A Planning-based Architecture for 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 novel use of planning in Reconfigurable Manufacturing. Authors considered the nodes of a manufacturing plant as individual AI-based agents able to reason on continuously updated representation of their domain model, plan their own actions, and execute them. The paper aims at clarifying the role of planning, its connection with both a goal selection mechanism, and the agent's knowledge. It describes in detail how a planning system has been customized for the task of planning and execution and shows results of a realistic simulation on a manufacturing plant.

Introduction

The need of matching frequent product modifications and shorter product life cycles forces manufacturers to invest in competitive factors such as short lead time, reactivity to market frequent changes and cost effective production maintaining high quality of products. Reconfigurable Manufacturing Systems (RMSs) (Koren et al. 1999) represent a viable solution to competitively operate in such dynamism. They are equipped with a set of reconfigurable enablers (Koren and Shpitalni 2010) that can be related either to the single component of the system (e.g., a mechatronic device) or to the entire production cell and system layout (a transportation system or machines topology). The role of each enabler is to implement the correct system reconfiguration in response to changes of the production requirements.

The GECKO project proposes an adaptive control infrastructure for RMSs in which the production environment is modeled as a community of autonomous, self-declaring and collaborating GECKO nodes encapsulating the physical mechatronic equipments. The nodes coordination solution relies on auction-based techniques (Carpanzano et al. 2015) while the reconfigurability concept is applied to the control of both the logical and the physical aspects of the single nodes of the plant. Here, the focus is on *logical reconfigurations* of the control that may result from the need of adjusting the functionalities and control policies after a production change (e.g., new part type), an event affecting the physical equipment (e.g., anomalous behaviors) or a change in the production goals (e.g., minimization of energy consumption vs. minimization of idle times). Although AI based approaches have been considered as local enablers in some works (e.g., see (Crawford et al. 2013; Ruml, Do, and Fromherz 2005)), the design of control models that codify all the possible failures and possible changing situations remains a crucial problem for such systems. Moreover, relevant structural modifications in an agent configuration entail a re-design of the control strategies, a step generally hard to manage on the fly.

To address the above issue, a dedicated research initiative has been started, as described in (Borgo et al. 2014), to create a knowledge-based control architecture. (Borgo et al. 2015) describes an initial sketch of the knowledge and planning control loop. The current paper presents the concrete application of automated planning integrated with ontology based technology to address plant nodes reconfigurability at production level within the GECKO agent architecture. The proposed solution is the combination of a knowledge management module and a timeline-based planning and execution system in which each GECKO node gathers the information from the environment, reproduces an abstraction of the shop-floor and interprets the production dynamics. This potentially allows to evaluate, configure and tune the GECKO nodes capabilities by automatically activating the control functions and implementing the needed temporal plans to pursue the current goals. In other words, the control system is a wrapping agent that implements the basic sense-plan-act cycle around the mechatronic part of the plant node enriched with a knowledge-base in the loop that guarantees flexibility with respect to a number of unexpected events, enabling the possibility of regenerating the planning domain specification when needed. The main goal here is to demonstrate how modular physical equipments together with knowledge-based software can better support high reconfigurability at the logical (i.e., reasoning) level. A key aspect of this specific application is the wide range of situations the model can capture and the ability to cope with them without stopping the plant but adapting the current working system through the interplay between knowledge representation and planning though the recomputing of new planning domains and problems. Experimental results collected during tests performed on the GECKO pilot plant show the practical feasibility of the integrated solution when facing increasingly complex instances of a real-world manufacturing case study.

Paper organization. The next section introduces the targeted manufacturing domain, while the subsequent one presents a general agent architecture that integrates a knowledge manager and a planning system in closed loop. Then the ontological approach, key aspect of the knowledge manager, and the knowledge processing mechanism are described showing how they are operationalized in a specific knowledge-based framework. Furthermore, the mapping that generates a planning model from the declarative knowledge description is presented. The complete agent at work in an RMS scenario is described and a complete experimental evaluation of the knowledge management processes presented that shows the feasibility of the control approach. Some conclusions end the paper.

The Manufacturing Case Study

In the paper, the pilot plant from the GECKO project is exploited to elicit a case study to test the proposed integrated solutions and to assess the planning capabilities. The target plant aims at recycling Printed Circuit Boards (PCBs). It is composed of different machines for loading/unloading, testing, repairing and shredding of PCBs and of a conveyor system that connects them. The conveyor is implemented through a Reconfigurable Transportation System (RTS) composed of a set of reconfigurable mechatronic components, called *Transportation Modules* (TM), see Figure 1. The goal of the plant is to analyze defective PCBs, to automatically diagnose their faults and, depending on the type of the malfunctions, attempt an automatic repair or send them to waste.

The proposed agent architecture is wrapped around each of the TMs hence its functionalities are introduced with more details. Each of the TMs combines three Transportation Units (TUs). The units may be unidirectional or bidirectional, with bidirectional units enabling also movements from side to side (cross-transfers) from/to other TMs, see Figure 1. The TM can support two main transfer services, forward and backward, and zero to many cross-transfer services. Different configurations can be deployed varying the number of cross-transfers components and thus enabling multiple I/O ports. TMs can be connected back to back to form a set of different conveyor layouts.

