Planning with Flexible Timelines in the Real World

Alessandro Umbrico, Marta Cialdea Mayer
University Roma TRE, Italy
alessandro.umbrico@uniroma3.it

Andrea Orlandini
CNR - National Research Council of Italy
Institute of Cognitive Science and Technology

Abstract

Planning is a core field of Artificial Intelligence since its beginnings. Broadly speaking, planning techniques aim at providing artificial agents with the capability to autonomously solve "complex" problems. Several planning techniques have been introduced in the literature that employ different approaches for modeling and solving planning problems.

My PhD research activities concern timeline-based approach to planning. Timeline-based planning is a particular Temporal Planning paradigm which has been successfully applied to solve real-world problems. Despite it practical success there is not a shared view of this planning approach. There are several timeline-based frameworks that have been introduced in the literature each of which applies its own *interpretation* of timelines, timeline-based plans and planning domains.

In this regard the objective of my PhD is to analyze the features of the different existing timeline-based systems and to provide a complete characterization of timeline-based approach by providing a semantics for the related planning concepts, defining a methodology to model domains and problems and by defining domain independent heuristics and develop a suited timeline-based planning framework, called EPSL.

Introduction

Timeline-based planning has been introduced in early 90s (Muscettola 1994), it 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 in several real-world contexts (especially in space applications) thanks to take into account the temporal aspects of the problem. 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), APSI-TRF (Cesta et al. 2009) or IXTET (Laborie and Ghallab 1995).

Despite its practical success, there is a lack of formalization of timeline-based planning related concepts. There is not a uniform and shared view of concepts like timelines, timeline-based plans, domains and problems. Every framework applies its own *interpretation* of timeline-based planning. This results in different ways of considering timeline-based problems and also, different ways of modeling and solving such problems.

In this context, my PhD research goal is to characterize timeline-based planning approach from different point of views and develop a suited planning framework, called EPSL - the Extensible Planning and Scheduling Library. Namely, we aim at understanding timeline-based planning by providing an acceptable semantics for the related planning concepts, defining a methodology to model and solve problems by means of timelines.

The following sections introduce timeline-based planning approach by exploiting the formalization we have proposed in some recent works (Cialdea Mayer, Orlandini, and Umbrico 2015; 2014) and the EPSL planning framework that I'm currently developing. Later sections describe an interesting application of EPSL (and the timeline-based approach) to a manufacturing real-world context for the development of a knowledge-based control module (KBCL - the Knowledge-Based Control Loop). In particular, this application gave an important contribution for the definition of a hierarchical modeling approach for timeline-based planning and a domain independent heuristic that we have implemented and tested in the EPSL framework.

Finally we briefly present some ongoing works on the comparison of our "vision" of timeline-based planning and EPSL with EUROPA which is one of the most known timeline-based software environment in the literature.

Timeline-based Planning Approach

The main result concerning the formalization of timeline-based planning approach is represented by our work (Cialdea Mayer, Orlandini, and Umbrico 2015). The key contribution of this work, which builds and extends previous works (Cialdea Mayer, Orlandini, and Umbrico 2014), is to provide a formal stand-alone definition of the main concepts of timeline-based planning and the relative *controllability properties*, independently from the concrete structure exploited to represent timelines.

The importance of this feature is due to the fact that representing a flexible timeline-based plan as a *temporal network* entails a sort of simplification of the associated plan structure, thus causing a loss of information on the causal/temporal "dependencies" among its components. Indeed, such information can be useful for planning engines

(for instance to define suited heuristics as we'll see in the next sections) and in general, for supporting a more detailed analysis of the relevant features enclosing in the generated plans.

The timeline-based approach pursues the idea that planning and scheduling for controlling complex physical systems consists of the synthesis of desired temporal behaviors (i.e. *timelines*). Thus, a *planning domain* is modeled as a set of features with an associated set of temporal functions on a finite set of values.

The time-varying features of the domain can be modeled by means of *multi-valued state variables*. The possible evolutions of these features are described by some causal laws and limited by domain constraints. These are specified in a *domain specification*. A timeline-based planner must find a sequence of decisions that brings the timelines into a final desired set, satisfying the domain specification and goals. Thus, a domain specification must provide the set of causal and temporal constraints that specify which value transitions are allowed for the state variables, and the minimal and maximal duration of each valued interval.

