UNIVERSIDAD DE ALCALÁ
DEPARTAMENTO DE AUTOMÁTICA

TESIS DOCTORAL

REPRESENTACIÓN Y PLANIFICACIÓN DE TAREAS CON TIEMPO Y RECURSOS

REPRESENTING AND PLANNING TASKS WITH TIME AND RESOURCES

María Dolores Rodríguez Moreno

Directores
Dr. Daniel Borrajo Millán   Dr. Daniel Meziat Luna

DICIEMBRE DE 2003

A Rosa

Caminante, son tus huellas
el camino, y nada más;
caminante, no hay camino,
se hace camino al andar.

Al andar se hace el camino,
y al volver la vista atrás
se ve la senda que nunca
se ha de volver a pisar.

Caminante, no hay camino,
sino estelas en la mar.

(Antonio Machado - Poema Caminante)

Agradecimientos

Cuando uno empieza un camino nunca sabe qué largo va a ser y cuántas personas se cruzarán en él. En esta tesis muchas son las personas que directamente o indirectamente han hecho posible que llegara a su fin.

La mayor parte del trabajo se ha desarrollado en colaboración con el grupo de Inteligencia Artificial Scalab de la Universidad Carlos III de Madrid. Quiero agradecer en primer lugar, a mis directores de tesis Daniel Borrajo y Daniel Meziat por la ayuda y la confianza que han depositado en mi. Asimismo, existen personas de este grupo de investigación que me han animado cada vez que pasaba por allí y hacían más agradable mi visita: Ricardo y David siempre dispuestos a escucharme, responder a mis preguntas y hacerme sonreir, y Fernando por el soporte con SHAMASH.

Otra parte de mi trabajo la he realizado en BRITISH TELECOM Adastral Park y en el ISTC-CNR. En BT quiero agradecer dentro del grupo *Intelligent Business System* a Paul Kearney, Paul O'Brien, Jamie Stark y Simon Thomson; y fuera de éste a Simon Martin, por todo la ayuda que me ofrecieron.

Y en el ISTC-CNR, dar mi más profundo agradecimiento a Amedeo y Angelo por las largas conversaciones mantenidas, dándome ánimos, muy buenos consejos y sugerencias (si la tesis está escrita en LaTeX es gracias a vosotros). Pero tampoco puedo olvidarme de la cálida acogida de los miembros del grupo: a Riccardo y Simone por las charlas y debates que mantuvimos con aspectos relacionados con scheduling y con O-OSCAR, a Federico por su ayuda con el dominio Robocare, a Gabriella por sus clases de italiano y a Nicola por darme todo lo necesario para iniciarme en LaTeX.

También, al grupo de Ingenieros de HISPASAT por su ayuda y colaboración durante el desarrollo de la herramienta CONSAT, especialmente a Arseliano Vega, Pedro Luis Molinero y Jose Luis Novillo.

No quiero olvidarme tampoco de mis amigos y de mis compañeros del pasillo *Uno* del Dpto. de Automática por su apoyo.

Agradecer por último, las sugerencias que Marc Cavazza y Enrico Giunchiglia han aportado a esta tesis.

Y por supuesto a las personas que han vivido más de cerca este trabajo, mi familia. A Carmelo por aguantar sin reñistar que le *robara* su PC para mis experimentos y a Rosa, mi madre, a la que dedico este trabajo. Tanto tú como yo sabemos que no hubiera sido posible nada de esto si no hubieses estado ahí apoyándome, escuchándome y alentándome en todo momento: ¡Lo hemos conseguido!.

Resumen Ampliado

Cualquier técnica de resolución de problemas en Inteligencia Artificial (IA), como rama de la informática, debe tratar dos aspectos a la hora de plantear soluciones a problemas: la representación del conocimiento y las técnicas que se pueden aplicar. Entre las técnicas con más actualidad dentro de la IA está la planificación de tareas.

En esta tesis hemos abordado como **primer objetivo**, el problema de la representación del conocimiento para planificación en dos dominios reales. Uno de los obstáculos en aplicar técnicas de planificación a dominios reales es la dificultad en modelar dominios. Generalmente se necesita que las personas que han participado en su desarrollo deban llevar a cabo la fase de modelado ya que en muchos casos la representación depende bastante del conocimiento del funcionamiento interno de la herramienta.

Uno de los dominios pertenece a los sistemas de planificación de actividades de negocio, como son los sistemas de gestión de procesos conocidos como sistemas de *Workflow*. El otro pertenece al dominio de los satélites, en concreto se ha realizado la modelización de las operaciones nominales que desde tierra se deben llevar a cabo en el operador español de telecomunicaciones por satélite HISPASAT.

En los sistemas de gestión de procesos se han dedicado esfuerzos para la definición de lenguajes que permitan a usuarios no expertos, introducir facilmente el conocimiento de los procesos en las herramientas con las que trabajan. Se ha utilizado la herramienta de modelado de procesos SHAMASH y el sistema de workflow en BRITISH TELECOM llamado COSMOSS para introducir el conocimiento. Dicho conocimiento será traducido en términos de lógica de predicados (por ejemplo al lenguaje estándar de representación de dominios de planificación PDDL2.1). Después de esta conversión, se podrán validar y generar automaticamente modelos reales utilizando cualquier planificador que soporte PDDL2.1. Con la unión, los planificadores ganan un lenguaje fácil de utilizar por cualquier usuario y a los sistemas de *Workflow* les permiten generar y validar los modelos de los procesos que se deben realizar a diario en sus empresas.

