22nd International Conference
on Automated Planning and Scheduling

June 25-29, 2012, Atibaia – Sao Paulo – Brazil

PSTL 2012

Proceedings of the
Workshop on Planning and
Scheduling with Timelines

Edited by
Gérard Verfaillie and Roman Barták

Editors

Gérard Verfaillie, ONERA, France
Gerard.Verfaillie@onera.fr

Roman Barták, Charles University in Prague, Czech Republic
bartak@ktiml.mff.cuni.cz
Roman Barták is supported by the Czech Science Foundation under the contract P103/10/1287.

Organization

Gârard Verfaillie, ONERA, France
contact email: Gerard.Verfaillie@onera.fr

Roman Barták, Charles University, Czech Republic
contact email: bartak@ktiml.mff.cuni.cz

Program Committee

Roman Barták, Charles University, Czech Republic

Amedeo Cesta, ISTC-CNR, Italy

Steve Chien, JPL, USA

Brad Clement, JPL, USA

Juan Manuel Delfa Victoria, Darmstadt Technical University, Germany

Jeremy Frank, NASA, USA

Simone Fratini, ESA, Germany

Felix Ingrand, LAAS-CNRS, France

Nicola Policella, ESA, Germany

Cédric Pralet, ONERA, France

Kanna Rajan, MBARI, USA

Gérard Verfaillie, ONERA, France

Foreword

Timelines are a very natural way of representing how the state of a physical system evolves with time. System evolution is represented via a set of concurrently evolving timelines. Each timeline represents the evolution of a specific state variable. Each variable may have a finite, discrete, or continuous domain of values. Each variable may evolve in a discrete or continuous way, due to instantaneous events occurring at some times or due to natural time evolution. Constraints on and between timelines, either on timeline values or on times of change, represent physical limitations or user requirements on the concurrent system evolution.

This representation is at the basis of many existing planning and scheduling systems, either specific or generic. It has been widely used for the high-level control of fielded systems, that is for planning, scheduling, and controlling activities of space, air, ground, or underwater mobile robots. As opposed to other classical planning systems which focus on actions and their durations, preconditions, and effects, planning and scheduling systems that are based on timelines focus on concurrent system evolutions following actions and time evolution.

The short-term objective of the workshop is to gather people that are working with this kind of representation in order to take stock of what exists and of what remains to be done. The long-term objective would be to reach an agreement on a common timeline representation framework and on a common language, on top of which different algorithms could be developed. Mid-term realistic objectives would be the edition of a book or a special issue of a journal and the creation of a common Web site where any material related to timeline-based planning and scheduling systems (articles, presentations, software, benchmarks . . .) could be found.

Gérard Verfaillie and Roman Barták
PSTL 2012 Organizers
June 2012

Contents

EUROPA: A Platform for Timeline-based AI Planning, Scheduling, Constraint Programming, and Optimization

Javier Barreiro*, Matthew Boyce*, Jeremy Frank†, Michael Iatauro*
Paul Morris†, Tristan Smith*, Minh Do*

* SGT Inc., NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035
† NASA Ames Research Center, Mail Stop 269-3, Moffett Field, CA 94035

Abstract

EUROPA is a class library and tool set for building and analyzing planners within a Constraint-based Temporal Planning paradigm. This paradigm has been successfully applied in a wide range of practical planning problems and has a legacy of success in NASA applications. *EUROPA* offers capabilities in 3 key areas of problem solving: (1) Representation; (2) Constraint-based Reasoning; and (3) Search. *EUROPA* is a means to integrate advanced planning, scheduling and constraint reasoning into an end-user application and is designed to be open and extendable to accommodate diverse and highly specialized problem solving techniques within a common design framework and around a common technology core. While *EUROPA* is a complete tool set, in this paper, we will mostly concentrate on its timeline-based plan representation and the least-commitment partial-order planning approach operates on top of that representation.

Introduction

EUROPA (Extensible Universal Remote Operations Planning Architecture) is a class library and tool set for building timeline-based planners (and/or schedulers) within a Constraint-based Temporal Planning paradigm (Frank and Jonsson 2003). Constraint-based Temporal Planning is a paradigm of planning based on an explicit notion of time and a deep commitment to a constraint-based formulation of planning problems. This paradigm has been successfully applied in a wide range of practical planning problems and has a legacy of success in NASA applications.

As a complete Planning & Scheduling platform, *EUROPA* offers capabilities in 3 key areas of problem solving:

1. **Representation**: Externally, *EUROPA*'s main input modeling language is the New Domain Definition Language (NDDL) (pronounced 'noodle'). NDDL is a high-level object-oriented modeling language that can describe a number of concepts based on Variables and Constraints. Internally, *EUROPA* allows a rich representation based on *durative tokens* on *timelines* for actions, states, resources and constraints that allows concise declarative descriptions of problem domains and powerful expressions of plan structure.
2. **Constraint-based Reasoning**: *EUROPA*'s main reasoning module contains constraint-processing and inference algorithms to enforce domain rules/constraints and propagate consequences as updates are made to the problem state. Specialized techniques for reasoning about temporal quantities and relations included in *EUROPA* are particularly useful to deal with real-life problem domains.
3. **Search**: by default, *EUROPA* supports chronological-backtracking search with the ability to integrate heuristics into the basic search algorithm and for developing new customized search algorithms.

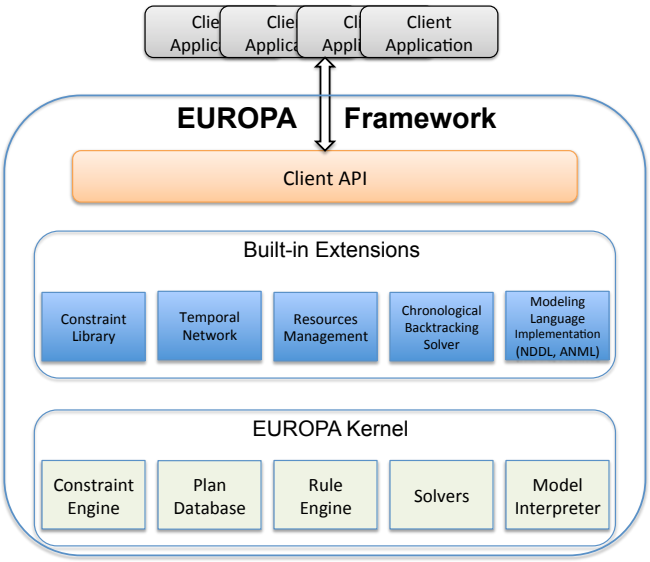


Figure 1: *EUROPA* Architecture

EUROPA is designed to be open and extendable to accommodate diverse and highly specialized problem solving techniques within a common design framework and around a common technology core. Figure 1 shows the main components of *EUROPA* and the hierarchical relationships between them: through the *Client API*, applications can utilize or extend either the core components of the *EUROPA kernel* or the built-in commonly used components that are extensions of the kernel components. *EUROPA* is now at version 2.6 which embodies significant evolution from the original *EUROPA*, which in turn was based upon HSTS (Muscettola et al. 1998). It has been made available under an open-source license. The source code and extensive documents on *EUROPA* are available at: *http://code.google.com/p/europapso/*.

Timeline-based Planning

Modeling: The NDDL input modeling language representation includes state and activity descriptions, as is common in planners using traditional modeling languages like the Planning Domain Definition Language (PDDL) that support the variable-value formalism. *EUROPA* thus takes its heritage from planning formalisms like IxTeT and SAS+. The NDDL models are parsed/translated into an internal representation with the following core components:

- **Timeline & Token:** *EUROPA* state variables are represented using *timelines*, and the changing values on timelines represent sequences of states. Specifically, states are represented as a set of temporally extended predicates called *tokens* and the temporal and logical constraints between them. Each token consists of a proposition and a list of parameters, which by default includes the *start*, *end* and *duration* temporal variables. Timelines consist of totally ordered sequences of connected tokens; hence, a timeline can be in only one state at any instant. Note that while tokens mostly represent temporally-extended predicates, which resemble world-state, tokens also represent actions in *EUROPA* and thus may not be on some particular timelines.
- **Compatibility:** The final component of NDDL model is a set of compatibilities/rules that govern the legal arrangements of states on, and across, timelines. These compatibilities are logical implications asserting that if a timeline is in a state (represented by a given token), then other timelines must be in one of a set of compatible states. Compatibilities can incorporate (1) explicit logical and arithmetic constraints on the parameters of the states/tokens or (2) temporal constraints between tokens. *EUROPA* provides a library of such constraints (e.g., all of the Allen's Temporal Relations), and this library can be extended if new constraints are needed.

Least-Commitment Partial Planning Refinement Search: EUROPA follows the *lifted plan-space refinement* planning approach. It uses a series of refinements to convert an initial partial plan into a final plan that is complete with respect to the requirements of the planner. A partial plan consists of a set of timelines containing a partially instantiated and ordered set of tokens with the possible flaws of: (1) unbound variables; (2) open condition; and (3) temporal threats. Flaws are resolved one at a time following user-defined flaw filter and flaw selection strategy and a complete plan is returned when there are no flaws. The main flaw types and the resolution strategy for each of them are:

- **Unbound Variable:** a variable in the partial plan whose domain is not a singleton. A unbound variable flaw is resolved by specifying a value from the domain of that variable.
- **Open Condition:** when a new token is added to the partial-plan, its status is *inactive* and it represents an open-condition flaw. This type of flaw can be resolved by: (1) *merge* with an existing active token; (2) *activate* and add to timeline; or (3) *reject* (if this option is allowed for the flawed token). When a token is "activated", it can introduce a new set of (inactive) *slave* tokens. For example, if the activated token represents a durative action, then (slave) tokens representing (pre)conditions and effects are added as new inactive tokens (new open conditions).
- **Threat:** once a token has been placed in the partial plan it may impact other tokens indirectly through possible overlapping requirements on objects, or by creating resource oversubscription. Threats are resolved by imposing ordering constraints among tokens.

There are extendable built-in flaw-filter and flaw-selection strategies to decide which flaw to be handled next. For the built-in chronological backtracking (depth-first) search solver, when a flaw is selected, it represents a branching point in the search graph with the next child search node to be selected/explored basing on the resolution-selection strategy (which *EUROPA* also provides extendable built-in options). The flaw-resolution strategy is backed by strong constraint propagation routines with special emphasis on temporal and resource reasoning.The search process stops when there is no additional flaw to handle. The built-in solver is only one of many search strategies that can be implemented using *EUROPA* all of the solver components are designed to allow the *EUROPA* user to implement other search algorithms and heuristics as required by the problem domain being addressed.

Current State & Future Work

EUROPA and its supporting tools have been going through a long period of development. Besides the core modeling and reasoning capabilities, *EUROPA* also provides a streamlined API to integrate with native client applications, Eclipse's plugins to help build and debug NDDL/ANML models, and visualization capabilities to support plan comprehension and diagnosis. Overall, *EUROPA*'s current main strengths include: (1) proven track record of addressing real life planning and schedulingproblems; (2) expressive modeling capability; (3) flexible framework; (4) strong support for integration with other applications; (5) open-source license. It has been used for a variety of missions, mission-oriented research, and demonstrations, including: DS1: RAX Remote Agent Experiment, Support for operation of the International Space Station's solar arrays, Bedrest study at Johnson Space Center, MER Tactical Activity Planning, Advanced Spaceflight Training Systems Development, MBARI's underwater autonomous vehicle, and Willow Garage's autonomous robot navigation.

Nevertheless, we still have a long list of improvements for *EUROPA*. The most important ones in our opinion are: significant improvements for search (especially heuristic guidance) and inference capabilities, support the ANML and PDDL modeling languages, improve the visualization and debugging tools and allow *EUROPA* extensions to be written in other languages. Given that *EUROPA* is open-source software, we welcome contributions from planning and scheduling researchers and practitioners.

References

Frank, J., and Jonsson, A. K. 2003. Constraint-based attribute and interval planning. *Journal of Constraints Special Issue on Constraints and Planning* 8(4).

Muscettola, N.; Nayak, P.; Pell, B.; and Williams, B. 1998. Remote agent: To boldly go where no ai system has gone before. *Artificial Intelligence* 103:5–47.

The APSI Framework: A Platform for Timeline Synthesis

Simone Fratini
ESA/ESOC
European Space Agency
European Space Operations Centre
Darmstadt, Germany
simone.fratini@esa.int

Amedeo Cesta
ISTC-CNR
Italian National Research Council
Via S. Martino della Battaglia, 44
I-00184 Rome, Italy
amedeo.cesta@istc.cnr.it

Abstract

The paper addresses to the current discussion aimed at reaching an agreement for a common representation framework and language for modeling problems with timelines. This paper is grounded on the experience of the 10 years collaboration between ISTC-CNR and ESA/ESOC for timelines-based planning, scheduling and applications deployment. The paper introduces the key concepts of our timeline modelization called the TRF (Timeline Representation Framework) and describes how the TRF is implemented in the ESA APSI (Advanced Planning and Scheduling Initiative) software platform.

Introduction

A solution of a planning and/or scheduling problem consists not only of crisp algorithms, but rather entails the development of a piece of software able to provide data structures and services necessary to implement the algorithm and create a comprehensive usable service. This software development effort is involved both in applied and in experimental research: in applied research the final goal is produce a piece of software that is identified by the customer as a "system", in experimental research the need for good and competitive results leads often to a significant effort in terms of deployment around the core algorithm that is the main objective of the experimentation.

Timeline based algorithms are not an exception in this respect. In fact the deployment and use of a timeline-based algorithm for planning and/or scheduling usually requires the representation and management of (at least): temporal information, timelines, values of the timelines (often represented as predicates of continuous and/or discrete parameters over time intervals) and various high-level relationships among temporal and value information (referred to as the domain theory).

Pre-existing works, (Muscettola 1994; Frank and Jónsson 2003; Fratini, Pecora, and Cesta 2008; Verfaillie, Pralet, and Lematre 2010), do not follow a standardized approach with a consequent objective difficulty in spreading and re-using information, software and languages across the community. There has been a proliferation of platforms all aimed at providing a set of similar services to implement planning and scheduling algorithms as well as complete "end-to-end" applications. Despite the fact these platforms have been using different underlying technologies, different modeling assumptions and different languages for representing and using the relevant information, they have in common that they usually provides high-level support for (or for a subset of): (1) Represent and Manage Domains and Timelines (2) Model and Represent Domain Theories, Problems and Solutions (3) Problem Solving with Timelines (4) Timeline V&V and (5) Timeline Execution.

In our work we have synthesized a reference framework, called TRF (Timeline Representation Framework), which aims at providing the theoretical background for modeling and solving generic problems with timelines. Additionally, we have deployed an implementation of the TRF, namely the APSI framework, for rapid prototyping of timeline-based planning and scheduling solutions for space applications.

APSI's design has been driven by the need of a fast and easy integration of general, domain independent planning and scheduling services with specific problem solving algorithms, because very often in space domains there is the need of modeling specific features not easy (or not worth being pursued for efficiency reasons) to be handled with domain independent P&S techniques. For this reason the APSI framework (Cesta et al. 2011; Fratini, Pecora, and Cesta 2008) is an architecture designed to allow a decomposition of the problem solving process in a hierarchy of interacting solvers, each of them devoted to the solution of a specific problem with timelines, and provides for a modeling language designed for abstracting the problem of integrating these solvers as a problem of timeline synchronization.

The software architecture and the modeling language of the APSI framework are based on the TRF's theoretical background which provides, with the concepts of *timeline*, *event*, *synchronization* and *solving step*, the abstraction level to entail this integration. This paper provides a theoretical definition of these key concepts, a sketchy implementation for them and an example of modeling.

Behaviors and Events Representation

The timelines-based approach inherits a lot from control theory. In fact a common background among all systems based on timelines is an underlying assumption that the world is modeled as a set of entities whose properties can vary in time (such as one or more physical subsystems) according with some internal logic or as a consequence of external in-

puts. The intrinsic property of these entities, namely *components* in the TRF, is that they "evolve over time" and that their evolutions can be affected by external inputs, namely *events* in the TRF. The analogy with control theory can be extended to conceive the problem solving with timelines as a problem of controlling components with external inputs in order to achieve a desired behavior.

Hence the assumption underlying the TRF theoretical framework is that whatever is the specific problem to model (planning, scheduling, execution or more specific tasks), it is generically casted as a problem of identifying a set of inputs and relations among them that, once applied to the components modeling a domain with a given initial set of possible temporal evolutions, namely *behaviors* in the TRF, will lead to a set of final behaviors that satisfy some properties (for instance feasible sequences of states, or feasible resource consumption, as well as more complex properties).

In the TRF a behavior σ for a component C is a piecewise function defined in an interval $\mathcal{H}$[1] by means of a finite set of samples of $\mathcal{H}$ and values in a set $\mathcal{D}_C$ associated to them. Hence we model a behavior σ for a component C as a tuple $\langle T_\sigma, N_\sigma \rangle$ where T_σ is finite set of ordered time instant in $\mathcal{H}$[2] and $N_\sigma \in \mathcal{D}_C^{|T_\sigma|}$ is an assignment of values to the time instants in T_σ. An event e is a pair $\langle \tau_e, \nu_e \rangle$ where τ_e is the time frame of the event, i.e. a sub-set of $\mathcal{H}$, and ν_e is its value. We denote as $\mathcal{V}_C$ the set of values for the events applicable to a component C, hence we have $\nu_e \in \mathcal{V}_C$, and we assume $\mathcal{V}_C$ be a set where is defined a composition operator $\oplus$[3].

In the APSI framework, an implementation of the TRF theoretical framework, behaviors are restricted to be piecewise constant functions, where a sample in $\langle t_i \in T_\sigma, v_i \in N_\sigma \rangle$ represents the constant value of the behavior in $[t_i, t_{i+1})$, or piecewise linear functions, where the behavior is linear between two consecutive samples. Furthermore time frames for events are restricted to be intervals in $\mathcal{H}$, hence an event is a pair $\langle [t_s, t_e), v \in \mathcal{V}_C \rangle$ with $t_s, t_e \in \mathcal{H}, t_s > t_e$.

The APSI framework provides for the implementation of components for domain independent planning and scheduling, including *state variables* (Muscettola 1994) and various types of resource (defined below), as well as provides for specialized components implemented during various deployments for ESA (implemented from scratch to model specific features of the domain, like rechargeable batteries for instance, or wrapping pre-existing domain specific applications conceptually based on timelines, for efficiency reasons or to preserve legacy software).

State variables. A state variable is defined in the APSI framework as a component C_{SV} taking piecewise constant behaviors over a set of symbolic values $\mathcal{D}_{C_{SV}}$. The co-domain $\mathcal{D}_{C_{SV}}$ of the behaviors for a state variable is represented by means of a tuple $\langle \mathcal{S}, \mathbb{D}, \mathcal{V} \rangle$ where:

- $\mathcal{S} = \{s_1, \ldots, s_n\}$ is a finite set of symbols;
- $\mathbb{D}_v = \{\mathcal{D}_1, \ldots, \mathcal{D}_n\}$ is finite set of parameter domains;
- $\mathcal{V} = \{v_1, \ldots, v_n\}$ is a finite set of states. A status $v \in \mathcal{V}$ is a pair $\langle s_v \in S, X_v = \langle x_1^v, \ldots, x_m^v \rangle \rangle$. $X(v)$ is the set of parameters of a status v, $\mathcal{D}(x) \in \mathbb{D}_v$ is the domain of the parameter x.

We use bounded integer parameters and enumerated parameters, constituted by a finite set of symbols. We define $\mathcal{D}_{C_{SV}} \equiv \mathcal{V}_g$, where $\mathcal{V}_g$ is the set of all the possible groundization of values in $\mathcal{V}$[4].

For a state variable only some transitions between the states are allowed and there is a minimal and maximal allowed temporal duration for each status (see also Figure 1). Valid behaviors for a state variable components are defined by means of a tuple $\langle \mathcal{T}, \mathcal{W}, \mathcal{D} \rangle$ where:

- $\mathcal{T} \subseteq \mathcal{S} \times \mathcal{S}$ is the transition function that denotes legal transitions between pair of states;
- $\mathcal{W} : \mathcal{T} \rightarrow \mathcal{R}$ is the function that associates to possible transitions between states, sets of relations among the parameters of the states. Being $\langle v_i = \langle s_i, X_i \rangle, v_j = \langle s_j, X_j \rangle \rangle$ a transition such that $\langle s_i, s_j \rangle \in \mathcal{T}$ and $X = X_i \cup X_j$ the union of v_i and v_j variables, we have $\forall r \in \mathcal{R}, r \subseteq \mathcal{D}(x_0) \times \cdots \times \mathcal{D}(x_{|X|})$;
- $\mathcal{D} : \mathcal{S} \rightarrow \mathbb{N} \times \mathbb{N}$ is the value duration function, i.e. a function that specifies the allowed duration of values in $\mathcal{V}$ (as an interval $[lb, ub]$);

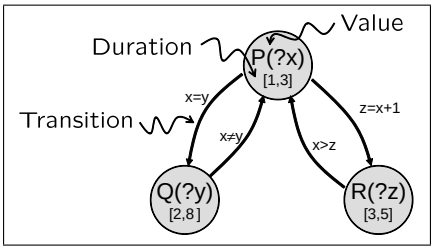


Figure 1: State Variable

Events applicable to a state variable component model choices of values over time intervals (as done in similar architectures as EUROPA and HSTS, where a "token" is conceptually similar to an event for a state variable component). Hence for a state variable component, $\mathcal{V}_{C_{SV}} \equiv 2^{\mathcal{D}_{C_{SV}}}$. The operator $\oplus$ is defined for $\mathcal{V}_C$ as the set intersection $\cap$ among the disjunctions of values of the state variable according with the semantic that the combined effect of a set of events stating a disjunction of values allowed in an interval of time is the intersection of the allowed values. Supposing for instance a state variable with three possible values $\{v_1, v_2, v_3\}$ and two events $e_1 = \langle \tau, \nu_1 = v_1 \vee v_2 \rangle$ and

[1] $\mathcal{H} = [t_0, t_H) \subseteq \mathbb{R}$, with $t_0, t_H \in \mathbb{R}, t_H > t_0$.

[2] $T_\sigma = \{t_i | t_i \in \mathcal{H}\}$. We assume, $|T_\sigma| \geq 2$, $t_0 = t_O$, $t_{|T_\sigma|} = t_H$ and $t_i < t_{(i+1)}$.

[3] The operator $\oplus$ is provided to define the combined effect of two events $e_i = \langle \tau, \nu_i \rangle$ and $e_j = \langle \tau, \nu_j \rangle$ on the same time frame τ. We denote as $e_i \oplus e_j$ the event e such that $\tau_e = \tau$ and $\nu_e = \nu_i \oplus \nu_j$.

[4] $v_g = \langle s, \overline{X} = \langle \overline{x}_0 \in \mathcal{D}(x_0), \ldots, \overline{x}_n \in \mathcal{D}(x_n) \rangle \rangle \in \mathcal{V}_g$ denotes a groundization of a state variable value $v = \langle s, X = \langle x_0, \ldots, x_n \rangle \rangle \in \mathcal{V}$.

$e_2 = \langle \tau, \nu_2 = v_2 \vee v_3 \rangle$, we have $\nu_1 \oplus \nu_2 = \{v_2\}$. In this case the combined effect of e_1 and e_2 would be an event $\langle \tau, \nu = v_2 \rangle$.

Resources. Regarding resources, a common sense for what a resource is is any physical or virtual entity of limited availability needed for something that can generate competition. This concept can be captured by a component whose behaviors are numerical functions of time, piecewise, linear, exponential or even more complex, depending on the accuracy of the model. Each behavior represents a different profile of resource allocation in time. Given that in the APSI framework are currently available resources implemented as components taking piecewise constant numeric behaviors. The concept of limited availability is defined by means of minimum and maximum availability bounds: a temporal function representing a behavior of a resource is a consistent behavior if it is always between allowed bounds in the interval of planning and scheduling. There are two classes of resources modeled as two different components: reusable resource component C_R and consumable resource component C_C. The difference stems in the type of event applicable. In both cases an event capture the concept of resource usage, but for a reusable resource an event represents an amount of resource booked on a temporal interval while for a reusable resource an event is an amount produced or consumed in a time instant. Given that a generic event for a reusable resource is defined over an interval and can take values in $\mathbb{N}$ (this is basically an activity that states a resource usage over the time frame) while a generic event for a consumable resource is defined in a time instant and takes values in $\mathbb{Z}$, affecting the resource allocation profile from that point to the horizon (negative events are considered consumptions). The operator $\oplus$ is defined as simple algebraic sum, being the cumulative effects of bookings and production/consumptions the algebraic sum of the amounts booked or produced/consumed.

An example. As an illustrative example of the concepts introduced so far, we will sketch here a model designed for a robotic scenario[5]. The goal of the experiment was to control the rover to take a set of pictures, store them on board and dump the pictures when a communication channel was available.

The first step of the experiment was to model the system to be controlled. The rover is equipped with a pan-tilt unit, two stereo cameras (mounted on top of the pan-tilt unit), a communication facility and a limited memory to store pictures[6]. The rover is able to autonomously navigate the environment, move the pan-tilt, take pictures and communicate images to a remote orbiter. Hence we consider, for this experiment, as interesting sub-systems to model the rover: a mobility system MS, a pan-tilt unit PTU, a camera CAM, a communication system COMM and a memory MEM.

We consider in the model the rover able to move between two points in space given their coordinates $\langle x, y \rangle$, taking into account that the rover may get stuck in between two points. Hence the mobility system can be modeled as a state variable taking the following values: AT($?x, ?y$) when the rover is standing in $\langle x, y \rangle$, GOTO($?x, ?y$) when the rover is moving toward $\langle x, y \rangle$ and STUCKAT($?x, ?y$) when the rover is stuck in $\langle x, y \rangle$. We assume that a transition GOTO($?x, ?y$) $\rightarrow$ AT($?x, ?y$) denotes a successful move to $\langle x, y \rangle$, while a transition GOTO($?x, ?y$) $\rightarrow$ STUCKAT($?x', ?y'$), with $?x \neq ?x'$ or $?y \neq ?y'$ denotes an unsuccessful move to $\langle x, y \rangle$ with the rover stuck in $\langle x', y' \rangle$ while moving. A transition AT($?x, ?y$) $\rightarrow$ GOTO($?x', ?y'$) denotes the rover starting to move from a point $\langle x, y \rangle$ to a point $\langle x', y' \rangle$.