State variables A state variable models a particular feature of the domain that must be controlled over time. Formally it is characterized by four components: the set V of values representing the possible state or actions the feature can assume or perform over time; a function T mapping each value $v \in V$ to the set of values that are allowed to follow v; a function γ tagging each value $v \in V$ with information about its controllability; a function D setting upper and lower bounds on the duration of each value of the variable. In particular, the *controllability tagging fucntion* γ tags each value $v \in V$ as controllable $\gamma(v) = c$ or not $\gamma(v) = u$. If a value v is *controllable* it means that the planner (or the executor of the plan) can decide the actual duration of the value. If a value v is *uncontrollable*, instead, the planner can decide its start time but the planner cannot decide its end time. Namely, the planner cannot make any hypothesis about the actual duration of uncontrollable values during the solving process.

Synchronization rules A domain specification must provide global constraints that coordinate the temporal behaviors of the state variables. Such relations are specified by means of *synchronization rules* that constrain values of different state variables. Namely, synchronization rules specify how the domain features must behave in order to perform some complex tasks.

Formally, a synchronization rule is an expression of the form:

$$a_0[x_0 = v_0] \rightarrow \exists\, a_1[x_1 = v_1] \ldots a_n[x_n = v_n]\, .\mathcal{C}$$

where (i) $a_0, \ldots, a_n$ are distinct *token variables*; (ii) for all $i = 0, \ldots, n$, x_i is a state variable and $v_i \in values(x_i)$; and (iii) $\mathcal{C}$ is a positive boolean formula where only the token variables $a_0, \ldots, a_n$ occur.

Token variables represents particular instances of values of the domain state variables. It is important to point out that the use of token variables allows to specify multiple instances of the same value of a state variable in the right-hand part of a synchronization rule. The left-hand part of the synchronization rule, $a_0[x_0 = v_0]$, is called the *trigger* of the rule and represents the value the rule can be applied to.

Timelines A timeline represents the temporal evolution of a system component up to a given time (the *horizon*). It is made up of a sequence of valued intervals, called *tokens*, each of which represents a time slot in which the variable assumes a given value. It is important to point out that, when planning with timelines, time flexibility is taken into account by allowing token durations to range within given bounds. A token for a variable $x = (V, T, \gamma, D)$ is completely described by representing its start and end "times" with temporal intervals as follows:

$$x^i = (v, [e, e'], [d, d'], \gamma(v))$$

Thus a timeline FTL_x is a for a state variable $x = (V, T, \gamma, D)$ is a finite sequence of tokens of the form:

$$FTL_x = x^1 = (v_1, [e_1, e'_1], [d_1, d'_1], \gamma(v_1)), ..., x^k = (v_k, [e_k, e'_k], [d_k, d'_k], \gamma v_k)$$

It is important to point out that once a token x^i is embedded in a timeline, the time interval to which its start points belongs can be easily computed by considering the end time of the previous token, $start\_time(x^{i+1}) = end\_time(x^i)$ (where the start time of the first token x^0 is $[0, 0]$).

A *scheduled timeline* is a particular case where each token has a fixed end time $[t, t]$. A *schedule* of a timeline FTL_x is essentially obtained from FTL_x by narrowing down to singleton (i.e. time points) the end times of the tokens. The schedule of a token corresponds to one of the valued intervals it represents which is obtained by choosing an exact end point in the allowed interval without changing its duration bounds. In this regards, a scheduled timeline is a sequence of scheduled tokens that satisfy their duration bounds.

Flexible plans The main component of a flexible plan is a set of timelines representing different sets of scheduled ones. It may be the case that not every scheduled timelines satisfy the synchronization rules of the domain. In order to guarantee that every set of scheduled timelines represented by a given flexible plan π (i.e. the different ways of executing π) is valid with respect to the underlying planning domain, the plan has to be equipped with additional information about the temporal relations that have to hold in order to satisfy the synchronization rules of the domain. Namely, the representation of a flexible plan must also include information about the relations that must hold between tokens in order to satisfy the synchronization rules of the planning domain.

In general, a flexible plan includes a set of temporal constraints ($\mathcal{R}$) on tokens $\pi = (\mathbf{FTL}, \mathcal{R})$. When there are different ways to satisfy a synchronization rule by the same set **FTL** of flexible timelines, there are also different (valid) flexible plans with the same set of timelines **FTL**; each of them represents a different way to satisfy synchronizations.

