The philosophy underlying the framework presented in this paper is inspired by classical control theory, in that the planning and scheduling problem is modeled by identifying a set of relevant components whose temporal evolutions need to be controlled to obtain a desired behavior. Components are primitive entities for knowledge modeling, and represent logical or physical subsystems whose properties may vary in time. An intrinsic property of components is that they evolve
over time, and that decisions can be taken on components which alter their evolution. The approach shortly summarized here is the approach followed by the OMPS planner and scheduler, that has been presented in previous works (e.g., see (Fratini, Pecora, & Cesta 2008) among others).

**Components and their behaviors.** During any interval of time a component can have one or more possible behaviors. Each behavior describes a different way in which the component’s properties vary in time during the temporal interval of interest. However, not every function over a given temporal interval can be taken as a valid behavior for a component. The evolution of components in time is subject to “physical” constraints. We call consistent behaviors the ones that actually correspond to a possible evolution in time according to the real-world characteristics of the entity we are modeling, i.e., those behaviors which adhere to the “physical” constraints. For instance, if the component is a state variable, its behaviors are sequence of symbolic values. A set of rules that specify which transitions between allowed values are legal, and a set of lower and upper bounds on the duration of each allowed value are used to state which behaviors are legal (these constraints are often synthesized in a finite state machine). In the case of a component that model a resource, behaviors are numerical functions of time (to trace the resource availability at each time instant) and a lower and upper bound of resource availability state which behaviors are consistent (those that are always between the bounds) and which are not.

Now that we have introduced the concept of component as the fundamental building block of the component-based approach, the next step is to define how component behaviors can be altered (within the physical constraints imposed by the consistency notion). A decision is a temporally tagged event. When applied to a component, a decision alters the component behaviors. In the case of state variables, a decision might be a choice of a value in a given temporal interval, while in a resource component a decision might be a resource consumption in a given interval (an activity) or a production/consumption (that positively or negatively alters the resource availability after a given time point). What is important to point out here, is that nothing is stated a-priori in general component definition about how actually a decision alter the component behavior. The idea is that, given a set of components, a set of decisions determine their behaviors, and each component must provide the implementation for computing its own behaviors once that a set of decisions have been stated. Moreover, since the notion of consistency is component-dependent as well, a component must be able to compute, given a set of decision, its own consistent behavior and must provide the implementation for adjusting (whenever possible) the decisions to avoid inconsistencies.

**Domain theory.** After defining components in isolation we select some of them to describe more complex target domains. When components are put together 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 others behavior. The domain theory specifies such inter-component relations. As the notion of consistency describes which behaviors are consistent for each single component with their own “internal physical model”, the domain theory defines which combinations of behaviors of all components are acceptable with respect to decisions taken on other components. We represent such requirements by means of synchronizations. A synchronization specifies the “consequences” of decisions. A synchronization could for instance specify that a given amount of resource must be booked when a certain value is taken by a given state variable, or could specify a value and/or temporal relationship among the sequence of values taken by two different state variables. Once again it is important to notice that the notion of synchronization is general, and states a temporal and/or value relations among two or more decisions. In case of synchronization among state variables, the notion collapses into the compatibility concept widely used in timeline-based planning. More in general the definitions of component, decision and synchronization in the component-based approach generalize the definitions of state variable, token and compatibility used in both HSTS or Europa planners.

**A comment.** The purpose of the pursued generalization is very similar to the principle behind the object-oriented approach that generalizes the notion of primitive data types. As an object abstracts a primitive type by providing a data representation and a set of methods for managing data, the component abstracts the notion timeline by providing both a representation of the behaviors and a consistency notion, and providing a set of methods for computing the effects of decisions (and methods for managing them). As an object-oriented application is built by defining objects and making them to interact, similarly a component-based domain is built by defining components and specifying rules for putting them together and making them able to interact (the domain theory). This approach carries all the advantages typical of the object-oriented approach: modularity, reusability of components, possibility of building a library of planning and scheduling components reusable in different domains and so on. The TRF provides the enabling technology for building component-based planning and scheduling applications.