The rover pan-tilt unit can be pointing a given angle $\langle \alpha, \beta \rangle$ or can be moving from two angles. Hence the unit can be modeled with a state variable taking the following values: POINTINGAT($?pan, ?tilt$) when the unit is pointing an angle $\langle \alpha = ?pan, \beta = ?tilt \rangle$ and MOVINGTO($?pan, ?tilt$) when the unit is moving toward an angle $\langle \alpha = ?pan, \beta = ?tilt \rangle$.

The rover camera can take pictures with the pan-tilt unit (in the current position of the rover) and store the picture on an on-board memory with a given file id. Hence the camera can be modeled with a state variable taking the following values: CAMIDLE(), when the camera is not taking pictures and TAKEPIC($?file\_id$) when the unit is taking a picture that will be stored in a file with id = $?file\_id$.

The communication system can dump a file with a given id. Hence it can be modeled with a state variable taking the following values: COMMIDLE(), the idle status, and DUMP($?file\_id$) when the unit is dumping a picture stored in a file with id = $?file\_id$.

The memory has a fix amount of cells available to store pictures. A cell is occupied when a picture is stored and freed when a picture is transmitted to the orbiter. Hence it can be modeled with a consumable resource with unary consumption and production events.

The second step was to model the controller to operate the rover. Interesting user goals were (among the others): (1) to take a picture in a given position $\langle x, y \rangle$ with the pan-tilt unit in $\langle \alpha, \beta \rangle$, storing it with a given file id and dump it when the orbiter is visible for some periods and (2) to drive the rover in a given position $\langle x, y \rangle$. Hence we modeled the controller using two more state variables: a mission timeline state variable MT and a visibility dump window state variable VW.

The mission timeline state variable can take the values TAKEPICTURE($?x, ?y, ?pan, ?tilt, ?file\_id$), to model the goal of taking a picture in $\langle ?x, ?y \rangle$, with the pan-tilt unit pointing to $\langle ?pan, ?tilt \rangle$, store the picture in a file with the id $?file\_id$ and dump it as soon as possible, the value GOTO($?x, ?y$) to model the goal of just moving the rover to $\langle ?x, ?y \rangle$ and IDLE() (an idle status).

[5]The model was designed for an experiment done at LAAS-CNRS in the context of the ESA GOAC project (Goal Oriented Autonomous Controller). The target platform was DALA, an iRobot ATRV rover that has been successfully used to show an autonomy operation experiment. See (Fratini et al. 2011) for a detailed description of the specific domain models. The on-board experiment has been performed using an APSI domain independent planner integrated with an executive layer.

[6]Memory was not in the original GOAC model, it has been added here to show an example of resource modeling.

The visibility window state variable models the temporal windows where the communication channel is available. For the sake of simplicity, we modeled a binary state variable with only two values: AVAILABLE() and NOTAVAILABLE(). This component was an *uncontrollable* feature, i.e. not subject to planning. In fact the orbiter visibility was computed in advance and ingested as a component with an already completely specified behavior.

Timelines and Timeline Extraction Procedures

The relation between component behaviors and events that influence them is modeled by means of *timelines* and *timeline extraction procedures*. A timeline in the TRF is a finite set of variables representing ordered points in a temporal interval $\mathcal{H}$ and variables representing values related to them. Formally, a timeline $\mathcal{TL}$ for a component C over a temporal interval $\mathcal{H}$ is a tuple $\langle \mathcal{T}, \mathcal{V}, T, N \rangle$ where:

- $\mathcal{T}$ is a finite, ordered set of variables ranging in $\mathcal{H}$;
- $\mathcal{V}$ is a set of variables ranging in $\mathcal{D}_C$ such that $|\mathcal{V}| = |\mathcal{T}|$;
- $T \subseteq \mathcal{H}^{|\mathcal{T}|}$ is a set of possible temporal occurrence assignment to variables in $\mathcal{T}$ such that $\langle \mathcal{T}, T \rangle$ defines a fragmentation[7] of $\mathcal{H}$;
- $N \subseteq \mathcal{V}^{|\mathcal{T}|}$ is a set of possible assignment of values in $\mathcal{V}$ to variables in $\mathcal{T}$.

A timeline $\mathcal{TL} = \langle \mathcal{T}, \mathcal{V}, T, N \rangle$ for a component C defines a set of partitions of $\mathcal{H}$ into intervals $\tau_i = [t_i, t_{i+1})$ and associated values $\nu(\tau_i)$, one for each assignment of variables in $\mathcal{T}$ and $\mathcal{V}$, and we denote that as $\langle \tau_i, \nu(\tau_i) \rangle \in \mathcal{TL}$. These partitions define an envelope of C's behaviors, having $\mathcal{TL} = \{\sigma_i = \langle T_\sigma, N_\sigma \rangle | T_\sigma \in T \wedge N_\sigma \in N\}$.

A timeline extraction procedure defines the effects of envelopes of events on envelopes of component behaviors. An envelope of events $\mathcal{E}v$ for a component C is a tuple $\langle \mathcal{E}, N, T \rangle$, where $\mathcal{E}$ is a set of variables $\epsilon = \langle \tau_\epsilon, \nu_\epsilon \rangle$ ranging over the set of events applicable for C, $T \subseteq (2^{\mathcal{H}})^{|\mathcal{E}|}$ is a set of temporal occurrences assignments to the time frames of the variables in $\mathcal{E}$ and $N \subseteq \mathcal{V}_C^{|\mathcal{E}|}$ is a set of possible assignments to the values of variables in $\mathcal{E}$.

The definition of a timeline extraction procedures is based on the principle of *temporal fragmentation*. A set of time frames $\{\tau_1, \ldots, \tau_n\}$ is a *fragmentation* of a time frame τ' if $\tau' = \bigcup \tau_i$. We denote as $\mathcal{F}(\tau') = \{\tau_1, \ldots, \tau_n\}$ the fragmentation of τ'. A timeline $\mathcal{TL}$ defines a fragmentation of a time frame τ' if $\mathcal{TL}$ defines a partition of $\mathcal{H}$ into a set of time frames $\{\tau_1, \ldots, \tau_n\}$ and it exists a subset of it that is a fragmentation of τ'. A timeline $\mathcal{TL}'$ defines a fragmentation of a timeline $\mathcal{TL}$ if, being $\{\tau_1, \ldots, \tau_n\}$ the partition of $\mathcal{H}$ defined by $\mathcal{TL}$, for each τ_i in $\{\tau_1, \ldots, \tau_n\}$, $\mathcal{TL}'$ defines a fragmentation of τ_i. Figure 2 gives an idea of the fragmentation principle. In the figure $\mathcal{TL}'$ is a fragmentation of a timeline $\mathcal{TL}$ (with two fragments having values ν_1 and ν_2) and a fragmentation of the temporal occurrences of two events ϵ_1 and ϵ_2 (with values $\nu_{\epsilon 1}$ and $\nu_{\epsilon 2}$). $\mathcal{TL}'$ has 8 fragments. The figure shows also the fragmentation of ϵ_1.

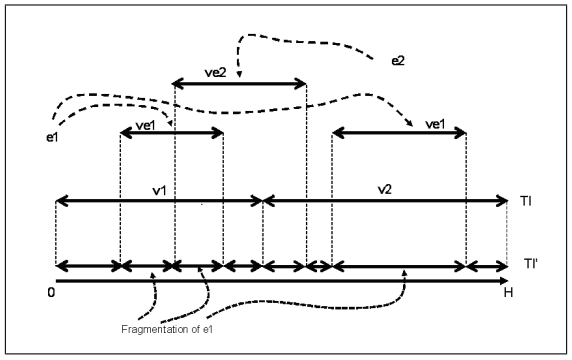


Figure 2: Temporal Fragmentation

A timeline extraction procedure models the effects of an envelopes of events over an envelope of component behaviors. Given a timeline $\mathcal{TL}$ modeling an envelope of component behaviors and an envelope of events $\mathcal{E}v$, in the TRF $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$ denotes the effect of the events in $\mathcal{E}v$ over the behaviors in $\mathcal{TL}$. $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$ is a timeline such that:

1. $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$ is a fragmentation of $\mathcal{TL}$;
2. $\forall \epsilon \in \mathcal{E}v$, $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$ is a fragmentation of τ_ϵ;
3. $\forall \langle \tau_i, \nu(\tau_i) \rangle \in \mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$, $\nu(\tau_i) = f_C(\nu(\tau), \oplus \nu_{\epsilon_j})$, where τ is the fragment of if $\mathcal{TL}$ such that $\tau_i \in \mathcal{F}(\tau)$ and ϵ_j are the events in $\mathcal{E}v$ such that $\tau_i \in \mathcal{F}(\tau_{\epsilon_j})$.

The function $f_C : 2^{\mathcal{D}_C} \times 2^{\mathcal{V}_C} \rightarrow 2^{\mathcal{D}_C}$ denotes, for a component C, the effects of an event over a component behavior in the time frame of occurrence of the event.

Timelines in APSI. In the case of state variable components implemented in the APSI framework for instance, the function $f_{C_{SV}}$ is defined as a set intersection. Supposing for instance a fragment $\langle \tau, \{v_1, v_2\} \rangle$ of a timeline for a state variable and an event $\epsilon = \langle \tau', \nu_1 = v_2 \vee v_3 \rangle$. The timeline extracted will take the value $f_{C_{SV}}(\{v_1, v_2\}, v_2 \vee v_3) = \{v_2\}$ in any fragment that belongs to the fragmentation of τ and τ'. The functionals f_{C_R} and f_{C_C} for reusable and consumable resource components are defined as algebraic sums. For a reusable resource, the effect of an event ϵ occurring in an interval τ is to decrease the resource level in any fragment of the timeline that is part of the fragmentation of τ, while for a consumable resource the effect of a production/consumption is to increase/decrease the resource level in any fragment of the timeline that is part of the fragmentation of the interval occurring from the time of the event occurrence till the planning horizon.

The APSI framework provides for an implementation of *flexible multi-valued timelines*. A flexible multi-valued timeline is a TRF timeline $\langle \mathcal{T}, \mathcal{V}, T, N \rangle$ where variables in $\mathcal{T}$ are time points of a Simple Temporal Problem with specified minimal and maximal distance between two consecutive points. Any solution of the STP satisfying the distance

[7] $\langle \mathcal{T}, T \rangle$ is a fragmentation of an interval $\mathcal{H} = [t_0, t_H)$ if, for each assignment $\overline{t} = \langle \overline{t_0}, \ldots, \overline{t_i}, \ldots, \overline{t_H} \rangle \in T$, $\overline{t_0} = 0$, $\overline{t_H} = H$ and $\overline{t_i} < \overline{t_j}, \forall i < j$.

constraints between two consecutive points is a valid assignment of temporal occurrences to variables in $\mathcal{T}$, hence T is defined as the set of solutions of the STP.

Envelopes of events over a time interval $\mathcal{H}$ are implemented as set of event variable with time frames that are either time points of an STP or intervals with starting and ending time points of an STP. Temporal relationships among them are defined via a quantified Allen interval algebra directly translatable into STP temporal constraints.

Value variables assigned to time points in a timeline or to time frames of events in an envelope and the set of valid assignments of values to the variables are defined trough a CSP over enumerated or numerical parameters. Timelines of state variables assign to time points states of the state variable possibly with constraints among parameters (see Figure 3), while resource timelines assigns to time points integer parameters, possibly with linear constraints among them.

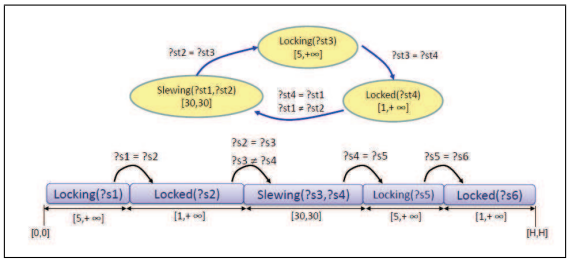


Figure 3: State Variable Timeline

The timeline extraction procedure implemented in the APSI framework allows to: (1) test if it is possible to extract the timeline, (2) identify subsets of the timeline and events temporal assignments to make possible a timeline extraction and (3) calculate $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$.

Regarding the first point, a timeline can be extracted from a timeline $\mathcal{TL}$ and an envelope of events $\mathcal{E}v$ if the times points of the STP underlying the timeline and the envelope of events are completely *ranked*, i.e., supposing available from the solution of the STP the distances d^{ij}_{min} and d^{ij}_{max} between any pair of time points (this requires an All Pair Shortest Path propagation of the temporal network), a set of time points is ranked if it does not exist any pair of points t_i and t_j in the set such that $d^{ij}_{min} < 0$ and $d^{ij}_{max} > 0$.

If the time points of the STP underlying the timeline and the envelope of events are not ranked, a scheduling procedure is necessary to enforce an order among them to allow the extraction of the timeline. Hence a timeline extraction procedure first schedules the time points by adding precedence constraints to the STP, then extract the timeline.

To extract the timeline, the time points are classified on the basis of their reciprocal distance: two time points belonging to the same class have a null distance to each other, while time points belonging to a class with a given index have a positive distance with respect to any time point belonging to a class with a smaller index. Formally a classification of time points into n classes $\mathbb{O} = \langle \mathcal{O}_1, \ldots, \mathcal{O}_n \rangle$, such that having $t_i \in \mathcal{O}_n$ and $t_j \in \mathcal{O}_m$, $n = m \leftrightarrow d^{ij}_{min} = d^{ij}_{max} = 0$ and $n > m \leftrightarrow d^{ij}_{min} \geq 0$. $\mathbb{O}$ defines a fragmentation of the planning horizon $\mathcal{H}$ into $|\mathbb{O}|$ fragments, that is also a fragmentation of all the events in the envelope and a fragmentation of the timeline $\mathcal{TL}$. In fact all the timeline transition points and all the time points that define the time frames of the event belong to the sets in $\mathbb{O}$. A set $\mathcal{O}_i$ is part of the fragmentation of an interval based event if its starting point belongs to a set $\mathcal{O}_j, j \leq i$ and its ending point belongs to a set $\mathcal{O}_k, k > i$. A set $\mathcal{O}_i$ is part of the fragmentation of the timeline fragment starting at a transition point t if it contains the time point representing t or if t belongs to $\mathcal{O}_j, j < i$ and does not exist any transition point in any set $\mathcal{O}_k, j < k < i$. To take a representative time point for each set in $\mathbb{O}$ and to associate it the composition of the values of the events and timeline fragments of which it is part of the fragmentation, define $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$.

Domain Theory, Problems and Solutions Modeling

More than a component in isolation, what is needed for modeling is a set of components synchronously evolving over the same temporal interval $\mathcal{H}$. When we use TRF components to model a real domain they cannot be considered as reciprocally decoupled, rather we need to take into account the fact that they influence each other's behavior. Hence the need of defining tuples of component behaviors acceptable considering the fact they represent the evolution of a whole system once put all together.

Last but not least when modeling a system it is usually needed to specify also how events have to be synchronized, because in fact physical and technical constraints influence also the type of input that can be provided to the system. Hence the need of defining also valid patterns for the events that can be applied to modify the behavior of the system.

Domain Theory. A way for practically defining valid patterns of applicable events and/or valid combination of behaviors (namely a definition of the *domain theory* that specifies what the system is and how it can be used) is by means of *temporal properties* on timelines and envelopes of events.

A *temporal property* ρ is a pair $\langle \pi_\tau, \pi_\nu \rangle$ where $\pi_\tau : 2^{\mathcal{H}} \rightarrow \{\top, \bot\}$ is a boolean function on time frames of $\mathcal{H}$ and $\pi_\rho : \mathcal{V} \rightarrow \{\top, \bot\}$ is a boolean function on a set of values $\mathcal{V}$.

The semantic of a temporal property depends on to what π_ν is related to: (1) it can define a property on timeline fragments and values associated to them or (2) it can define a property on the time frames of occurrence and values of events in an envelope. Let $\mathcal{T}(\rho, \mathcal{TL})$ the set of fragments of a timeline $\mathcal{TL}$ that verify π_τ and whose associated values verify π_ν[8]. Similarly, let $\mathcal{E}(\rho, \mathcal{E}v)$ be the set of events in an envelope $\mathcal{E}v$ whose frames of occurrence verify π_τ and associated values verify π_ν[9]. We say that a temporal property ρ is *pertinent* to a timeline $\mathcal{TL}$ (or to an envelope of events $\mathcal{E}v$) if $\mathcal{T}(\rho, \mathcal{TL}) \neq \emptyset$ (or $\mathcal{E}(\rho, \mathcal{E}v) \neq \emptyset$). We define a predicate $\mathcal{P}ert(\rho, \mathcal{TL})$ (or $\mathcal{P}ert(\rho, \mathcal{E}v)$) that is true if and only if

[8] $\mathcal{T}(\rho, \mathcal{TL}) = \{\langle \tau, \nu \rangle \in \mathcal{TL} | \pi_\tau(\tau) \wedge \pi_\nu(\nu(\tau))\}$.
[9] $\mathcal{E}(\rho, \mathcal{E}v) = \{\epsilon \in \mathcal{E}v | \pi_\tau(\tau_\epsilon) \wedge \pi_\nu(\nu_\epsilon)\}$.

a temporal property is pertinent to a timeline or an envelope of events.

Temporal properties are the building blocks of *synchronizations*. A synchronization is a conditional statement that links in a cause-effect relationship sets of temporal properties on timelines and envelopes of events.

A synchronization γ is a statement $\rho^r \rightarrow \langle \mathcal{S}, \mathcal{R}_\tau, \mathcal{R}_\nu \rangle$ where ρ^r is a temporal property called "reference", $\mathcal{S} = \{\rho_i^{\mathcal{S}}\}$ is a set of temporal properties called "supports", $\mathcal{R}_\tau = \{r_\tau(\rho_1, \ldots, \rho_n) : (2^{\mathcal{H}})^n \rightarrow \{\top, \bot\}\}$ is a set of n-ary relations on time frames related to temporal properties $\rho_i \in \mathcal{S} \cup \{\rho^r\}$ and $\mathcal{R}_\nu = \{r_\nu(\rho_1, \ldots, \rho_n) : \mathcal{V}^n \rightarrow \{\top, \bot\}\}$ is a set of n-ary relations on timeline values or event values related to temporal properties $\rho_i \in \mathcal{S} \cup \{\rho^r\}$.

A synchronization is a rule with a *reference* (that triggers the application of the synchronization), some *supports* and some n-ary relations on time frames and timelines and/or event values (that describe the consequence triggered by the reference). Two concepts have to be formalized to define how a synchronization is applied: when it is *applicable* to a timeline or to an envelope of events and when it is *satisfied* by a timeline and an envelope of events.

Intuitively the idea is that a synchronization is applicable when a timeline or an envelope of events contains fragments or events that verify the temporal property of the synchronization reference and it is satisfied by a timeline and an envelope of events if they contain fragments and symbolic events that verify the supporting temporal properties as well as the n-ary relations between them and the fragments or events that verify the reference. Formally the definitions are based on the pertinence of temporal properties: a synchronization $\gamma = \rho^r \rightarrow \langle \mathcal{S}, \mathcal{R}_\tau, \mathcal{R}_\nu \rangle$ is *applicable* to a timeline $\mathcal{TL}$ or to an envelope of events $\mathcal{E}v$ if $\mathcal{P}ert(\rho^r, \mathcal{TL})$ or $\mathcal{P}ert(\rho^r, \mathcal{E}v)$. We define a predicate $\mathcal{A}pp(\gamma, \mathcal{TL}, \mathcal{E}v) \leftrightarrow \mathcal{P}ert(\rho^r, \mathcal{TL}) \vee \mathcal{P}ert(\rho^r, \mathcal{E}v)$.

A synchronization $\gamma = \rho^r \rightarrow \langle \mathcal{S}, \mathcal{R}_\tau, \mathcal{R}_\nu \rangle$ applicable to a timeline $\mathcal{TL}$ or to an envelope of events $\mathcal{E}v$ is *satisfied* by $\mathcal{TL}$ and $\mathcal{E}v$ if it exists, for each assignment $\mathcal{A}(\rho^r, \langle \tau, \nu \rangle)$ of time frames and values pertinent to the reference[10], an assignment $\mathcal{A}(\rho_i^{\mathcal{S}}, \langle \tau_i, \nu_i \rangle)$ of time frames and values pertinent to each support, such that:

- $\forall r_\tau(\rho_1, \ldots, \rho_n) \in \mathcal{R}_\tau,\ r_\tau(\tau_0, \ldots, \tau_n) = \top,\ \tau_i = \mathcal{A}(\rho^i, \langle \tau_i, \nu_i \rangle)$;
- $\forall r_\nu(\rho_1, \ldots, \rho_n) \in \mathcal{R}_\tau,\ r_\nu(\nu_0, \ldots, \nu_n) = \top,\ \nu_i = \mathcal{A}(\rho^i, \langle \tau_i, \nu_i \rangle)$.

We define a predicate $\mathcal{S}at(\gamma, \mathcal{TL}, \mathcal{E}v) = \top$ if and only if it exists an assignment that verifies the above properties.

The definition of domain theory is based on the definitions of applicability and satisfiability of synchronizations. A *d*omain theory $\mathcal{DT}$ for a domain constituted by a set of components is a set Γ of synchronizations. A set of timelines $\mathcal{TL}$ for D and an envelope of events $\mathcal{E}v$ for D *satisfy* a domain theory $\mathcal{DT}$ if any applicable synchronization in $\mathcal{DT}$ is satisfied by $\mathcal{TL}$ and $\mathcal{E}v$. We define a predicate $\mathcal{S}at(\mathcal{DT}, \mathcal{TL}, \mathcal{E}v) = \top \leftrightarrow \forall \gamma \in \Gamma, \mathcal{A}pp(\gamma, \mathcal{TL}, \mathcal{E}v) \rightarrow \mathcal{S}at(\gamma, \mathcal{TL}, \mathcal{E}v)$.

Problems and Solutions. A problem can be defined in general by stating (1) an envelope of behaviors that describe (by means of a set of timelines $\mathcal{TL}_0$) the initial status of the system, (2) an envelope of behaviors and events that describe (by means of a pair $\langle \mathcal{TL}_g, \mathcal{E}v_g \rangle$) the goal condition, in terms of behaviors considered acceptable and events that have to be allocated in a solution of the problem and (3) a domain theory $\mathcal{DT}$ specifying the temporal properties required for valid timelines and events applied to the system. Formally A *p*roblem $\mathcal{P}$ in a TRF domain is a tuple $\langle \mathcal{DT}, \mathcal{TL}_0, \mathcal{E}v_g, \mathcal{TL}_g \rangle$ where:

- $\mathcal{DT}$ is a domain theory;
- $\mathcal{TL}_0$ is an initial timeline;
- $\mathcal{E}v_g$ is a goal envelope of events;
- $\mathcal{TL}_g$ is a goal timeline.

The goal of a TRF solver is to identify an envelope of events $\mathcal{E}v_s$ containing at least one instantiation of all the events in the envelope $\mathcal{E}v_g$ such that (1) allows an extraction of a timeline from $\mathcal{TL}_0$, the timeline extracted $\mathcal{TL}_s$ describes a subset of acceptable behaviors $\mathcal{TL}_g$ and $\langle \mathcal{TL}_s, \mathcal{E}v_s \rangle$ satisfies the domain theory. Formally a *solution* $\mathcal{S}ol(\mathcal{P})$ of a problem $\mathcal{P}$ is a tuple $\langle \mathcal{TL}_s, \mathcal{E}v_s \rangle$ where:

- $\mathcal{E}v_g \subseteq \mathcal{E}v_s$[11];
- $\mathcal{TL}_s = \mathcal{E}xtr(\mathcal{TL}_0, \mathcal{E}v_s)$;
- $\mathcal{TL}_s \subseteq \mathcal{TL}_g$;
- $\mathcal{S}at(\mathcal{DT}, \mathcal{TL}_s, \mathcal{E}v_s)$.

General definitions of domain theory, problem and solutions can be instantiated to derive specific definitions for planning with state variables, scheduling with resources and integrated planning and scheduling problems with state variables and resources. For instance, a timeline based planning problem with state variables (Muscettola 1994; Frank and Jónsson 2003) can be defined for a domain with a set of state variable components by defining synchronizations among values taken by fragments of the state variables timelines and specifying a goal timeline $\mathcal{TL}_g$ where in some intervals the value taken by the state variable is specified while in other intervals there is no specified value (this timeline does not satisfy the domain theory, because some intervals are not specified). A solution to a timeline based planning problem with state variables can be provided as a timeline $\mathcal{TL}_s$ that satisfy the domain theory and that is compliant with the fragment specified in the problem (this property is guaranteed by the fact that $\mathcal{TL}_s \subseteq \mathcal{TL}_g$). Goal of the planner in this case would be to find the proper set of events to narrow the values taken by the timeline in unspecified intervals till a timeline that satisfies all the synchronizations among the values is found.

[10] An assignment $\mathcal{A}(\rho, \langle \tau, \nu \rangle)$ of pertinent time frames and values to a temporal property ρ is a pair $\langle \rho, \langle \tau, \nu \rangle \in \mathcal{T}(\rho, \mathcal{TL}) \rangle$ if ρ is a temporal property on a timeline, or a pair $\langle \rho, \langle \tau_\epsilon, \nu_\epsilon \rangle, \epsilon \in \mathcal{E}(\rho, \mathcal{E}v) \rangle$ if ρ is a property on an envelope of events.

[11] $\mathcal{E}v = \langle \mathcal{E}, T, N \rangle \subseteq \mathcal{E}v' = \langle \mathcal{E}', T', N' \rangle \leftrightarrow \mathcal{E} \subseteq \mathcal{E}'$ and the projections of T', N' over elements in $\mathcal{E}$ are subsets of T and N.

A scheduling problem can be defined for a domain with a set of resource components by specifying an initial availability profile for the resources as a timeline and an envelope of events containing the activities to schedule or the production/consumptions to schedule as a goal condition. A solution to a timeline based scheduling problem would be the temporal relations added to the envelope of activities to be scheduled in order to eliminate any overlap among their frames of occurrence that can lead to fragments of the resulting profile timeline where there is an overconsumption.