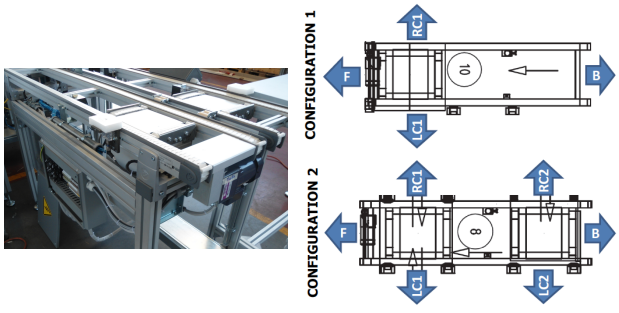

Figure 1: A picture of a Transportation Module (on the left) of the RTS and two possible configurations (on the right).

The manufacturing process requires each PCB to be loaded on a fixturing system (a pallet) in order to be transported by the TMs and processed by the various machines of the RMS. 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. At each point in time a pallet is associated with a given destination and the RTS allows for a number of possible routing solutions. The next destination of a pallet carrying a PCB can change over time as operations are executed (e.g., by the test station, the shredding station, the loading/unloading cell). The new destination is available only at execution time.

The GECKO proposal was to realize a distributed control infrastructure composed by a *community* of autonomous agents (Borgo et al. 2014) able to cooperate in order to define the paths the pallets must follow to reach their destinations. Thus, 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. The decisions of the coordination algorithm (see (Carpanzano et al. 2015) for further details) acts as goal injection for the planning mechanism of each agent. Hence according to our pursued abstraction, the plant is a set of TMs endowed with independent capabilities to carry on their goals, by analyzing the current situation, synthesizing a planning domain and problem, then planning and executing the plan for such goals. It is worth observing that a plan-based controller can endow an agent with the desired autonomy (i.e., deliberative capabilities), but given the particular dynamic nature of RTSs it does not guarantee a continuous control process capable to face all the particular situations/configurations. Indeed it is not easy (or hardly possible) to capture all the dynamics of the production environment in a unique planning domain. The specific capabilities of a TM in the RTS are affected by many factors, e.g., a partial failure of the internal elements of a TM, a reconfiguration of the RTS plant or maintenance activities of some TMs of the plant. Thus, it is not always possible to design a plan-based controller which is able to efficiently handle all these situations. The higher is the complexity of the planning domain the higher is the time needed to synthesize the plans and the latency of the control architecture must be compatible with the latency of the plant.

The key direction in GECKO project has been the one of endowing the control architecture of an agent with a knowledge-reasoning mechanism capable of representing the actual capabilities of the related TM of the plant. This allows to adapt the planning model specification by considering only the actual capabilities of the TM to control. In particular the knowledge reasoning mechanism allows to realize a continuous control process by dynamically generating an updated planning model every time a change in the capabilities of the TM or in the production environment is detected.

Knowledge and Controller in a Loop

The key integration of distinct cognitive functions in the agent architecture is shown in Figure 2. At a higher abstraction, the figure shows the integration of two “big boxes”

called here "Knowledge Manager", that contains the know-how of the agent, and "Deliberative Controller" that represents the plan-based control architecture – *a-la* (Lemai and Ingrand 2004) to set the stage. To make the whole idea operational we need to open the boxes and describe what is needed to allow the two functionalities to work together.

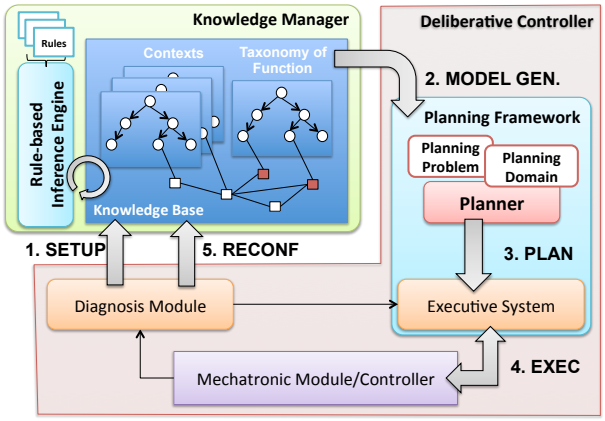

Figure 2: The Knowledge-based Control Loop.

Following a careful analysis of the reasoning needs, the **Knowledge Manager** relies on a suited ontology which models the general knowledge of manufacturing environments. The ontology contains (i) a classification of relevant information in three distinct *Contexts* – namely *Global*, *Local* and *Internal* (see later) – and (ii) a *Taxonomy of Functions* which classifies the set of functions the agents can perform according to their effects in the environment (see later).

The Knowledge Manager exploits the ontology to build and manage the Knowledge Base (KB) of the particular agent to control. The KB represents an abstract description of the structure and the capabilities of the agent and also of the production environment (from the agent's point of view). Namely, the KB represents the "instantiation" of the general knowledge to the particular agent to control. In this context the Rule-based Inference Engine is a specific module which is responsible for processing KB information by inferring additional knowledge about the functional capabilities the agent is actually able to perform (see later for further details). Thus, given a TM of the RTS of the case study, the KB contains information concerning the devices that compose the TM (e.g. the cross transfers, the conveyor engines, the port sensors), the set of other TMs and/or working machines directly connected (i.e. the set of collaborators) and information concerning the whole production environment from the agent perspective (e.g. the topology of the shop floor). Then the Inference Engine analyzes the structure of the TM and its collaborators in order to add to the KB information about the set of transportation functions the TM is actually able to perform.