**Timeline-based Representation Framework**

The TRF software architecture consists of layers organized in a hierarchy. Each layer is responsible for dealing with a particular aspect of the problem, and each layer uses the services provided by the underlying layers to implement its functionalities. The constraint-based nature of the approach is extremely visible in the way the different layers exchange information: constraints are posted on the underlying levels as a consequence of decisions taken on higher levels, and decisions are taken on higher levels by analyzing the domains of the variables in the underlying levels. The architecture has been conceived to be easily extensible by adding components. This capability is very important to achieve
a good balance between general, domain independent planning (easily customizable to various domains) and specialized, efficient reasoning (often needed in real world domains for efficiency reasons).

TRF’s architecture is subdivided in three levels: a Time/Parameters layer, a Component layer and a Domain layer. Layers are organized according to the hierarchy shown in Figure 1. The TRF design reflects the approach induced by the component based approach. The planning domain is modeled as a set of concurrent threads (the timelines) and the problem is to synthesize a set of decisions to obtain a desired behavior and to synchronize the threads. Hence the TRF structure, where a common lower level represents the information shared among the timelines, temporal information and parameter information, a middle level that represents the extension point where the modeler plugs the components, and an upper level that provides a unified, shared representation of the plan.

**Time and Parameters Layer.** This is the lowest layer in the TRF’s architecture. Temporal and parameter information is managed at this level. The interface provided by this level is simple and straightforward. Higher levels create temporal elements and parameters, impose constraints on them and query the database to access the information on events temporal positions and parameters values. The temporal information is managed in shape of Temporal Constraint Networks (TCNs) (Dechter, Meiri, & Pearl 1991). TCNs allow representing events, also called time points, and temporal constraints that represent distances, separation constraints, etc. This layer is endowed with propagation algorithms to maintain the consistency of the possible value assignments to time points. The current implementation in is based on the Simple Temporal Problem (Dechter, Meiri, & Pearl 1991).

Two propagation algorithms are defined: the first one implements an All Pair Shortest Path algorithm (less efficient but provides information about time points distances), the second one implements a Single Source Shortest Path algorithm (more efficient, but only time points lower and upper bound are provided). Parameters are managed through an external CSP solver, CHOCO (CHOCO 2008).

**Component Layer.** The component layer is the point of expansion of the TRF architecture. In this architecture a component is a software module that encapsulates the logic for (1) computing a timeline resulting from decisions; (2) evaluating the consistency of the computed timeline with respect to a set of given rules and (3) computing a set of temporal and/or parameter constraints and further decisions to solve (if possible) any threat to the consistency of the computed timeline. A couple of practical examples might help in understanding the three points. Referring again to the state variable and to the reusable resource component previously mentioned, a state variable component encapsulates the logic for computing, given a set of value choices, the resulting timeline. Temporally intersecting decisions must require the same values, otherwise the resulting timeline will be inconsistent. If two decisions that require $P(x)$ and $P(y)$ happen to overlap, the state variable component must be able for instance to deduce $x = y$ to ensure the consistency. In the case of a reusable resource component, it encapsulates the logic for computing resource profiles given a set of allocated activities. An inconsistency is detected when $n$ overlapping activities requires a total amount of resource greater than the maximum availability. In this case the resource component must be able to analyze the situation and to compute a set of temporal constraints between them that will solve the conflict. It is worth pointing that search is not supposed to be performed at this level: only necessary constraints deduced by propagation should be automatically posted within the components. Point of choice should be forwarded back to higher levels in the hierarchy of the architecture.

A component provides to higher levels basic timeline-management primitives (like timeline extraction and inconsistencies detection). It is a point of expansion because the components make the architecture independent from the actual implementation of the functionalities they provide, encapsulating component-specific algorithms and hiding differences about behaviors, inconsistency detection and reso-

---

**Figure 1:** The layered implementation of the TRF.
In the current implementation the TRF provides two types of standard components: state variables and reusable resources. State variables have behaviors that are piecewise constant functions over a finite, discrete set of symbols which represent the values that can be taken by the state variable. Each behavior represents a different sequence of values taken by the component. The consistency notion is stated as a sequence of constraints, i.e., a set of rules that specify which transitions between allowed values are legal, represented as a timed automaton (Allur & Dill 1994). Inconsistency detection and resolution are implemented as in (Fratini, Cesta, & Oddi 2005). Resources behaviors are real functions over time. Each behavior represents a different profile of resource consumption. Propagation algorithms from (Laborie 2003; Baptiste & Le Pape 1995) are implemented and used for consistency detection and timeline extraction.