An example. Giving back to the rover domain introduced above, we can provide some examples of synchronizations. In fact there are some constraints that had to be satisfied in order to correctly use the rover. The first one (C1) was that when the rover is taking a picture, it must be stable at a given location. The second (C2) was that when the rover is moving, the pan-tilt unit must be in a "rest" position (i.e., at angle $\langle 0, 0 \rangle$). We have modeled these requirements stating the following constraints:

CAM.TAKEPIC($?file\_id$) DURING MS.AT($?x, ?y$) (C1)
MS.GOTO($?x, ?y$) DURING PTU.POINTINGAT($0, 0$) (C2)

The goal TAKEPICTURE($?x, ?y, ?pan, ?tilt, ?file\_id$) can be achieved by the rover by: (a) taking a picture with id = $?file\_id$, with the rover in $\langle ?x, ?y \rangle$ and the pan-tilt unit pointing to $\langle ?pan, ?tilt \rangle$ and (b) dumping the picture. To take a picture on a given position and pan-tilt unit orientation, can be achieved by (a.1) moving the rover to $\langle ?x, ?y \rangle$, (a.2) moving the pan tilt unit to $\langle ?pan, ?tilt \rangle$ (a.3) taking a picture with the camera with an id = $?file\_id$. To dump a picture with a given id can be achieved by (b.1) dumping the picture when the dump window is available. Hence we add to the model the following synchronization:

MT.TAKEPICTURE($?x, ?y, ?pan, ?tilt, ?file\_id$)
CONTAINS
$\langle$ CAM.TAKEPIC($?file\_id$) BEFORE COMM.DUMP($?file\_id$),
CAM.TAKEPIC($?file\_id$) DURING MS.AT($?x, ?y$),
CAM.TAKEPIC($?file\_id$) DURING
PTU.POINTINGAT($?pan, ?tilt$),
COMM.DUMP($?file\_id$) DURING VW.AVAILABLE() $\rangle$

The goal GOTO($?x, ?y$) can be achieved by driving the rover to $\langle ?x, ?y \rangle$. Hence we need to add to the model just the following synchronization:

MT.GOTO($?x, ?y$) MEETS MS.AT($?x, ?y$)

The memory management is specified by means of synchronizations between the actions of taking a picture (which consume memory) and the action of dumping a picture (which free the memory):

CAM.TAKEPIC($?file\_id$) START-AT MEM.CONSUME(1)
COMM.DUMP($?file\_id$) END-AT MEM.PRODUCE(1)

Figure 4 shows a graphical representation of the synchronizations involved in the action to take a picture. In this domain a problem is stated by specifying the initial value for the component timelines and an envelope of events stating the goals to be achieved. Figure 5 shows a problem where the initial conditions describe a rover in a rest position in $\langle 0, 0 \rangle$ and two goals to be achieved: (1) to take a picture in a position $\langle 2, 3 \rangle$ with the pan-tilt unit in $\langle -35, -45 \rangle$ and a picture id "$pic1$" and (2) to go in $\langle 0, 0 \rangle$ after having taken the picture. Figure 6 shows a solution for the problem.

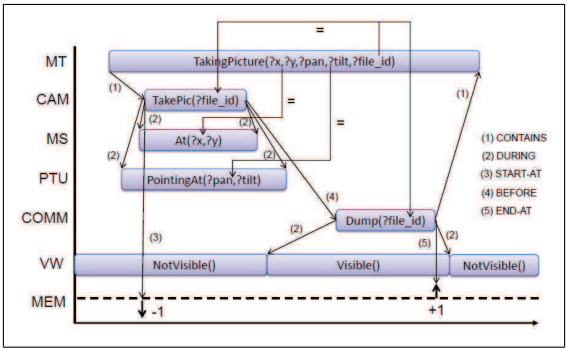


Figure 4: Synchronization

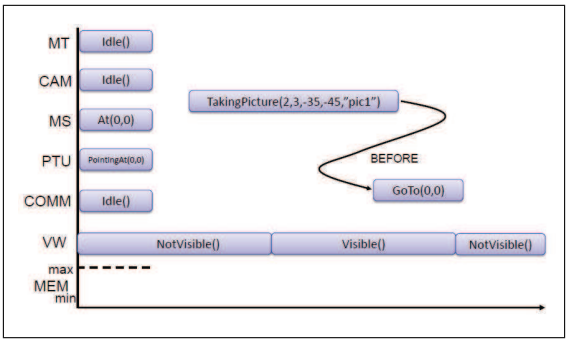


Figure 5: Problem

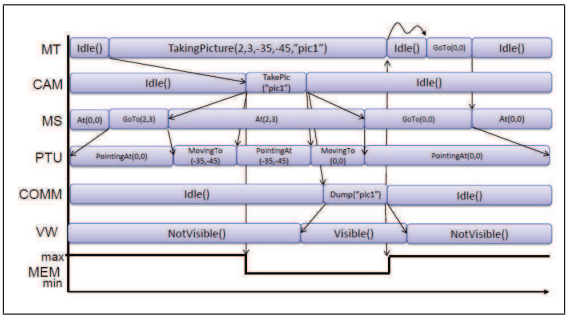


Figure 6: Solution

Problem Solving with Timelines

Problem solving in the TRF can be described as a process of identifying *flaws*, i.e. problems on timelines and envelopes of events regarding the satisfiability of the domain theory, the extractability of the timelines or the compliance with goal conditions, and *solving steps*, i.e. proper actions to remove flaws.

We define, given a set of timelines $\mathcal{TL}$, an envelope of events $\mathcal{E}v$ and a domain theory $\mathcal{DT}$, three types of flaws: a *d*omain theory flaw ϕ_{DT} is a tuple $\langle \gamma, \mathcal{TL}, \mathcal{E}v \rangle$ where $\gamma \in \mathcal{DT}$ is a synchronization such that $\mathcal{A}pp(\gamma, \mathcal{TL}, \mathcal{E}v) \wedge \neg\mathcal{S}at(\gamma, \mathcal{TL}, \mathcal{E}v)$. A *t*imeline extraction flaw ϕ_{TL} is a tuple $\langle \mathcal{TL}, \mathcal{E}v \rangle$ such that $\mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$ does not exist. A *g*oal condition flaw ϕ_{GC} is a tuple $\langle \mathcal{TL}, \mathcal{E}v, \mathcal{TL}', \mathcal{E}v' \rangle$ such that $\mathcal{TL} \not\subseteq \mathcal{TL}' \vee \mathcal{E}v \not\subseteq \mathcal{E}v'$.

The timeline based solver can remove a flaw by modifying the envelope of events that influences the behaviors of the system and re-extracting the timeline that describe the new set of behaviors in order to obtain a new envelope and a new timeline without the flaw. We define a generic solving step as a function of a flaw and an envelope of events. Formally given a flaw ϕ and an envelope of events $\mathcal{E}v$, a solving step Ω for ϕ in $\mathcal{E}v$ is a function $\Omega(\phi, \mathcal{E}v) = \mathcal{E}v'$.

Depending on the type of action taken, a solving step can be a *scheduling step* if it only adds relations among existing events or a *planning step* if it also adds events to the envelope. Depending on the type of flaw, different types of solving steps are appropriate to remove the flaw.

A domain theory flaw $\phi_{DT} = \langle \gamma, \mathcal{TL}, \mathcal{E}v \rangle$ can be removed by adding events to $\mathcal{E}v$ or by adding relations among the events in $\mathcal{E}v$ in order to either make γ not applicable to $\langle \mathcal{TL}, \mathcal{E}v \rangle$ or to make it satisfied by $\langle \mathcal{TL}, \mathcal{E}v \rangle$. Hence either a planning or a scheduling step are appropriate for this type of flaw. A timeline extraction flaw $\phi_{TL} = \langle \mathcal{TL}, \mathcal{E}v \rangle$ can be removed only by adding relations among the events in $\mathcal{E}v$ in order to narrow their time frames of occurrence to allow their fragmentation. Hence a scheduling step is appropriate for this type of flaw. A goal condition flaw $\phi_{GC} = \langle \mathcal{TL}, \mathcal{E}v, \mathcal{TL}', \mathcal{E}v' \rangle$ can be removed either by adding events to $\mathcal{E}v$ in order to directly meet goal conditions specified as events in $\mathcal{E}v'$ missing in $\mathcal{E}v$ or in order to generate fragments in $\mathcal{TL}'$ missing in $\mathcal{TL}$ trough the timeline extraction process or by adding relations to the envelope of events in order to cut instantiations of events or timeline frames that violate the goal conditions. Hence either a scheduling or a planning step can be appropriate to solve this type of flaw.

A general solving process with timelines is an iterative process of timeline extraction, flaw collection and planning or scheduling steps based on (1) a $CollectFlaws$ procedure that analyzes the current status of the timelines and the envelope of events against the goal condition and the domain theory and produces a list of flaws; (2) a $ChooseFlaw$ heuristic procedure that analyzes the flaws and choose which one has to be resolved first and (3) a $ChooseSolvingStep$ procedure that on the basis of the flaw and the status of the timelines and envelope of events select a planning or scheduling step to be applied in order to remove the flaw.

The building blocks of the solving algorithms can be designed for being general, domain independent (up to a given extent) and reusable with different configurations of the components in a domain (an example of implementation for state variable and resource components is the OMPS planner described in (Fratini, Pecora, and Cesta 2008)) or can be designed on purpose for a specific problem modeled with timelines and synchronizations among them.

Function solve($\mathcal{P} = \langle \mathcal{DT}, \mathcal{TL}_0, \mathcal{E}v_g, \mathcal{TL}_g \rangle$)

$\mathcal{TL} = \mathcal{TL}_0$;
$\mathcal{E}v = null$;
$\Phi = CollectFlaws(\mathcal{TL}, \mathcal{E}v, \mathcal{DT}, \mathcal{E}v_g, \mathcal{TL}_g)$;
while $\Phi \neq \emptyset$ **do**
  $\phi \leftarrow ChooseFlaw(\Phi)$;
  $\Omega \leftarrow ChooseSolvingStep(\phi, \mathcal{TL}, \mathcal{E}v)$;
  $\mathcal{E}v \leftarrow \Omega(\phi, \mathcal{E}v)$;
  $\mathcal{TL} \leftarrow \mathcal{E}xtr(\mathcal{TL}, \mathcal{E}v)$;
  $\Phi = CollectFlaws(\mathcal{TL}, \mathcal{E}v, \mathcal{DT}, \mathcal{E}v_g, \mathcal{TL}_g)$;
end
return $\langle \mathcal{TL}, \mathcal{E}v \rangle$;

Conclusions

The formulation presented in this paper, although at a very high level, aims at identifying building blocks of the general process of modeling and solving a generic problem with timelines by means of planning and scheduling steps. The lack of standardization of the timeline-based approach has lead to an objective difficulty in spreading and re-using information, software and languages, but such a standardization has to be grounded on a widely accepted definition of basic concepts like "timeline", "domain theory", "problem" and "solution of a problem". Without this agreement, it is impossible to design a general language to compare different planners or to analyze similarities and differences among different architectures. This paper describes and formalizes how timelines are represented and used in the ESA APSI software platform, with the aim of specifying requirements for a future standardization that would be compatible with the algorithms and architectures that we have been proposing over time.

Acknowledgments. Amedeo Cesta is partially supported by MIUR under the PRIN project 20089M932N (funds 2008).

References

Cesta, A.; Cortellessa, G.; Fratini, S.; Oddi, A.; and Bernardi, G. 2011. Deploying interactive mission planning tools - experiences and lessons learned. *JACIII* 15(8):1149–1158.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8(4):339–364.

Fratini, S.; Cesta, A.; De Benedictis, R.; Orlandini, A.; and Rasconi, R. 2011. Apsi-based deliberation in goal oriented autonomous controllers. In *ASTRA 2011. 11^{th} Symposium on Advanced Space Technologies in Robotics and Automation*.

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.

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

Verfaillie, G.; Pralet, C.; and Lematre, M. 2010. How to model planning and scheduling problems using constraint networks on timelines. *Knowledge Eng. Review* 25(3):319–336.

Design Concepts for a new Temporal Planning Paradigm

J. M. Delfa Victoria*
Technische Universität
Darmstadt, Germany

Nicola Policella
ESA-ESOC, Germany

Yang Gao
University of Surrey, UK

Oskar von Stryk
Technische Universität
Darmstadt, Germany

Abstract

Throughout the history of space exploration, the complexity of missions has dramatically increased, from Sputnik in 1957 to MSL, a Mars rover mission launched in November 2011 with advanced autonomous capabilities. As a result, the mission plan that governs a spacecraft has also grown in complexity, pushing to the limit the capability of human operators to understand and manage it.

However, the effective representation of large plans with multiple goals and constraints still represents a problem. In this paper, a novel approach to address this problem is presented. We propose a new planning paradigm named HTLN, intended to provide a compact and understandable representation of complex plans and goals based on Timeline planning and Hierarchical Temporal Networks. We also present the design of a planner based on HTLN, which enables new planning approaches that can improve the performance of present real-world domains.

1 Introduction

In the past decades Automated Planning & Scheduling (P&S) has become a well studied field. Nevertheless, there is an important gap between academic and real-world systems that needs to be continuously bridged in both directions to make planning theory aware of the complexity of real-world problems and to transfer innovations in theory to applied planners. In this paper we consider the planetary rover as an example of a real-world problem dealing with critical operations, uncertainty and complex systems and goals that can be easily generalized to many other real scenarios on Space and Earth like rescue robots or autonomous vehicles.

The scope of this work is to define a new timeline planning paradigm for real-world scenarios such as the rover-world problem, aimed to manage temporal problems with uncertainty. The objective is to create a Planning & Scheduling System able to generate robust plans for execution, or more specifically, responsive to the uncertainty and dynamics of the environment. A second objective is to produce more understandable plans for human experts.

Regarding its design, the system must retain sufficient generality in order to be used to design a knowledge-based, domain-independent timeline planner which can take advantages from different planning techniques such as HTN, CSP or MC.

After studying the problem, we have identified a number of key planning technologies from both the academic and applied worlds that represent the ingredients of a new planning paradigm called Hierarchical Timeline Networks (HTLN). It is based on hierarchical hypergraphs to represent the structure of problems, where complex goals in the upper layers of the hierarchy are decomposed in more specific subgoals, grouped together in sub-hypergraphs in lower level layers.

The paper is organized as follows: first, the planetary rover problem is described, then the proposed approach is discussed and the mathematical background of HTLN is introduced. Next, the design of a planner based on HTLN paradigm is presented. The paper concludes with some final remarks and a discussion of future work.

2 Planning for Autonomous Rover Missions

The planetary rover is a type of robot equipped with a locomotion system, typically wheeled, to move across hazardous terrain. Its hardware is divided between *payload* and *platform*. The former includes all the instrumentation dedicated to perform science, while the remaining sub-systems in support of these activities are considered the platform. They can serve different purposes both on Earth and space, such as rescue missions, surveillance or planetary exploration.

We use as a reference a planetary rover scenario with a single robot that must perform a traverse through uneven, unknown terrain towards a target, which can be a rock or geographical feature. The rover then performs a number of scientific activities such as taking pictures or studying the chemical composition of samples extracted with a driller.

This scenario shares with other problems in the robotic domain a number of specific characteristics, listed in Table 1, making it very hard from a planning point of view.

In this context, additional effort is required in the management of the plan complexity for two reasons. First, as the complexity of plans increases, the capability of humans to understand and manage them decreases. However, human operators need to understand the outcome of the planners and the reasons leading to generate it. This problem can be addressed by making the planner goal-based (Dvo-

Property	Description	Field
Uncertainty - Dynamic environment	The environment can spontaneously change its state due to external events	Both
Uncertainty - Partial observability	Some aspects of the state of the world are unknown. It has three consequences: Planning based in a complete understanding of the world is not feasible, some of the assumptions considered during planning might be wrong and new relevant information for the plan might be discovered only during execution time	Both
Uncertainty - Non determinism	Not possible to estimate with precision the outcome of the robot actions	Both
Hw/Sw/Problem complexity	The complexity of space missions has increased exponentially (Dvorak 2009; Bajracharya, Maimone, and Helmick 2008)	Both
Highly constrained	Robotic operations required highly constrained models to avoid malfunctions	Both
Restricted communications	Some scenarios do not allow continuous or real-time communications with the robot. For example, the round trip of a radio signal from Earth to Mars can take more than forty minutes	Both
Low CPU performance	Space-oriented processors have much lower performance than those integrated in conventional computers (Berger 2009)	Space
Failure recovery	The possibilities to recover a mission in case of a spacecraft failure are very limited. Therefore, safety and V&V play a major roll in space missions	Space

Table 1: Properties of autonomous robots exploration on space and Earth

rak, Amador, and Starbird 2008), where goals are organized in a hierarchical structure that contains different levels of abstraction (Ghallab, Nau, and Traverso 2004). Navigating deeper in the structure allows the operator to learn how complex goals decompose in sub-tasks. Second, algorithms might be also benefited, as it is easier to isolate and fix parts of the plan that fail as a group. Different techniques such as intelligent backtracking and fast-forwarding can be used in this context.

A higher level of autonomy is crucial to increase the science return and decrease mission costs at the same time (Muscettola et al. 1998; Chien et al. 2006). However, autonomy comes at a price: the higher the level of autonomy desired, the higher the complexity of the system and level of detail in the knowledge to be added. The problem is that during the design of a planning system, part of the knowledge of the human experts is not captured. For this reason the planner should be able to collaborate with experts in a mixed-initiative strategy (Bresina et al. 2003), where humans can manually add new elements to the problem and force the planner to satisfy them.

Finally, it must be possible to express the temporal evolution of events and actions. Temporal planners combined with CSP techniques have demonstrated to be very effective for problems involving dynamic environments and have become the main technology in the space domain (Frank and Jónsson 2003; Fratini, Pecora, and Cesta 2008; Ghallab and Laruelle 1994). In order to increase the robustness at execution-time, a number of techniques like temporal flexible plans or continuous replanning are applied. As a consequence, the planner should be able to generate partially defined plans to be further completed once the information required is available, possibly at execution time, based on the least commitment principle.

3 Related work in AI Planning

Even though the techniques mentioned before are suitable to address some of the problems presented in Table 1, none of them completely satisfies all the characteristics and requirements described above. In the specific case of space robotics, the high cost of a mission plus the fear of losing the spacecraft due to software malfunctions have so far prevented a deeper integration of these technologies with the exception of two notorious attempts: Deep Space 1 (Muscettola et al. 1998; Jonsson et al. 2000) and Earth Observation 1 (Chien et al. 2004).

Significant work has been conducted in the field of CSP planning, where different versions of arc and path-consistency algorithms have been used in several planners (Mackworth 1977; Bessière 1994; Singh 1995). Even though stating the planetary rover as a pure CSP problem is possible, this is not straightforward and can result in a complex representation. For this reason, alternative techniques should be taken under consideration to represent time, uncertainty or goal decomposition.

With respect to hierarchical task networks, HTN (Erol, Hendler, and Nau) planners have been successfully applied in real problems (Wilkins and desJardins 2000) such as SIPE-2 (Wilkins et al. 1995) or O-Plan (Tate, Drabble, and Kirby 1994) but they show some limitations in dealing with uncertainty as well as temporal domains and do not allow interaction with human experts. Even though SIPE-2 can define vagueness with respect to interval relations in terms of minimum and maximum duration, our objective is to have a more powerful mechanism to represent partial plans.

Regarding timeline planners, there are several examples that have been used in the space sector, such as Aspen (Chien et al. 1997; Fukunaga et al. 1997), Europa (Frank and Jónsson 2003) or IxTet (Ghallab and Laruelle 1994), but they are not completely oriented to uncertainty or do not provide hierarchical task representation.

For planning under uncertainty two techniques have been widely used: Markov Decision Processes (MDP) (Cassandra, Kaelbling, and Littman 1994) and Model Checking (MC) (Clarke, Long, and McMillan 1989; Bertoli et al. 2001). The rover problem presents uncertainty in all its possible dimensions (Table 1) making it a crucial aspect of the planner design. By assigning costs and rewards, it is possible to represent desires about goals, while non determinism is expressed by means of probabilities assigned to the different choices available to achieve a goal. However, in MDP the policies (pairs states-actions) are defined beforehand, which do not represent a good approximation for real scenarios where the number of states might be infinite. It seems better to provide a model of the system and let the planner calculate how to achieve a specific state, taking into account the model in a similar way to MC, where policies only contain the states involved in transitions to achieve the goals.

4 Theoretical background

HTLN relies on the idea of merging timeline planning and HTN (Erol, Hendler, and Nau) techniques. With respect to timeline planning, HTLN is based on the formalism of APSI (Fratini, Pecora, and Cesta 2008). Regarding HTN, we have used cyclic hypergraph structures to represent the hierarchical decomposition of goals into sub-goals (Figure 2).

4.1 Timeline planning principles

In APSI, a planner requires two inputs: **model** and **problem** and produce one output, the **plan**. The model contains a formal description of all the systems which activities must be planned. Each system is modeled as an automaton (named $component$ in APSI) composed of states (component decisions (cd) in APSI) and relations (rlt) between the cd's, which represent a super-set of the classical transitions in automata theory. Each component has related a timeline that represents a more or less flexible sequence of cd's that represent the plan.

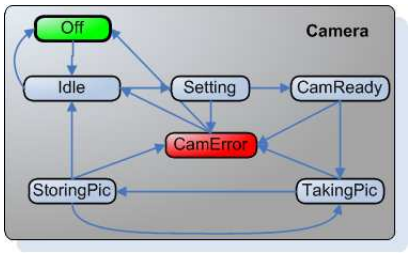


Figure 1: Component Camera automaton

A problem is represented as a decision network (dn), that is, a graph that contains a set of initial conditions ic representing facts that the planner does not need to justify, and $goals$, which the planner must justify using the model. Both ic's and $goals$ are represented as sets of cd's (the nodes) and/or rlt's (the edges) of the graph.

In case all goals are satisfied, the result is a fully supported dn called plan from which the timelines are extracted.

4.2 Hypergraph theory

Hypergraphs have been widely studied and applied in different areas such as the definition of complex data structures or optimization (Berge 1990; Gallo et al. 1993). They combine graph and set theories (Rugg 1983): a hypergraph is a generalization of a graph where a hyperedge can connect any number of nodes; it can be also considered as a set of hyperedges, where each hyperedge is itself a set which elements are the nodes it connects. It is formally defined as $H = (N, E)$, where N is the set of nodes/hypernodes and E is the set of hyperedges. The model of hypergraph used here diverges from the traditional as it contains in addition a second way for grouping nodes (besides the hyperedge) called hypernode (Damaschke 2009). While hyperedges are used to specify relations (such temporal or parameter relations) between nodes, a hypernode is used to represent the group of nodes in which a complex goal is decomposed. Therefore, a hypernode can be managed by the planner as a single node (seen as a black box) or as a sub-problem.

HTLN structure is organized horizontally with nested hypergraphs and vertically by mean of directed hierarchical hypergraphs.

The former are used to represent the decomposition of a complex element (problem or goal) in sub-elements. A super-hypergraph can contain several sub-hypergraphs and so forth until all the elements of each sub-hypergraph are simple. The idea behind this structure is to help the planner to manage all types of element (complex or simple) in a common way. However, this structure is not sufficient to represent the evolution of a problem. During the planning process, each complex goal is replaced by one of its possible decompositions (sub-goals). It is important to keep track of this process to backtrack in case the planner finds no solution. To represent this relation, a tree data structure seems to be more appropriate than nesting hypergraphs. Nevertheless, a tree can be generalized as a directed hypergraph removing the constraints by which nodes must have at most one parent and no cycles allowed. The resulting structure is a directed hierarchical set of hypergraphs. Each hypergraph has a set of parents (except the root) and children (except the leaves), where a parent represents a less evolved version of the child.

Figure 2 represents the hierarchical hypergraph structure. Ellipses represent problems and sub-problems while circles represent goals. Arrows represent the decomposition of goals into sub-goals while lines represent binary relations (temporal and parameter). For the sake of simplicity, there is no relation involving more than two nodes. The nomenclature used (represented as labels in the figure) is the following:

- p_i and p_{i+1}: Problem at level i and its decomposition
- cd^s: Simple node (definition in Section 5)
- cd^c: Complex node (definition in Section 5)
- $d^{\uparrow}$ and $d^{\downarrow}$: Node to be decomposed (parent) and its decomposition (child). Formally: $d^{\uparrow}(to) = d^{\downarrow} \wedge d^{\downarrow}(from) = d^{\uparrow}(to)$
- dn^{sup} and dn^{sub}: $super - dn$ and one of its $sub - dn$'s
- r_{dec}: Decomposition relation, where the source is a cd^c and the target is a $sub - dn$. Even though a cd^c can be decomposed in different $sub - dn$'s, just one is used in the problem, being the rest inactive $d^{\downarrow}$
- r_{temp}, r_{param}: Temporal and parameter relations respectively

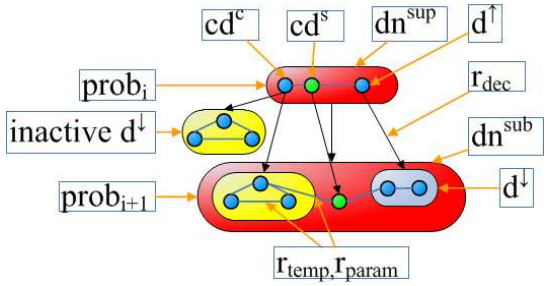


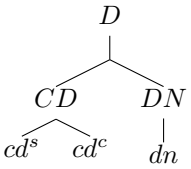
Figure 2: HTLN structure

Following section will present the novelties of HTLN and some formal definitions.

5 Hierarchical Timeline Networks (HTLN)

HTLN extends the APSI formalism in the following areas:

- Uses a HTN approach, allowing the definition of complex goals and its decomposition in sub-plans. HTLN defines three types of decisions which organization is displayed in the tree bellow:
  - cd^s: Simple component decision that can be directly "executed". It is represented as a node of the graph that cannot be further decomposed
  - cd^c: Complex component decision that must be decomposed in order to be "executable". It is represented as a node of the graph, that is, a node with a set of decomposition relations
  - dn: A decision network represents a special type of decision represented as a hypernode
- Common representation for all relation types, extended as n-ary relations between sets of cd's
- HTLN allows partially defined, partially ordered plans in order to handle uncertainty (see Section 6)



5.1 Component Decision (cd)

A cd on a $component$ defines that a state of the component automaton holds for a given interval of time and can be used to express goals and constraints. It is formally defined as follows:

$$cd = \left\langle id, type, \{values\}, \{rlts\}, \{props\} \right\rangle \quad (1)$$

where:

- id: Identifies uniquely the cd
- $type$: Type of cd as defined in Section 5
- $\{values\}$: Each state has associated a value defined over a qualitative or quantitative domain. Notice that the same cd representation is used to represent actions of a component or consumption/production of certain resource
- $\{rlts\}$: Set of relations that affect this cd
- $\{props\}$: A cd can have specific properties such as relevance, uncertainty, parameters, etc. which values are assigned or retrieved via specific functions, formally: $f_{property}(cd)$

Like in dn's, a cd can be added or deleted by the planner or human user.

dn's as cd's

A decision network $dn_1^{\downarrow}$ that represents a decomposition of a goal $d_1^{\uparrow} \in CD^c$ inherits the values, relations and properties of $d_1^{\uparrow}$.