The **Planning Framework** provides deliberative features relying on the planning model generated from the KB to actually control the mechatronic device. More specifically, it is a wrapper of the planning and execution system employed (i.e., a timeline-based system in this work) to provide deliberative capabilities. It is responsible to integrate KB information with Planning by automatically generating the model of the mechatronic device to control. Indeed, planning domain and problem specifications are dynamically generated from the KB and an off-the-shelf planning and execution system is activated to synthesize signals for the actuators that control the mechatronic device. It is important to point out that the generation process encodes our modeling approach and that the ontology classification of the KB's information determines a particular and useful structure to the (timeline-based) planning domain as described later.

The **Mechatronic Module** is the composition of a *Control Software* and a *Mechatronic Component* (e.g., a transportation module, a working machine, etc.). In our case the control software is based on standard reference models (e.g., IEC61499) and each mechatronic component is then represented by dedicated hardware/software resources encapsulating the module control logic.

In what follows we call *Knowledge-based Control Loop (KBCL)* the overall process which allows to integrate the elements described above in a unique control infrastructure. Thus, the resulting control process enables an agent both to dynamically represent its capabilities, the detected environmental situation and to infer the set of available functions on which a coherent planning model is generated.

The KBCL Process at Runtime

The management of the KB, the generation of the planning domain and the continuous monitoring of the information representing the actual status of the agent and the environment are the rather complex activities that are properly managed by the KBCL process at runtime. In this regard the KBCL process is organized in the following phases: (i) the *setup phase*; (ii) the *model generation phase*; (iii) the *plan and execution phase*; (iv) the *reconfiguration phase*.

The *setup phase* generates the KB of the agent by processing the raw data received from the *Mechatronic device* through a *Diagnosis Module*. The resulting KB completely describes the structure of the particular module to control, the set of TMs the module can cooperate with and the set of functions the module is actually able to perform in order to support the production flow. Then, the *model generation phase* exploits the KB of the agent to generate the timeline-based planning model the *Deliberative Controller* needs to actually control the device.

When the planning domain is ready the *planning and execution phase* starts, and the *Deliberative Controller* continuously builds and executes plans. During this phase the KBCL process behaves like classical plan-based control systems. The *Planning Framework* builds the plan according to some tasks to perform. *Soft changes* in the plan execution are directly managed by the *Deliberative Controller*, e.g., temporal delay of some planned activities. Conversely whenever the *Diagnosis Module* detects a structural change of the agent and/or of its collaborators e.g., a total or partial failure of a cross transfer of the TM to control (i.e., *Hard changes*), the *reconfiguration phase* starts.

The reconfiguration phase determines a new iteration of the KBCL process cycle. The KB of the agent is updated by detecting the new state of the mechatronic device and its production environment as well as inferring the updated set of functions the TM can perform according to the new state. As before, once the KB of the agent is complete, the planning model of the *Deliberative Controller* is also updated w.r.t. the new state of the module. It is worth observing that the KB and the planning model are updated only when structural changes that impede the execution of the plan are detected.

The Knowledge Manager

In a changing environment the agents must coherently share information relevant to the tasks. We thus used an ontological analysis to build reliable information systems that exploit different information types and contexts. The result is a general mechanism to *dynamically generate* a high-level description of agent's capabilities and system's situations.

Extending DOLCE Ontology

The Contexts. The KB collects information like capabilities, other agents' types and action constraints. Starting from foundational works (Borgo and Masolo 2009; Guarino and Welty 2009), we introduced three contexts to classify the agent's knowledge: *Global* (knowledge on the environment), *Local* (knowledge on connected collaborators), *Internal* (knowledge on itself). The contexts help to find the relevant information depending on the agent's aims.

Global context. Since the agents act in a complex environment, we need to ensure that they reliably exchange organizational and production information. On top of basic communication channels and protocol(s), the agents must agree on a vocabulary (terms and relations) to communicate and reason on information at the local and global views: the vocabulary ensures the meaning of an identifier, a request for bidding, an operation, and so on. SUB-CONTEXT 1: FACTORY LANGUAGE. It contains the high-level language (vocabulary, rules, semantic constraints etc.) used by the agents to share information about functionalities like moving (an item), joining (two or more items), testing (input-output parameters) and their relationships, e.g., carrying is a specialization of moving and welding of joining, as well as information about timing, requests for actions, and committed plans. SUB-CONTEXT 2: FACTORY SHOP FLOOR. It contains information on agents, products, tools in the factory, and information exchanged like requests for action or availability, e.g. toDo(item #123, resistor impedance test) states what needs to be done to item #123 and hasFunctionality(impedance test, machine #M98) that machine #M98 can perform impedance tests. SUB-CONTEXT 3: FACTORY REGULATION AND PERFORMANCE. It contains information on general constraints (e.g., production policies, safety regulations, preconditions for operations) and on the performance of cells and the factory as a whole (e.g., productivity, consumption, throughput).