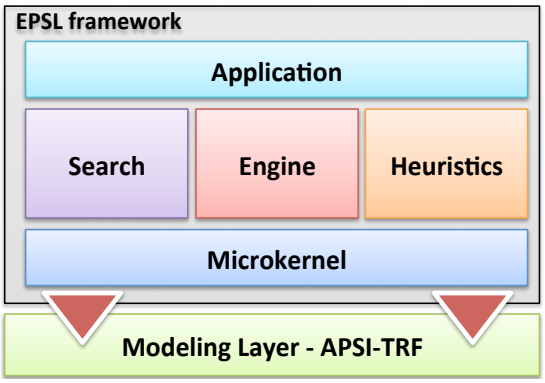

Figure 1: EPSL architecture

The Extensible Planning and Scheduling Library

Timeline-based applications, typically, are strictly related to the specific domain they have been designed for. It is hard to exploit "past experience" in order to adapt already developed applications to different context with different features. Thus, it is usually start developing new applications from scratch.

In this regard, the EPSL (the Extensible Planning and Scheduling Library) (Umbrico, Orlandini, and Cialdea Mayer 2015) is a layered software framework (built on top of APSI-TRF (Cesta and Fratini 2008)) which aims at defining a flexible software environment for supporting the design and development of timeline-based applications. The key point of EPSL is its interpretation of a planner as a *modular* solver which combines together several elements to carry out its solving process.

Figure 1 shows the main architectural elements of the EPSL architecture. In particular, the *Engine Layer* is the architectural element responsible for managing the portfolio of algorithms (called *resolves*) a planner can use to actually solve timeline-based problems. The higher is the number of available resolver the higher is the solving capability of the framework. Indeed, the solving process of an EPSL-based planner consists of a plan refinement procedure which iteratively refines the plan by solving "undesired" conditions, called *flaws*. A flaw represents a particular condition which threats the completion or the consistency of the current plan (e.g. a planning goal). Every resolver is responsible for detecting and solving a particular type of flaw. The set of available resolvers determines what an EPSL-based planner can actually do to solve a problem. Similarly, the *Heuristic Layer* is the architectural element responsible for managing the set of available criteria an EPSL-based planner can use to select flaws when solving.

Thus, the EPSL solving approach can be "easily" adapted to the particular problem to address by changing the set of resolvers and heuristics the planner can use (i.e. the planner configuration). As a matter of fact, the particular strategy or heuristics applied can strongly affect the behavior and the performance of a planner.

Knowledge-Based Control Loop

The KBCL is a knowledge-based control module developed within the GECKO project I have collaborated to during my PhD. The reader may refer to several works (Stefano et al. to appear; Borgo et al. 2015; 2014a; Carpanzano et al. 2015; Borgo et al. 2014b) for a detailed description of the approach and the specific application context.

The key direction of the GECKO project has been to endow the control architecture of an agent with a knowledge reasoning mechanism capable of representing the actual capabilities of the related agent. Thus we made a tight integration between knowledge representation techniques with and timeline-based planning by exploiting the EPSL framework.

In general, a plan-based controller can endow an agent with the deliberative capabilities needed to autonomously perform complex tasks. In particular contexts like the RTS (Reconfigurable Transportation System) of the manufacturing case study we have considered within the GECKO project, the dynamic nature of the system we want to control does not guarantee a continuous control process capable to face all the particular situations/configurations.

Indeed, the specific capabilities of a Transportation Module (TM) of the RTS can be affected by many factors, e.g. a partial failure of the internal elements or a reconfiguration of an RTS plant (see cited works for further details). Thus it is not always possible to 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.

Our proposed solution was to "extend" the control loop of an agent with a knowledge-reasoning mechanism capable of representing its actual capabilities. In this way, it is possible to simplify the planning model specification by considering only the actual capabilities of the agent (e.g. a TM of the RTS) to control. In particular, the knowledge reasoning mechanism allows to realize a continuous control process by dynamically synthesizing the timeline-based planning model every time a change in the capabilities of the agent (i.e. the TM) occurs.