Para la empresa de telecomunicaciones por satélite HISPASAT hemos desarrollado la herramienta CONSAT (CONtrol de SATélites) para ayudar a dar solución al problema de planificar las operaciones que desde tierra deben llevar a cabo el grupo de ingenieros. Cada año este grupo genera toda la documentación necesaria a mano y en papel. Existen dos tipos de documentos, uno que proporciona una visión de las operaciones que se deben realizar cada día del año, y otro que representa con más detalle las operaciones que se deben realizar cada semana.

Una vez generados los documentos, se revisan y se verifican. Debido al incremento del número de satélites (actualmente cuatro y en el futuro dos más), esta tarea necesita ser automatizada. CONSAT da solución a estos problemas gracias a los tres subsistemas de los que consta:

- **El subsistema de usuario:** se encarga de controlar el acceso de los usuarios así como la manipulación de todos los ficheros y datos de entrada necesarios para la planificación de todas las operaciones.

- **El subsistema razonador:** una vez que los datos han sido introducidos correctamente, un planificador se encargará de dar solución al problema.

- **El subsistema generador:** es responsable de mantener la coherencia entre las dos posibles representaciones que los ingenieros necesitan: la *anual* y la *semanal*. Cualquier modificación que se realice en una representación será actualizada automaticamente en la otra, evitando los fallos de incongruencias muy habituales al generarse a mano. Además, este subsistema genera la solución en el tipo de formato que utilizan en HISPASAT, permite comparar dos soluciones diferentes o generarlas en HTML.

Pero en la planificación de estos dominios se necesita trabajar con tiempo y recursos. Dentro del área de planificación se distinguen básicamente dos grandes campos: la planificación y el *scheduling*.

Por planificación se entiende la selección de una secuencia de actividades de tal forma que satisfagan una o varias metas y un conjunto de restricciones impuestas por el dominio. La mayoría de las investigaciones realizadas en planificación se han centrado en encontrar un conjunto ordenado de acciones que satisfagan una o más metas.

Por *scheduling* se entiende la asignación de recursos y tiempos de inicio de las actividades, obedeciendo a las restricciones temporales de las actividades y las limitaciones de capacidad de los recursos compartidos. *Scheduling* es también una tarea de optimización donde recursos limitados se disponen a lo largo del tiempo entre actividades que se pueden ejecutar en serie o en paralelo de acuerdo con el objetivo de, por ejemplo, minimizar el tiempo de ejecución de todas las actividades.

Como **segundo objetivo** en este trabajo, se han identificado los puntos débiles y fuertes de la planificación y *scheduling* y se ha propuesto una solución que integra ambos campos.

Se han presentado distintos modelos de integración: desde la utilización de un planificador para resolver los problemas de los dominios expuestos anteriormente, pasando por la integración en línea de un planificador y un *scheduler*, hasta el enfoque más integrado que entremezcla planificación y *scheduling*: IPSS (Integrated Planning and Scheduling System).

En IPSS el razonamiento se divide en dos niveles. El planificador se encarga de la selección de acciones (puede optimizar según una métrica de calidad diferente al tiempo o recurso) y el *scheduler* de la asignación del tiempo y los recursos. Durante el proceso de búsqueda, cada vez que el planificador decide aplicar un operador, consulta al *scheduler* para comprobar su consistencia temporal y de recursos. Si es inconsistente el planificador vuelve hacia atrás y genera otra secuencia de acciones.

IPSS consta de tres capas:

- **La capa** *Deordering*: tiene como misión convertir el plan incompleto totalmente ordenado en un plan incompleto parcialmente ordenado.
- **La capa** *Ground*-CSP: se encarga de crear una Red Temporal, añadiendo los operadores y las retricciones causales y temporales entre ellos.
- **La capa** *Meta*-CSP: razona sobre los conflictos de recursos que pueden ocurrir entre pares de actividades que consumen el mismo recurso.

Los resultados demuestran la eficacia de IPSS frente a los enfoques de integrar planificación y *scheduling* en línea y respecto a otros sistemas de planificación actuales cuando se puede realizar una separación entre los recursos y predicados lógicos y se puede imponer una duración máxima al plan solución.

Abstract

Any problem solving technique in Artificial Intelligence (AI), as an area in Computer Science, must face two aspects when providing computational solutions to problems: knowledge representation and the methods used to reason. One of the solving techniques that has gained recent importance inside AI is planning and scheduling.

In this Ph.D. thesis we study these two aspects for AI planning and scheduling. The first goal will be to face the problem of knowledge representation in two real domains. First we have dealt with workflow domains by means of the SHAMASH workflow modeling tool and the COSMOSS workflow system at BRITISH TELECOM. Second, we have used the satellites domain for HISPASAT, a Spanish multi-mission system in charge of satisfying national communication needs. For this domain, we will be in charge of providing a solution to the ground nominal operations for its satellites.