**Domain Layer.** The Domain layer presents to the TRF’s users primitives to represent both domain theory and plans, and in general allows to use components. A plan is represented as a decision network: given a set of components $C$, a decision network is a graph $(V, E)$, where each vertex $\delta C \in V$ is a component decisions defined on a component $C \in C$, and each edge $(\delta C_1, \delta C_2) \in E$ is a relation among component decisions $\delta^{m}_{C_1}$ and $\delta^{n}_{C_2}$.

![Figure 2: Decision Network.](image)

In Fig. 2 is represented a decision network for a domain with two state variable components, $C_1$ and $C_2$, with two decisions for each component. A decision specifies a disjunction of required values for the corresponding state variable timeline (for instance $\delta^1_{C_1}$ requires either a value $A(x_1)$ or a value $B(x_2)$ for the state variable $C_1$). The fundamental tool for defining dependencies among component decisions are relations, of which the TRF provides three types: temporal, value and parameter relations. Given two component decisions, a temporal relation is a constraint among the temporal elements of the two decisions (continuous arrows in Fig. 2, where for example the DEADLINE constraint for $\delta^2_{C_1}$, and $\delta^2_{C_2}$ or the BEFORE constraint between $\delta^1_{C_1}$ and $\delta^2_{C_2}$ are temporal relations). A temporal relation among two decisions $A$ and $B$ can prescribe temporal requirements such as those modeled by Allen’s interval algebra (Allen 1983), e.g., $A \text{EQUA}L B$, or the hybrid variant qualitative/quantitative, e.g., $A \text{OVERLAPS} [l,u] B$. A value relation between two component decisions is a constraint among the values of the two decisions. A value relation among two decisions $A$ and $B$ (the dashed line in Fig. 2 between $\delta^1_{C_1}$ and $\delta^2_{C_2}$ for instance) can prescribe requirements such as $A \text{EQUALS} B$, or $A \text{DIFFERENT} B$ (meaning that the value of decision $A$ must be equal to or different from the value of decision $B$). Lastly, a parameter relation among component decisions is a constraint among the values of the parameters of the two decisions (the continuous line between $\delta^1_{C_2}$ and $\delta^2_{C_2}$ that states $x_2 = 2x_1$ in Fig. 2 for example). Such relations can prescribe linear inequalities between parameter variables. For instance, a parameter constraint between two decisions with values “available(‘antenna’, ‘?bandwidth’)” and “transmit(‘?bitrate’)” can be used to express the requirement that transmission should not use more than half the available bandwidth, i.e., $\text{?bitrate} \leq 0.5 \cdot \text{?bandwidth}$.

The domain level is where concurrent threads represented by each component in the underlying level are put together to constitute the component-based domain. The decision network provides a unified vision of the current solution, while the synchronizations that constitute the domain theory provide a unified means for expressing the constraints that the decisions must satisfy. In the current TRF implementation an extension of the DDL_3 language (Fratini, Pecora, & Cesta 2008) is used for specifying the domain theory.

It is worth pointing that the decision network and the domain theory are flexible enough for representing a wide range of different problems. It is possible to model a timeline-based planning problem and represent a plan. In fact, a timeline-based domain independent planner as OMPs can be easily refactored as a solver on top of the TRF with state variables. The solver implements search procedure and heuristics, while the TRF maintain the planner search space, also providing powerful functionalities for helping in building such a search space. But also a pure scheduling problem (and its solution) can be represented with a decision network. In fact the next section presents some use cases of the TRF, presenting also a scheduler for RCPSP/max problems built on top of the framework.

**Case Studies**

In this section we survey three different cases in which we have used the TRF to develop specific solvers. The first is a deployed space application, named MrSPOCK, recently developed within the APSI project of the European Space Agency, the other two are more research oriented in nature: STRIPLINE is a simple planner for STRIPS domain recently synthesized as a step of an internal research activity and a family of constraint-based schedulers developed to import in the same framework previous research results.