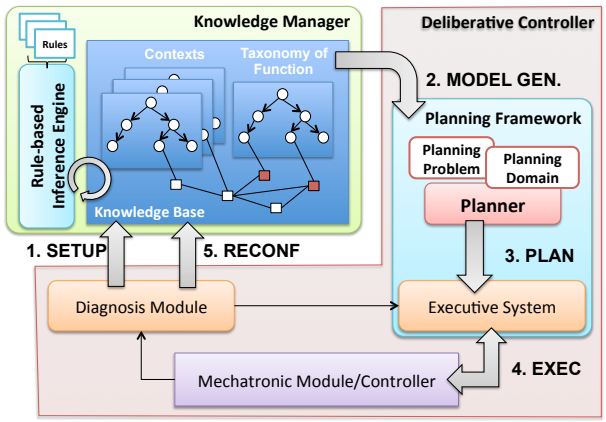

Figure 2: The KBCL module architecture

Figure 2 shows the extended control loop which results

from the integration of two ”big boxes”. The *Knowledge Manager* which contains the knowledge about the agent to control, and the *Deliberative Controller* which represents the ”classical” plan-based control architecture where EPSL is integrated in an execution environment for the synthesis and execution of timeline-based plans.

The ontological approach The Knowledge Manager of Figure 2 relies on a suited ontology which captures the general knowledge of the manufacturing environment. The ontology contains (i) a context-based classification of the information about the agent and production environment, and (ii) a taxonomy of function which classifies the set of functions a generic agent can perform in a manufacturing environment.

The Knowledge Manager exploits the ontology to build and manage the Knowledge Base (KB) of the specific agent (i.e. TM) to control. In particular the context-based approach classifies information according to three contexts that characterize the agent from three different point of views. The *internal context* characterizes the internal structure of the agent and its components. The *local context* characterizes environment in terms of other agents or elements of the production environment the agent must interact with. The *global context* contains information of interest for all the agent composing the shop floor, e.g. the type of product to work or information about the performance of the factory.

The KBCL in action The management of the KB, the generation of the planning domain and the continous monitoring of the information concerning the actual status of the agent (and its environment) are complex activities that must be properly managed by the KBCL process at runtime. In this regard the KBCL is composed by the following phases (depicted in Figure 2: (i) the *setup* phase; (ii) the *model generation* phase; (iii) the *plan and execution* phase; (iv) the *reconfiguration* phase.

Broadly speaking, the *setup phase* generates the KB of the agent by processing the raw data received by the agent (a TM in the GECKO project) through the *Diagnosis Module* of Figure 2. The resulting KB completely describes the agent to control in terms of its structure, its capabilities and the related production environment. The *model generation* phase exploits the KB of the agent to automatically generate the timeline-based planning domain needed by the *Deliberative Controller* to actually control the device.

When the planning model has been generated the *plan and execution* phase starts. The KBCL process behaves like a classical plan-based controller during this phase. However, whenever a structural changes occurs in the agent or its environment e.g. a failure of an internal component or a failure of a collaboratore of the shop floor, the *reconfiguration* phase starts. The *reconfiguration* phase determines a new iteration of the KBCL cycle and a new version of the KB and a new version of the timeline-based planning model are generated.

A Hierarchy-based Modeling Appraoch

When applying timeline-based approach usually, we must control an ”artificial agent” able to perform some complex tasks in a specific working environment e.g. a TM of the GECKO project. In order to provide a suited timeline-based model it is necessary to capture all the features, the operational and temporal constraints that characterize a specific domain. In this regards, exploiting the context-base analysis described in the previous section to characterize the knowledge about the functional capabilities of an agent, the modeling approach we propose, follows a functional decomposition of the domain by identifying three relevant types of state variables. They are (i) the *functional variables*, (ii) *primitive variables* and (iii) *external variables* (see (Umbrico, Orlandini, and Cialdea Mayer 2015) for further details).

A *functional variable* provides a logical representation of the agent in terms of the high-level task the agent can perform, notwithstanding its internal composition. A *primitive variable* models a specific physical/logical component of the system. Values of such a variable correspond to concrete state/actions the related element is actually able to assume/perform over time. Finally, an *external variable* provides a logical view of an element whose behavior is not under the control of the system but affect the execution of its functionalities. Such a variable models conditions that must hold in order to successfully perform internal activities.

Synchronization rules specify constraints between different variables of the planning domain. These rules allow to further constrain the behaviors of the domain state variables in order to safely realize complex tasks. In this regard, given the described modeling approach, synchronization rules can be used to specify how the high-level functionalities (i.e. the values of the functional state variables) are implemented by the agent. A synchronization rule specifies the set of *primitive* and/or *external* values and the needed temporal relations that allow the agent to successfully perform the related high-level function (i.e. the *functional* value the synchronization applies to).