But these domains need the integration of techniques from planning and scheduling. Traditionally, there is a clear subdivision of techniques and roles that belong to planning and scheduling. Planning systems select and order sets of activities such that they achieve one or more goals and satisfy a set of domain constraints. For most part, planning research has focused on finding a feasible chain of actions that accomplish one or more goals. Scheduling systems are in charge of assigning resources and time for activities, obeying their temporal restrictions and the capacity limitations of shared resources. Scheduling is an optimisation task where limited resources are allocated over time among both parallel and sequential activities, such that deadlines and makespan are minimised.

The second goal will be to study the weak and strong points of both fields. Planning systems have a rich representation of the problem descriptions but they have limitations to reason about time and handle cumulative resources. Scheduling systems can perfectly handle temporal and resources reasoning, but there is a lack of expressive language to represent the problem.

From this perspective, by combining scheduling and planning systems synergistically these weaknesses can be solved. We present here different models to integrate planning and scheduling: from the stand alone approach of using a planner, to the pipeline integration of using a planner and a scheduler, towards the more integrated approach of interleaving planning and scheduling. This last approach will be compared against the other two models and some state of the art planners, showing its effectiveness.

Contents

List of Figures

List of Tables

Part I

Introduction

Chapter 1

Introduction

In this chapter we present an introduction and a frame of reference to the work developed in this Ph.D. We first describe the motivation of this work, then the objectives that we want to achieve and finally the structure of this dissertation.

1.1 Motivation

In the last decades the advances in computer science have been translated into tools and techniques to automate processes that until then where performed and supervised by humans. Without any doubt, Artificial Intelligence (AI) is one of the computer science areas with more expectation created in the last years. AI is a scientific discipline which tries to operationalise human intellectual and cognitive capabilities in order to make them available through information processing systems.

In order to do so, it must face two aspects: the knowledge representation and the reasoning. In this dissertation, we will focus on these two aspects in relation to one of the problem solving techniques that has gained some relevance now inside AI: planning and scheduling. Traditionally these both fields have evolved separately even if they have a lot in common. In industry in general, the term planning has been used to describe a wide variety of problems, including problems referred to by the AI community as scheduling. In AI, the term planning is used to describe the generation of an ordered set of activities such that they achieve one or more goals and satisfy a set of domain constraints. For most part, planning research has focused on finding a feasible ordered set of actions that accomplish one or more goals. Examples of planning domains can be real time robot planning, travel planning, planning for information gathering and Database Queries, satellite planning operations and financial planning, among others.

Scheduling is in charge of assigning resources and time for activities, obeying the temporal restrictions of activities and the capacity limitations of shared resources. Scheduling is an optimisation task where limited resources are allocated over time among both parallel and sequential activities such that deadlines and makespan are minimised. Examples of scheduling domains include classical job-shop, manufacturing and transportation scheduling.

To solve any of the planning or scheduling domain examples mentioned above, we need to represent all the information in order to find good solutions efficiently. In AI, the study of languages in which we say things about the world belong to the research area of Knowledge Representation (KR). But one of the problems that this area presents is that representations influence on problem resolution. When solving problems, different models/representations of the problem provide different quality solutions or even the intractability of achieving them. Real domains as logistics, robotics, satellites or workflow (these two last ones, used in this work) require of a rich representation to handle activities, time and resource constraints. To provide solutions to this problem, several languages have appeared in the AI community. The PDDL2.1 language [68] is becoming a standard in the planning field for representing domains and problems. Although PDDL2.1 and other predecessor planning languages allow to represent this type of real problems, in many cases some assumptions have to be made (i.e., considering that changes in the world are only produced by the planner or having a complete and accurate knowledge of the starting situation) and in some cases the problem must be reduced to a subset of the original problem.

Integrating planning and scheduling is currently a hot topic in the AI community. By combining scheduling and planning systems synergistically the weaknesses in both fields can be solved. The planning systems could deal with time and resources thanks to the scheduling search procedure that interleaves refinement solution (assigning values to variables) and constraint propagation that computes implications of the assignments to other variables and eliminates inconsistent values, and the planning system can supply to the scheduler the language and the precedence constraints for the activities. In order this integration to be possible better and richer models are needed, so this will be one of the goals we want to achieve in this work.

1.2 Objectives

The work presented in this dissertation is motivated on one side by the limitations that non expert-users in AI encounter when modeling domains and problems for using planning and scheduling tools and techniques, and on the other side the need to integrate planning and scheduling for solving real world problems.

The objectives of this PhD. thesis are:

1. Study from a knowledge representation perspective what is missing in previous modeling languages on planning and scheduling that would allow to apply those techniques to two real world domains: workflow and satellites.

2. Use of Business Process Reengineering languages to face the problem of the knowledge representation in AI planning and scheduling for workflow domains. These languages allow non-experts to enter knowledge on company processes and their internal organisation. Once the user has created a representation of the problem, an automatic translation into AI planning domain

languages can be done, having in mind the different elements that compose both systems.

- As a subobjective, we want to build a tool to help on the automation of the generation of process models for workflow domains by using planning techniques.

3. Modeling of a real satellite domain with time and resources using a planner. Abstract the way the problem is solved in order to generalise it to other domains that need an integrated approach of planning and scheduling and the representation of this information in several planning languages such as a specific planners input language (QPRODIGY in this case), in the PDDL2.1 syntax and formulated as a Constraint Satisfaction Problem (CSP).