**MrSPOCK**

MrSPOCK, the “MARS EXPRESS Science Plan Opportunities Coordination Kit”, is a new tool which required a combination of various research aspects from the planning and scheduling area. The system solves an interesting multi-objective optimization problem that requires the satisfaction of a number of temporal and causal constraints to produce long term plans for the MARS EXPRESS spacecraft activities. An interesting aspect is the hybrid combination of a
constraint-based representation that supports timeline-based planning and scheduling, an optimization algorithm that exploits such representation and an interaction front end which has multiple features. The system has been first deployed to end users during May 2008 and it is currently being refined to perfectly match the details of the daily use. Apart from the fielded application it is worth highlighting here the interesting leverage we obtained with respect to our previous experience in ESA projects, e.g., (Cesta et al. 2007), due to the use of the TRF. This general framework has allowed us to capture a number of constraints with a basic domain description language. Additionally the use of the timeline-based representation as a central concept for the user interaction front end demonstrates its particular suitability to capture the way of working of human planners in space domains.

The problem. The open problem we addressed at ESA with the MrSPOCK application was to support the collaborative problem solving process between the science team and the operation team of the space mission. These two groups of human planners iteratively refine a plan containing all activities for the mission. The process starts at the long term plan (LTP) level – three months of planning horizon – and is gradually refined every two days to produce final executable plans. Goal of MrSPOCK is to develop a pre-planning optimization tool for spacecraft operations planning and, specifically, the generation of a pre-optimized skeleton LTP which will then be subject to cooperative science team and operation team refinement (see (Cesta et al. 2008) for a more detailed description of the whole work).

A critical point in developing an application to produce the MARS EXPRESS skeleton LTP is the need to consider a great number of operational constraints that cannot be easily removed after four years of daily mission operation practice. In order to capture the work practice we had to cope with very specific constraints that are difficult for the general purpose solving framework but more easily be taken into account in a domain specific solver, hence the choice of creating such solver on top of the TRF. In general it is worth underscoring that in developing application of planning and scheduling in real context the trade-off generality/specificity is a relevant one even if it is usually not mentioned in official literature. In our previous experience to produce the ME@AR tool (Cesta et al. 2007) we have used a model-based representation centered on timelines and several principles of mixed-initiative planning that are research products of our area but the whole implementation was done on purpose for the application. In MrSPOCK the amount of the general purpose modules used in the implemented system is quite high with respect to our previous work. It is also worth mentioning that the development of a solver entirely based on domain independent solver would require the customization of an amount of specific knowledge in the domain description with a consequent production of a rather cumbersome domain model. Our choice has been to use TRF for clean modeling purposes while relying on a specific module for driving an efficient problem solving.

Use of the TRF. MrSPOCK uses the TRF domain modeling capabilities to capture the main entities of the Long Term Plan domain within the MARS EXPRESS mission. In order to describe the components we used to model the problem it is important to remind an important distinction between: (1) Controllable Components, whose temporal behavior is decided by the solver. They define the search space for the problem, and their timelines ultimately represent the problem solution; (2) Uncontrollable Components the evolution of which is give to the solver as input. They represent values imposed over time which can only be observed; they are seen as additional/external data and constraints for the problem.

Figure 3 shows how the MARS EXPRESS LTP domain is captured in the current release of MrSPOCK. In particular in this case we only use the component type “state variable”.

A single controllable state variable models the spacecraft’s pointing mode (Pointing), which specifies the temporal occurrence of Science and Maintenance operations as well as the spacecraft’s Communication to Earth. The values that can be taken by this state variable, their durations (represented as a pair \([\text{min}, \text{max}]\)) and the allowed transitions among the possible states are synthesized by the automaton shown in the right side of Figure 3.

As uncontrollable variables we represent ground stations (GS) availability and the occurrence of the key orbit events (Apocentre and Pericentre). The temporal occurrences of pericentres and apocentres are shown in Figure 3 (“Apo” and “Peri” values on the timeline, left/top part of the picture) and are defined in time according to an orbit event file decided by the flight dynamics team. The other state variable maintains the visibility information of three ground stations (“MAD”, “CEB” and “NNO” timelines left/bottom part of the figure). The allowed values of these state variables are: \(\{\text{Available(?rate,?ul\_dl,?antennas)},\ \text{Unavailable()}\}\), where the \(?rates\) parameter indicates the bitrate at which communication can occur, \(?ul\_dl\) indicates whether the station is available for upload, download or both, and the \(?antennas\) parameter indicates which dish is available for transmission.