As a consequence, a planner can apply its normal operators over a set of cd's viewed as a special node called hypernode represented by the dn. This approximation makes the planner more powerful, as it can perform planning over groups of cd's, and simpler, as no special code is required to handle dn's.

Type	Constraint	Directed	Arity
Parameter	Yes	No	n-ary
Temporal	Yes	No	n-ary
Decomposition	No	Yes	n-ary

Table 2: Relations in HTLN

5.2 Relation (rlt)

A relation is used to describe a common property between some cd's. It is represented as a hyperedge that joins together the cd's involved in the relation. A $constraint$ is a special type of relation that limits the possible states in which a $component$ can be and the duration of this status. All rlt's in HTLN are n-ary in order to construct more understandable problems and plans for humans (Little and Ghafoor 1990).

A relation is formally defined as follows:

$$rlt = \left\langle id, type, \{elements\}, \{props\} \right\rangle \quad (2)$$

where:

- id: Identifies uniquely the rlt
- $type$: The relations supported by HTLN are showed in Table 2
- $\{elements\}$: A relation contains a number of elements $e_i \mid 0 < i \leq n$. A n-ary relation is applied iteratively to each consecutive pair of elements in the following way: $rlt = \left\langle id, type, \{e_i, e_{i+1}\}, \{props\} \right\rangle \in [0, n)$. An element is formally described as follows: $e_i = \left\langle cd, \{values\}, \{constraints\} \right\rangle$
  - cd: Component decision affected by the relation. It can be a cd^c, cd^s or dn
  - $\{values\}$: Similar to the $cd's$, this field is used to define a domain of values that the relation can assume. Depending on the type of relation, these values can be:
    - Parameter: Qualitative or quantitative domain that delimits the value of the parameter
    - Temporal: Define an interval $[l, u]$ of real values that delimit the duration of the relation
    - Decomposition: Not applicable (see Section 5.2)
  - $\{constraints\}$: Each relation might have appended some constraints
- $\{props\}$: A rlt, as a cd, can have specific properties, formally: $f_{property}(rlt)$

Each of the relations are introduced in detail below.

Parameter (f_{param}). A parameter relation is used to constraint the value between the parameters of different cd's of the dn. It is represented as a n-ary non-directional constraint. Formally

$$rlt = \left\langle id, f_{param}, \{elements\}, \{props\} \right\rangle \quad (3)$$

where:

- f_{param}: There are two type of relations: $f_{param} \in (=, \neq)$
- $\{elements\}$: List of parameters involved in the constraint

Temporal (f_{temp}). In HTLN, temporal constraints represent a superset covering other types of constraints like transitions in conventional automata theory or synchronizations between components (Muscettola 1994). Temporal relations can be applied to any kind of $node$, that is, to dn's, cd^c's and cd^s's.

A temporal constraint is represented as a n-ary, non-directional constraint. Formally:

$$rlt = \left\langle id, f_{temp}, \{elements\}, \{props\} \right\rangle \quad (4)$$

where:

- f_{temp}: HTLN uses the set of Allen temporal relations: $Equals$, $Before\,[l,u]$, $Meets$, $Starts\,[l,u]$, $Ends\,[l,u]$, $Overlaps\,[l,u]$ and $Contains\,[l_1,u_1],[l_2,u_2]$, being $[l,u]$ a time interval, where l is the lower bound and u the upper bound. In the special case of $Contains$, the first interval defines the time relation between the starting points and the second the relation between the ending points
- $elements$: List of elements involved in the constraint

Decomposition (f_{dec}). It represents a relation that is applied not only to decisions $d \in D$, but also to other relations in order to generate a more evolved version of the problem. It is represented as a $1:n$ directional relation from a decision $n_1 \in D$ to a set of decisions $\{n_2\}$. The type of n_2 depends on the type of n_1 as follows:

- $n_1 \in CD^s \Rightarrow n_2 \in CD^s$: All simple elements are just copied in the evolved version of the problem
- $n_1 \in CD^c \cup DN \Rightarrow n_2 \in DN$: A complex element, either a cd^c or dn will be decomposed as a dn

The decomposition relation is formally represented as follows:

$$rlt = \left\langle id, f_{prop}, \{\{from\},\{to\}\}, \{props\} \right\rangle \quad (5)$$

The decomposition function relies in some supporting functions that are presented bellow:

- $n_1(from)$: Returns the list of parents of n_1
- $n_1(to)$: Returns the list of children of n_1
- $n_1(active)$: Indicates whether the decision n_1 is active or not
- $f_{sup}(n_1)$: Returns the list of $super-dn$ of n_1
- $f_{sub}(n_1)$: Returns the list of $sub-dn$ of n_1
- $x(nodes)$: List of nodes of x, where x might be a relation or decision network
- $d(edges)$: List of relations of a decision $d \in D$
- $d(params)$: List of parameters of the decision d
- f_{dec}: Decomposition function (which follows the rules defined before)
- $f_{temp}^{Starts}(n_1,n_2)$: Defines a $Start\_Start$ temporal relation between two decisions
- $f_{temp}^{Ends}(n_1,n_2)$: Defines an End_End temporal relation between two decisions
- $f_{search}(domain,n_1)$: Return the decomposition dn's for n_1
- $x \stackrel{+}{=} y$: Addition of the element y to the list $\{x\}$
- $x = y$: Assigns to x the value of y

In addition to the elements presented in Section 4.2, two more are used:

- $domain$: Model of the system containing a description of all components, its simple decisions $cd^s \in CD^s$, complex decisions $cd^c \in CD^c$, decompositions $dn \in DN$ and relations $rlt \in R$
- $\chi(rlt)$: Log that stores all the decompositions performed during planning in support of backtracking in case the planner cannot find a solution

The following algorithms are in charge of decomposing a dn into a more evolved version. This decomposition represents an extra move of the planner in order to transform a problem into a plan.

Algorithm 1: $f_{decNode}(d^{\uparrow}, domain)$

```
begin
    d↓(active) = true
    f_createDec(d↑, d↓)
    dn↑ = f_sup(d↑)
    if dn↑ ≠ null then
        dn↓ = dn↑(to)
        dn↓(nodes) ± d↓
    else
        dn↓ = d↓
    if d↑ ∈ DN then
        ∀d ∈ d↑(nodes), d ∈ D ⇒ f_decNode(d, domain)
        ∀f_r ∈ d↑(edges), f_r ∈ {R_param, R_temp} ⇒
        f_decRel(f_r)
```

Decompose a decision $d^{\uparrow} \in D$ in $d^{\downarrow} \in D$. The algorithm is illustrated in Figure 3.

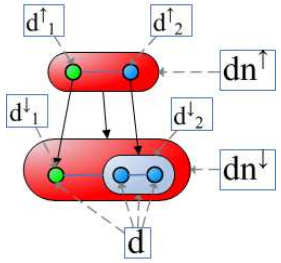


Figure 3: Decomposition of a dn

The algorithm receives as inputs the node to be decomposed ($d^{\uparrow}$) and the decomposition node ($d^{\downarrow}$), which will be selected in different ways depending on the type of $d^{\uparrow}$ (see 5.2:

- $d^{\uparrow} \in CD^s$ (simple decision): $d^{\downarrow}$ contains a copy of $d^{\uparrow}$
- $d^{\uparrow} \in CD^c$ (complex decision): A list of dn candidates to decompose $d^{\uparrow}$ is retrieved from $domain$. One dn is heuristically selected from this list
- $d^{\uparrow} \in DN$ (decision network): A new dn is created to store the decomposition elements of $d^{\uparrow}$

The decision $d^{\downarrow}$ selected for the decomposition is marked as active to distinguish it from the rest of options in which $d^{\uparrow}$ could be decomposed. Then a decomposition relation between $d^{\uparrow}$ and $d^{\downarrow}$ is defined. Following, the $super-dn$'s of $d^{\uparrow}$ and $d^{\downarrow}$, named $dn^{\uparrow}$ and $dn^{\downarrow}$ respectively, are identified. Notice that $d^{\uparrow}$ represents the problem p_i in case it has no $super-dn$. In this case, $d^{\downarrow}$ has been previously initialized to a new dn which represents the evolution of the problem called p_{i+1}. In case $dn^{\uparrow} \neq$ null, then $d^{\downarrow}$ is added to $dn^{\downarrow}$, that means, the decomposition node of $d^{\uparrow}$ is added to a problem that represents an evolution of the problem in which $d^{\uparrow}$ is defined. In case $d^{\uparrow}$ represents a dn, we further decompose each of its cd's and rlt's.

Algorithm 2: $f_{decRel}(f_r^{\uparrow})$

begin
- new $f_r^{\downarrow} \in R$
- $\forall d^{\uparrow} \in f_r^{\uparrow}(nodes)$
- **begin**
  - $d^{\downarrow} = d^{\uparrow}(to)$
  - $f_r^{\downarrow}(nodes) \stackrel{+}{=} d^{\downarrow}$
- $dn^{\downarrow}(edges) \stackrel{+}{=} f_r^{\downarrow}$

Decompose a relation $f_r^{\uparrow} \in R$**.** The algorithm is illustrated in Figure 4.

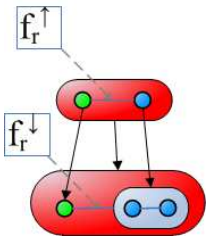


Figure 4: Decomposition of a relation

First, a new relation $f_r^{\downarrow}$ is created of the same type as $f_r^{\uparrow}$. For each node $d^{\uparrow}$ involved in $f_r^{\uparrow}$ we search its decomposition node $d^{\downarrow}$ in $dn^{\downarrow}$ and assign it to $f_r^{\downarrow}$. Once $f_r^{\downarrow}$ contains all the decomposition nodes of $f_r^{\uparrow}(nodes)$, the relation $f_r^{\downarrow}$ is added to the list of relations of $dn^{\downarrow}$.

Regarding parameter relations, the decomposition of $f_r^{\uparrow}$ in $f_r^{\downarrow}$ is very simple as $d_2^{\downarrow}$ will inherit the parameters of $d_2^{\uparrow}$. With respect to temporal relations (see Algorithm 4), the decomposition of $f_r^{\uparrow}$ in $f_r^{\downarrow}$ is equally simple, as $d_2^{\downarrow}$ will inherit the temporal relations (including the temporal duration) of $d_2^{\uparrow}$.

Propagate a relation $f_r \in \{R_{param}, R_{temp}\}$**.** During planning, two more algorithms will be required to propagate parameter and temporal relations between a dn and its cd's. In order to study how they propagate consider, without loss of generality, a relation like $f_r^{\uparrow}$ in Figure 4 involving two elements, one of which is a cd^c. The generalization of this example to n-ary relations involving different cd^s's and cd^c's is straightforward.

Algorithm 3: $f_{propParamRel}(dn^{sup}, d^{new})$

begin
- $\forall par_i \in d^{new}(params) : \exists par_j \in dn^{sup} \wedge par_i = par_j$
- **begin**
  - new $f_{param}^{equals} \in R_{param}$
  - $f_{param}^{equals}(from) = par_i$
  - $f_{param}^{equals}(to) = par_j$
  - $dn^{\downarrow}(edges) \stackrel{+}{=} f_{param}^{equals}$

The propagation of parameter relations is presented in Algorithm 3. In case there is a parameter par_i in the new cd equal to a parameter par_j of the dn (for example, both refer to the speed of the rover), an $equal$ parameter relation is created between them and added to the set of relations of dn^{sup}.

Regarding temporal relations, in order to maintain the consistency between $d_2^{\downarrow}$ and its cd's, the temporal relation must be propagated any time a new element is added to $d_2^{\downarrow}$. Algorithm 4 takes care of it.

Algorithm 4: $f_{propTempRel}(dn^{sup}, d^{new})$

begin
- **if** $dn^{sup}(order)$ *change_to* $\neg fullyOrdered$ **then**
  - $dn^{sup}(edges) \stackrel{=}{=}$ $\{f_{temp}^{Starts}(dn^{sup}, n_{first}), f_{temp}^{Ends}(dn^{sup}, n_{last})\}$
  - $dummy_1, dummy_2 =$ new $cd \in CD^s$
  - $dn^{sup}(nodes) \stackrel{+}{=} \{dummy_1, dummy_2\}$
  - $f_{temp}^{Starts}(dn^{sup}, dummy_1)$
  - $f_{temp}^{Ends}(dn^{sup}, dummy_2)$
  - $dn^{sup}(edges) \stackrel{+}{=}$ $\{f_{temp}^{Starts}(dn^{sup}, dummy_1), f_{temp}^{Ends}(dn^{sup}, dummy_2)\}$
- **else if** $dn^{sup}(order)$ *change_to* $fullyOrdered$ **then**
  - $dn^{sup}(edges) \stackrel{=}{=}$ $\{f_{temp}^{Starts}(dn^{sup}, dummy_1), f_{temp}^{Ends}(dn^{sup}, dummy_2)\}$
  - $dn^{sup}(nodes) \stackrel{=}{=} \{dummy_1, dummy_2\}$
  - $f_{temp}^{Starts}(dn^{sup}, n_{first})$
  - $f_{temp}^{Ends}(dn^{sup}, n_{last})$
  - $dn^{sup}(edges) \stackrel{+}{=}$ $\{f_{temp}^{Starts}(dn^{sup}, n_{first}), f_{temp}^{Ends}(dn^{sup}, n_{last})\}$

In case a new element is added to dn^{sup}, the status of dn^{sup} is checked. If it has changed to $\neg fullyOrdered$ (see Section 6), the temporal relations between the first/last cd

and dn^{sup} are removed and two dummy cd^s's are created to enforce the temporal relations between dn^{sup} and its cd's:

- $f_{temp}^{Starts}(dn^{sup}, dummy_1)$: Indicates that dn^{sup} will start at the same time as the first of its cd's
- $f_{temp}^{Ends}(dn^{sup}, dummy_2)$: Indicates that dn^{sup} will end at the same time as the last of its cd's

If the status of dn^{sup} has changed to $fullyOrdered$, the $dummy$ cd's are replaced by the first and last cd of dn^{sup} and the new relations $f_{temp}^{Starts}(dn^{sup}, n_{first})$, $f_{temp}^{Ends}(dn^{sup}, n_{last})$ are added to the list of edges of dn^{sup}.

5.3 Decision Network (dn)

A dn is represented as a hypergraph which nodes are a set of cd's and edges are the set of rlt's. In order to allow the definition of hierarchical structures, it can be decomposed in two different ways (see Figure 2):

- Vertical (level of abstraction): A parent dn ($dn^{\uparrow}$) represents a less evolved version of the child ($dn^{\downarrow}$), that is, $dn^{\downarrow}$ contains at least the decomposition of one $dn^{\uparrow}$ complex goal. Each dn can have several parents, as two $dn^{\downarrow}$ can be unified in one single dn, and several children, because a dn might be decomposed in different ways. There is one single dn at the highest level in the hierarchical structure, called $problem_0$, which represents the root of the structure, and several dn's at the lowest level, called $problem_i$ (where i is the level in the hierarchy) which represents the leaves of the structure. In case the planner finds a solution, it will be in any of the leaves of the structure
- Horizontal (nesting): A $super - dn$ (dn^{sup}) can be decomposed in an unlimited number of $sub - dn$'s (dn^{sub}), nested in different levels. At the same time, each dn can belong to several dn^{sup}, as different dn^{sup} can share common goals. There is one single dn at the highest nesting level, called $problem_i$, where i represents the hierarchical level and several dn's at the lowest level, called $sub-problem_{ij}$, where j represents the nesting depth. A dn represents itself a complex goal that can be managed as any other goal by the planner

  It is formally defined as follows:

$$dn = \Big\langle id, \{values\}, \{interval\}, \{nodes\}, \{edges\}, \{props\} \Big\rangle \quad (6)$$

  where:

- $\{values\}$: Values associated to the dn inherited from its parent $d^{\uparrow}$, in case dn represents the decomposition of a cd^c with value
- $\{interval\}$: Indicates the range of minimum and maximum time that the execution of this dn should take. It is either defined by the user (for the initial dn that represents the problem) or derived from its parent $d^{\uparrow}$ as depicted bellow in Algorithm 4
- $\{nodes\}$: Set of cd's and dn^{sub}'s involved in the dn
- $\{edges\}$: Set of rlt's in which either the dn or any of its elements are involved
- $\{props\}$: A dn inherits the properties of cd's. The value of each property is assigned or retrieved with a specific function, formally defined as follows: $f_{property}(dn)$

A dn is partially ordered in case any of its cd's are not ordered by means of temporal relations with a predecessor and successor. The predecessor of the first element and the successor of the last element is their dn, related by means of $Starts$ and $Ends$ temporal relations respectively (see Algorithm 4). During the specification of a problem or during planning time, a dn can be added or deleted by the planner or human user.

6 Design of a planner based on HTLN

In this section we summarize the guidelines we identified in order to exploit the HTLN paradigm. Traditional timeline planners search for fully justified plans, which are complete, fully ordered and valid:

- Complete: All variables are grounded. The plan always specifies how to proceed (the timelines have no gaps)
- Fully ordered: All the decisions are sequenced
- Valid: All the constraints are satisfied

This definition might represent an unachievable condition for real-world P&S systems. In order to deal with non deterministic, dynamic and partially observable environments, more flexibility is required. Our PSS is based on the concept of Sufficient Plan, defined as follows: *All variables and relations are sufficiently grounded, all fully grounded relations are satisfied, all sub-plans are sufficiently decomposed and all the mandatory goals can be achieved for at least one specific instantiation of the sufficient plan.*

The underlying concepts of this definition are:

- Sufficiently grounded: All decisions $d \in D$ and relations $r \in R$ that appear as goals in the problem must specify whether they should be grounded or not at planning time. A cd is grounded when its value and parameters are grounded; a relation is grounded when all its elements are grounded. A partially grounded relation has two important consequences: (1) the relation cannot be satisfied, (2) in case of temporal relations, the resulting dn is partially ordered
- Sufficiently decomposed: A cd^c should also specify whether it must be or not fully decomposed. A cd^c is fully decomposed if all its $sub-cd$'s are fully decomposed and partially decomposed in other case
- Valid plan: If there is one instantiation of the partial plan where every decision and relation can be grounded, all constraints satisfied, all sub-plans can be fully decomposed and all mandatory goals are achieved

This represents an extension of the definition provided in (Frank and Jónsson 2003), where a partial plan is fully defined up to a certain point called plan horizon, ignoring activities that fall outside it. In our case, any goal, decision or constraint might be partially defined according to an initial definition, giving the responsibility to the executive to fill in the gaps prior to the execution.

During the plan expansion, the planner has to add (and eventually delete) cd's in order to justify the already existing goals in the network. As dn's are managed in HTLN like cd's, the planner can reason over groups of goals. In fact, the planner can add $sub - dn$'s stored in the KB using the same methods applied to cd's.

By reasoning over the underlying graph of an HTLN problem, it is possible to identify separate sub-problems, allowing the use of parallel planning. As explained in (Dechter and Pearl 1987), a graph $G = (V, E)$ has a separation vertex v if there exist vertices a and b, $a \neq v$, $b \neq v$ such that all the paths connecting a and b pass through v. A graph that has a separation vertex is called separable. Let $V' \subseteq V$, the induced subgraph $G' = (V', E')$ is called a non-separable component if G' is non-separable and if for every larger V'', $V' \subseteq V'' \subseteq V$, the induced subgraph $G'' = (V'', E'')$ is separable. An efficient algorithm to generate valid temporal plans (not considering resource consumption) and computing the minimal network is to first find the non-separable components $C_1..C_m$ and then solve each one of them independently. If all components are valid, then the entire network is valid and the minimal networks of the individual components coincide with the overall minimal network. Taking advantage of HTLN structure, we can use this technique to isolate independent sub-problems dn^{sub} and perform parallel planning, where each planner takes care of a different independent sub-problem.

6.1 Planner design

In classical planning, two types of operators are used to build a solution from a partial plan: expansion (horizontally) of the partial plan and fixing flaws. However, in HTLN a plan is a hierarchical structure which divides the problem in different levels of abstraction. In order to generate a valid plan, the planner must generate a valid dn at each level of abstraction. Therefore, a planner for HTLN should have a third type of procedure, the decomposition, which expands the plan vertically.

At the same time, the planner requires the following properties for the cd's: $f_{isSGround}(cd)$, $f_{isSDec}(cd)$, which specify whether the cd is sufficiently grounded or decomposed, respectively, and $f_{active}(cd)$, that specifies which cd is active between a set of exclusive cd's. With respect to rlt's, $f_{isSatisfied}(rlt)$ indicates whether the relation is satisfied or not.

Given a problem and model as inputs to the planner (presented in Algorithm 5), it first divides the problem in independent sub-problems by means of the separation vertices. Each sub-problem is then evolved in parallel and added to the plan, which is refined while taking into consideration all the partial modifications.

The strategy to evolve a problem consists of two steps. First, the flaws (threads or open-goals) at one level of the problem are computed and fixed (if possible). Once the layer is valid, the problem is evolved to the next level decomposing a complex goal in subgoals. This task is undertaken by the $vDecomposition$ method shown in Algorithm 6, which decomposes cd's and rlt's using the algorithms shown in Section 5.2.

Algorithm 5: HTLNPlanner(problem, domain, time)

```
begin
    while (time) do
        subdns = calcSeparationVertex(problem)
        [parallel]
        plan ± evolveP(subdns_i, domain, time)
        refinePlan(plan)
        if (plan.score > best.score) then
            best ← plan
    if (¬best) then
        return fail
    return solution
```

Algorithm 6: evolveP(problem, domain, time)

```
begin
    while (time) do
        flaws = computeFlaws(problem, domain)
        fixFlaws(problem, domain, flaws, time)
        if (flaws ≠ ∅) then
            return fail
        f_decNode(problem, domain)
        if (¬problem.isSDec ∨ ¬problem.isSGround)
        then
            return fail
        return problem
    return fail
```

7 Conclusions and Future Work

Real-world problems, and particularly space robotics, manage complex and critical systems that present important divergences with respect to theoretical problems. In order to introduce automated P&S systems in this area, the experts need to understand and trust the outcomes of the planners. At the same time, we have to increase the flexibility of planners and plans in order to increase its robustness during plan execution in uncertain environments with noise information. Finally, it is becoming crucial to provide instruments to represent and store the knowledge that will lead the planner through NP-complete search-spaces where completeness or soundness cannot be guaranteed. We consider that the way problems and plans are defined and represented constitutes a major concern that might help to address these issues. This is the reason that motivated the conception of HTLN as a new paradigm that defines complex goals as the building blocks of problems and provide advanced features to define and connect them. In the future, we are planning to extend the HTLN paradigm in order to include fuzzy temporal intervals and relations as a complementary tool to represent uncertainty in execution. A planner which exploits the HTLN paradigm and follows the guidelines presented in Section 6 is also under development.

Acknowledgements. This research has been co-funded by the Networking/Partnering Initiative (NPI) between ESA-ESOC and TU Darmstadt and by the Spanish Trainee program, funded by the Ministry of Spain. It also receives support from the German Research Foundation (DFG) within the Research Training Group 1362 "Cooperative, adaptive and responsive monitoring in mixed mode environments".

References

Bajracharya, M.; Maimone, M.; and Helmick, D. 2008. Autonomy for mars rovers: Past, present, and future. *Computer* 41(12):44 –50.

Berge, C. 1990. Optimisation and hypergraph theory. *European Journal of Operational Research* 46(3):297–303.

Berger, R. 2009. Development of electronics for use in outer space. Presentation.

Bertoli, P.; Cimatti, A.; Roveri, M.; and Traverso, P. 2001. Planning in nondeterministic domains under partial observability via symbolic model checking. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Bessière, C. 1994. Arc-consistency and arc-consistency again. *Artif. Intell.* 65:179–190.

Bresina, J. L.; Jonsson, A. K.; Morris, P. H.; and Rajan, K. 2003. Mixed-initiative activity planning for mars rovers. *Challenges* 2–3.

Cassandra, A. R.; Kaelbling, L. P.; and Littman, M. L. 1994. Acting optimally in partially observable stochastic domains. In *Proceedings of the twelfth national conference on Artificial intelligence (vol. 2)*, AAAI'94, 1023–1028. Menlo Park, CA, USA: American Association for Artificial Intelligence.

Chien, S.; Rabideau, G.; Knight, R.; Sherwood, R.; Engelhardt, B.; Mutz, D.; Estlin, T.; Smith, B.; Fisher, F. W.; Barrett, T.; Stebbins, G.; and Tran, D. 1997. Aspen - automated planning and scheduling for space mission operations. *Jet Propulsion* 1–10.

Chien, S.; Sherwood, R.; Tran, D.; Cichy, B.; Rabideau, G.; Castano, R.; Davies, A.; Lee, R.; Mandl, D.; Frye, S.; Trout, B.; Hengemihle, J.; D'Agostino, J.; Shulman, S.; Ungar, S.; Brakke, T.; Boyer, D.; Van Gaasbeck, J.; Greeley, R.; Doggett, T.; Baker, V.; Dohm, J.; and Ip, F. 2004. The eo-1 autonomous science agent. In *Proceedings of the Third International Joint Conference on Autonomous Agents and Multiagent Systems - Volume 1*, AAMAS '04, 420–427. Washington, DC, USA: IEEE Computer Society.

Chien, S.; Doyle, R.; Davies, A.; Jonsson, A.; and Lorenz, R. 2006. The future of ai in space. *Intelligent Systems, IEEE* 21(4):64 –69.

Clarke, E.; Long, D.; and McMillan, K. 1989. Compositional model checking. In *Logic in Computer Science, 1989. LICS '89, Proceedings., Fourth Annual Symposium on*, 353 –362.

Damaschke, P. 2009. Multiple hypernode hitting sets and smallest two-cores with targets. *J. Comb. Optim.* 18(3):294–306.