Local context. This context is specific to an agent and depends on its type. It collects information about neighbor agents, active connections, coordinated activities and commitments. It maintains an updated list of the physical connections (port_i, agent_j) and agreed plans, e.g.: "receive from port_1 at time t_5", possibly including time constraints and tolerance.

Internal context. It collects information about the agent itself: identifier, components, capacities and related information (e.g., time to start an action, size of processable pieces etc.), possible change-overs, maintenance schedule etc. For example, at booting the agent states its identifier, its ports, engines (for cross-transfer, conveyor etc.), sensors and their tasks like s_i is the engine_j's left stopper. This gives a list of capacities that are verified at run-time by self-diagnosis. E.g., if engine_j cannot reverse direction the updated context will contain (port_1, delivering) but not (port_1, receiving). By comparing knowledge of previous states, the agent can identify (partial) malfunctioning and actual capabilities.

The Ontology of Functions

An agent is itself a functional element that potentially changes in time. To manage changes in the agent's capacities, we use an ontology of functions whose main task is to enable executable combinations (plans) of available functions given the agent's goals. Note that at this stage we primarily work with the function taxonomy since this suffices from the perspective of the agent.

There is a large literature on function and function classification in engineering design. Our ontology builds on three approaches: the FOCUS/TX (Kitamura et al. 2011), the Functional Basis (FB) (Pahl et al. 2007), and the Function Representation (FR) (Chandrasekaran and Josephson 2000). In particular, 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" and "glue" as ways to perform the (ontological) "join" function (Kitamura et al. 2011). Our function classification has two components: the ontological component contains functions like "join", "move" and "communicate" Figure 3; the implementation component (not discussed here) assigns a set of possible ways to execute an ontological function. Note that the ontological component remains stable over time while the implementation component is updated whenever the agent's capabilities change.

Figure 3 shows the ontological functions where, for instance, "reclassify" stands for the function "change the classification of an operand" as when changing the status of a product after performing a test; "change-over" for "change its own parameters" which occurs, e.g., when an agent acts on itself to activate/deactivate some component; "channel" for "move an operand", that is, to change its location; "stabilize" for "maintain relational parameters" like when regulating the input-output relationship in electronic components; "sense" for "test an operand", i.e., acquiring information without altering the operand; finally, "send" for "output information" like when sending a signal that a failure occurred.

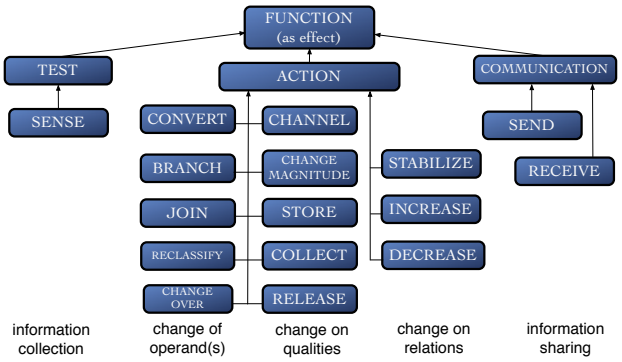

Figure 3: The function ontological taxonomy and its rationale.

The Knowledge Base lifecycle

The management of the KB of the agent, briefly described above while introducing the Knowledge Manager, relies on a knowledge processing mechanism implemented by means of a *Rule-based Inference Engine* (see Figure 2). The *Inference Engine* exploits a suited set of inference rules in order to process the KB of the agent. The knowledge processing mechanism classifies data received from the *Diagnosis Module* and infers additional knowledge concerning the functional capabilities of the agent. This mechanism involves two reasoning steps as sketched in Figure 4, i.e., (i) a *low-level* reasoning step and (ii) a *high-level* reasoning step.

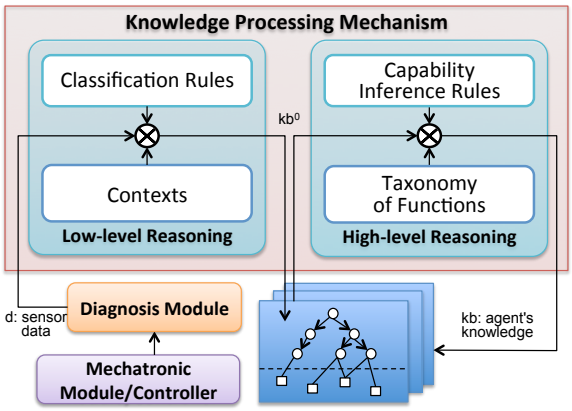

Figure 4: The knowledge processing mechanism

The *low-level* reasoning step builds an initial version of the KB by classifying sensor data on the basis of Context categorization. Thus, this first step initializes the KB by adding a suited set of individuals representing the devices that compose the agent and its collaborators. Given a TM in the RTS, the KB resulting from the *low-level* reasoning step will contain information concerning the cross-transfers, the conveyor engines and the ports that compose the module, their status, their internal connections (i.e., the internal topology of the module) and the set of the TMs connected through its ports. Namely, the resulting KB describes the internal and local contexts of the agent (i.e., the components and collaborators of the agent).