Any valid plan needs temporal synchronizations among the pointing timeline and the uncontrollable variables. These synchronization constraints are represented as dotted arrows in the figure: Science operations must occur during Apocentres, Maintenance operations must occur during Apocentres.
and Communication must occur during ground station visibility windows. As mentioned, in addition to those synchronization constraints, the Pointing timeline should satisfy the transition constraints among values specified by the automaton and the minimal and maximal duration specified for each value (in the automaton as well). A solution is obtained when a set of consistent timelines for the controllable component are defined and all the operational constraints are satisfied.

**Solver.** A distinctive aspect of MrSPOCK is in the synthesis of a problem solver directly connected to the timeline representation. In this way we exploit the TRF constraint engines for propagating several types of constraints, while using specialized search engines partly general partly tailored to the problem. In particular, MrSPOCK integrates a greedy one pass constructive search procedure with a generic optimization cycle that uses a genetic algorithm approach as discussed in (Cesta et al. 2008). One of the interesting achievements in this work is the hybridization of a timeline based general purpose approach with a wrapping module that implements a genetic optimization search. It is worth underscoring again how the TRF is endowed with propagation algorithms hence it is not just a bookkeeping data structure rather it has an active role as is current practice of constraint satisfaction engines. In creating a complete architecture we situate MrSPOCK at an intermediate stage between generic timeline-based planners, e.g., our own OMPS architecture, and the domain specific timeline-based solver used in MEXAR2.

**STRIPLINE**

STRIPTLE is a planner for simple STRIPS domains recently implemented on top of the TRF. It pursues the key idea of translating the STRIPS problem instances in terms of timelines and temporal constraints. Considering a STRIPS instance \( P_{ST} = (P,O,I,G) \) where \( P \) is a set of ground literals, \( O \) is a set of actions, \( I \subseteq P \) is the set of literals true in the initial state and \( G = (M,N) \) describes the final state, where \( M \subseteq P \) is the set of literals that must be true in the final state, \( N \subseteq P \) is the set of literals that must be false and \( M \cap N = \emptyset \). An action \( a \in O \) is a tuple \( \langle \alpha, \beta, \delta, \gamma \rangle \), where these four sets specify, in order, which literals must be true for the action to be executable, which ones must be false, which ones are made true by the action and which ones are made false. An instance of a STRIPS problem \( P_{ST} \) can be translated into a timeline planning problem \( P_{TL} = (C,S,D) \) where \( C \) is a set of components, \( S \) is a set of synchronizations and \( D \) is a set of decisions that describe the initial values of the timelines. This is described in the following.

**Domain model.** We need to represent with timelines both the actions that constitute a plan and the state of the world resulting from the application of that sequence of actions. Since a STRIPS plan is simply an atemporal sequence of actions, while in the TRF we represent only temporally placed events, we discretize the TRF’s underlying temporal representation by focusing on \( H \) events temporally placed over an interval \([0,H]\) at \( t_0 = 0, t_1 = 1 \) and so on until \( t_{H-1} = H-1 \). We suppose the application of a STRIPS operator \( a_i \) at each event \( t_i, i > 0 \). Under this assumption, we introduce an *impulsive state variable* component, the *agent*, with values in \( O \) to represent with a timeline a STRIPS plan. The impulsive state variable, similar to a concept proposed in (Verfaillie, Pralet, & Lemaitre 2008), is a component similar to a state variable, whose decisions modify the component behavior in a given time point, instead of in a given time interval as in the case of state variables. As a consequence, the timeline for this component takes values in some time points (when decisions occur) while are undefined elsewhere. STRIPLINE plans the behavior of this timeline by choosing which action \( a_i \) is applied at each time point \( t_i \). The STRIPS state of the world is represented through ”classical” state variables, by defining for each literal in \( P \) a state variable with values in \{true,false\}. At each time instant \( t_i \), the values taken by these timelines in the interval \([t_i,t_{i+1}]\) model the state of the world of the STRIPS domain after the application of the action \( a_i \) at \( t_i \) (modeled in the agent timeline). On the other side, the values taken in the interval \([t_{i-1},t_i]\) model the state of the world where the action \( a_i \) is applied.