Dechter, R., and Pearl, J. 1987. Network-based heuristics for constraint-satisfaction problems. *Artificial Intelligence* 34:1–38.

Dvorak, D. L.; Amador, A. V.; and Starbird, T. W. 2008. Comparison of goal-based operations and command sequencing.

Dvorak, D. L. 2009. Nasa study on flight software complexity. Technical report, NASA.

Erol, K.; Hendler, J.; and Nau, D. S. Htn planning: Complexity and expressivity. In *In Proceedings of the Twelfth National Conference on Artificial Intelligence (AAAI-94*, 1123–1128. AAAI Press.

Frank, J., and Jónsson, A. 2003. Constraint-based attribute and interval planning. *Constraints* 8:339–364.

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.

Fukunaga, A. S.; Rabideau, G.; Chien, S.; and Yan, D. 1997. Aspen: A framework for automated planning and scheduling of spacecraft control and operations. In *Proceedings of the International Symposium on AI, Robotics and Automation in Space*.

Gallo, G.; Longo, G.; Pallottino, S.; and Nguyen, S. 1993. Directed hypergraphs and applications. *Discrete Applied Mathematics* 42(2-3):177 – 201.

Ghallab, M., and Laruelle, H. 1994. Representation and control in ixtet, a temporal planner. In *Proceedings of the International Conference on AI Planning Systems (AIPS)*, 61–67.

Ghallab, M.; Nau, D.; and Traverso, P. 2004. *Automated Planning: Theory and Practice*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Jonsson, A. K.; Morris, P. H.; Muscettola, N.; Rajan, K.; and Smith, B. D. 2000. Planning in Interplanetary Space: Theory and Practice. In *Artificial Intelligence Planning Systems*, 177–186.

Little, T. D. C., and Ghafoor, A. 1990. Synchronization and storage models for multimedia objects. *IEEE J. Sel. Areas Commun.* 8(3):413–427. NewsletterInfo: 38.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99 – 118.

Muscettola, N.; Nayak, P. P.; Pell, B.; and Williams, B. C. 1998. Remote agent: to boldly go where no ai system has gone before. *Intelligence* 103(1-2):5–47.

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

Rugg, R. D. 1983. Building a hypergraph-based data structure. *AUTOCARTO 6* 2:211220.

Singh, M. 1995. Path consistency revisited. In *Proceedings of the Seventh International Conference on Tools with Artificial Intelligence*, TAI '95, 318–. Washington, DC, USA: IEEE Computer Society.

Tate, A.; Drabble, B.; and Kirby, R. 1994. O-plan2: an open architecture for command, planning and control. *Intelligent Scheduling* 1:213–239.

Wilkins, D. E., and desJardins, M. 2000. A call for knowledge-based planning. *AI MAGAZINE* 22:99–115.

Wilkins, D. E.; Myers, K. L.; Lowrance, J. D.; and Wesley, L. P. 1995. Planning and reacting in uncertain and dynamic environments. *Journal of Experimental Theoretical Artificial Intelligence* 7(1):121–152.

Propagating Temporal Constraints on Sets of Intervals

Jonas Ullberg and **Federico Pecora**
Center for Applied Autonomous Sensor Systems
Örebro University, SE-70182 Sweden
{jonas.ullberg,federico.pecora}@oru.se

Abstract

In this paper we propose a method of propagating quantitative Allen interval constraints on sets of intervals defined by polygons in a two dimensional space. The method is used to solve the problem of inferring timelines of human activities from timelines representing traces of sensor data. The main advantage of this method over others is that it allows a more general description of the events that the intervals are taken to reflect during inference. This paper deals with the algorithmic issues underlying the timeline recognition process. In this context, we compare the performance of our method to that of a state of the art approach based on classical temporal constraint reasoning techniques (Dousson and Maigat, 2007).

Introduction

In this paper we provide a new way of performing context recognition from sensory data given as intervals on a set of timelines. We apply this technique to the problem of inferring human activities from data coming from a set of heterogeneous sensors in an apartment. Our goal is to construct a system for inferring context based on a given model of how this context correlates to sensor traces. An example use of such a system is to infer human activities from a set of sensors located throughout an apartment. This and other applications benefit from the ability to specify the model based on which context is inferred in a flexible and compact way. An easily specifiable model would allow, for instance, to easily configure a context recognition system to infer human activities and situations on a per-user and environment basis. Furthermore, the system should be able to be configured by a person without extensive knowledge of the underlying algorithm.

For this purpose we have found it useful to represent the states of the sensors and the inferred activities as intervals on different timelines. The model that describes the causal relationships between the states of the sensors and the inferred activities is provided as a set of quantitative Allen's interval algebra constraints. These constraints are posted between intervals representing sensor readings. This approach was first described by Ullberg, Loutfi, and Pecora (2009) and then subsequently extended by Pecora et al. (2012). However, in this prior work, we found that although constraints taken from Allen's interval algebra are a convenient way to describe such relations, they are also very brittle in the sense that small deviations in how the raw sensory data is interpreted and placed on the timelines can prevent activities from being inferred. One possible way of overcoming this is through the use of fuzzy Allen's interval constraints (Mansouri, 2011). In this work, constraint violations are allowed to some degree, thus allowing activities to be inferred with a "low likelihood" in case the model is not fully supported by the sensor readings. This approach, however, introduces problems when interpreting the inferred activities. Specifically, one has to provide a threshold on the likelihood of an inferred activity in order to be able to act in response to it, and this choice seems to lack rationale.

In this paper we tackle the problem of brittle inference in the opposite way, that is, by using non-fuzzy, quantified Allen interval constraints, but instead admitting many interpretations of sensor timelines. This is achieved by performing temporal inference on multiple intervals contextually. That is to say, each sensor reading is represented as a *set* of flexible temporal intervals rather only one. In order to assess whether temporal constraints hold among sets of intervals, we define a new temporal constraint propagation algorithm. This algorithm constitutes the basis of an abductive inference system which decides the overall context recognition problem.

The paper is organized as follows. First, we introduce the context recognition problem and the overall abductive inference procedure. We then describe the theory behind temporal constraint propagation with multiple intervals. We focus first on how to represent multiple intervals, and provide a semantics for quantified Allen's interval constraints among multiple intervals. We then detail the how these intervals are filtered using an arc consistency algorithm. We conclude with a comparison of the resulting system for context recognition with a state of the art technique based on similar abductive process but which uses classical temporal inference.

Problem statement

We illustrate the context recognition problem by giving a simple example of how inference is done by Pecora et al. (2012). In this work, intervals are generated from rich sen-

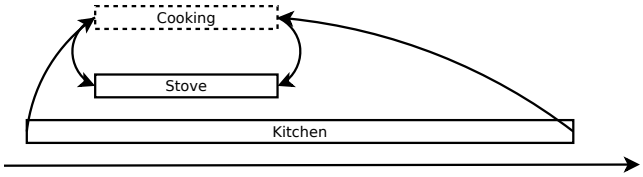


Figure 1: An example of a pattern in the sensory data that allows an activity to be inferred.

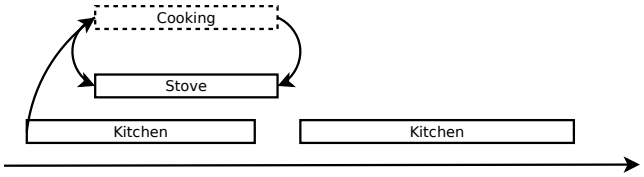


Figure 2: An example of a pattern in the sensory data that prevents an activity from being inferred.

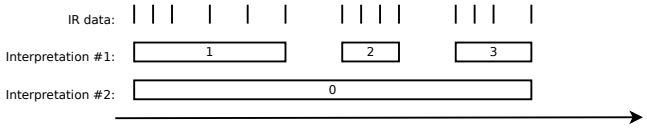


Figure 3: Two timeline representations of the same sensory data.

sory data that is provided by several sensors in a home environment. The goal is to provide a higher-level representation of what is happening in the world that is rich enough to reason about, but is unencumbered by unnecessary detail. Each of the generated intervals represents a fact that holds true during a limited period of time, for instance that the humidity in the bathroom was **high** between 14:00 and 14:15.

In order to infer context, the generated set of intervals are combined with a model consisting of rules of the form: **Cooking** `Equals` **Stove** $\wedge$ **Cooking** `During` **Kitchen**. Such rules define how context, in this case the activity of **Cooking**, can be inferred from intervals representing sensed data. These rules define abstract patterns of constraints in Allen's interval algebra (Allen, 1983) that should be satisfied in order for an activity to be recognized. The inference itself is done by iteratively trying to constrain a Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl, 1991) in which the intervals representing sensed data and inferred activities are managed as pairs of timepoints, representing the start and end time of the interval in question.

The high level Allen interval constraints are represented as simple distance constraints between timepoints in the STP. In the example above, the two constraints reference 6 timepoints in the STP; the start and end times of the three intervals **Stove**, **Kitchen** and **Cooking**. A constraint such as **Cooking** `Equals` **Stove**, is represented by two simple distance constraints in the STP that constrain the start and end times of the **Cooking** and **Stove** intervals so that they can only take on the same values. During inference, each combination of intervals that are referenced by such a pattern must be tried or pruned away by a search procedure. For instance the **Stove** might have been turned on several times in the past, and the person has been in the **Kitchen** at multiple times in the past. In this case, each combination of choices from these two groups of intervals have to be evaluated. Each such possible combination is tried by propagating a STP, and if the STP has a consistent solution the pattern is considered satisfied, and the **Cooking** activity is thus inferred.

Figure 1 and Figure 2 show two potential scenarios that could unfold, in the former the **Cooking** activity is successfully recognized and in the latter it cannot be recognized due to the introduced discontinuity in the **Kitchen** timeline. The key point is that even though these two scenarios are visually similar, they are very different from the point of view of the constraint-based inference. Clearly, the rule is written with the scenario that unfolds in Figure 1 in mind, and small deviations from the optimal scenario can prevent us from recognizing the activity.

This example is inspired by a real world deployment of sensors in an apartment and could easily arise when using a Passive Infrared (PIR) sensor for instance (a motion sensor often used in burglar alarms). This sensor is characterized by the fact that it emits a Boolean reading at regular intervals reflecting if movement has been sensed or not. When forming intervals out of these readings, we must interpret them to reflect if a person is in a room or not. This can be done, for instance, by allowing a fixed temporal window of discontinuities among consecutive readings indicating that movement has been registered. This situation is illustrated in Figure 3, which shows a set of Boolean readings indicating movement, and also two timelines that have been formed out of this data but with different thresholds of allowed discontinuity. In Interpretation #1, the translation of the discrete readings into intervals is quite strict so that discontinuities are easily introduced, whereas Interpretation #2 is resilient enough to only create one interval out of these readings.

The problem illustrated in Figure 2 could possibly be overcome by altering the rule so that **Cooking** is required to be `Overlapped-By` **Kitchen** rather than occur `During` **Kitchen**. This constraint would however not be satisfied if being in the **Kitchen** is first sensed after the **Stove** is turned on.

In order to overcome this problem we want to use a wider range of possible intervals as support for the constraint-based context inference procedure. For instance, we might wish to use an interval (generated by a relaxed interpretation of the sensory data, biased towards generating large continuous intervals rather than introducing discontinuities) and all of its sub-intervals as support for the inference. Thus, we would like to be able to reason on multiple *interpretations* of the same sensory data, each providing additional support for inferring an activity.

Formal problem statement

Our context recognition problem can be described as a constraint satisfaction problem (CSP) Tsang (1993) of the form $\langle V, CA \rangle$. Here, $V = \{v_0, \ldots, v_l\}$ is a set of variables, each

representing the timeline of a sensor or of one inferred activity. The domain of each variable, $v = \{i_0, \ldots, i_m\}$, is a set of (possibly overlapping) temporal intervals of the form $i = [s, e]$, where s is the start time of the interval and e its end time. Each such interval either denotes that a sensed fact holds true, or that an activity was performed as the interval's time-span states (not during, but precisely starting at time s and ending at time e).

$CA = \{c_0, \ldots, c_n\}$ is a set of constraints on the variables in V of the form $c_j = \{(v_0 \texttt{ AllenC } v_1) \wedge (v_0 \texttt{ AllenC } v_2) \wedge \ldots\}$, where each c constrains the domains of two variables. v_0 is a variable representing the timeline of an inferred activity, and $v_{i \neq 0}$ is either a variable representing the timeline of a sensor or of another inferred activity. Note that this implies a dependency graph among timelines of inferred activities which has no loops, i.e., a tree. `AllenC` is a quantitative binary Allen constraint of the form v_a `CONSTRAINT` $[l, u]$ v_b. These constraints defines temporal relations between the variables that should hold in order for an activity to be inferred.

A solution to the problem $\langle V, CA \rangle$ is an assignment of values (i.e., sets of intervals) to variables (i.e., activity and sensor timelines). A solution to the context recognition problem is the projection of a solution to the CSP on the variable representing the inferred activity. In other words we are not interested in the interpretations of sensors readings necessary to support inferred activities.

In the CSP, we maintain only one variable representing an activity to be inferred. The reason has to do with constraint propagation. Let an activity to be inferred be A. Propagating the constraints in the CSP may reduce the domain of a variable representing a sensor, S, which is necessary to support A. However, this reduction only reflects the fact that some intervals in the domain of S are not relevant for inferring A, and not that they represent incorrect knowledge about the sensor readings. The intervals filtered out due to the requirements of A could be used to infer another activity B. This is not possible if the CSP contains variables representing both A and B.

In the following sections we present a geometric representation of the domains of the variables in V. This representation allows us to define a propagation algorithm which achieves arc-consistency. As we will see, arc-consistency is complete under certain assumptions because of the tree structure mentioned above.

Representing multiple intervals

In order to increase the number of activities that we can successfully identify it makes sense to perform temporal inference on batches of interpretations contextually, or even more useful, on an entire spectrum of interpretations including these. In a naïve way, the former could be accomplished by admitting several overlapping intervals on the same timeline. For instance, by merging Interpretation #1 and #2 in Figure 3. This would however only work to a limited extent since it would also increase the complexity of searching for matching patterns in the data. This problem affects all approaches to context recognition which rely on an explicit representation of each interval in memory.

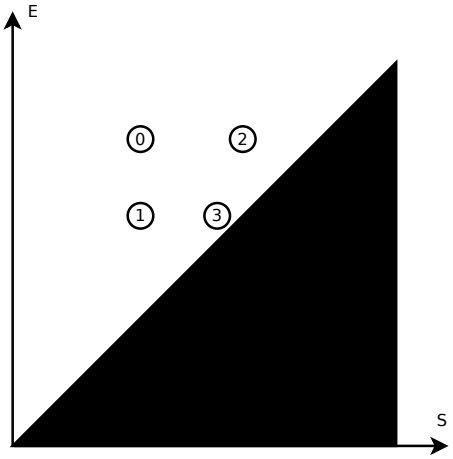


Figure 4: A 2 dimensional representation of a set of enumerated intervals.

A more intelligent strategy is to propagate constraints on a spectrum of interpretations contextually. This requires changing the way in which we represent sets of intervals. The most straightforward way of representing a set of intervals would be to interpret an interval as a single point in a 2 dimensional graph as in Figure 4. Each of the 4 points (intervals) in this figure corresponds to one of the intervals in Figure 3. In this figure, each point's projection onto the x-axis defines the interval's start-time, and its projection onto the y-axis the end time. Naturally, an interval is not permitted to end before it has started, therefore no interval is allowed to reside in the lower-right part of this figure. Interpreting intervals as points in a 2-dimensional space was first done by Rit (1986), who described how qualitative Allen Interval constraints could be used and propagated on such representations[1] This representation was later discussed by Pujari, Kumari, and Sattar (2000) and has subsequently only been briefly mentioned in other work (Duftschmid, Miksch, and Gall, 2002; Aigner and Miksch, 2006). The reason for this lack of attention is most likely because of the introduction of alternative problem formulations such as the STP, TCSP (Dechter, Meiri, and Pearl, 1991) and DTP (Stergiou and Koubarakis, 1998). However, in our problem the constraint networks are relatively simple compared to the ones in most contemporary work, whereas the sets of intervals that we want to reason about is large. Thus there is reason to believe that this representation is better suited for our particular problem.

Representing intervals as points in a two dimensional space not only serves as a visual aid, but more importantly, this representation can also be used to "generalize away" the usage of enumerated sets of intervals and instead consider groups of intervals. Figure 5 visualizes such a set of intervals. Specifically, the gray triangular area protruding from the diagonal in this figure corresponds to the set of all intervals that are `Contain`ed within Interval 0 in Figure 3. For

[1]The problem identified by Rit (1986) was named Sets of Possible Occurrences (SOPOS).

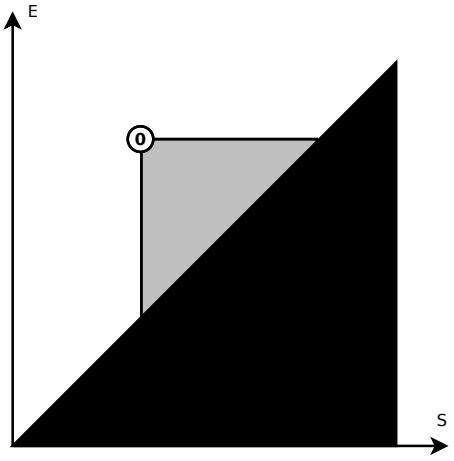


Figure 5: The set of intervals that occurs `During` interval 0 in Figure 4.

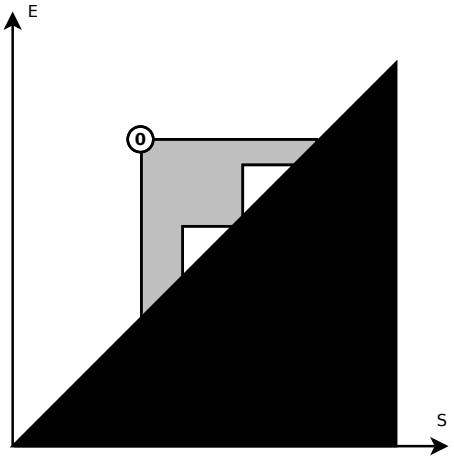


Figure 6: The set of intervals that occurs `During` interval 0 in Figure 4, excluding all intervals contained within the two "gaps" in the timeline.

an interval to be contained within another, the requirement is that the interval starts after and ends before the "containing" interval. These two requirements corresponds to the bounds of the gray area in the figure. Thus, this area contains all the intervals found in Figure 4.

By looking at Figure 3 and Figure 4 we can also notice that it might be meaningful to reason about the set of intervals that are fully contained within Interval 0, with the exception of the sub-intervals that are contained in the two "gaps", during which we received no indication of this being true. Figure 6 illustrates the mentioned set. The rationale behind this might be that we want to be more general in our description of the state of the world and use facts such as "The person was in the kitchen between 13:00 and 14:00" (contained in interval 0, arbitrary picked time not illustrated in any figure) or "between 13:00 and 13:15" (contained in interval 1), but not "between 13:20 and 13:25" (if this corresponds to the gap between Interval 1 and 2). Thus, this representation allows temporal constraints to be supported by general descriptions of the events, i.e., the person was mostly in the kitchen between 13:00 and 14:00, but not by more precise queries that we have reasons to doubt, e.g., being in the kitchen between 13:20 and 13:25.

Constraints among multiple intervals

The representation introduced above would be useless unless it was also possible to propagate temporal constraints on the intervals defined by these sets. Fortunately this can be done, although under certain assumptions as we will see. We can directly outline the admissible set of intervals B that a single interval i allows given a constraint as illustrated in Figure 7. This figure shows one single interval i along with

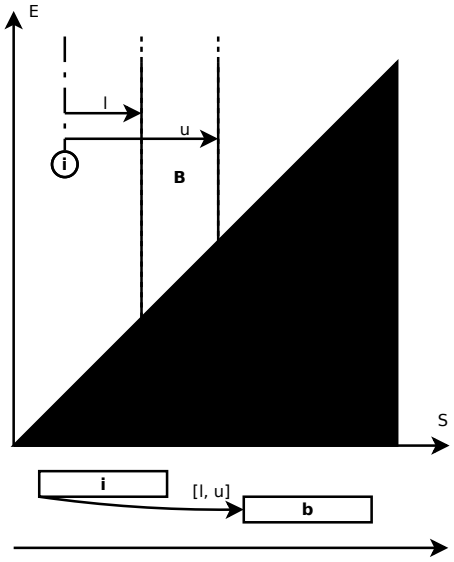


Figure 7: A $(\texttt{Starts} \vee \texttt{Started-By})\,[l, u]$ constraint.

the set of intervals B that satisfies the temporal constraint $i\ (\texttt{Starts} \vee \texttt{Started-By})\,[l, u]\,b$, so that any interval b in B starts at least l and at most u time units after i. Note that since this constraint does not limit the allowed end time of any interval in B, the set of allowed intervals stretches up towards infinity in the figure. For *mixed* constraints, i.e., constraints in which one interval's start time constrains another interval's end time or vice-versa, the geometric representation involves a projection onto the diagonal. An example of this is illustrated in Figure 8, the semantics here is that the end time of an interval i constrains the start time of another interval b. Thus, the start time of i is projected onto the diagonal to translate it into an end-time. The diagonal intersection is then used to constrain B in a similar way as in Figure 7.

Furthermore, Figure 9 illustrates a constraint that involves more than a single timepoint in the STP, i.e., the start or the end time, taken from each of the intervals. Here, the start time of i limits the possible time of occurrence of the end time of B. The distinguishing factor here is that the start time of any b is also limited to occur after the start time of i.

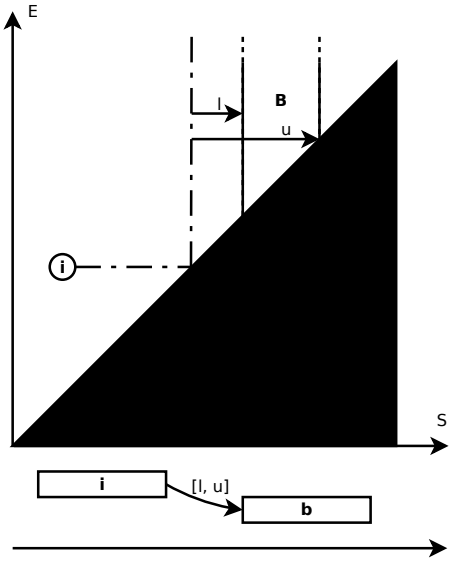


Figure 8: An `After`$[l, u]$ constraint.

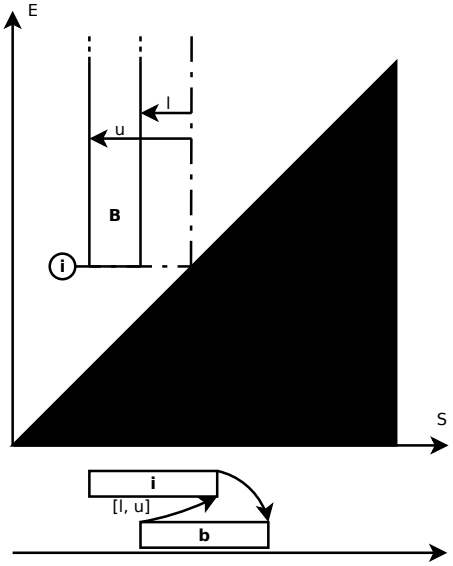


Figure 9: The `Overlaps`$[l, u]$ constraint.

Propagation

The propagation algorithm that is used to solve the context recognition problem is basically a reimplementation of the AC-3 algorithm (Mackworth, 1977), that is adapted to work on geometric sets of intervals. Like AC-3, the algorithm keeps a work list containing the arcs in the constraint network that should be propagated. This set is initialized to contain all the variables in the domain. Similarly, during propagation, arcs are removed from this list and processed. If this reduces the domain of a variable, arcs involving this variable are reintroduced into the list.

Algorithm 1 shows how we propagate one arc, $A \overset{c}{\leftarrow} B$, in the constraint network, i.e., how the algorithm removes

Algorithm 1

```
1:  function propagate_arc(A, c, B)
2:      P ← ∅
3:      A^convex ← convex_subsets(A)
4:      for p in A^convex do
5:          I ← ∅
6:          for i in p do
7:              I ← I ∪ evaluate(i, c)
8:          end for
9:          p ← convex_hull(I)
10:         P ← P ∪ p
11:     end for
12:     B ← B ∩ P
13: end function
```

values from the domain of a variable B with the help of the constraint c and the values in the domain of variable A. Traditionally, this is done by checking each value in the domain of B and searching for a variable in the domain of A that satisfies the constraint. If one exists, the value in B is kept, otherwise it is filtered. In our case, this is not possible since we do not maintain an explicit representations of the intervals in the domains of the variables. Instead, given A and c, we calculate all possible values of B. This is done by calculating the *Minkowski Sum* (de Berg et al., 1997) of the interval A and (dynamically) each convex set of intervals that are formed by evaluating the constraint on the vertex intervals in A. We will refer to this set as the *convolution* of A given c, and we will use it to geometrically intersect the previous domain of B in order to remove inconsistent values. This is done as shown in Algorithm 1.

Algorithm 1 calculates $A \overset{c}{\leftarrow} B$ and takes as an argument two variables, A and B, and a constraint c. It starts by initializing a new empty set of polygons (line 2) that will be used to store the convolution of A.

The next step in the algorithm is to calculate a convex decomposition A^{convex} of the polygons defining the set of intervals in A with the function $convex\_subsets$ (line 3). This function takes as input a set of possibly non-convex polygons and returns a larger (or equally sized) set of convex polygons that defines the same regions (line 4). This kind of decomposition is handled using an algorithm such as the one by Keil (1985). Note that optimal decomposition of a simple polygon can be done in $O\left(r^2n^2\right)$ time (Keil, 1985), where n is the total number of vertices and r is the number of notches (reflex angles). Note also that we can do this in $O\left(n\log(n)\right)$ time Hertel and Mehlhorn (1985) with a guarantee that we do not get more than four times more convex pieces than the optimum.

The convolution itself is driven by solving a small STP containing two intervals (i.e., four timepoints). In the STP, these timepoints are initially constrained to each other with simple distance constraints reflecting the Allen interval constraint defined by c. Furthermore, one pair of timepoints are constrained to the start and end time of interval i respectively. The STP is then propagated with the Floyd–Warshall all pairs shortest path algorithm (Floyd, 1962), which re-

duces the temporal domain of the remaining pair of timepoints.