The *high-level* reasoning step extends the KB elicited at the previous step to infer the capabilities the agent is actually able to use on the basis of its current status and the current production environment. Furthermore, also causal/temporal relationships existing among such operands are inferred and represented as relationships among instances in the KB. For instance, let us consider a TM "T1" composed by a port "F" which connects this module to the module "T2" of the plant and a port "B" which connects the module to the module "T3" of the plant and that port "F" and port "B" are internally connected by a conveyor. In such a case the initial KB will contains information about components port "F", port "B" and the conveyor, their connections and information about collaborators "T2" and "T3". Then the *high-level* reasoning step will infer that "T1" is able to perform two channel functions, one channel goes from port "F" to port "B" and one goes from port "B" to port "F".

When the two reasoning processes end, the resulting KB represents all the information needed to create suited models for the planning system exploited by the *Planning Framework*.

Deliberative Control with Timelines

The *Planning Framework* element in Figure 2 provides the KBCL process with deliberative capabilities by exploiting a timeline-based planner (Umbrico, Orlandini, and Cialdea Mayer 2015). The planner relies on a timeline-based planning model automatically generated from KB's information. Before describing the details of the process which generates the planning domain, this section provides a brief description of the timeline-based planning and the pursued modeling approach.

Timeline-based Planning in a Nutshell. Timeline-based approach to planning has been introduced in early 90s (see for instance (Muscettola 1994)) and takes inspiration from the classical control theory. It models a complex system by identifying a set of relevant features that must be controlled over time. This approach has been successfully applied to real world contexts (especially in space applications) and several planning frameworks have been developed for the synthesis of timeline-based P&S applications, e.g., EUROPA (Barreiro et al. 2012), ASPEN (Chien et al. 2010), and APSI-TRF (Cesta et al. 2009).

Broadly speaking timeline-based applications aims at controlling a complex system by synthesizing temporal behaviors of its features in shape of timelines. A timeline consists of a sequence of states/actions the related domain feature (e.g., a component of the device to control) must assume/perform over time. Every value on a timeline is temporally allocated and represents the value/action the feature assumes/perform during the related temporal interval. *Temporal flexibility* allows to allocate values to flexible temporal intervals, i.e. intervals with flexible start time and end time. The resulting timeline represents an envelop of possible temporal evolutions of the related feature. Thus a timeline-based plan, which consists of the union of all the timelines of the domain, represents the sets of all possible temporal evolu-

tions of the domain features. It is important to point out that the temporal flexibility in such a plan can be exploited at execution time by an executive system to gain robustness (see e.g., (Py, Rajan, and McGann 2010)).

Modeling Approach. The features of the system to be controlled are modeled by means of *State Variables*. A state variable describes the temporal behaviors of a specific feature by means of causal and temporal constraints. More specifically, it describes the values the feature can assume over time, with their duration constraints and allowed transitions. In this regard, the ontological analysis of the functional capabilities and the structure of agents, described in previous sections, give us a relevant contribution for the definition of a modeling methodology of timeline-based planning domains. The key idea is that the planning domain is to describe the functional capabilities of the system we want to control, the features of the elements that compose the system and the features of the working environment that must be taken into account in order to successfully carry out system's functions. Indeed, three different classes of state variables can be identified as relevant from the control perspective: (i) *functional* state variables; (ii) *primitive* state variables; (iii) *external* state variables. The *Functional state variables* model the system as a whole in terms of the functions it can perform (notwithstanding its internal structure). The *Primitive state variables* model the physical and/or logical elements that compose the system. In particular these state variables model the elements we must actually control to execute system functions. The *External state variables* model elements of the domain whose behavior is not directly under the control of the system. Namely these variables model conditions that must hold in order to successfully perform system's functions.

The behavior of state variables must be further constrained by specifying inter-component causal and temporal requirements, called *synchronization rules*. These rules specify additional constraints that allow to coordinate the behaviors of the domain features in order to perform some desired complex tasks (i.e., planning goals). Following a hierarchical approach, synchronization rules map the high-level functional values into a set of constraints among primitive and/or external values that guarantee the proper functioning of the overall system and its elements. Namely synchronization rules specify how high-level tasks are implemented by the system. Thus these rules describe dependencies between the functional state variables and the primitive/external state variables determining a hierarchy among them.

The Model Generation Process

Key role for the dialogue between the *Knowledge Manager* and the *Deliberative Controller* is the *Model Generation* process (Step 2 in Figure 2). The KB of the *Knowledge Manager* provides an abstract representation of the capabilities, the structure and production environment of the agent as described above. The generation process analyzes the KB to generate a timeline-based planning domain of the agent. The process encodes the hierarchical modeling methodology described in the previous section. Algorithm 1 describes the pseudo-code of the main procedure for the overall model generation procedure[1].

Algorithm 1 Procedure generating the planning model.