**Domain Theory Translation.** The STRIPS domain theory synthesizes, with the action schema, under what conditions an action can be applied and what the effects of the application are on the state of the world. In the translated domain these rules are both modeled as temporal synchronizations among the values taken by the agent timeline and the values taken by the timelines that model the state of the world. The required action applied in a time instant \( t_i \) is translated into a value requirement for the proper timelines that model the required literals over an interval ending at \( t_i \), while an effect of an action is translated into a value requirement over an interval that starts at \( t_i \). Let us consider for instance a simple Blocks World Domain with 3 blocks \( B_1,B_2 \) and \( B_3 \). An hypothetical action \( a \) to move \( B_1 \) from \( B_2 \) to \( B_3 \) would require as a precondition that \( P1 \) \( B_1 \) is on \( B_2 \), \( P2 \) \( B_1 \) is clear and \( P3 \) \( B_3 \) is clear. As effect we will have that \( E1 \) \( B_1 \) is no longer on \( B_2 \), \( E2 \) \( B_2 \) is on \( B_3 \), \( E3 \) \( B_2 \) becomes clear and \( E4 \) \( B_3 \) is not clear any longer. This action schema will be translated, supposing the timelines \((A)\) for the agent and \((T1)\) OnB1B2, \((T2)\) OnB1B3, \((T3)\) CLEARB1, \((T4)\) CLEARB2 and \((T5)\) CLEARB3 for the literals involved in the mentioned action schema, into the following synchronizations: \((S1)\) \((A) = a \text{ at-end (T1)} = \text{true and (A)} = a \text{ at-start (T1)} = \text{false to model (P1)}\) and \((E1), (S2)\) \((A) = a \text{ at-start (T2)} = \text{true to model (E2)}\), \((S3)\) \((A) = a \text{ during (T3)} = \text{true to model (P2)}\), \((S4)\) \((A) = a \text{ at-start (T4)} = \text{true to model (E3)}\) and \((S5)\) \((A) = a \text{ at-end (T5)} = \text{true and (A)} = a \text{ at-start (T5)} = \text{false to model (P3)}\) and \((E4)\).

**Timeline initialization.** For each literal in \( I \) a value **true** is put starting at \( t_0 \) in the corresponding timeline. For each literal in \( P-I \) a value **false** is put starting at \( t_0 \) in the corresponding timeline. For each literal in \( M \) a value **true** is put ending at \( t_H \) in the corresponding timeline. For each literal in \( N \) a value **false** is put ending at \( t_H \) in the corresponding
Planning algorithm. The translation presented is quite straightforward in the TRF. We had only to implement the impulsive state variable component deriving it from the state variable component. The planning algorithm is quite simple and straightforward as well. It incrementally builds a plan from \( t_1 \) choosing applicable actions by querying the TRF about the current values of the timelines and updating the timelines once that an action has been chosen (again exploiting TRF’s functionalities):

1. Check all executable actions at \( t = t_n \). This is performed by checking all those values of the agent timeline with synchronizations that require, for the timelines that describe the state of the world, values in the interval \([t_{n-1}, t_n)\) compatible with the current partial solution;
2. Execute one of them and check if the problem is solved. A plan has been found if all the values required at \( t_n \) have been achieved at \( t_{n+1} \). If this is not the case, extend the plan increasing \( t \). The frame axiom induce some additional temporal constraints on the decisions for the timelines that represent literals not affected by the executed action. In fact the effects of these decisions have to be extended since they have not been modified;
3. If no further actions are available and the problem is not solved, then backtrack.

STRIPLINE represents a preliminary work that still has not produced a complete planner, even for simple STRIPS problems. It is rather a first intermediate output of an ongoing work that aims at studying the possibility of planning temporal PDDL domains within a timeline based approach. The current STRIPLINE is not efficient and allows to perform only linear planning. Nevertheless is an example of how the TRF can supports everyday research activity enabling us to pursue new research directions.

