and other elements composing the module. The *bridging process* can build the related *Primitive Component*s of the timeline-based domain by exploiting this information. For instance, it builds a *Primitive Component* which models the main conveyor engine of the module. The values of this component represent the possible states of the conveyor: *forward()* if the conveyor is moving towards the *front* direction (F), *backward()* if the conveyor is moving towards the *back* direction (B) or *still()* if the conveyor is not moving. The *Local Context* contains the information needed to build the *External Component*s of the module. *External Component*s represent elements whose behaviour is not within the control of the agent but affects the execution of its functionalities. Namely, these components model some functional dependencies with the *environment* that make the agent able to perform its activities or not, e.g. the neighbor nodes of a transportation module. The values of these components represent the possible states of the neighbour agents: *Online()*, the node is working properly; *Maintenance()*, the node is performing a maintenance task and cannot support the production flow; *Malfuctioning()* if the node has an internal failure limiting its capabilities.

The *Functional Component*s provide a logical view of the system as a whole by modeling the high-level functionalities of an agent. They define the system in terms of what it can do in the environment independently of its internal details. The information required to build these components can be extracted by reasoning about information in the *Global Context* and the *Ontology of Function* of the KB. It is worth saying that it is also necessary to take into account data from the *Local Context* (e.g., a transport module endowed with three cross-transfer units is able to transport pallets towards any directions). However if the module T is connected with other two neighbor modules only, e.g., a module Tx on port F and a module Ty on port B, then the module T can transport pallets only towards the related directions (i.e., directions F and B). Namely, the actual capacity of the module to perform its functionalities is subject to the status of its ports and its neighbor agents.

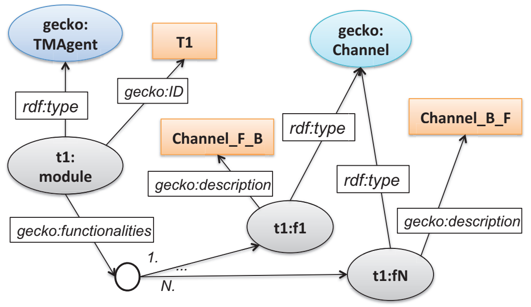

Figure 3: A partial view of KB's information

To exemplify the bridging process discussed above, the definition of the *functional component*s for the timeline-based model is briefly discussed and a portion of the KB representing information related to the functional capabilities of the transport module is depicted in Figure 3 (The notation is based on the RDF syntax). Figure 3 shows that the node is an agent of the type "TMAgent" (i.e. a Transportation Module agent) with ID "T1". The module is able to perform different "implementations" of the *Channel* functionality (w.r.t. taxonomy in Figure 2). For example, "T1" can perform a *Channel* functionality labelled "Channel_F_B" which corresponds to the capability of receiving a pallet from port "F" and transferring it towards the port "B". This example shows how to "interpret" KB's information exploiting its schema as the defined KB's schema (ontology and contexts) is the key feature here for providing data with semantics. Then, reasoning processes can exploit such semantics in order to interpret data and discover node capabilities. In Alg. 1, a simple procedure for building the functional components for the timeline-based model is summarized.

The procedure *buildFunctionalComponents()* accesses the KB in order to define the *Functional Component*s of the timeline-based domain. First, the procedure extracts the available functionalities in the KB w.r.t. the *Global Context* (row 3). For each functionality (rows 5-13) the procedure finds the functionalities actually implemented by the agent (row 7). Namely for each functionality in the ontology (e.g. *Channel*) the procedure extracts the available implementations (e.g. "Channel_F_B") by reasoning on the *Local Context* of the agent. If the module implements the functionality (rows 8-12) then the procedure creates a new *Functional Component* (row 10) by adding a distinct value (rows 11-12) for each available implementation of the related functionality, e.g. "Channel_F_B". Similarly, primitive and external components are defined by means of procedures omitted here due to space limitations.

Algorithm 1 Functional Component Builder Algorithm

```
 1: function BUILDFUNCTIONALCOMPONENTS($KB$)
 2:     // extract available functionalities
 3:     $fs \leftarrow getFunctionalities(KB)$
 4:     for all $f \in fs$ do
 5:         // get agent's implementations of the functionality
 6:         $impls \leftarrow getImplementations(KB, f)$
 7:         if $\neg Empty(impls)$ then
 8:             // create a component for the functionality
 9:             $sv \leftarrow createFunctionalComponent(f)$
10:             // add a value for each implementation
11:             for all $impl \in impls$ do
12:                 $addValue(sv, impl)$
13:             $add(svs, sv)$
14:     return $svs$
```

After components definition, it is necessary to further constrain their behaviors in order to coordinate components together by means of *synchronization rules*. These rules model causal and temporal constraints for the agent while performing complex task. Synchronization specification follows a hierarchical approach. Usually these rules map the high-level functionalities of the agent into a sequence of *primitive* activities enforcing a set of operational constraints that guarantee the safety of the overall system Namely synchronizations allow to specify *how* the functionalities, modeled by *Component*s, are internally executed by the system. The dynamic generation of rules requires to combine information in different contexts of the KB.