1: **function** BUILDCONTROLMODEL(KB)
2: // extract agent's information and initialize the model
3: $agent \leftarrow getAgentInformation(KB)$
4: $model \leftarrow init(KB, agent)$
5: // define components of the model
6: $svs \leftarrow buildFunctionalComponents(KB, agent)$
7: $svs \leftarrow buildInternalComponents(KB, agent)$
8: $svs \leftarrow buildExternalComponents(KB, agent)$
9: // define synchronization constraints
10: $s \leftarrow buildSynchronizations(KB, agent)$
11: // update planning model
12: update(model,svs,s)
13: **return** $model$
14: **end function**

The procedure exploits the KB information concerning the *inferred* functional capabilities of the agent in order to build the functional state variables of the model (row 6). Then, the procedures exploits KB information concerning the internal and local contexts of the agent to build the primitive and external state variables respectively (rows 7-8). Finally, the model is completed by generating the synchronization rules (row 10) that describe how the module can perform its functions. Then, the process generates a timeline-based model like the one depicted in Figure 5, which represents a TM with only one cross-transfer unit available. In general, a TM can perform several transportation activities (i.e., *channel functions* according to the taxonomy in Figure 3) to move pallets on the plant. Thus, the model contains a functional state variable (the *TM-Channel*) which models the set of channel functions the TM is actually able to perform (e.g., *Channel-F-B* or *Channel-F-R*) depending on its particular configuration and the neighbors available.

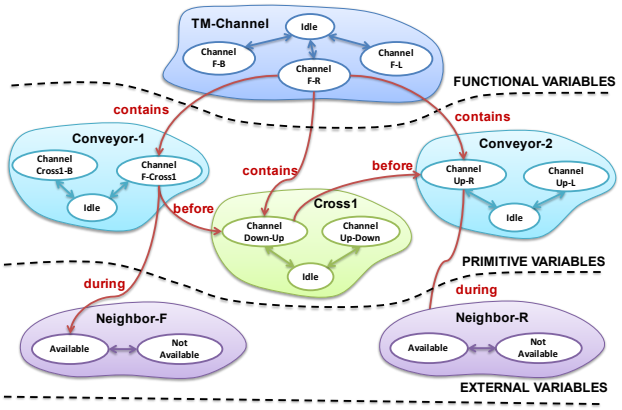

Figure 5: A (partial) view of the timeline-based model generated for a transportation module equipped with a cross transfer unit.

[1]A more detailed description of the whole model generation process may be found here: https://db.tt/FobSCp4c. The exploited ontology with examples for KB and generated planning model is available here: https://db.tt/u2Z1Nw90

The primitive variables of the model represent the internal devices composing the TM, such as the conveyors (*Conveyor-1* and *Conveyor-2*) and the cross-transfer unit (*Cross1*). These variables model the capabilities of the TM in terms of the *primitive functions* (i.e., the actions/commands) the module can directly execute. For example, *Conveyor-1* can perform a *primitive channel* to move a pallet from *port-F* to *Cross-1* (*Channel-F-Cross1*). The external variables of the model represent other modules of the plant the TM can directly collaborate with (e.g., *Neighbor-F* or *Neighbor-B*). A channel function of the TM can be considered as a *compound capability* which correlates the TM with other modules of the plant. In this regard, a *channel function* is carried out by means of a set of *primitive channels*. The arrows in Figure 5 represents the temporal constraints entailed by a synchronization rule which allows the TM to transport a pallet from *Neighbor-F* to *Neighbor-R*, i.e., the function *Channel-F-R*. Namely, the synchronization rule specifies the set of *primitive functions* with the related temporal constraints the TM must follow to perform a particular channel function. Thus, the generated timeline-based planning model provides a functional characterization of the system we want to control. The planning goals represent the functions the agent may perform. These functions are described in terms of the *atomic* operations (i.e., primitive functions) the agent can perform by means of its components and its collaborators (i.e., the module's neighbors).

The Knowledge-Based Control Loop in Action

This section reports on a set of tests on the KBCL with different TM configurations. All the different physical configurations of a TM have been considered, from zero to three cross-transfer modules. These configurations are referred to as *simple*, *single*, *double* and *full*, respectively. A different configuration also entails a different number of connected TM neighbors. Clearly, the more complex scenario is the one with the highest number of cross-transfers (the full TM) and neighbors. Also, three reconfiguration scenarios (*reconf-a, reconf-b* and *reconf-c*) have been developed considering different external events, namely an increasing number (from 1 to 3) of TM neighbors momentarily unable to exchange pallets, plus two scenarios related to internal failures (*reconf-d* and *reconf-e*) due to a cross-transfer engine failure and to a local failure on a specific port.

The experiments were carried out to evaluate the performance of the following aspects of a TM: (i) the knowledge processing mechanism; (ii) the planning model generation; (iii) the synthesis of plans to manage a set of pallet requests. The final aim is to evaluate the feasibility of the KBCL approach by showing that the performance are compatible with execution latencies of the RMS[2].

Figure 6 shows the timings in the Setup phase for the KBCL module operation, i.e., to build the KB exploiting the classification and capability inference process (left-hand side of Figure 6), and to generate the timeline-based planning specification for the TM (right-hand side of Figure 6). On the one hand, the results show that an increase in the complexity of the TM configurations does not entail a degeneration of the knowledge processing mechanism: the inference costs are almost constant (around 1.3 secs). This behavior was expected since the number of instances/relationships in the KB is rather low notwithstanding the physical configuration of the TM; thus, the performance of the inference engine deployed here is not particularly affected. On the other hand, the model generation is linearly affected spanning from 0.8 secs in the simple configuration, up to a maximum of 2.2 seconds in the full configuration. The model generation process entails a combinatorial effect on the number of instances/relationships needed to generate components and synchronizations leading to larger planning models and, thus, to higher process costs. When a reconfiguration scenario occurs, the knowledge processing costs are negligible. Among all the considered reconfiguration cases (i.e., reconf-a-b-c-d-e), the time spent by the knowledge processing mechanism to (re)infer the enabled functionalities is just a few milliseconds. In fact, both the classification and capability inference steps are applied to a slightly changed KB and, then, minimal changes in the functionalities can be quickly inferred in the system and represented in the new KB. For what concern the planning model generation after a reconfiguration, each reconfiguration scenario (both external and internal) leads to a strong reduction of functionalities and, thus, the related costs are relatively small. For instance, in the case of the full TM configuration, the cost for model generation is not greater than 0.8 seconds.