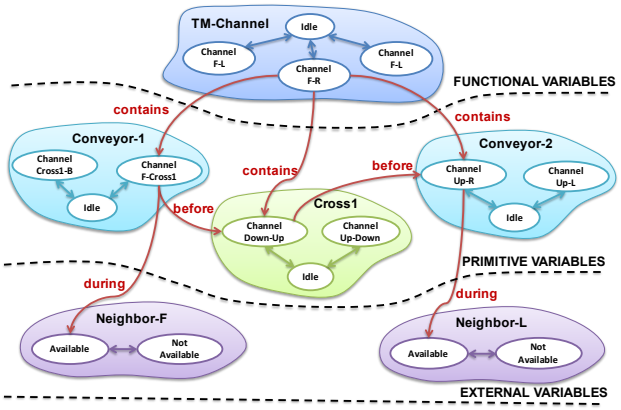

Figure 3: A planning domain example

Figure 3 shows an example of a planning domain obtained by applying the modeling approach described to a Transportation Module (TM) of the manufacturing plant in the

GECKO project. The functional state variable *TM-Channel* models the transportation tasks the module is able to perform (e.g. *Channel-F-B*, *Channel-R-L*). The primitive state variables model the internal component of the TM in terms of the operations they can perform (e.g. a conveyor of the module is able to move a pallet between two internal position of the module). The external state variables model the status of other agents (i.e. other TMs) of the plant the TM must cooperate with.

Finally the read arrows of Figure 3 models constraints among values of the state variables that must be satisfied to perform the tasks. Specifically, the constraints describe the way the TM can implement the possible transportation tasks. E.g. the TM performs a "Channel-F-R" by moving the pallet from position "F" to position "Cross1" by means of "Conveyor-1". Then the "Cross1" moves the pallet from position "down" to position "up". Finally, the "Conveyor-2" moves the pallet from position "up" to position "L".

A domain-independent heuristic It is possible to observe that a *synchronization rule* basically, represent a *dependency* between two or more variables and their timelines. Thus, given a timeline A and a timeline B, a synchronization rule $S_{A,B}$ from a token $x \in A$ to a token $y \in B$ implies a dependency between these timelines. Namely, tokens on timeline B are subject to tokens on timeline A.

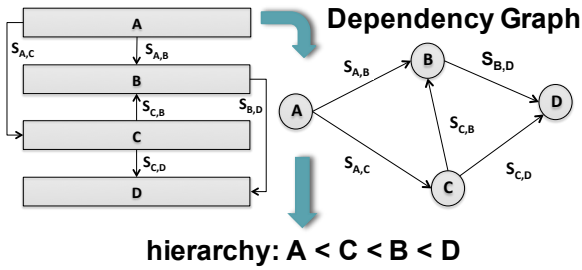

Figure 4: Extracting hierarchy from synchronizations

Given this assumption, it is possible to build a *dependency graph* (DG) among timelines by analyzing synchronization rules. Figure 4 shows a set of timelines with synchronization rules and the resulting dependency graph. The nodes of the graph represent timelines (or state variables) and edges represent dependecies between them (i.e. at least a synchronization rule between tokens of the related timelines exists).

Given the DG, it is possible to extract the hierarchy of the domain. An edge from a node A to a node B in the DG represents a dependency between timeline A and timeline B. Consequently, the hierarchy level of timeline A is not lower than the hierarchy level of timeline B. If not path in the DG connects B to A (i.e. no cycle is detected) then the hierarchy level of A is higher than the hierarchy level of B. Thus if the DG contains a cycle among two or more nodes then the related timelines have the same hierarchy level and they are said to be hierarchically equivalent.

In this regard, the heuristic we have defined exploits hierarchy information to assign "priority" to the flaws detected during the solving process. The work (Umbrico, Orlandini, and Cialdea Mayer 2015) shows some promising results concerning the application of this heuristic to improve the solving capabilities of EPSL-based planners. In particular, the heuristic can be represented as a *pipeline* of filters that extract the most *promising* flaws to solve first as follows:

$$\Phi^0(\pi) \xrightarrow{f_h} \Phi^1(\pi) \xrightarrow{f_t} \Phi^2(\pi) \xrightarrow{f_d} \Phi^3(\pi) \rightarrow \ \phi^* \in \Phi^3(\pi)$$

Given a partial plan π with an initial set of flaws $\Phi^0(\pi)$, each filter f of the pipeline extract the subset of flaws to solve according to a particular criteria. The first filter f_h applies the hierarchy by selecting the flaws that belong to the most *independent* timelines (i.e. the timeline with the highest hierarchy level). The flaws composing the last set represent equivalent choices from the hierarchy point of view. Thus the planner can randomly selects one of these flaws to solve $\phi^* \in \Phi^3(\pi)$.