At this stage, the mutual temporal relationship between the second pair of timepoints in the STP reflects the set of intervals that are admissible given the constraint c and the interval i. This corresponds to the situation illustrated in Figures 7–9. In these figures, the pair of timepoints that are initially constrained represent the start and end time of i, and after propagation the remaining temporal flexibility in the second pair defines area B. The admissible region for B according to the interval i and the constraint c is then extracted by analyzing the remaining temporal flexibility of the second pair of timepoints. Each such convolution of a single interval i generates four new intervals (i.e., the vertices of the admissible area of B in Figure 9). This process is done for each interval representing a vertex of A, and for each convex sub-polygon of A a set of intervals I is formed. This situation is shown in Figure 10.

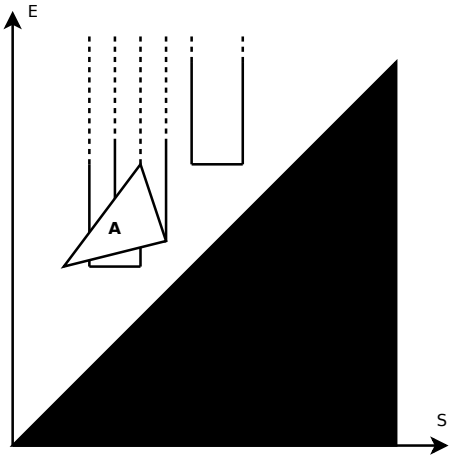


Figure 10: The individual convolutions of the intervals defined by region A with the constraints shown in Figure 9.

Redundant (interior) intervals are then removed from each I by taking the set's convex hull. This creates a (convex) polygon p (line 9) that defines the convolution of the convex subset of A, A^{convex}, with respect to the constraint c such as the one illustrated in Figure 11. The convex hull is retrieved with a Graham scan Graham (1972) which has a complexity of $O\left(n\log\left(n\right)\right)$ where n is the number of intervals (vertices) in I.

Finally, all such convex polygons form one set of polygons P that completely define the admissible values of B with respect to A and the constraint c. This set is then used to intersect the previous domain of B (line 12), which effectively removes values from B that cannot be satisfied with respect to A and the constraint c. Like the union, the intersection is calculated with a clipping algorithm such as the one outlined by Greiner and Hormann (1998), which has a complexity of $O\left(mn\right)$, where m and n are the number of vertices (i.e., intervals) in two polygons. However, in practice, this can usually be done much faster if some sort of partitioning scheme is used.

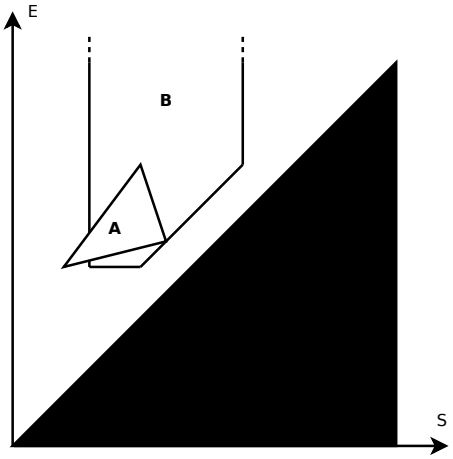


Figure 11: The convex hull of the individual convolutions shown in Figure 10.

Finally, just like in AC-3, when the domain of one variable is reduced, all of its outgoing arcs are reasserted in the work list. Furthermore, when the work list becomes empty the constraint network has been fully propagated. The algorithm outlined here provides the necessary functionality to propagate Allen's interval constraints in a network where the domains of the variables are sets of temporal intervals defined by polygons.

Experimental evaluation

In this section of the paper we compare the performance and quality of the inference of the geometric approach against the results that can be obtained using the chronicle recognition system suggested by Dousson and Maigat (2007). The data used for this evaluation was collected in the home of a researcher during a two-week long period with the help of a few wireless sensors. The sensors provide samples measuring movement, humidity, illumination and the usage of furniture such as the couch and the bed.

In the approach originally describe by Dousson and Maigat (2007), dubbed chronicle recognition, so-called patterns of sensor events, called chronicles, are recognized. In our approach, as well as other constraint based approaches to context recognition (Pecora et al., 2012), sensor readings and inferred activities are represented as intervals, i.e., pairs of timepoints. Nevertheless, the chronicle recognition technique can be easily extended to deal with intervals. Such an extension is what we are comparing against — which effectively makes chronicle recognition functionally identical to the approach described by Pecora et al. (2012).

The scenario used for this evaluation consisted of inferring an activity **WatchingTV** from two sensors whose data is taken to reflect if someone is **InLivingroom** and **InCouch**. Furthermore the requirement for inferring **WatchingTV** is {**WatchingTV** `Contains` **InCouch** $\wedge$ **WatchingTV** `During` **InLivingroom**}. Finally, any interval in the domain of the **InLivingroom** variable is required to have a duration of at least 5 minutes.

The data used to construct the **InCouch** timeline consisted of 120,581 samples from a pressure sensor mounted underneath a couch, and the **InLivingroom** timeline was constructed from 50,133 samples from a PIR sensor.

In the experiment we used a simple discretization method to generate intervals out of the raw data coming from a sensor. The discretization method used consisted of finding all intervals in which the mean of the samples is above a predefined threshold. Specifically, given a set of samples $s_0, \ldots, s_n$ for a sensor s, the samples' respective time $t_0, \ldots, t_n$ and a threshold T, we formed a set I_s containing all the intervals $[t_i, t_j]$ where

$$\sum_{x=i}^{j} \frac{s_i \left(t_x - t_{x-1}\right)}{t_j - t_i} \geq T.$$

Furthermore, any interval that could be fully contained in another was discarded.

When applied to the samples, the discretization method generated 3,186 intervals for the **InLivingroom** variable and 4,764 intervals for the **InCouch** variable. In the case of the chronicle recognition, we inferred activities from these sets of intervals as they were. However, when inferring from these sets with the geometric approach, we used the convex hull of each cluster of overlapping intervals in I_s. The idea is that the convex hull provides us with a more general description of the data that does not contain unnecessary detail, a description that is difficult to obtain for an approach that uses enumerated sets of intervals as input. The sets created for the **InLivingroom** and **InCouch** variable consisted of 1,036 intervals in 175 polygons and 170 intervals in 26 polygons respectively.

During inference, the geometric approach took 2.6 s to finish propagating its network. In the chronicle recognition system, the corresponding operation took 154.4 s (average of 10 runs). After propagation, the domain **WatchingTV**, was defined by 1,086 intervals in 174 polygons. In contrast, the chronicle recognition system recognized the pattern 101,615 times (each one using a unique combination of intervals in the input data as support).

Clearly, the chronicle recognition system is not conceived to deal with input data representing alternative views of the sensor readings, and forcing it to do so obviously affects performance heavily. However, this comparison is intended to show the difference in the quality of the inference. Figure 12 shows the solutions to the context recognition problem obtained with the geometric approach and the chronicle recognition system. In this figure, the filled areas show the possible intervals in the domain of the **WatchingTV** variable inferred using the geometric approach. The dots that are farthest from the diagonal show 799 unique "earliest start time, latest end time" solutions obtained by the chronicle recognition system. I.e., since a solution to the STP contains flexibility in the timepoints, we have extracted the earliest possible start time and the latest possible end time of each interval recognized by the chronicle recognition system[2]. Similarly,

[2]Note that these solutions are however not guaranteed to be a valid in the general case.

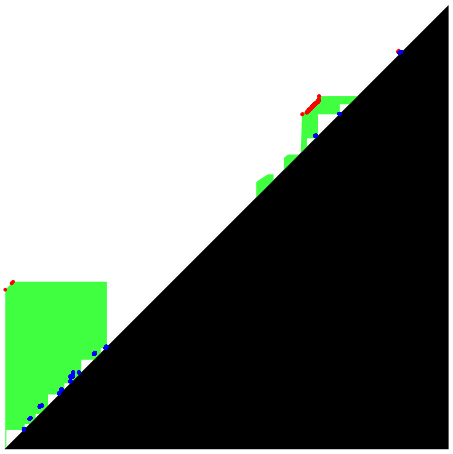


Figure 12: Comparison of the sets of intervals in the domain of **WatchingTV** that were recognized with the chronicle recognition system and the geometric approach. The green areas show the set of possible intervals obtained by the geometric approach and the red and blue dots show the earliest start-/latest end-time and earliest start-/earliest end-time solutions of the chronicle recognition approach.

1,675 earliest start/end time solutions were extracted and are represented by the dots in the figure closest to the diagonal. Overall, the dots represent two sets of unique solutions obtainable with the chronicle recognition approach. We can further interpret the result of the chronicle recognition system to encompass a continuum of intervals between the two sets of dots. These intervals would be most likely supported by the input data. Note, though, that our approach, which can reason in terms of implicitly defined sets of intervals, achieves a result that is by far more informative. The geometric approach provides a result that is comparable to the chronicle recognition system's result, but is less brittle. Furthermore, note that there are two distinct sets of intervals (slightly to the right of the middle in the figure) that the geometric approach recognizes but the chronicle recognition fails to find.

As the propagated domain of the activity variable is represented in a two-dimensional space, one cannot directly use the domain of the activity variable to visualize the occurrences of events on a "traditional" timeline. This difficulty is, however, not exclusive to the geometrical approach. Note in fact that, since patterns matched with Allen interval constraints often contain residual flexibility, one timeline only offers a partial representation of the state of the world. The typical solution to this is to extract the earliest start-/ earliest end-time solution and take this to represent the precise occurrence of activities. This can of course be done also with the two-dimensional results of the our approach, however note that we would be discarding much more information: not only is it necessary to choose the bounds of intervals to represent on the timeline, but also the intervals themselves.

Conclusion

In this paper we have shown how quantitative Allen interval constraints can be propagated onto sets of intervals defined by polygons in a two dimensional space. This approach was described in detail and also evaluated against the chronicle recognition system proposed by Dousson and Maigat (2007). In the evaluation we found that the quality of the inferred intervals is comparable, but the inference is much more efficient with the geometric approach proposed here.

Future work will evaluate the quality of the inferred activities with respect to real-world activity recognition problems. Furthermore, we intend to investigate the possibility of using more advanced clustering methods for the generation of intervals from sensory data. This could provide better descriptions of the real world events on which inference is performed, and thus increase the adherence of the obtained timelines to reality.

Also, the formal properties of the algorithm described here remain to be analyzed, and the possible usages of this representation should be evaluated in different contexts. Here we have focused on activity recognition, but the technique is applicable in other contexts where temporal constraint reasoning over multiple intervals is useful. As noted, we are also interested in exploring how timelines should be extracted from the geometric representation, as this adds the issue of interval selection to the process of extracting a timeline.

References

Aigner, W., and Miksch, S. 2006. Carevis: Integrated visualization of computerized protocols and temporal patient data. *Artificial Intelligence in Medicine* 37(3):203–218. Knowledge-Based Data Analysis in Medicine.

Allen, J. F. 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM* 26:832–843.

de Berg, M.; van Kreveld, M.; Overmars, M.; and Schwarzkopf, O. 1997. *Computational geometry: algorithms and applications*. Secaucus, NJ, USA: Springer-Verlag New York, Inc.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Dousson, C., and Maigat, P. L. 2007. Chronicle recognition improvement using temporal focusing and hierarchization. In *Proceedings of the 20th international joint conference on Artifical intelligence*, IJCAI'07, 324–329. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc.

Duftschmid, G.; Miksch, S.; and Gall. 2002. Verification of temporal scheduling constraints in clinical practice guidelines. *Art. Intelligence in Medicine* 25(2):93–121.

Floyd, R. W. 1962. Algorithm 97: Shortest path. *Communications of the ACM* 5:345–348.

Graham, R. L. 1972. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters* 1(4):132–133.

Greiner, G., and Hormann, K. 1998. Efficient clipping of arbitrary polygons. *ACM Transactions on Graphics* 17(2):71–83.

Hertel, S., and Mehlhorn, K. 1985. Fast triangulation of the plane with respect to simple polygons. *Information and Control* 64(1-3):52–76.

Keil, J. M. 1985. Decomposing a polygon into simpler components. *SIAM J. Comput.* 14(4):799–817.

Mackworth, A. K. 1977. Consistency in networks of relations. *Artificial Intelligence* 8(1):99–118.

Mansouri, M. 2011. Constraint-Based Activity Recognition with Uncertainty. Master's thesis, Örebro University, School of Science and Technology.

Pecora, F.; Cirillo, M.; Dell'Osa, F.; Ullberg, J.; and Saffiotti, A. 2012. A Constraint-Based Approach for Proactive, Context-Aware Human Support. *Journal of Ambient Intelligence and Smart Environments*. (accepted).

Pujari, A. K.; Kumari, G. V.; and Sattar, A. 2000. Indu: An interval duration network. In *Proceedings of Sixteenth Australian joint conference on AI*, 291–303. SpringerVerlag.

Rit, J.-F. 1986. Propagating temporal constraints for scheduling. In *Proceedings of the 5th National Conference on Artificial Intelligence*.

Stergiou, K., and Koubarakis, M. 1998. Backtracking algorithms for disjunctions of temporal constraints. *Artificial Intelligence* 120:248–253.

Tsang, E. 1993. *Foundations of constraint satisfaction*. Computation in cognitive science. Academic Press.

Ullberg, J.; Loutfi, A.; and Pecora, F. 2009. Towards Continuous Activity Monitoring with Temporal Constraints. In *Proceedings of the 4th Workshop on Planning and Plan Execution for Real-World Systems at ICAPS09*.

Resource-based Planning with Timelines

Debdeep Banerjee[1,2] and **Jason Jingshi Li**[3]
Quintiq[1], The Australian National University[2], EPFL[3]
debdeep.banerjee@quintiq.com, jason.li@epfl.ch

Abstract

Real world planning applications typically involve making decisions that consumes limited resources, which requires both planning and scheduling. In this paper we propose a new approach that bridges the gap between planning and scheduling by explicitly modeling the problem in terms of resources, state variables and actions. We show that it is an intuitive way to formulate real world problems with complex constraints, and that solutions can be found by compiling the problem into a constraint satisfaction problem.

Introduction

Real world planning problems typically involve actions with complex temporal constraint, where different consequences of the action come to effect at different phases of the same actions. Consider the example of the turning of a spacecraft in order to point at a target as described in (Smith, 2003). The reaction control system (RCS), must fire the thrusters to provide angular velocity, then the spacecraft coasts until it points to the destination target, then the RCS thrusters are fired again to stop the angular motion of the spacecraft. It means the firing of the thrusters happens in the beginning and the end, and is controlled by the controller. Each time the thrusters are fired, propellants are consumed and it creates vibrations which may prevent some other operation on the spacecraft. It is a complex action that has influences on various domain objects at different times. A challenge for the automated planning research is to create formalisms that efficiently model and solve problems involving such actions.

The main representation language for the planning community are PDDL and its variants (AIPS-98 Planning Competition Committee, 1998; Fox and Long, 2003). In general, the planning models are based on a description of the world in terms of propositional and numeric variables, a set of functions that defined over them, and a set of actions that changes the state of the world. Although PDDL is widely used in the planning research community and is Turing-Complete, it is difficult to use it to model many practical problems due to its lack of support for modeling different kind of resources and temporal constraints that occur in many real world settings. In particular, as argued in (Smith, 2003), complex actions that have intermediate effects are particularly difficult.

Alternatively, one can represent the problem as a scheduling problem in terms of the available resources and the durative activities that have different requirements over these resources. Alternative options to achieve the goals are represented by different modes of action execution. However, the scheduling approach lacks a high-level representation language, and in-depth domain knowledge is needed to model the problems.

In this paper, we aim to bridge the gap between planning and scheduling. Our approach is to frame a planning-scheduling problem in terms of resources, state-variables and actions over a timeline. The actions describe how the resources and state variables evolve over time. Throughout the paper we use a manufacturing setting to illustrate our approach. We show that our approach is an easier way of modeling a planning problems with complex constraints that appear in typical industry-related problems; provides a simple and intuitive semantics for actions and state transitions; and one can encode the problem as a constraint satisfaction problem (CSP) in a straight forward manner.

The work in this paper can be considered as extension of of (Banerjee, 2009), where the former work describes modeling temporal planning problems using actions and transitions. The approach in this paper explicitly models resources, and creates an action representation to allow for delayed effects. The constraint model for resource transition in our paper is related to the support-link scheduling described in (Banerjee and Haslum, 2011). The main contribution of this paper is to show that everything can fit together under a unifying framework that allow us to model resource-based planning problems, and we empirically evaluate our approach with a new solver for such problems.

Our approach is related to the ANML language, where both approaches describe a planning problem in terms of actions and multi-valued state variables. The key difference being that ANML provides temporal qualifiers to represent expressive actions, whereas we represent actions by a set of transitions. Although this restricts our representation from describing more expressive actions effects in ANML, it helps us to develop an efficient and straight forward way to encode and solve real world, resource-based planning problems as constraint satisfaction problems.

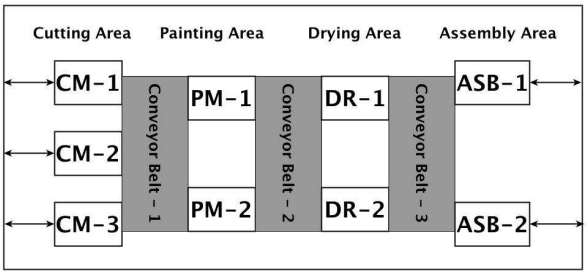


Figure 1: Illustration of a factory with 3 cutting machines (CM), 2 painting machines (PM), 2 dryers (DR) and 2 assembly areas(ABS)

The Setting

We consider the class of planning problem that require the manipulation of some scarce resource. A problem is a tuple $< R, S, A, H, I, G >$, where R denotes the resources involved in the problem, S the set of state variables, A the set of actions, H the planning horizon, I the initial state, and G the set of goal states. We will illustrate our modeling approach through the following example in a practical setting.

Example

Consider a simple manufacturing plant with 4 areas: cutting, painting, drying and assembly. In the cutting area there are a number of cutting machines that are used to produce a fixed type of parts from raw materials; the painting area has some painting machines to paint the parts with a specific color; the drying area has some drying units to dry the recently painted parts; and finally the assembly area where the parts are being assembled together on a number of assembly desks. Cutting area, painting area and assembly areas of factory floors are connected via conveyor belts, each part after being cut in the cutting machine, travels via these conveyor belts from one location to other location. Each order (o) consists of a number of different order-parts (op), each of which that needs to be cut, painted, dried and assembled with other parts. Automated conveyor-belts move the order parts from one area to another. An order is completed if all its parts are assembled.

The cutting machines are used to cut the raw material into parts. The machine has its own configuration time and offload time, where it needs a worker to configure it before it can cut a part, and a worker to offload the part and send it to the painting machines once it is done. The cutting operation also produces some waste byproducts, which must be cleaned after some iterations.

Each painting machine in the painting area is capable of painting a part with a color. If a machine has to change the color of its paint, it would require some set up time to wash the nozzles depending on what the next color would be. Setup times for changing a color from other color is given.

After the part is painted, it must be sent to the drying area to be dried. A drying unit, while it is running, can dry an unlimited number of parts. However, it may only be in continuous operation for a certain length of time, after which they would need to be switched off to be cooled for a fixed period before it can be switched back on again.

After all the parts of an order are cut, painted and dried, they are sent to an assembly desk where they are put together by a number of workers. Once it is done, the order is completed. Workers are needed in cutting area to configure machines and offloading parts when they are processed. They are also needed in the assembly areas to complete orders. For a worker going from one location to other takes time.

Resource

In our approach, at the center of the problem representation are resources. In essence, resources are domain objects in the planning world that has a finite capacity, and that they are required in order for an action to be executed. The availability of a resource is also reflected on the timeline, where at any instant in time we have both the amount of the resource available and its maximum capacity.

We divide resources in two broad categories: *Reservoir Resources (RSR)*, which are either consumed or produced by an action; and *Reusable Resources (RUR)*, which are borrowed by an action at the beginning of its execution, and returned when the action is completed. So in our example, RSRs are raw materials and waste by products in the plant, and RURs are factory machines and worker pool.

A given resource r has the following attributes:

- $capacity(r)$: an integer that denotes the maximum amount of resource units;
- $type(r)$: the type of the resource, being either Reservoir or Reusable;
- $level(r, t)$: the amount of resource used at time t, with the constraint $0 \leq level(r, t) \leq capacity(r)$;
- $freeSpace(r, t)$: the amount of resource remain available at time t.

In our example, we have the following resources with their associated capacities:

- Reservoir Resources:
  - CuttingMachinWaste: $capacity = X$ (cleaning interval)
- Reusable Resources:
  - CuttingMachine: $capacity = 1$
  - WorkersPool $capacity = Y$ (number of the workers)
  - PaintMachine $capacity = 1$

State Variables

State variables are the domain objects in the planning world that can be in one of many finitely possible states at any given point in time. In contrast to resources, they do not have a capacity. However, they may still be conditions for which actions may be executed.

A state variable sv has the following attributes:

- $dom(sv)$: the set of possible domain values of sv;
- $state(sv, t)$: the domain value of sv that holds at time t.

In our example, there are a number of orders to complete, each of which consists of multiple order parts. For each order (o), order part (op), drying unit (du), we have the following state variables with the associated domain values to denote their status:

- OrderStatus(o): {*incomplete, completed*}
- OrderPart(op): {*uncut, cut,painted,dried,assembled*}
- DryingUnit(du): {*on, off*}

We view state variables and resources as *timelines* as described in control-based modeling of planning and scheduling problems. That means by timelines of state variable and resource we will mean their evolution over time in terms of states and resource availability.

Actions and Transitions

Actions are the components that manipulate both resources and state variables. In our approach, we break down an action into the individual effects of an action on either a resource or a state variable, each with its start time and duration on the timeline. We call such individual effect a *Transition*, which may be interpreted as a temporal constraint on a specific domain object. Hence, we represent an action as a set of transitions, which is a set of synchronized durative effects with different durations. This is in contrast to the PDDL-based representation where each action has its durations and all effects takes place either at the beginning or the end of the action. Our approach allows s to intuitively model actions with multiple delayed effect, which is ubiquitous in real world applications.

A transition T has the following attributes:

- $act(T)$: the action that T is a part of;
- $req(T)$: the requirement for T to commence execution;
- $dur(T)$: the duration of T;
- $start(T)$: the start time of T;
- $end(T)$: the end time of T;
- $offset(T)$: the time delay between the start of action and the start of transition.

A transition involves only a single domain object, being either a state variable or a resource. It is typed according to its effect on the domain object. If the transition involves a state variable, it can be of either an *EFFECT* transition, one that changes the assignment of the variable from one value to another; or a *PREVAIL* transition, one that preserves the assigned value of a state variable for its duration. If the transition involves a resource, then it can either *BORROW* a certain amount of resource at the start and return it at the end; *CONSUME* the resource or *PRODUCE* the resource. In the following section we will describe in detail each type of transitions.

EFFECT Transitions For a given EFFECT transition on state variable sv, written as T^E_{sv}, $req(T^E_{sv})$ is a tuple $< s_{from}, s_{to} >, s_{from}, s_{to} \in dom(sv)$. The requirement denotes the value of sv before and after the transition. More specifically, for the transition T^E_{sv} be valid, the following constraints must be satisfied:

1. sv must be assigned to s_{start} at the start of the transition;
$$state(sv, start(T^E_{sv})) = s_{start} \quad (1)$$
2. sv must be assigned to s_{end} at the end of the transition;
$$state(sv, end(T^E_{sv})) = s_{end} \quad (2)$$
3. sv must be undefined between the start and the end of the transition.
$$\forall t \mid start(T^E_{sv}) < t < end(T^E_{sv}) : state(sv, t) = \emptyset \quad (3)$$

Given an EFFECT transition T^E_{sv} on a state variable, $\mathrm{pre}(T^E_{sv})$ denotes the pre-condition,i.e $\mathrm{pre}(T^E_{sv}) = s_{from}$ and $\mathrm{post}(T^E_{sv})$ denotes the post-condition of T^E_{sv}, i.e $\mathrm{post}(T^E_{sv}) = s_{to}$. We say the T^E_{sv} achieves the state $\mathrm{post}(T^E_{sv})$ from the state $\mathrm{pre}(T^E_{sv})$.

PREVAIL Transitions For a given PREVAIL transition on state variable sv, written as T^P_{sv}, $req(T^P_{sv})$ is a tuple $< s_p >, s_p \in dom(sv)$. The requirement denotes that sv must remain s_p for the entire duration of the transition. More specifically, for T^P_{sv} be valid, the following must constraints be satisfied:

$$\forall t \mid start(T^P_{sv}) \leq t \leq end(T^P_{sv}) : state(sv, t) = s_p \quad (4)$$

Note that for a PREVAIL transition pre- and post-conditions are the same, i.e. $\mathrm{pre}(T^E_{sv}) = \mathrm{post}(T^P_{sv}) = s_p$.

PRODUCE Transition A PRODUCE transition on resource r, written as T^R_r, reserves the amount of free-space as described by $req(T^R_r)$ at the beginning of its execution $start(T^R_r)$ for its entire duration, and produces $req(T^R_r)$ amount of resource when it is completed at time point $end(T^R_r)$. The free-space is consumed at the end of the transition.

CONSUME Transition A CONSUME transition on resource r, written as T^C_r, is the complement of a PRODUCE transition. It consumes $req(T^C_r)$ amount of resource levels at at the beginning of its execution $start(T^C_r)$, while reserves the same amount of free-space. It releases the free-space at the end of its execution.

BORROW Transition A BORROW transition T^B_r on a reusable resource r uses the resource for its duration, and at the end gives back the resource. It may be interpreted as a CONSUME transition, but returns the resource at the end of its execution.

Actions in the Example Our manufacturing example contain many actions. They include cutting, painting and drying the order parts, assembling the parts together for an order, clean a cutting machine, and switch on and off a dryer. Here we will describe the specific actions to illustrate our approach.