Figure 6: KB initial inference and planning domain generation.

Finally, we evaluate the planning costs when facing both setup and reconfiguration scenarios. Figure 7 shows the trend of the planning time in the Setup scenario considering all the TM configurations and an increasing number of pallet requests (randomly generated), i.e., planning goals, to be fulfilled. Planning costs span from few seconds up to nearly 30 seconds when planning for 10 pallet requests within a 15 minutes time horizon. In general, the more complex the planning model, the harder the plan synthesis problem. Thus, the planning costs follow the complexity of the configurations of the specific TM agent.

Discussion. The experimental results show the practical feasibility of the KBCL approach in increasingly complex

[2] All the experiments have been performed on a workstation endowed with an Intel Core2 Duo 2.26GHz and 8GB RAM.

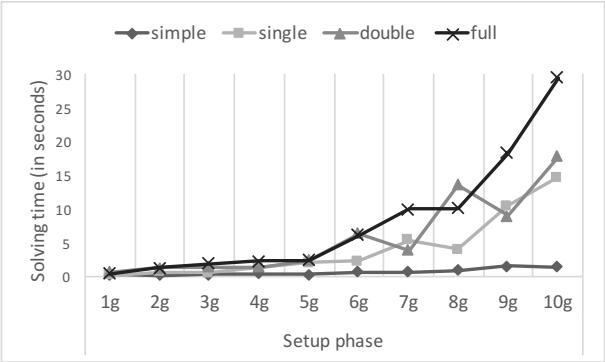

Figure 7: Timings for planning an increasing number of goals for different TM configurations during Setup phase.

instances of a real-world manufacturing case study. The collected data for the initialization (or the update) of a generic agent's KB (considering both knowledge processing and model generation) and the cost for planning synthesis have a low impact on its performance during operation. In fact, in order to face production periods of 15 minutes –and the management of 10 pallet requests– no more than 5 seconds are required by the Knowledge Manager while less than 30 seconds are required by the Planner to generate a suitable plan. Such performance are compatible with system latency usually involved in this type of manufacturing applications. It is worth reminding how the role of the KBCL is to avoid major overhauls of the control policies (e.g., control code revisions deployed too often in concrete cases) to cope with adaptation to variations or plant reconfigurations.

Related Works

Ontology for information management and planning specification have been exploited in Manufacturing and in Robotics to enable agents to reason about their environment and to optimize their activities. In this section we provide a brief discussion of some of the most relevant works in the literature w.r.t. to our work. In manufacturing, ontologies have been applied to increase flexibility in modeling and planning of, e.g., mechatronic devices (Balakirsky 2015), resources in collaborative environments (Solano, Rosado, and Romero 2013) and to manage information of distinct types (Hristoskova et al. 2013). In all these cases, planning specifications, if considered, are fixed and neither automatically generated nor subsequently adapted. This shows that the flexibility provided by ontological representation is still underestimated in the domain. In robotics, ontologies have been more widely exploited. An ontology-based knowledge framework (OMRKF) (Suh et al. 2007) was exploited to infer knowledge and to clarify missing or uncertain data produced by noisy sensors. The used ontology is rich and shows the potentialities for information control. Unfortunately, the organization is not good for functional reasoning, e.g., finding and turning are both functions but are in different classes. In our work we focus on the control elements in the deliberative layer (where planning capabilities come into play) and need a careful organization of the functional information. (Hartanto and Hertzberg 2008) uses Description Logic to represent the agent knowledge and the environment is explicitly modeled. Filters are used to extract the part of the environment representation needed for a given planning problem. Basically, this means to simplify a fixed planning specification. Our goal, instead, is to produce the environment model by exploiting actual data to dynamically generate planning models that are optimized for that agent in that environment. Tools like (Bell et al. 2013) represent offline and static environments for authoring models, while our work aims at implementing an on-line control loop. Finally, other frameworks that use ontology in robotics, like KnowRob (Tenorth and Beetz 2015) and ORO (Lemaignan et al. 2010), are not directly comparable to our work since there ontologies are aimed to obtain an action-based knowledge representation for cognitive functionalities such as domain features learning, symbol grounding, etc. We conclude that our approach makes a relevant step forward to pursue the increased flexibility needed in RMSs

Conclusions

This paper describes a novel planning-based application for the manufacturing domain. In particular we started from observing that a crucial need in reconfigurable manufacturing is the flexibility with respect to keeping the pace with changes in the execution environment without stopping the plant when updating the different processes. Given the opportunity of the GECKO project we have addressed the problem with a solution that integrates knowledge reasoning and plan-based control. The time constants demonstrated by the experiments are in line with those of the real plant we are working at hence the path for a real use has been fully paved. In solving our problem we have also contributed with a different use of ontological representation and reasoning jointly with planning with respect to previous proposals (as described in the related works section). Generalizing the KBCL idea to other domains is one of the directions for future work.