   - As a subobjective, we want to build a tool to help on the generation of schedules for on ground control operations for satellites maintenance using planning and scheduling.

4. Analyse some approaches that integrate planning and scheduling and extend/ improve PDDL2.1 or any planner language in order to support this integration.

5. Study different approximations for integrating planning and scheduling such as:

   - Building an integrated system able to reason about actions, time and resource constraints.
   - Evaluation of different approaches to integrate planning and scheduling, varying from a stand-alone planning system able to cope with some temporal and resource information towards the more integrated approach of interleaving planning and scheduling.
   - Evaluation of the built integrated system against state of the art planners.

1.3 Structure and Contents

This subsection provides an outline of the chapters and appendices that compose this dissertation. We have subdivided it in four parts: Introduction, Modeling for Planning and Scheduling, Integrating Planning and Scheduling, and Conclusions and Future Work.

- **Chapter 1** describes the motivation, objectives and contents of this PhD. thesis.

- **Chapter 2** starts with a general overview of the two main problems in AI: the problem representation and the methods used to efficiently solve those problems. Then, it focuses on the areas of planning and scheduling and revises the state of the art languages and tools for knowledge engineering in these two fields. Next, it explains what planning and scheduling are, and it describes the techniques developed in both areas. Lastly, it enumerates some historic and modern approaches of the integration of planning and scheduling.

- **Chapter 3** introduces the two problem solvers used in this dissertation: the planner QPRODIGY and the scheduler O-OSCAR.
- **Chapter 4** describes the features of workflow domains, in particular two workflow tools: SHAMASH, a workflow modeling tool, and COSMOSS, the workflow system at BRITISH TELECOM. It describes the advantages of using AI planners for modeling processes in workflow systems, and it studies what AI planners can gain using these tools.
- **Chapter 5** explains some features of satellites domains through the nominal operations of the Spanish satellite company, HISPASAT. We also describe the features, architecture and functionality of CONSAT, the tool developed for the HISPASAT company in order to schedule the nominal ground operations.
- **Chapter 6** faces the problem of knowledge representation by abstracting the way the satellites domain in HISPASAT is solved in order to generalise it to other domains that need an integrated approach of planning and scheduling. The temporal and resources information, generalised by the Allen primitives and multicapacity resources required to deal with real domains, has been represented in PRODIGY and formulated as a Constraint Satisfaction Problem (CSP). In some cases we have also used the PDDL2.1 syntax.
- **Chapter 7** presents three approaches to integrate a planning system and a constraint-based scheduling system. It varies from a stand-alone planning system able to cope with some temporal-resource information towards the more integrated approach of interleaving planning and scheduling, the IPSS system. It describes the algorithm used by IPSS[1], its features and the advantages of this approach over the others.
- **Chapter 8** shows empirical results on the IPSS problem solving performance obtained in a domain of the IPC-2002, ZENOTRAVEL and in other two domains: the BT domain, a reduced version of installing a new telephone line at BRITISH TELECOM and the ROBOCARE domain, a multi-agent system which generates user services for human assistance. This domain intends to give a user service in a closed environment such as a health-care institution or a domestic environment. We also compare IPSS with other state of the art planners.
- **Chapter 9** presents conclusions and discusses some future research directions.
- *Appendices and Bibliography*
  - Appendix A provides a framework of reference with the terminology used in the SHAMASH tool and in the PRODIGY planner.
  - Appendix B summarises the O-OSCAR input file.
  - Appendix C shows the operations implemented in the HISPASAT domain by means of the PDL4.0 syntax (the language used by PRODIGY and QPRODIGY).

[1]Stands for: Integrated Planning and Scheduling System.

– Appendix D represents the different robocare domain versions in the PDDL2.1 syntax considering robots as other logical predicates or as resources, and augmenting gradually the difficulty of the problem.
– Appendix E represents the BRITISH TELECOM (BT) domain in the PDDL2.1 syntax considering workers as other logical predicates and actions with and without durations.
– Appendix F presents the PLATA application. It is in charge of the translation between PDDL2.1 and QPRODIGY syntax as well as helping the user in creating domains and problems in both languages through an easy-to-use interface.
– Appendix G presents a temporal and resource extension for the PDDL2.1 syntax.

Chapter 2

State of the Art

The goal of AI [127] is to study how to build machines that perform tasks normally performed by human beings. Basically we can say that intelligence needs knowledge. Therefore, to make a machine think we need on one hand to consider the problem of representing the knowledge and on the other hand the reasoning processes, that is, the methods to solve those problems. Then, the main topics in this area are how to represent the information and automatic methods for solving problems. Within the AI field, our work focuses on *AI planning and scheduling*.

In this chapter we describe the previous work related to this Ph.D. thesis. We analyse, from the Knowledge Representation point of view, different modeling languages in planning and scheduling and their features. Then, we revise the main techniques in both fields and the representative systems in each category. Finally, we present the systems that have faced the problem of integrating planning and scheduling.