The Mapping in Action

Operationally, the bridging process presented above is exploited to adapt the control model of a generic node in the pilot plant during two phases: (1) the *set up* phase, to provide the controller with the key information about the initial physical configuration of the specific node, i.e., the internal equipment and the associated functional capabilities, and (2) the *reconfiguration* phase, dynamically adapting the control model while operating after a reconfiguration either in the internal configuration or in the connections with other nodes in the RTS. In general, when the functional capabil-

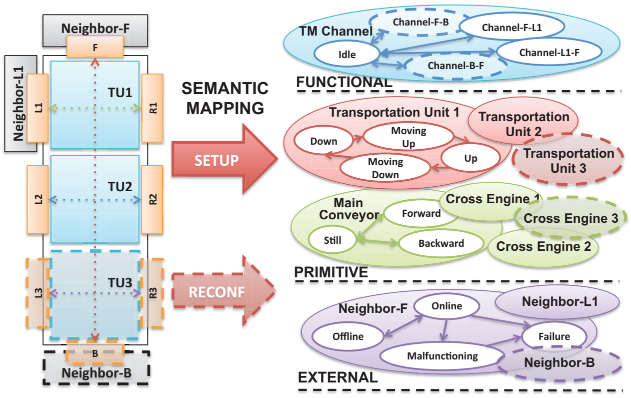

Figure 4: Conceptual schema of a specific TM (on the right) and associated control model (on the left). Components with dotted borders are removed in the model after a failure of TU3 unit.

ities of a node are modified during an operative period, a reconfiguration phase is required. The modifications of the node structure are detected and the control node has to update the KB and the information in the different contexts accordingly. Then, the generic control node takes advantage of the KB Control Loop to dynamically adapt the timeline-based model and to (re)establish a suitable control configuration. Let us consider a transport module depicted in left part of Fig. 4. An initial control model is generated as a result of the KB control loop after the setup phase (see the whole right part of Fig. 4). During operation, an internal HW failure occurs for the element "TU3" and the module becomes not able to perform *Channel* functionalities involving "TU3". Thus, the initial P&S model becomes obsolete and it is updated by removing the values related to functionalities no more available (e.g., "Channel_F_B" and "Channel_B_F") from the *Channel* functional component as well as the components related to the failing TU (see dotted components in Fig. 4).

Conclusions

This paper presented an integration step described as a "Knowledge-based Control Loop" that allows an agent architecture supervising mechatronic nodes of a RMS to represent the relevant knowledge about the plant and to map such knowledge in a domain description for a time-based planning and execution module. Before closing it is worth underscoring how, differently from the previous works, a knowledge-based control loop is envisaged aiming to implement a dynamic knowledge processing mechanism capable of allowing the automatic generation of adaptive behaviors when facing changes occurring in the actual system.

Acknowledgments. CNR authors are supported by MIUR/CNR under the GECKO Project (FdF-SP1-T2.1).

References

Balakirsky, S. 2015. Ontology based action planning and verification for agile manufacturing. *Robotics and Computer-Integrated Manufacturing* 33(0):21 – 28. Special Issue on Knowledge Driven Robotics and Manufacturing.

Borgo, S., and Masolo, C. 2009. Foundational choices in DOLCE. In Staab, S., and Studer, R., eds., *Handbook on Ontologies*, International Handbooks on Information Systems.

Borgo, S.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014. Towards a cooperative knowledge-based control architecture for a reconfigurable manufacturing plant. In *19th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*.

Chandrasekaran, B., and Josephson, J. 2000. Function in Device Representation. *Engineering with Computers* 16(3/4):162–177.

Fratini, S.; Pecora, F.; and Cesta, A. 2008. Unifying Planning and Scheduling as Timelines in a Component-Based Perspective. *Archives of Control Sciences* 18(2):231–271.

Guarino, N., and Welty, C. 2009. An overview on Ontoclean. In *Handbook on Ontologies*.

Hartanto, R., and Hertzberg, J. 2008. Fusing dl reasoning with htn planning. In Dengel, A.; Berns, K.; Breuel, T.; Bomarius, F.; and Roth-Berghofer, T., eds., *KI 2008: Advances in Artificial Intelligence*, volume 5243 of *LNCS*. Springer Berlin Heidelberg. 62–69.

Hirtz, J.; Stone, R. B.; McAdams, D. A.; Szykman, S.; and Wood, K. L. 2001. A functional basis for engineering design: Reconciling and evolving previous efforts. *Res. in En. Des.* 13(2):65–82.

Kitamura, Y.; Segawa, S.; Sasajima, M.; and Mizoguchi, R. 2011. An Ontology of Classification Criteria for Functional Taxonomies. In *IDETC/CIE*. ASME.

Koren, Y.; Heisel, U.; Jovane, F.; Moriwaki, T.; Pritschow, G.; Ulsoy, G.; and Brussel, H. V. 1999. Reconfigurable manufacturing systems. *CIRP Annals - Manufacturing Technology* 48(2).

Pahl, G.; Beitz, W.; Feldhusen, J.; and Grote, K. 2007. *Engineering Design. A Systematic Approach*. Springer.

Py, F.; Rajan, K.; and McGann, C. 2010. A systematic agent framework for situated autonomous systems. In *Int. Conf. on Autonomous Agents and Multiagent Systems (AAMAS)*.

Solano, L.; Rosado, P.; and Romero, F. 2013. Knowledge representation for product and processes development planning in collaborative environments. *International Journal of Computer Integrated Manufacturing* 27(8):787–801.

Tenorth, M., and Beetz, M. 2009. Knowrob - knowledge processing for autonomous personal robots. In *Intelligent Robots and Systems IROS 2009. IEEE/RSJ*, 4261–4266.

Wiendahl, H.-P.; ElMaraghy, H. A.; Nyhuis, P.; Zäh, M. F.; Wiendahl, H.-H.; Duffie, N.; and Brieke, M. 2007. Changeable manufacturing-classification, design and operation. *CIRP Annals-Manufacturing Technology* 56(2):783–809.