- CutOrderPart(order, part, cutting_machine): Cutting a part for an order on the cutting machine CM. It has 5 main transitions (Fig. 2)
  - Transition-1: A BORROW transition on the resource "Worker" for the time it takes to configure part on the cutting machine

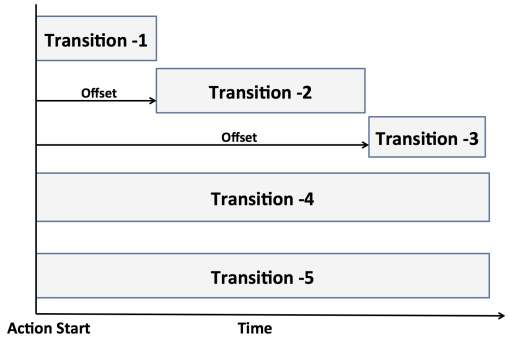


Figure 2: Action CutOrderPart(o, op, cm) and its transitions.

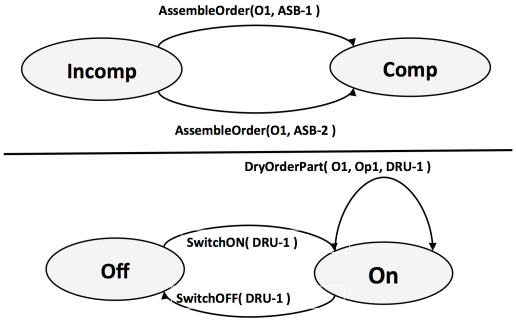


Figure 3: Transitions on order and dryer

  - Transition-2: After configuration time, it has an EFFECT transition on the state variable "Order-Part" where it changes its state from *uncut* to *cut* for the duration when cutting machine cut the part.
  - Transition-3: After cutting machine finishes the processing, it has a BORROW transition on the resource "Worker" for them it takes to offload the part from the machine.
  - Transition-4: During the whole time, it has a BORROW transition on the resource cutting_machine
  - Transition-5: During the time of processing it has a PRODUCE transition on the resource CuttingMachineWaste, where it produced 1 unit of waste.
- ColorOrderPart(order, part): The color part action 2 transitions: one BORROW transition on the resource painting_machine, and an EFFECT transition on the state variable "Order-Part" where it changes state the state *cut* to *painted*.
- DryOrderPart(order, part): This action has 2 state variable transitions: one PREVAIL transition on the state variable "DryingUnit", that needs the state *on*, and an EFFECT transition on the state variable "Order-Part" where it changes the state *painted* to *dried*.

Other actions for which we would not elaborate here include: AssembleOrderPart(order, asb_desk), CleanCuttingMachine(machine), SwitchOnDryer(dryer) and SwitchOffDryer(dryer). Figure. 3 describes how actions changes the assigned values of state variables.

Modeling Setup Constraints

Setup constraints that defines how much time must elapsed between two consecutive tasks. For a transition T, $\mathrm{Setup}(T)$ denotes the setup state of the transition. In our example, there are three scenarios where setup constraints apply; first between any two painting task of different color in a painting machine, second for each worker moving from one location to other, and thirdly between cutting and painting, painting and drying, and drying to assembly tasks of a part as it has to travel via conveyer belts from one area to other. To model setup constraints, we add a setup matrix to each state variable and resource, and assign a setup state to each transition. Setup matrices describes the time delay between pair of setup states. For example, we add a color setup matrix to each painting machine resource. If there are 3 different colors, C1 to C3, then the color setup matrix defines the time needed to change color from a pair of colors. Each resource transition on the painting machine must have a setup state among C1 to C3. For example, each ColorOrderPart(order, part, PM) action has a transition on the resource PM. If the color needed for the part is C1, then the setup state of the transition must be C1.In addition to the color matrix added to the painting machines, in the running example we add a distance location matrix to the resource "WorkerPool" that defines the time needed for each worker to travel from one location to other. All transitions on the resource "WorkerPool" must have one of the locations (locations of individual machines) as the setup state. A distance matrix is added to the state variable "OrderPart" that defines the time to travel from one area to other area via conveyor belt for a part, all transitions on the state variable must one of the areas $:cutting, painting, drying, assembly$ as setup state.

Initial State, Goal State and Planning Horizon

Similar to PDDL, a planning problem contain the description the initial state, which denote the state of the world we start in; and the goal state, the things that we want to be true. It may also contain the description of a planning horizon, which is the maximum time we allow the plan to reach the goal state. Abusing the notation, we would also say that the value a particular domain object (state variable or resource) is in the initial/goal state, if the value assignment is part of the description of the initial/goal state.

The Solution to a Planning Problem

In our setting, the solution to a planning problem is a *valid flexible plan*, which denotes the set of actions to be executed from an initial state to reach a goal state, and maintaining that no constraint is violated at any time during its execution.

Flexible Plan A flexible plan is simply a set of tuples $(a, [X, Y])$, where a is an action, and $[X, Y]$ is a time interval where X and Y are specific time points, which that denotes the range of the possible starting time of action a.

In executing a flexible plan, the agent chooses a starting time from the specified time interval of each action. When

all the starting time of the actions are specified, the resulting plan is a realization of the flexible plan.

Schedule A realization of a flexible plan creates a schedule for every state variable and resource, where a **schedule** is a sequence of transitions with fixed start time. A schedule of a state variable is 'em valid, if and only if the preconditions of the first EFFECT transition is in the initial state, the postconditions of the last EFFECT transition is in the goal state, and the following holds at any given time:

- If an EFFECT transition is in execution, then no other transitions are in execution on this state variable;
- If a PREVAIL transition is in execution, then other PREVAIL transitions in execution on this variable must also require the same state.

A schedule of a resource is valid if and only if the following holds:

- At any time, there exists no set of transitions in execution such that the total resource requirement of the set is greater than the capacity of the resource;
- Immediately after the execution of all the transitions that starts at the initial time point, the level of resource must be less than or equal to the initial level of the resource;
- at the end of the planning horizon the amount of resource in r is within the range defined in the goal state.

Hence, we say that a realization is *valid*, if and only if it creates a *valid schedule* for every state variable and resource, and a flexible plan is *valid* if and only if every possible realization of the plan is a valid realization.

Encoding the Problem as a CSP

In our approach, the search for the solution of a planning problem is to find a *valid flexible plan*, which represents a set of valid schedules for every state variable and resource. This corresponds to a set of temporal constraints on every domain object. Therefore, it is natural to model the planning problem as a Constraint Satisfaction Problem (CSP) on the domain objects.

The constraint model for each state variable can be thought of as the constraints for **causal-links** between pairs of state variable transitions. First introduced in Partial-Order-Planning (McAllester and Rosenblitt, 1991), a *causal-link* $a[p]a'$, represents the fact that action a achieves the pre-condition p for action a'. In our approach, a causal link on a state variable sv, written as $T_{sv}[s]T'_{sv}$, denotes the precedence relation between two transitions. T_{sv} is an EFFECT transition that make the precondition of the latter transition T'_{sv} true. Solving the constraint problem on every state variable is analogous to deciding which causal links hold in the final plan, where all the precedence constraints between those pair of transitions are satisfied.

The constraint model for each resource is based on deciding the **support-links** between pairs of transitions on the resource. On a resource r, a *support-link*, $T_r[\delta]T'_r$, denotes that transition T_r provides δ amount of resource towards the requirement of T'_r. If $\delta = 0$, it means T_r does not provide any support to T'_r. If $\delta > 0$, then the support link implies a precedence relation between T_r and T'_r. By deciding how transitions provide support to other transitions, i.e. creating support links, we build a schedule on each resource.

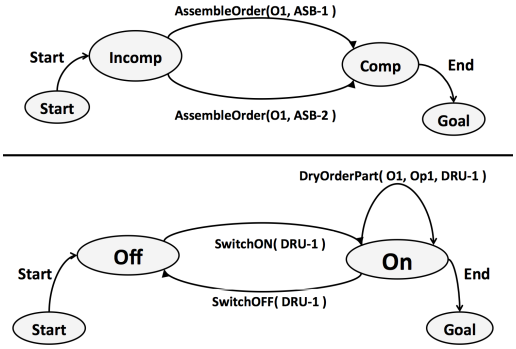


Figure 4: Additional states and actions

Each *causal* and *support link* implies a precedence or ordering relation between a pair of transitions. A precedence relation between two transitions $T \rightarrow T'$ means that T' starts after T finishes its execution. Since each transition is non-preemptive, and the start times of transitions and their corresponding actions are synchronized, each precedence constraint implies a precedence relation between the actions of the transitions. The constraint model for actions maintains the transitive closure of these precedence relations.

Preprocessing

We first introduce a preprocessing step before encoding the problem as a CSP. It introduces new states, actions and transitions such that it allows the resulting CSP to be solved in a more efficient manner.

Additional States for State Variables For each state variable $sv \in SV$ we add two additional states to its domain of possible states: $start_{sv}$ and end_{sv}.

Dummy Start **and** End **actions** We add two dummy actions Start and End into the set of actions A, where Start and End mark the achievement of the initial state and goal respectively. The Start action is constrained to appear at the beginning of the plan, before any other action in the plan, and the End action is constrained to appear at the end of the plan, after every other action. Introducing these dummy Start and End actions is a standard practice in modeling partial order causal link (POCL) planning (McAllester and Rosenblitt, 1991). Note that all transitions of all dummy actions, Start, End and other dummy actions that we introduce below, have duration 0.

On each state variable $sv \in SV$, the Start action has an EFFECT transition T_{sv}^{start} that changes the state of sv from the dummy $start_{sv}$ state to the initial state $\mathrm{init}(sv)$, representing the achievement of the initial state of sv. The End action has an EFFECT transition T_{sv}^{end} on each state variable sv that changes its state form either the goal state to the end_{sv} state (Fig. 4).

Similar to state variables, on each resource $r \in R_{reserve} \cup R_{reuse}$, the Start action has a resource transition T_r^{start}, and

the End action has a resource transition T_r^{end}.

The DUMMY start action's transitions will be referred as initial transitions and the DUMMY end actions' transitions will be referred as goal transitions in the following sections.

CSP Variables and Domains

To formulate the problem as a CSP we create the following CSP variables:

- **next[T]**: For each state variable transition T, except for the goal transitions, the CSP variable next[T] represents which EFFECT transition immediately follows T. Domain of next[T] contains all EFFECT transitions that can immediately *follow* T. That is, domain of next[T] contains all state variable transitions T′ such that $\mathrm{post}(T) = \mathrm{pre}(T')$. Note that for a transition T, all transition in the next[T] domain are the transitions on the same state variable as T.
- **previous[T]**: For each transition T, except for the initial transitions, previous[T] represents which EFFECT transition is immediately before T. Domain of previous[T] is the set of EFFECT transitions that can appear immediately *before* T. That is, domain of previous[T] contains all T′ such that $\mathrm{pre}(T) = \mathrm{post}(T')$ where both T and T' are on the same state variable.
- **inplan[A]**: For each action A, inplan[A] represents if the action A is in the plan or not. There are two possible values for inplan[A], *true* or *false*.
- **support[T_r,T'_r]**: For each pair of resource transitions $< T_r, T'_r >$, where T_r and T'_r are on the same resource and T_r is not a goal transition and T'_r is not an initial transition, the variable $\mathrm{support}(T_r, T'_r)$ represents the amount of resource (a non-negative integer) T_r provides to T'_r. Note, on a reservoir resource, there can be two types of transitions: PRODUCE transitions and CONSUME transitions. A PRODUCE transition T_p produces $\mathrm{req}(T_p)$ amount of resource at the end, and a CONSUME transition T_c consumes $\mathrm{req}(T_c)$ amount of resource at the start. For this reason we say that a PRODUCE transition can only provide support to a CONSUME transition and vice versa. Note that both PRODUCE and CONSUME transition can provide support to goal transitions on resources. So what all this means is the for each $\mathrm{support}(T_r, T'_r)$ variable we define, if T_r is a PRODUCE transition, then T'_r must either a CONSUME transition or goal transition, and if T_r is a CONSUME transition then T'_r must be a PRODUCE transition or goal transition. If δ_{T_r,T'_r} denotes the maximum amount of resource that T_r can provide to T'_r, then

$$\delta_{T_r,T'_r} = \min\left(\mathrm{req}(T_r), \mathrm{req}(T'_r)\right)$$

The domain of each $support(T_r, T'_r)$ is the interval $[0, \delta_{T_r,T'_r}]$, where 0 indicates that T_r does not support T'_r.

Note, that although either the next or the previous variables alone are sufficient for the encoding, using both provides an opportunity for better propagation. Each assignment of the *next* and *previous* variable creates a *causal-link*, and assignment of *support* variables represents a *support-link*.

There are two additional variables for each transition T: **start[T]**, which represents the start time of T, and **end[T]** representing the end time of T. Similarly, for each action A, a variable **start[A]** represents the start time of A and **end[A]** represents the end time of the action.

We maintain two sets for each transition T, **before(T)** and **after(T)** to represent precedence relations between transitions on the same domain object, where before(T) contains all the transitions T' where $T' \rightarrow T$ holds, and similarly, after(T) contains all the transitions T'' where $T \rightarrow T''$ holds.

For each pair of actions A and B, we maintain a variable dist(A,B) that represents the distance in time from start of A to start of B, i.e. dist(A,B) = Start(B) - Start(A).

The next[T] and previous[T] variables can be assigned to a **not-in-plan** value $\perp$, which will denote that the transition T will not be part of the final plan.

Constraints

1. **EFFECT Position Constraints**: If an EFFECT transition T appears before another EFFECT transition T', then T' must appear after T and vice versa, i.e. $\forall T', T' \in$ EFFECT

$$previous[T'] = T \Leftrightarrow next[T] = T'$$

2. **PREVAIL Position Constraints**: The following constraints holds for all PREVAIL transition T_p that can appear next to T_e and before T_a, where T_e and T_a are EFFECT transitions.

$$previous[T_p] = T_e \wedge next[T_p] = T_a \Rightarrow next[T_e] = T_a$$
$$previous[T_p] = T_e \wedge next[T_e] = T_a \Rightarrow next[T_p] = T_a$$
$$next[T_p] = T_a \wedge next[T_e] = T_a \Rightarrow previous[T_p] = T_e$$

Note that all next and previous variables' domains are consists of only EFFECT transitions, not PREVAIL transitions. This is case because PREVAIL transitions do not change a state, so they can't appear in the left side of the causal link $T[s]T'$. The next and previous variables model the causal links.

3. **Action Synchronization Constraints**: If an action is in the plan then all the transitions caused by the action must also be in the plan and vice versa, i.e for all action A,

$$inplan[A] = true \Leftrightarrow \forall_{T.act=A} T : \neg(next[T] = \perp).$$

Note that this constraint is bi-directional, i.e. if $\perp$ is removed from a next variable then it implies that the corresponding action is included in the plan.

4. **Transition Exclusion Constraint**: If a transition T is excluded from the plan, then no transition can appear before or after it, i.e. for all transition T [1],

$$next[T] = \perp \Leftrightarrow \text{previous[T]} = \perp.$$

5. **Action Time Synchronization Constraints:** Start times of transitions must be consistent with the start time of their corresponding actions and vice versa.

$$start[A] = \forall_{\mathrm{act}(T)=A} T : start[T] - offset(T)$$

[1] both EFFECT and PREVAIL transitions

Similarly, each action's end time is must be equal to the maximum of the end times of its transitions.

$$end[A] = \max\{\forall_{\mathrm{act}(T)=A} T : end[T]\}$$

6. **Support Constraints**: Each assignment of a next variable implies a precedence constraint between the transitions.

$$next[T] = T' \Rightarrow T \rightarrow T'$$

Similarly, each assignment of the support variables also implied a precedence constraint between the transiitions.

$$support[T, T'] > 0 \Rightarrow T \rightarrow T'$$

7. **Temporal Position Constraints**: For each precedence constraint $T \rightarrow T'$ we post the following temporal constraints

$$inplan[T] \Rightarrow \mathrm{dist}(\mathrm{act}(T), \mathrm{act}(T')) \geq \mathrm{dur}(T) + \mathrm{offset}(T) - \mathrm{offset}(T') + setuptime(\mathrm{Setup}(T), \mathrm{Setup}(T'))$$

Recall that $\mathrm{dist}(\mathrm{act}(T), \mathrm{act}(T'))$ represents the distance between the starlings of the two actions: $\mathrm{start}(\mathrm{act}(T')) - \mathrm{start}(\mathrm{act}(T)$, and the $setuptime(\mathrm{Setup}(T), \mathrm{Setup}(T'))$ denotes the time delay needed between the given setup states.

8. **Non-preemptive Transition Constraints:** Since transitions are non-preemptive, the following condition must hold for all transition T.

$$end[T] - start[T] = T.duration$$

In additions to these constraints we maintain the transitive closure of the precedence relations conditioned on the inclusion of transitions. That means, for transitions T, T' and T'', if $T \rightarrow T'$ and $T' \rightarrow T''$, we only post the precedence relation $T \rightarrow T''$ if and only if inplan[T']= true.

Preliminary Evaluations

We implemented a simple constraint solver to solve such planning problems in C++. As we are not aware of any public available planning benchmarks with complex temporal constraints on resources, nor solvers readily available to easily model and solve such problems, we evaluated our own solver with a set of randomly generated benchmark representing completing orders in the factory depicted in the example in Fig. 1. Our solver ran on servers with AMD Opteron(TM) Processor 6272 at 2.4GHz. We enforce a time limit of 30 minutes and memory limit of 2GB per instance.

We tested our factory setting with 5, 10, 15 and 20 orders at 50 instances each. The table in Fig. 5 reports the number of transitions, the average cpu time for the solved instances, and the number of unsolved instances. Fig.6 shows the performance of our solver against cpu time. The problem is solved almost instantaneously for small instances, and the difficulty of the problem increases with its size. We note that our solver is still under development, and its performance should improve once more powerful propagators are used.

orders	transitions	avg cpu time	failure
5	188.44	2.37	0/50
10	347	32.69	0/50
15	649.95	133.54	8/50
20	855.87	299.87	4/50

Figure 5: Solver statistics for factory with 5 to 20 orders.

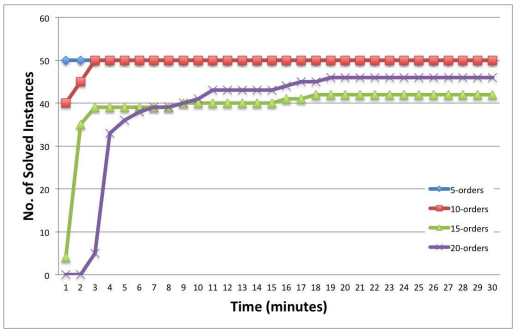


Figure 6: Number of solved instances against CPU time.

Summary

We proposed a new approach to model and solve planning problems with resources. Our approach extends an action-based planning planning domain description that provides an easy and direct way to model practical problems with complex temporal constraints. It concisely compiles the planning problem as a constraint satisfaction problem, and provides an interesting alternative to the current state of the art in modeling a wide ranges of possible planning applications. Possible future work includes improving the solver performance by improving the efficiency of constraint encoding and propagation, and evaluate our solver against other existing solvers on a set of expressive benchmark problems.

References

AIPS-98 Planning Competition Committee. PDDL - The Planning Domain Definition Language. Technical Report, Yale Center for Computational Vision and Control, 1998.

D. Banerjee. Integrating Planning and Scheduling In a CP Framework : A Transition-based Approach In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

D. Banerjee and P. Haslum. Partial-Order Support-Link Scheduling. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

M. Fox and D. Long. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelligence Research*, 20: 61-124, 2003.

David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634639, 1991.

D. Smith The case for Durative Actions: A Commentary on PDDL2.1 *Joural of Artificial Intelligence Research*, 20:149-154, 2003.

D. Smith, J. Frank and W. Cushing. The ANML Language In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling*, 2008.

Maintaining Timelines with Hybrid Fuzzy Context Inference

Masoumeh Mansouri and **Federico Pecora** and **Alessandro Saffiotti**
Center for Applied Autonomous Sensor Systems
Örebro University, SE-70182 Sweden
{mmi,fpa,asaffio}@aass.oru.se

Abstract

Timelines allow to represent temporally-rich information about plans as well as the current execution status of plans. Recent work has addressed the related issue of inferring timelines representing contextual information — often useful for informing planning and/or plan execution monitoring processes. The present article addresses the particular issue of inferring context from given models of how observations relate to context, and representing this context on timelines. We strive to abandon assumptions currently made on context recognition, namely that hypotheses are either confirmed or disproved. We propose a technique which allows to accept the inferred context on a timeline with a degree of possibility. The approach is based on fuzzy constraint reasoning, and captures two sources of uncertainty: uncertainty in the model that is used to infer context, and uncertainty in the observations. We also formulate the problem of searching for the most likely timeline as a Constraint Optimization Problem.

Introduction

The concept of timeline is central to many planning and scheduling approaches. Timelines allow to represent temporally-rich information about plans as well as the current execution status of plans. Recently, several continuous planning approaches have been proposed which use timelines to maintain the on-going state of the domain as it is observed by sensors. This allows the planner to infer courses of action which are contextual to the current status of the world. Applications in which the temporal context is used in a closed loop with planning include domestic activity management (Pecora et al., 2012) and unmanned aerial vehicles (Doherty, Kvarnström, and Heintz, 2010). In these works, plan generation and execution depends in non-trivial ways on the temporal relationships among observations. These relationships are modeled in languages based on temporal constraints, and the algorithms which allow to infer the timelines representing context are based on temporal constraint propagation techniques. Examples of these algorithms include chronicle recognition (Dousson and Maigat, 2007) and abductive temporal inference (Pecora and Cirillo, 2009). The present article addresses this particular issue, namely inferring context from given models of how observations relate to context, and representing this context on timelines.

Timeline generation and maintenance can take into account some degree of uncertainty in the temporal placement of values. This is typically achieved by means of temporal constraint reasoning techniques (e.g., Simple Temporal Problems, STP (Dechter, Meiri, and Pearl, 1991)) which are capable of maintaining temporal *bounds* on observations and inferred values on the timeline. However, current approaches to context inference lack the ability to account for two important sources of uncertainty, namely uncertainty in the model that is used to infer timelines that represent context, and uncertainty in the observations. To motivate why this is useful let us begin with an example. Suppose we have the ability to observe the location of a person at home, as well as the current state of the kitchen stove. We wish to recognize the occurrence of the observed person being in the context of "Cooking". We employ for this purpose a model which asserts that the user is cooking if the stove is on while he/she is in the kitchen. Now, suppose the scenario unfolds as follows: the user enters the kitchen and turns on the stove; after a while, he turns off the stove before leaving the kitchen. The stove state sensor is based on temperature: when the temperature goes below a certain threshold, the stove is considered to be off. Therefore, the sensor observations associated to the stove will indicate that it is still on after the user has left the kitchen. This situation falls outside the model stated above. The failure to recognize the cooking activity is not due only to sensor imprecision; it is also a consequence of the fact that model cannot anticipate all the possible states which can occur in a real situation. In other words, this failure is caused by the incapability of the inference process to deal with uncertainty in the model. Similarly, we may want to take into account the fact that the sensor providing the location observations gives uncertain readings. It may be the case, for instance, that its image processing algorithm returns an uncertain estimate of the person being in the kitchen, and that we want to factor this information into the context recognition process in order to provide a degree of belief of the cooking situation.

Two strategies can be adopted in order to extend the ap-

plicability of a model under uncertainty. First, to replace the modeling language with one that can explicitly include measures of uncertainty; second, to measure the applicability of a crisp model to the current situation using an underlying theory of uncertainty. In this work, we follow the latter approach. By doing so, we maintain the ability to use a simple model to represent domain knowlege, and we accommodate uncertainty by modifying the inference process. We use techniques based on fuzzy logic to measure the similarity between the situation reported by the sensor readings and the situation described in the model. We leverage fuzzy constraint based reasoning in an abductive reasoning process to recognize a context on the basis of a set of heterogeneous sensor readings, which may contain uncertainty. Note that we are not interested in how the uncertainty in the sensor readings is computed: we assume that sensors come with an appropriate sensor model. The use of fuzzy temporal inference techniques may result in general in the generation of multiple timelines which are compatible, to some degree, to the sensor data. Accordingly, in the last part of this paper we present an approach to extract a unique, most likely timeline.

This paper is organized as follows. First, the overall context recognition algorithm is explained. This algorithm relies on two constraint-based inference processes, namely propagation in a fuzzy value constraint network and in a fuzzy temporal constraint network. After detailing these constraint reasoning procedures, we investigate two important problems caused by accommodating uncertainty in the inference process, namely (1) that of determining quantified temporal bounds for the inferred context, and (2) that of determining the most likely timeline.

Constraint-based Context Modeling

In this work, we leverage a context inference algorithm similar to the one described by Pecora et al. (2012). A context is an inferred value (e.g., activity of a human) which is the result of inference based on the observable properties in an environment (e.g., sensor readings). Both observable properties and inferred values are represented as state variables. For instance, a state variable *Human* can represent the current activity of a user, e.g., {Cooking, Eating, Relaxing, Sleeping}, or model the possible values of observable properties of the environment. We refer to these values as sensor readings e.g., a state variable can represent a *Stove* whose values can be {On, Off}.

Correlations among the values of state variables are defined in a model in terms of two dimensions of knowledge. One is the value dimension, which refers to the possible values a state variable can assume (e.g., *Human* = Cooking). The other dimension is the temporal dimension, which is formulated as relations in Allen's Interval Algebra (Allen, 1984). These relations are qualitative, and model the relative placement in time of state variables values (e.g., *Human* = Cooking *During Stove* = On). In this example, a "=" relation is imposed on the values of both the *Human* and *Stove* state variables. Note that similar approaches (Pecora et al., 2012; Dousson and Maigat, 2007) implicitly assume an "=" relation on values. In our approach, we admit the "$\neq$" relation as well, e.g., it is possible to assert in the model that Cooking depends on *Location* $\neq$ Bedroom.

The collection of temporal and value relations constitutes the model based on which context inference occurs. Relations which assert the same value on a state variable that represents an inferred property of the environment (e.g., *Human*) are collected into so-called rules. These rules are from a domain model. For instance, the following rule describes one possible condition under which the human activity of Cooking can be inferred:

(*Human* = Cooking) *During* (*Location* = Kitchen) $\wedge$
(*Human* = Cooking) *Contains* (*Stove* = On)

Given a rule like the one above, we call *head* of the rule, the value of the state variable representing the inferred property (e.g., *Human* = Cooking), and we call *requirements* the relations and values of the other state variables involved in the rule.

The entire inference process is an abductive reasoning process, whereby observed or previously inferred values are explained by hypothesizing the occurrence of specific values and testing these hypotheses repeatedly by using the rules in the model. Abductive reasoning imposes the requirements of a rule as constraints between an interval h, which represents a hypothesis of the head of the rule occurring, and other intervals representing sensor readings as they have been observed. Intervals representing sensor readings are maintained in a so-called Sensor Constraint Network (SCN, Figure 1). The SCN contains all sensor readings and the relations among them as they are observed. The resulting set of constraints is propagated in order to decide whether the hypothesis is admissible.