Before ending, it is worth observing explicitly that the particular structure of the transportation module somehow "helps" (even "simplifies") our work with respect to a general case of using the KBCL with any robotic device (e.g., a mobile platform, a human-robot interaction scenario). Here the claimed contribution is with respect to the novel application area in manufacturing (i.e., RMSs) that is extremely relevant and extends what already done in (Ruml, Do, and Fromherz 2005) to an area in continuous expansion. Those authors extended the use of planning, and other AI techniques, in a complex machine. Here we have designed a decentralized use of planning in the nodes of our reconfigurable transportation network, focusing on the continuous need of re-formulating the plan domain description.

Acknowledgments. CNR authors are supported by MIUR/CNR under the GECKO Project (FdF-SP1-T2.1). We would also thank the anonymous ICAPS reviewers for valuable comments.

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.

Barreiro, J.; Boyce, M.; Do, M.; Frank, J.; Iatauro, M.; Kichkaylo, T.; Morris, P.; Ong, J.; Remolina, E.; Smith, T.; and Smith, D. 2012. EUROPA: A Platform for AI Planning, Scheduling, Constraint Programming, and Optimization. In *ICKEPS 2012: the 4th Int. Competition on Knowledge Engineering for Planning and Scheduling*.

Bell, S.; Bonasso, R. P.; Boddy, M.; Kortenkamp, D.; and Schreckenghost, D. 2013. PRONTOE - A case study for developing ontologies for operations. In *Proc. of the Int. Conf. on Knowledge Engineering and Ontology Development (KEOD-13).*, 17–25.

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. Springer.

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 2014)*. IEEE.

Borgo, S.; Cesta, A.; Orlandini, A.; and Umbrico, A. 2015. An ontology-based domain representation for plan-based controllers in a reconfigurable manufacturing system. In *Proc. of the 28th Florida Artificial Intelligence Research Society Conference (FLAIRS-15)*.

Carpanzano, E.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; Umbrico, A.; and Valente, A. 2015. Design and implementation of a distributed part-routing algorithm for reconfigurable transportation systems. *International Journal of Computer Integrated Manufacturing*.

Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2009. Developing an End-to-End Planning Application from a Timeline Representation Framework. In *IAAI-09. Proc. of the 21st Innovative Application of Artificial Intelligence Conference, Pasadena, CA, USA*.

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

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*.

Crawford, L. S.; Do, M. B.; Ruml, W.; Hindi, H.; Eldershaw, C.; Zhou, R.; Kuhn, L.; Fromherz, M. P.; Biegelsen, D.; de Kleer, J.; et al. 2013. Online reconfigurable machines. *AI Magazine* 34(3):73–88.

Guarino, N., and Welty, C. 2009. An overview on Ontoclean. In Staab, S., and Studer, R., eds., *Handbook on Ontologies*. Springer.

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 *Lecture Notes in Computer Science*. Springer. 62–69.

Hristoskova, A.; Aguero, E. C.; Veloso, M.; and De Turck, F. 2013. Heterogeneous Context-Aware Robots Providing a Personalized Building Tour. *Int. J. of Advanced Robotic Systems*.

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

Koren, Y., and Shpitalni, M. 2010. Design of reconfigurable manufacturing systems. *Journal of Manufacturing Systems* 29(4):130 – 141.

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).

Lemai, S., and Ingrand, F. 2004. Interleaving Temporal Planning and Execution in Robotics Domains. In *AAAI-04*, 617–622.

Lemaignan, S.; Ros, R.; Mosenlechner, L.; Alami, R.; and Beetz, M. 2010. ORO, a knowledge management platform for cognitive architectures in robotics. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, 3548–3553.

Muscettola, N. 1994. HSTS: Integrating Planning and Scheduling. In Zweben, M. and Fox, M.S., ed., *Intelligent Scheduling*. Morgan Kauffmann.

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 *AAMAS-10. Proc. of the 9th Int. Conf. on Autonomous Agents and Multiagent Systems*.

Ruml, W.; Do, M. B.; and Fromherz, M. P. 2005. On-line planning and scheduling for high-speed manufacturing. In *ICAPS-05. Proc. 15th Int. Conf. on Automated Planning and Scheduling*, 30–39.

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.

Suh, I. H.; Lim, G. H.; Hwang, W.; Suh, H.; Choi, J.-H.; and Park, Y.-T. 2007. Ontology-based multi-layered robot knowledge framework (OMRKF) for robot intelligence. In *IEEE/RSJ Int. Conf. on Intelligent Robots and Systems, 2007. IROS 2007.*, 429–436.

Tenorth, M., and Beetz, M. 2015. Representations for robot knowledge in the KnowRob framework. *Artificial Intelligence –*. (http://dx.doi.org/10.1016/j.artint.2015.05.010).

Umbrico, A.; Orlandini, A.; and Cialdea Mayer, M. 2015. Enriching a temporal planner with resources and a hierarchy-based heuristic. In *AI*IA 2015, Advances in Artificial Intelligence*. Springer. 410–423.