2.1 Knowledge Engineering in Planning and Scheduling

Knowledge Engineering (KE) refers to many topics as Knowledge Representation, Knowledge Modeling or Knowledge Validation. In the KE PLANET Roadmap [110] it is defined as the process that deals with the acquisition, validation and maintenance of domain models, and the selection and optimisation of appropriate machinery to work on them. As a representative tool for the acquisition, validation and maintenance of domain models we can mention the GIPO tool [111, 112, 165]. But, in this chapter we will only focus on Knowledge Representation (KR).

Theories of action proposed within the KR community typically address compact representations such as frames, semantic networks or fuzzy representations. The aim is often to provide natural and concise methods for specifying system dynamics. Similar motivations but very different methods are used by researchers in uncertainty, where the use of influence diagrams and belief networks are the norm.

KR in AI Planning and Scheduling relates to the analysis of knowledge used in planning and scheduling: ontologies and schemas of plans and planning processes and their use in knowledge acquisition and engineering; reusable and modular planning/scheduling components; and knowledge acquisition tools for planning/scheduling.

In the following subsections we will describe some of the main representation formalisms in planning and scheduling. The main question when evaluating them is its representation capability. In planning, we can consider the *Operators Based* representation as the most extended one, but we will also mention the HTN representation, the situation calculus, temporal logic, the representation based on ontologies or propositional logic. In scheduling, we will consider the constraint satisfaction representation approach and the representation based on ontologies.

2.1.1 Operators Based Representation for Planning

Within this representation the most widely used alternative for AI planning systems is the STRIPS representation. Derived from STRIPS, with a richer representation, are among others, the ADL and PDDL languages.

2.1.1.1 Classical Representation in Planning: STRIPS

The STRIPS (Standford Research Institute Problem Solver) representation was originally proposed by Fikes and Nilsson [63]. It was introduced to overcome what were seen as computational difficulties in using states to construct plans [109]. In the STRIPS representation, a world state is represented by a set of logical formulae ($\mathcal{A}$), the conjunction of which is intended to describe the given state.

Actions are represented by operators. An operator consists of preconditions (conditions that must be true to allow the action execution), and post-conditions or effects (usually composed of an add list and a delete list). The add list specifies the set of formulae that become true in the resulting state, while the delete list specifies the set of formulae that are no longer true and must be deleted from the description of the state. A transition function *f* maps states s into states s´ such that:

$s' = s - Del(a) + Add(a) \quad \forall\, a \in \mathcal{A}(s)$ in the progression space (that is, it starts from the initial state towards the goals)

As a result, the planner generates an ordered set of actions (serial or parallel) which, when executed in any world satisfying the initial state description, will achieve the goals. In STRIPS the cost of applying each action *a* in state *s*, *c(a,s)*, is 1. The optimal solutions of the problem are the optimal solutions of the state model. A possible way to find such solutions is by performing a search in such a space, or, in other approaches, to search in the possible space of plans. The key ingredient in both cases is the heuristic used to guide the search.

We can define a plan as an organised collection of operators. A plan is said to be a solution to a given problem if the plan is applicable in the problem's initial state, and if, after plan execution, the goals are true in the resulting state. The plan will be applicable if the preconditions for the execution of the first operator hold in the initial state.

Originally, the preconditions and effects of STRIPS operators were limited to a conjunctive list of literals. A more sophisticated language, ADL (Action Description Language) [132], has extended the STRIPS language to action schemata, allowing, among others, disjunction in the preconditions, conditions in the effects and universal quantification in preconditions and effects. Despite of this extension, STRIPS-based planners present some limitations in the way time and resources are handled:

- Time: there is not an explicit model of time, so they cannot represent durations, temporal constraints in operators or deadlines in the goals.

- Resources: requirements on resources or their consumption cannot be represented.

2.1.1.2 Standardizing Languages for Planner Comparison: PDDL2.1

Since STRIPS, many other planning languages have appeared, what do it very difficult to compare planners performance and efficiency. In order to unify these languages, the PDDL [113] language was born with this goal. PDDL, standing for Planning Domain Definition Language, is the current standard language for encoding planning domains. The first version of the language was developed by Drew McDermott and some other researchers for the 1998 AIPS Planning Competition. It is basically a first-order logic language with a Lisp syntax.[1] This early version supported the main features of the ADL [132], SIPE-2 [191], PRODIGY [180] and UCPOP [133] languages.

This version evolved into PDDL2.1 syntax [68], the official language used for the AIPS'02 competition. It has been extended to represent temporal features of domains by means of *durative actions* and plans that depend on concurrency and temporal structure.

In PDDL2.1 various levels to handle time have been considered (levels 3, 4, 5). The meaning of each level is as follows: levels 1 and 2 correspond to the original PDDL together with an extension to handle numeric variables and to update instantaneously their values. At level 3 (the one used in the competition), actions encapsulate continuous changes, that is, the correct values of any affected variables are guaranteed only at the end points of the interval. Instead, at level 4, discrete effects were allowed thanks to functions that can refer to instants on the time-line. Finally, level 5 [66] is a complete syntax and semantic to describe discrete state changes not included in the competition.

Durative actions can express initial and final effects and also, differentiate between invariant conditions and preconditions. Invariant conditions are conditions that must be true during the action execution. To achieve this, temporally annotated preconditions and effects are allowed. That is, we must state if the proposition is asserted *at the start*, *over all* or *at the end* of the interval. These actions take an amount of time not equal to zero to provide their effects. The duration can be expressed by a single numeric value or a complex function expression whose value is determined by conditions in the initial state.