Context inference with uncertainty

The context inference is done through the abductive reasoning process on patterns of sensor observations and places these patterns and temporal relations together with the model in a constraint network. Temporal constraint propagation ascertains whether what is hypothesized has occurred. However, in current approaches inference has a Boolean result: either the hypothesis is confirmed or it is disproved. These approaches would fail to recognize an activity like cooking in a real world scenario (see the example in Introduction). As mentioned, the lack of flexibility in the model is what caused the failure. One way to tackle this problem is to have a much larger set of rules that captures many occurrences of sensor patterns. This approach has important disadvantages. One is that it is cumbersome to model a collection of rules that are able to anticipate all the possible relative temporal relations among the sensor reading patterns. More importantly, increasing the number of rules enlarges the search space in an abductive reasoning process in which each single rule should be hypothesized and propagated within a constraint network. In our approach, we follow a different strategy to have a conservative extension of the crisp case, namely we employ uncertain inference to relax the model. Specifically, we employ the notion of fuzzy constrain satisfaction problem (Dubois, Fargier, and Prade, 1996). We replace crisp temporal constraints with fuzzy temporal constraints, therefore, computing a possibility de-

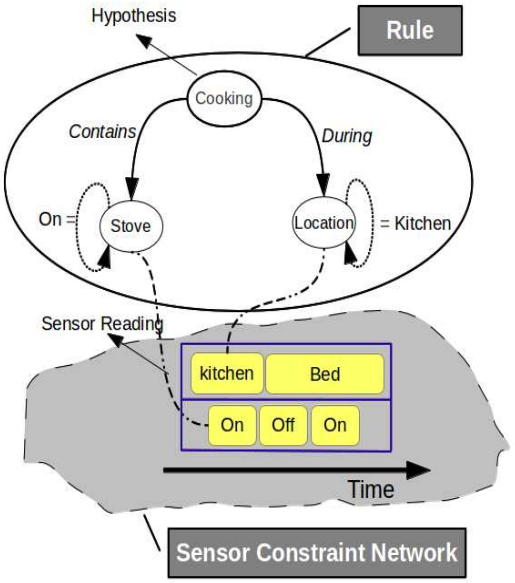


Figure 1: A rule models temporal and value relations that exist between sensor readings and inferred context expressed as sets of temporal and value constraints between state variable values.

gree of temporal admissibility. In addition to the temporal facet of the model, we include a fuzzy CSP for the values of state variables, so as to compute a possibility degree with respect to the value aspect.

Overall, we employ two constraint reasoning procedures, one for temporal inference and one for value inference. The former occurs in what we call *fuzzy value constrain network*, the latter in a *fuzzy temporal constraint network*. The steps of the fuzzy inference process are:

- Create a hypothesis which is the head of a rule, and constraint the required values in SCN.
- Compute the value consistency and the temporal consistency of the hypothesis by:
  - performing arc consistency (AC) on the fuzzy value constraint network (due to the structure of the fuzzy value constraint network, AC is sufficient to obtain the maximum value consistency (Mansouri, 2011)),
  - performing a fuzzified version of Allen's algorithm (path consistency) on the the fuzzy temporal constraint network and obtaining an optimal solution through backtracking.
- Estimate the temporal bounds of the hypothesis for the purpose of generating a timeline by:
  - creating a STP from the optimal solution of temporal constraint network,
  - propagating the STP with the Floyd-Warshall algorithm (Floyd, 1962) to obtain temporal bounds of the hypothesis interval.

Inference in the fuzzy value and temporal networks is explained in the following sections. Note that once an overall possibility degree for a hypothesis is obtained, we are left with the task of deciding temporal bounds for the inferred hypothesis. This requires the use of quantitative temporal constraints, which we propagate in a STP to obtain the bounds of inferred hypotheses according to their most likely temporal placement. This latter procedure yields timelines, each of which has an associated possibility degree. The problem of finding the timeline which has the maximum possibility degree, therefore the timeline representing the most likely overall sequence of inferred context, can be cast as a constraint optimization problem. This last aspect of our work is currently being investigated, and therefore we limit the description of the search of the most likely timeline to problem definition.

Fuzzy Value Constraint Network

A fuzzy constraint network (fuzzy CN) is a triple $\langle X, D, C\rangle$ where X and D are a finite set of variables and their domains, and C is a set of fuzzy constraints. The fuzzy value constraint network which is built for the purpose of this work includes two categories of variables. One consists of the variables which model the value requirements of a hypothesis, and the other represents sensor readings. The domains D of all variables are the symbols representing possible states (e.g., {On, Off} for the variable representing the *Stove*). A fuzzy constraint is a fuzzy relation R on a set of variables $V \subseteq X$ which is denoted as R_V. This relation, that is a fuzzy set of tuples, is defined by a membership function μ_{R_V} (Klir and Folger, 1988). Each tuple $t_V \in R_V$ is a assignment of values to the variables in V, and the membership function assigns a degree of possibility in $[0, 1]$ to each tuple. Notice that possibility degrees do not need to add to one: for example, the void constraint over the values of V is represented by $\mu_{R_V}(t_V) = 1$ for all t_V. The projection of a tuple t over a sequence of variables V is denoted by $t[V]$. A solution of a fuzzy CN is a complete assignment for all variables in X with satisfaction degree greater than 0. The satisfaction degree of a complete assignment t is

$$\deg(t) = \min_{R_V \in C} \mu_{R_V}(t[V])$$

The optimal solution $\hat{t}$ of a fuzzy CN is the complete assignment whose membership degree is maximum over all complete assignments (Ruttkay, 1994), that is,

$$\hat{t} = \arg\max_{t \in \prod_{x_i \in X} D_i} \deg(t)$$

Sensor processes continuously add to the SCN new variables which represent perceived sensor values. The degree of belief of a particular value is modeled as a soft unary constraint. For example, the belief that the stove is on with possibility α_1 and off with possibility α_2 is modeled with a unary constraint whose membership function imposes α_1 and α_2 on the values On and Off, respectively. A possibility degree of 0 or 1 is also assigned to each value of the state variables which model the value requirement. As shown in Figure 2 for instance, the variable "Stove2", representing the

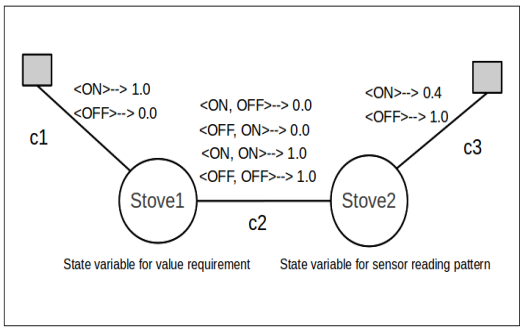


Figure 2: A fuzzy CSP with two state variables, three constraints (Soft unary constraint {c1, c3}, Hard binary constraint c2) and domain {On, Off}. 0.4 for On and 1.0 for Off are data obtained from sensor

perceived state of the stove, is constrained to assume values On = 0.4 and Off = 1.0. Another variable "Stove1" is used to represent the value requirement of a rule asserting that the stove should be on. If $X = \{x_1, ..., x_k\}$ is a finite set of k variables, the membership function for the unary fuzzy constraints imposed on every variable $x_i \in X$ is expressed as

$$\mu_{R_{x_i}}(t) \rightarrow [0, 1] \qquad t \in D_i$$

Binary constrains of the fuzzy value constraint network are hard constraints. This is because binary constraints represent the equality and inequality requirements of a rule (e.g., we want the *Stove* = On). For example, a tuple (a, a) which is a possible assignment for two variables, is assigned to the value 1 in case of an "=" constraint, and to value 0 for the case of the inequality constraint. The membership function for a binary hard constraint over variables $w_{ij} = \{x_i, x_{j \neq i}\}$ is defined as

$$\mu_{R_{w_{ij}}}(t) \rightarrow \{0, 1\} \qquad t \in D_i \times D_j$$

In the example shown in Figure 2, a binary constraint is used to enforce that the value requirement "Stove1" is equal to the perceived state represented by variable "Stove2".

In order to obtain a maximum possibility degree of the fuzzy constraint network, arc consistency and search are performed. Since the structure of the network in this specific application is a tree, the problem of finding the maximum satisfaction degree can be solved in polynomial time (Dechter, 2003). In fact, the structure of the network follows the semantics of unification, in which there is a value relation (= or $\neq$) obtained from a rule between the variable modeling the value requirement and the variable modeling a sensor reading, therefore, the structure is always a tree.

The maximum possibility degree of the network shown in Figure 2 is 0.4. If the binary constraint stated in the rule had been an inequality, the value possibility of the network would have been 1.0. The semantics of inequality are omitted here and explained in more detail by Mansouri (2011).

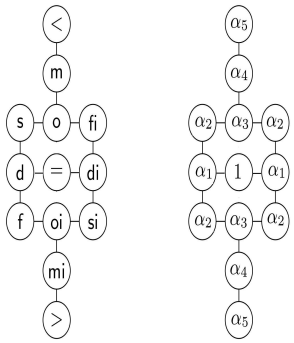


Figure 3: The Allen relations and their membership grades with respect to the relation *Equals*. In this figure, Allen's relations are defined as follows: *Meets*(m), *During*(d), *Before*(<), *Overlaps*(o), *FinishedBy*(fi), *Contains*(di), *StartedBy*(si), *Equals*(=), *Starts*(s), *Finishes*(f), *OverlappedBy*(oi), *MetBy*(mi), *After*(>). (Adapted from Guesgen (2002).)

Fuzzy Temporal Constraint Network

In the previous section, we addressed how to check the value eligibility. In addition to the value constraints, we have to consider the temporal requirements in the rules. As explained in the section "Constraint-based Context Modeling", the temporal constraint network is created by imposing temporal requirements of the hypothesis on SCN. We call this network, "original temporal constraint network". The objective is to find the consistent temporal constraint network that is closest to the original one. If the original temporal constrains network is not consistent, we find the closest consistent temporal constraint network through introducing uncertainty to the original network. The notion of flexibility and uncertainty for the temporal aspect of the model are provided by fuzzifying Allen relations (Guesgen, 2002). A fuzzy Allen relation is represented as a set of crisp Allen relations with an associated possibility degree. To define the membership grade, the notion of conceptual neighborhood is leveraged (Freksa, 1992). For instance, assume that two intervals $I1$ and $I2$, are in relation *Equals*, then by allowing the duration of the intervals to vary, we can change this relation to *During* or *Contains*. In this case, to make a fuzzy set including a pair of all thirteen Allen relations, we assign the membership grade 1 to the relation *Equals* and a membership grade less than 1 for the others. The membership grades are defined for each relation based on the closeness to the *Equals* relation. Figure 3 illustrates this example in the topological view of conceptual neighborhood with the membership grades, $1 = \alpha_0 \geq \alpha_1 \geq \alpha_2 \geq ... \geq 0$

We divide Allen relations in the temporal constraint network into two categories. The first category belongs to the relations imposed from a rule and the second is for the relations capturing the relative position in time of the sensor readings. Since we want to determine the degree of possibility of a rule in the model given existing sensory patterns, we fuzzify the relations of the first category in the way ex-

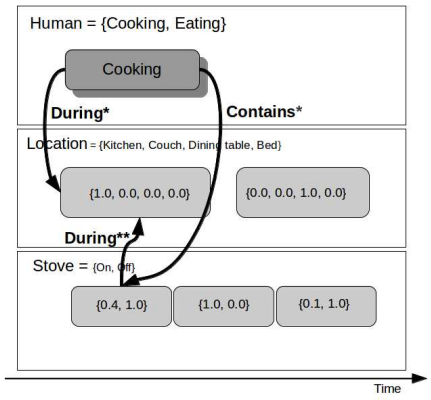
plained earlier. As for the second category, Allen relations remain as crisp constraints. In other words, for Allen relations that are responsible for placing sensor readings in time, the initial relation is assigned to 1, while the membership grade of the others is 0.

In order to have an optimal solution for the fuzzy Allen network (i.e., a crisp, consistent temporal constraint network that is closest to the original temporal constraint network), we use a generalization of the classical backtracking algorithm with incremental path consistency to the fuzzy framework (Badaloni and Giacomin, 2000). More precisely, by applying a fuzzy extension of Allen's algorithm, we obtain the maximum temporal possibility degree as well as the sets of complete assignments with the highest possibility degree. The maximum temporal possibility is calculated by taking the minimum over the maximum membership degree of each edge in the propagated fuzzy temporal constraint network.

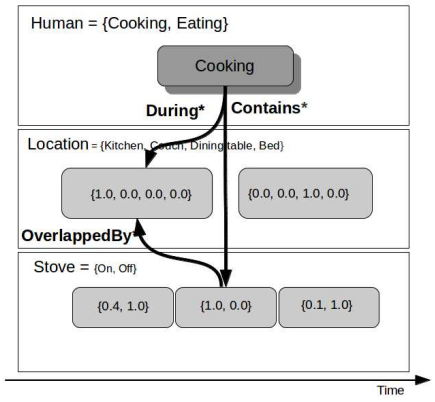
In crisp temporal constraint reasoning, finding a feasible solution to the interval constraint problem is solved with a combination of constraint propagation (path consistency) and search. The complexity of the fuzzy extension to Allen's propagation algorithm is augmented at most by a factor equal to the number of levels of preferences used to define the fuzzy Allen network (Badaloni and Giacomin, 2000). In our work, there are as many levels of preference as there are conceptual neighbors.

An example of fuzzy temporal reasoning is shown in Figure 4. With respect to Figure 4(a), the relation *During** is fuzzified as follows:[1] {(*Before*, 0.2), (*Meets*, 0.4), (*Overlaps*, 0.6), (*FinishedBy*, 0.4), (*Contains*, 0.6), (*StartedBy*, 0.4), (*Equals*, 0.8), (*Starts*, 0.8), (*During*, 1), (*Finishes*, 0.8), (*OverlappedBy*, 0.6), (*MetBy*, 0.4), (*After*, 0.2)} and the relation *During*** is a crisp relation with the membership degree of 1 for *During* and 0 for the rest of Allen's relations. By applying propagation on the fuzzy temporal network depicted in Figure 4(a), these membership degrees are updated. The optimal solution is built through the search process. In this example, we obtain the maximum possibility degree of 1 as a result of taking minimum over membership degrees of the relations in the optimal solution. Having maximum possibility degree of 1 indicates that this network is temporally consistent. Clearly, in this case, the closest temporal constraint network is the original one.

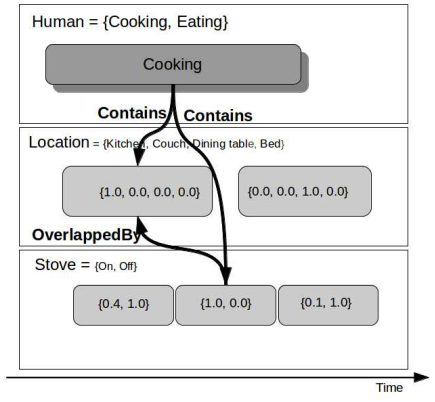
Suppose the case that the Allen relation *During*** which models the temporal relation of sensor reading patterns is replaced by an *OverlappedBy* relation. This case is compatible with the Cooking scenario explained in Introduction Section, in which the network was not temporally consistent (see Figure 4(b)). By fuzzifying this network and propagating the resulting network, we obtain a maximum possibility degree of 0.6. Several selections of Allen's relations entail this possibility degree. One of them is represented in Figure 4(c). In other words, the network shown in Figure 4(c) represents the "closest interpretation" of the temporal relations among sensor readings that would make the rule consistent. The "similarity" between this interpretation and the one required by the rule is 0.6.



(a)



(b)



(c)

Figure 4: Fuzzy Temporal constraint networks corresponding to the values of stove and location state variable

[1]Different choices of the α_i membership values can be used to reflect how far from the model we are willing to stray.

Determining the Overall Possibility Degree

We explained two fuzzy constraint networks which ascertain temporal satisfiability and value satisfiability. Now, to obtain an overall possibility degree of an inferred value from the two individual degrees, we are facing to the problem of multi criteria aggregation. Different operators can be employed to achieve this aggregation. A primary factor in the determination of the structure of such aggregation is the relationship between the criteria involved. Consider the cases of wanting, for example, "all" or "at least one of them" or "most" of the criteria to be satisfied. For each of these cases and many of which are not contemplated here, a specific operator is proposed, e.g., t-norm, t-conorm and ordered weighted averaging (OWA) operators (Yager, 1988). In our problem, we desire that both evaluating criteria be satisfied; so, a t-norm operator can be an appropriate choice. In this work, we employ the minimum possibility degree of the two constraint networks as the overall degree. One might also wish to keep the two possibility degrees separate, to be able to query on the degree of satisfiability for each dimension.

Computing Interval Bounds of Inferred Value

The propagation of qualitative Allen relations does not provide any knowledge about the time bounds of the intervals. I.e., we know the temporal and value possibility of an inferred activity occurring, but we do not know when it could occur, except for a relative qualitative placement in time of sensor readings. In fact, we know when sensor readings occur in time, but the lack of knowledge about the occurrence interval of inferred values makes it impossible to build a timeline. Since a timeline corresponding to a state variable is a sequence of values in time, the need of quantitative temporal reasoning arises; thus, we convert the obtained optimal temporal network containing a solution with maximum possibility degree to a STP. As noted earlier, the optimal solution contains a temporally consistent network with the highest possibility degree relative to the initial network, thus, the network which is converted to STP is consistent. Temporal propagation in the STP is done through the Floyd-Warshall algorithm, whose computational cost is O(n^3). Therefore, computing the temporal bounds for the inferred value can be done in polynomial time.

As a case in point, the network shown in Figure 4(c) is converted to a STP. This network is consistent as a result of performing the fuzzy Allen algorithm on the inconsistent network shown in Figure 4(b). Since the *Stove* and *Location* state variables represent sensor readings, we know the precise time in which these readings occurred. For instance, if the user was in the kitchen in the interval $[1, 12]$ and the stove was on in the interval $[5, 15]$, then the occurrence of Cooking would be $[0, 16]$. The cooking interval is calculated by choosing the earliest time of the timepoints in the STP which represent the beginning and ending of the Cooking activity.

Dealing with Multiple Timelines

The abductive reasoning process, by hypothesizing the head of rules, outputs hypotheses which can have different possibility degrees. The number of hypotheses is the number of combinations of state variable values in the SCN which can be unified with the value requirements of the applied rule. For instance, in the case that Cooking is hypothesized which is shown in the Figure 4, there are six possible combinations of state variables *Stove* and *Location*. Each combination entails an inferred hypothesis with a particular possibility degree. Moreover, there can be dependencies among the rules, therefore, each inferred value can be a requirement for the others. For instance, the model may state that Eating occurs after Cooking (among other requirements), thus, inferring the occurrence of Eating depends on a Cooking hypothesis which has been already inferred. Consequently, each Cooking hypothesis enlarges the number of combinations of values which is needed for the process of inferring Eating. With respect to the growing rate of inferences, we aim at recognizing a most likely timeline. The possibility of a timeline is expressed as the minimum of the possibility degree assigned to the values of state variables in the model.

We cast the problem of extracting the best timeline (i.e., the timeline with maximum possibility degree) as a Constraint Optimization Problem, COP (Dechter, 2003), in which variables are hypotheses referring to the possible values of a state variable; values of this constraint network are different ways to "support" a value of a state variable which itself is a constraint network (combinations of state variable values in the SCN). Constraints are the value constraints and the Allen temporal relations among values of state variables prescribed in the rules. Assigning a support to the variable has a cost which is calculated through the fuzzy inference process. An optimal solution to this problem, is a set of supports assigned to the variables and has the highest possibility degree with regard to the constraints. The collection of intervals associated to the hypotheses belonging to the optimal solution is the most likely timeline of a state variable.

Example

To extend our Cooking scenario, the human user leaves the kitchen and goes to the dining room in order to eat his lunch. The inference system aims at recognizing both Cooking and Eating activities. The value and temporal requirements of Eating are modeled in the rule shown below

(*Human* = Eating) *During* (*Location* = Dining table) $\wedge$
(*Human* = Eating) *After* (*Human* = Cooking)

An example of sensing process is depicted in the Figure 5 which is compatible with the above scenario. In this Figure, in addition to the timelines of state variables *Location* and *Stove*, we show the most likely timeline for state variable *Human* resulting from the overall process described in this paper.

The best two timelines of Human activity in terms of the possibility degrees are shown in Figure 6. The possibility degree associated to each value is the result of the fuzzy inference process which is introduced in this paper.

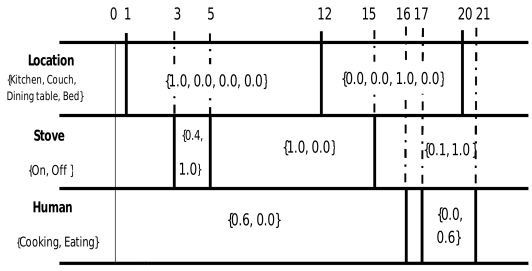


Figure 5: State Varibale Timlines

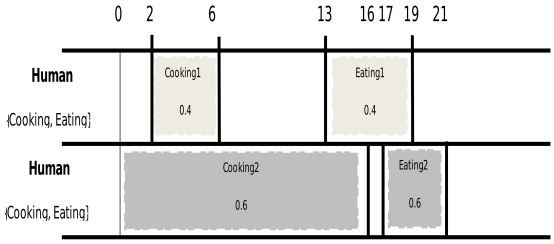


Figure 6: Human State Variable Timelines

Conclusion and Future Work

We have presented a fuzzy inference process for interpreting a crisp constraint-based model so as to accommodate uncertainty in sensor data and imprecision in the model. This fuzzy inference process covers two aspects of the model, namely value and temporal requirements. The problem of accommodating uncertainty into each aspect is solved using the notion of soft unary constraints in a fuzzy value constraint network and conceptual neighborhoods in a fuzzy temporal constraint network. We leverage state of the art algorithms for fuzzy qualitative temporal reasoning, and introduce quantified temporal bounds for the purpose of extracting timelines as a polynomial time post-processing step.

Our approach has been implemented in a system, which has been used on simple artificial scenarios like the ones shown in this paper. Our current work focuses on the evaluation of the system along two axes: first, time performance in larger domains; second, behavior in a real environment with sensors deployed in a smart home (Saffiotti et al., 2008).

While we have focused on qualitative temporal relations in this paper, it might also be interesting to investigate the use of quantitative temporal relations with uncertainty (Vidal and Fargier, 1999) and preferences (Khatib et al., 2001). Furthermore, it is essential to combine multiple hypotheses into candidate timelines. As described in this paper, we cast the problem of extracting a unique timeline as a COP, hence, different approaches for solving a COP which can be employed in this work should be studied.

Acknowledgements

This work was partly supported by the European Community's FP7 under contract # 287752 RACE "Robustness by Autonomous Competence Enhancement".

References

Allen, J. 1984. Towards a general theory of action and time. *Artificial Intelligence* 23(2):123 – 154.

Badaloni, S., and Giacomin, M. 2000. A fuzzy extension of allens interval algebra. In *AI*IA 99: Advances in Artificial Intelligence*, volume 1792. Springer Berlin , Heidelberg. 155–165.

Dechter, R.; Meiri, I.; and Pearl, J. 1991. Temporal constraint networks. *Artificial Intelligence* 49(13):61 – 95.

Dechter, R. 2003. *Constraint processing*. The Morgan Kaufmann Series in Artificial Intelligence. Morgan Kaufmann Publishers.

Doherty, P.; Kvarnström, J.; and Heintz, F. 2010. A temproal logic-based planning and execution monitoring framework for unmanned aircraft systems. *Journal of Automated Agents and Multi-Agent Systems* 2(2).

Dousson, C., and Maigat, P. L. 2007. Chronicle recognition improvement using temporal focusing and hierarchization. In *IJCAI*, 324–329.

Dubois, D.; Fargier, H.; and Prade, H. 1996. Possibility theory in constraint satisfaction problems: Handling priority, preference and uncertainty. *Applied Intelligence* 6:287–309.

Floyd, R. 1962. Algorithm 97: Shortest path. *Commun. ACM* 5:345.

Freksa, C. 1992. Temporal reasoning based on semi-intervals. *Artif. Intell.* 54(1-2):199–227.

Guesgen, H. W. 2002. Fuzzifying spatial relations. In Matsakis, P., and Sztandera, L., eds., *Applying soft computing in defining spatial relations*. Heidelberg, Germany, Germany: Physica-Verlag GmbH. 1–16.

Khatib, L.; Morris, R.; Morris, R.; and Rossi, F. 2001. Temporal constraint reasoning with preferences. In *Proc of the Int Joint Conf on Artificial Intelligence (IJCAI)*, 322–327.

Klir, G., and Folger, T. 1988. *Fuzzy sets, uncertainty, and information*. Prentice Hall.

Mansouri, M. 2011. Constraint-Based Activity Recognition with Uncertainty. Master's thesis, Örebro University, School of Science and Technology.

Pecora, F., and Cirillo, M. 2009. A Constraint-Based Approach for Plan Management in Intelligent Environments. In *Proc. of the Scheduling and Planning Applications Workshop at ICAPS09*.

Pecora, F.; Cirillo, M.; Dell'Osa, F.; Ullberg, J.; and Saffiotti, A. 2012. A Constraint-Based Approach for Proactive, Context-Aware Human Support. *Journal of Ambient Intelligence and Smart Environments*. (accepted).

Ruttkay, Z. 1994. Fuzzy constraint satisfaction. In *Fuzzy Systems, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the Third IEEE Conference on*, 1263 –1268 vol.2.

Saffiotti, A.; Broxvall, M.; Gritti, M.; Leblanc, K.; Lundh, R.; and Rashid, J. 2008. The PEIS-Ecology project: Vision and results. In *In Proc of the IEEE Int Conf on Intelligent Robots and Systems (IROS)*.

Vidal, T., and Fargier, H. 1999. Handling contingency in temporal constraint networks: from consistency to controllabilities. *Journal of Experimental and Theoretical Artificial Intelligence* 11:23–45.

Yager, R. 1988. On ordered weighted averaging aggregation operators in multi criteria decision making. *IEEE Trans. Syst. Man Cybern.* 18:183–190.