Ongoing Works

Currently we are making a comparison of EPSL framework with EUROPA which is one of the most known timeline-based planning framework in the literature. In particular our comparison aims at taking into account different aspects of the planning frameworks and not only their performances. Namely, our goal is to make a deep analysis of thedifferent approaches to timeline-based planning by considering their modeling capabilities, their expressiveness and their solving capabilities, in order to create a shared understanding of the meaning of both timelines and timeline-based plans.

In particular, we are taking into account two real world scenarios by defining the ROVER and the NEPTUS planning domains. Indeed, the selected domains represent two interesting real-world applications that are particularly relevant from the point of view of a plan-based control system. The core of both problems is to model and control a complex system which is able to perform some operations in a specific environment. The plan-based controller must provide the agent with the deliberative capabilities to autonomously synthesize and schedule the sequences of activities needed to perform high-level tasks.

The ROVER planning domain has been extracted from the scenario described on the EUROPA's web site concerning an autonomous exploration rover. This scenario represents a typical and well known application context in AI. It is relevant because it represents a classical single agent control scenario concerning the capability to provide a robotic device with autonomy in order to perform some complex tasks.

Similarly, the NEPTUS planning domain has been extracted from a real-world application senario, described in (Chrpa et al. 2015), where a number of AUVs must gather data about some known underwater phenomena. The problem of controlling an AUV may seem similar to the problem of controlling an autonomous exploration rover. However we have selected this domain for the *coordination* aspect involved. Indeed, in this context, the point is not just to control a single agent, but to safely coordinate several agents (i.e. the AUVs) in order to perform the tasks.

Initial results show relevant difference concerning their modeling approaches w.r.t. the structure of a timeline-based

planning domain, the type of elements the frameworks can model and the way a user must specify constraints to obtain the desired behavior of the system. Moreover there are also relevant differences concerning their interpretation of timeline-based plans and planning solutions.

Moreover I'm currently extending the EPSL planning framework by introducing the capabilities of modeling and reasoning about the temporal uncertainty of a planning domain. Thus, following the proposed formalization the planning framework must be able to model activities whose actual duration cannot be decided by the planner.

References

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*.

Borgo, S.; Cesta, A.; Orlandini, A.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014a. 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.; Rasconi, R.; Suriano, M.; and Umbrico, A. 2014b. A cooperative model-based control agent for a reconfigurable manufacturing plant. In *ICAPS-14, PlanRob - The 2nd ICAPS Workshop on Planning and Robotics*.

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 *The Twenty-Eighth International Flairs Conference*.

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* 1–18.

Cesta, A., and Fratini, S. 2008. The Timeline Representation Framework as a Planning and Scheduling Software Development Environment. In *PlanSIG-08. Proc. of the 27th Workshop of the UK Planning and Scheduling Special Interest Group, Edinburgh, UK, December 11-12*.

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*.

Chien, S.; Tran, D.; Rabideau, G.; Schaffer, S.; Mandl, D.; and Frye, S. 2010. Timeline-Based Space Operations Scheduling with External Constraints. In *ICAPS-10. Proc. of the 20th Int. Conf. on Automated Planning and Scheduling*.

Chrpa, L.; Pinto, J.; Ribeiro, M. A.; Py, F.; Sousa, J.; and Rajan, K. 2015. On mixed-initiative planning and control for autonomous underwater vehicles. In *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, 1685–1690.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2014. A formal account of planning with flexible timelines. In *The 21st International Symposium on Temporal Representation and Reasoning (TIME)*, 37–46. IEEE.

Cialdea Mayer, M.; Orlandini, A.; and Umbrico, A. 2015. Planning and execution with flexible timelines: a formal account. *Acta Informatica* 1–32.

Laborie, P., and Ghallab, M. 1995. Ixtet: an integrated approach for plan generation and scheduling. In *Emerging Technologies and Factory Automation, 1995. ETFA '95, Proceedings., 1995 INRIA/IEEE Symposium on*, volume 1, 485–495 vol.1.

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

Stefano, B.; Amedeo, C.; Andrea, O.; and Alessandro, U. to appear. A planning-based architecture for a reconfigurable manufacturing system. In *The 26th International Conference on Automated Planning and Scheduling*.

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 International Publishing. 410–423.