[1]ANSI X3.226: 1994 American National Standard for Programming Language Common Lisp

Also conditional effects are allowed but we cannot refer to the start of an action if the conditions are referring to invariant or end conditions (this is done in order to avoid that start effects depend on end conditions).

About plans, they have extended classical plans to specify: the instant of time at which the action is to be executed, the name of the instantiated action between parenthesis (as classical plans) and the action duration between square brackets (this field is omitted if the action is not a *durative action* because, by default, its value would be one).

As a summary, some PDDL2.1 features are:

- Modified treatment of numeric-valued fluents.
- Explicit representation of time and duration.
- Full semantics for temporal plans.
- Plan metric specifications as part of problems.

2.1.2 Other Representations for Planning

Among other representations for planning we can mention:

- HTN: planning problems and operators are organised into a set of tasks. High level tasks are reduced into a set of lower levels and the way to do it can be done in several ways as explains Yang in [196].
- Temporal Logic: converts operators into universally quantified formulae [10] (see also section 2.2.6 for details on the techniques used).
- The Situation Calculus: is defined in [108] as a formalism that handles situations as objects, that considers fluents that take values in situations, and events (including actions) that generate new situations from old ones. It is sometimes necessary to prove that certain properties are true in all world states accessible from the initial state.
- Propositional Logic: converts STRIPS like operators into propositional logic. Go to section 2.2.5 for details on the techniques used.
- Planning Ontologies: as an application of the increasing demand for formalising knowledge on the Web we can mention Web-PDDL [53], where AI representations (e.g., logics and frames) can be mapped to XML (included RDF and RDFSchema).[2] In Web-PDDL, an automatic translator between PDDL and RDF/DAML[3] has been built up. It is a new approach to implement ontology translation by ontology merging and automated reasoning conducted by the inference engine. It can translate an input PDDL file (either a domain, a situation, a problem definition or any combination of them) to a RDF/DAML file.

[2]http://www.w3.org/TR/2000/CR-rdf-schema-20000327/
[3]http://www.daml.org/

Another ontology approaches for planning are PLANET [77] and O-PLAN [129] (see section 2.4.1.3).

2.1.3 Scheduling Representations

In scheduling, less attention has been paid to the representation. Basically we can consider the Constraint Satisfaction approach[4] where the representation is very poor due to a very simple semantics (in Appendix B we can see an example of a constraint based scheduler input file) or the representation based on ontologies. We will describe here some of the main ontologies based approaches on scheduling.

2.1.3.1 Scheduling Ontologies: OZONE

OZONE [171] is an ontology considered as a framework for specifying models in a particular domain, that is, a meta-model that provides a vocabulary for formulating application models, as well as a set of constraints on how concepts can fit together to form consistent domain models.

The OZONE scheduling ontology adopts an activity-centered modeling viewpoint [170]. Scheduling is defined as a process that requires the use of *resources* by *activities* to satisfy *demands* over time. The problems are described in terms of this abstract domain model. A *demand* is an input request for one or more *products*, which designate the *goods* or *services* required. Satisfaction of demands centres around the execution of activities. An *activity* is a process that uses *resources* to produce *goods* or provide *services*. The use of *resources* and the execution of activities is restricted by a set of constraints.

Plans and schedules are represented as networks of *activities*, with an *activity* containing various decision variables as start time, end time and resources used. To construct a schedule that satisfies a given input *demand*, it is necessary to first instantiate a set of *activities* that will produce the *product*.

To schedule an activity, it is necessary to choose specific resources, which involves determining intervals where resources have capacity available to support the execution of the activity, and subsequently allocating capacity of chosen resources to ensure that they will not be used by other *activities*.

An important feature is its basic commitment to a constraint-based model of scheduling. The model defines a problem solving with two main components:

- A decision-making component, that makes the choices among alternative scheduling decisions and retracts the undesirable ones.

- A constraint management component, that propagates the consequences of the decisions made and maintains a representation of the current set of feasible solutions.

[4]The description of a CSP problem will be done on section 2.3.2.1.

2.1.3.2 Domain Description Language: DDL

The DDL representation [122] was designed to model the structure and dynamics of physical systems at multiple levels of abstraction and to represent evolutions of the state of the system over time.

DDL is subdivided into a set of system components, each of which has an associated set of properties. Each property represents an entry of a state vector, so only one value is considered at any point in time. The behaviour of the system is determined by the value of its dynamic properties called state variables of the system. Each value is constrained by a compatibility specification that consists of a set of compatibilities organised as an AND/OR graph. Compatibility implementation corresponds to precondition and postcondition satisfaction in classical planning.

For the Hubble Space Telescope problem (HST)[5] two levels of abstraction have been considered: the abstract level provides a basis for globally focus on planning details, and the detail level provides a basis for abstract predictions.

The primitive temporal description is the token, that is, a time interval identified by its start time and end time, over which a specified condition, identified by a type, holds. Each token can only represent a segment of the evolution of a single state variable.