Schedulers

One of the general goals in designing the TRF is to obtain a unified software framework that supports both planning and scheduling. Actually the idea of component-based architecture was initially inspired by the need to identify an abstract architectural concept able to capture the commonalities between early timeline-based planning (Muscettola 1994) and constraint-based scheduling a-la (Cesta, Oddi, & Smith 1998). Having resources as primitive components facilitates us in implementing a profile-based scheduler like the one based on conflict analysis of the earliest start time resource profile projection (for a detailed analysis see the ESTA algorithm in (Cesta, Oddi, & Smith 1998)). In TRF terminology this implies including a component for each resource in the problem and then inserting in an initial decision network the representation of the problem in form of the project graph with activities, temporal precedences among them and resource requirements for each of them.

Such a problem representation is then manipulated by a profile analysis procedure that return a set of conflict violations on the resource contention peaks (e.g., through sampling of minimal forbidden sets as in (Cesta, Oddi, & Smith 2002)) that are incrementally removed by posting precedence constraints. To sum up the TRF can be easily enriched by a version of the greedy ESTA algorithm that can be used to populate the algorithms family on top of the TRF. It is worth observing that the ESTA-approach is also generically used to detect conflicts among heterogenous components within the OMPs planner (Fratini, Pecora, & Cesta 2008). The possibility of dealing with actual scheduling decisions (like leveling a resource overcommitment) is in fact a key aspect in the planning and scheduling capability of OMPs.

Once defined the basic precedence-constraint posting scheduler we have further proceed and defined a meta-heuristic procedure that uses the one pass ESTA heuristics solver as the basic step. For example we have realized a variant of the iterative sampling procedure ISES described in (Cesta, Oddi, & Smith 2001) and are able to solve rather challenging resource constrained problems with a good efficiency and with performance in line with state of the art solvers in the area.

The TRF has been thought out as a generic development environment. Other colleagues have developed schedulers for different goals: (a) some colleagues of ours (Oddi & Rasconi 2008) have created an implementation of the IFLAT procedure introduced in (Cesta, Oddi, & Smith 2001) and then developed variants of the algorithm for carrying out research on scheduling on top of the TRF. At present, they have obtained competitive results on a new benchmark of 200 activities for the resource constrained problem; (b) other colleagues (Verfaillie & Pralet 2008) have synthesized a new scheduler and built an effective metaheuristic that solve huge instances of a real problem in the space domain as a case study within the APSI project.

Conclusions

In this paper we have presented our current work on timeline based planning and scheduling from a software architecture perspective. We have described the ideas underlying a basic module, called the TRF, we have developed for supporting timeline based reasoning and shown how such a module can be used as a development environment for supporting both application oriented and research oriented work. The key concept we are trying to underscore concerns the relevance, in P&S research, of effective software development environments. The TRF described here is a Java environment based on the general concept of timeline. The reason we are trying to develop our own environment (instead of, for example, modifying the EUROPA distribution (EUROPA 2008) resides in our interest for a principled development of the architecture grounded on the layered ontology introduced in this paper. In particular the generalization introduced with the concept of component and the description of heterogeneous domain in the decision network are interesting results contributed here to the general debate on timeline based planning and scheduling.
Our emphasis in the recent work has been to demonstrate the possibility of rapid prototyping in different directions. This paper gives an overview of such capability. The steps that we are currently pursuing to extend the TRF concern three directions: (1) creating a more complete knowledge engineering environment developing tools for supporting end users; (2) closing the loop between planning and real-time execution by adding an exec layer to the timeline solver; (3) addressing the problem of coping with uncertainty adding capabilities of plan repair and plan adaptation by importing our previous work in scheduling (Policella et al. 2007; Rasconi, Cesta, & Policella 2008) and extending it to planning.

Acknowledgments. Authors are currently supported by European Space Agency (ESA) within the Advanced Planning and Scheduling Initiative (APSI). APSI partners are VEGA GmbH, ONERA, University of Milan and ISTC-CNR. We acknowledge the role played by our colleagues in our current effort: thanks to Federico Pecora for his work in OMPS, to Gabriella Cortellessa and Angelo Oddi for their invaluable contribution to MrSPOCK, to Riccardo Rasconi for schedulers development and to Riccardo De Benedictis for his contribution to STRIPLINE.

References


Cesta, A.; Cortellessa, G.; Fratini, S.; and Oddi, A. 2008. Looking for MrSPOCK: Issues in Deploying a Space Application. In SPARK-08, Scheduling and Planning Applications workshop at ICAPS.