Asserting a value token does not guarantee that it will eventually be included in an executable plan. An executable token has also to find a time interval over which no other value token can possibly occur on the same state variable. A time line is a linear sequence of tokens that completely cover the scheduling horizon for a single state variable. In a completely specified plan/schedule, the time line consists of a sequence of value tokens with ground predicate types.

2.2 Planning Techniques

Planning is basically a search problem [8] where the choice of what space to search is critical to find good solutions efficiently. We can classify planning algorithms according to the following features:

- How the search is performed. It can be done through the space of world states (regression or progression) or through the space of plans (totally or partially ordered plans).
- Goal ordering. When solving a conjunctive goal, if the planner requires that all subgoals of one conjunct be achieved before subgoals of the others, it is called a *linear* planner; hence, planners that can arbitrarily interleave subgoals are called *nonlinear*.
- How the plans are generated. The plans generation can be done by refinement, retraction and transformation. Refinement is the process of gradually adding actions and constraints. Retraction eliminates previously added actions, and transformation interleaves refinement and retraction activities.

[5]The DDL language was designed for this particular domain.

- How the plans are built. If they do not use libraries of plans, that is, from scratch, they are called generative planners, but if they reuse previously generated plans, they are called case-based planners.

- How they handle uncertainty. Most of the classical planners use the assumption that there is not uncertainty in the world: every action has a predictable output. Another type of planners are the contingent planners. A contingent plan refers to a plan that contains actions that may or may not actually be executed, depending on the circumstances that hold at the time. A contingent planner must be able to produce plans even if the outcomes of some of the actions as well as the initial conditions, are not known *a priori*. Another way to handle uncertainty is done by the probabilistic planners; they use probabilities to represent and reason about uncertainty in planning domains [17].

- The use of domain specific knowledge [187, 188]. The planners which use algorithms expecting to work in reasonably large variety of application domains are called domain-independent, in opposition to the domain-dependent planners which use domain-specific heuristics to control the planner´s operation for a given domain.

Each method differs in its philosophy to expand the nodes and the structure of the search graph. To determine the adequate methods when solving planning or scheduling problems, we should have in mind:

- Soundness: the solution returned is a valid one.

- Completeness: if a solution exists the algorithm is able to find it.

- Optimality: it finds the best solution. In that case we should also define with respect to what criterion we want the solution to be optimal.

- Efficiency: how fast the solution is found.

Some of the most important techniques in this area are now described. For this reason, the same planners can be refered to in different sections. Another way of describing it, is presented in the PLANET repository [137].

2.2.1 Total Order Planners

We can define a Total Order (TO) planner as a planner that generates solutions that are sequences of total ordered actions. The TO distinction should also be kept separate from the distinction between state-based and plan-based planners. In a state-based planner, each search state corresponds to a state of the world and in a plan-based planner, each search state corresponds to a plan. While TO planners are commonly associated with state-based planners [18, 85, 180, 183], several TO planners have been plan-based [176, 185].

The most significant planners in this category are:

- PRODIGY is an integrated planning and learning system [180]. The system includes a planning algorithm and learning techniques such as case-based reasoning, analytical (EBL) or inductive techniques that can greatly increase the efficiency of the planner. It is a non-linear, domain-independent problem solver architecture. The language used is an extension of the STRIPS representation [180].
- VVPLAN [183] is a planner based on a forward state space search algorithm that uses the ADL language for representing domains and problems. Its main feature is to introduce a loop test relating to previously visited states that gives a significant increase in performance.
- As TO plan-based planners one can mention INTERPLAN [176] and WARPLAN [185].
- See also the planners in section 2.2.4

2.2.2 Partial Order Causal Link Planners

Partial Order Causal Link planners [187] search through the plan space. In this space, nodes represent partially specified plans, and edges denote plan-refinements operations such as the addition of an action to a plan. The planning algorithm implements a least commitment approach, that is, only the essential ordering decisions are recorded, there is no need to prematurely commit to a complete, totally sequence of actions.

In a Partial Order (PO) plan a step's precondition is possibly true if there exists a linearization in which it is true, and a step's precondition is necessarily true if it is true in all linearizations. A step's precondition is necessarily false if it is not possibly true. Steps on POP can be unordered with respect to each other. A linearization of a partially ordered plan is a total order over the plan's steps that is consistent with the existing partial order.

These planners must perform constraints satisfaction to ensure the consistency of the order constraints. In order to avoid interferences between actions, dependencies are recorded in a data structure called causal links. Causal links must be checked periodically during the planning process to make sure that no other action can threaten them. In case a plan contains a threat, some additional ordering constraints can be added to avoid it. The methods used are:

- Demotion: adding the constraint before the step that threats.
- Promotion: adding the constraint after the step that threats.
- Separation: adding a variable binding constraint.
- Confrontation: adding the negation of the conditional effects.

The search space in partial order planning expands quickly as the problem size increases: with the number of subgoals and initial conditions, as well as less countable factors such as operator ordering and subgoal interactions. Among the most representative PO planners, one can mention:

- UCPOP [133] it is the most well-known planner of this category. It uses a regressive algorithm that searches the space of plans. It plans by iteratively selecting and repairing flaws in the current plan. A flaw is repaired by adding steps and constraints to the plan. The search control strategy decides which partial plan to select for expansion. It handles a subset of Pednault's ADL action representation: conditional effects, disjunctive preconditions and quantified goals.

- CASSANDRA [140] and SENSp [55] are contingent, domain independent problem solver architectures based on UCPOP. They plan under uncertainty and differ in the way they represent it.

- ZENO [134] is a least commitment planner that handles actions (that can be simultaneous if the effects do not interfere) occurring over extended intervals of time. It supports: deadline goals, metric preconditions and effects and continuos changes. It uses a typed first-order language with equality to describe goals and the effects of actions. All types except time are assumed to be finite. A point-based model of time is adopted: temporal functions and relations use a Time Point as their arguments. ZENO relies on constraint satisfaction for all temporal and metric aspects. Linear equations are solved by Gaussian elimination, linear inequalities by the *Simplex* algorithm [6] and non-linear equations are delayed until they become linear via the solution of other equations and inequalities.

- VHPOP [197] is loosely based on UCPOP. Its key feature is its versatility. It implements all the common flaw selection strategies [139] as well as new ones as selecting open conditions based on heuristic cost. Because no single flaw selection strategy seems to be the best for all problems, VHPOP allows several flaw selection strategies to be used simultaneously. This makes it possible to combine the strengths of different strategies, allowing to solve more problems.

  It can support durative actions by adding a simple temporal network (STN) to the regular plan representation of a partial order planner. The STN records temporal constraints between actions in a plan, and supersedes the simple ordering constraints usually recorded by partial order planners. The use of STNs permits actions with interval constraints on the duration.

2.2.3 Graph Based Planners

The planners that belong to this category are derived from GRAPHPLAN [15]. Their search structure is a plan graph.

GRAPHPLAN alternates between graph expansion and solution extraction. The graph expansion extends the plan graphs forward until it has achieved a necessary condition for plan existence. The solution extraction phase performs a backward-chaining search on the graph, looking for a plan that solves the problem. If no solution can be found, the cycle repeats expanding the planning graph. The basic idea

[6]Created by George Dantzig in 1947, successfully used to solve linear programming problems.

is to perform a kind of reachability analysis to rule out many of the combinations and set of actions that are not compatible. The compatibility is done inferring binary mutual exclusion *mutex* relationships between incompatible actions (they have opposite effects, mutex preconditions, or the effect of one is the opposite of the other) and between incompatible propositions (opposite literals or all the actions that give rise to them are mutex at the previous step).

In the planning graph, there are two types of nodes: the proposition nodes (even levels) and action nodes (odd levels). The true propositions of the initial state are in level zero. Edges connect proposition nodes to action instances and actions nodes to subsequent propositions made true by the action's effects.

For the solution extraction, the planning graph has been extended to a level i at which all goal propositions are present and none are pairwise mutex. For each literal at level i, it chooses an action at level i-1 that achieves the subgoal. This choice is a decision point, if more than one action can be applied; it has to consider all possible ones to ensure completeness. If the action is consistent, then it proceeds with the next subgoal, otherwise backtracks. After finding a consistent set of actions at level i-1, it recursively tries to find a plan for the preconditions of the actions at level i-2. The process finishes when all the propositions are present at level zero. If after backtracking of all possible combinations fails, GRAPHPLAN extends the planning graph to an additional level and tries again to extract the solution.

In order to improve performance, different techniques have been applied to the graph expansion (regression focusing, handling of close world assumption and in-place graph expansion) and to the solution extraction (memoization, forward checking, dynamic variable ordering and explanation based learning). For a more detailed introduction of these techniques go to [188].

Among GRAPHPLAN descendants, one can mention:

- SGP [189] uses PDDL language syntax that solves contingent planning problems with uncertainty in the initial condition and with actions that combine causal and sensory effects.

- TGP [168] operates by expanding a planning graph representation. It adopts a simple extension of the STRIPS action language that allows each action to have a nonnegative start time, s, and a positive real-valued duration, d. The temporal model assumes that all preconditions must hold at the start, s, of the action; preconditions not affected by the action itself must hold through execution; and effects are undefined during execution and only guaranteed to hold at the final time point s+d.

- IPP [97] has extended GRAPHPLAN to handle resources. The basic resource representation language (BRL) is an extension of ADL that offers the following features:

  - Resource requirements of actions can be specified in preconditions.
  - Resource effects of actions can occur in its unconditional effects.

  – Database query schema support a compact representation of resource requirements and effects.

  When applying a BRL action to a state, the logical effects only change the set of ground literals that characterise the state, while the resource effects only change the values of the resource variables. Therefore the result of applying an action can be defined using two independent functions. The IPP search algorithm handles logical goals together with interval arithmetics to handle resource goals.

- TPSYS: Temporal Planning SYStem [73] is a temporal planner which combines the ideas of GRAPHPLAN and TGP to deal with level 3 durative actions of PDDL2.1. TPSYS [72] consists of three stages. The first stage calculates the static mutex relationships which always hold in the problem. The second stage extends a temporal planning graph, generating levels which are chronologically ordered by their instant of time. Finally, the third stage performs a backward search through the temporal graph and extracts a feasible, optimal plan. Although TPSYS is originally complete and optimal, it has currently substituted the backward search by a new process based on least-commitment and heuristic search. The new search significantly improves the scalability of TPSYS. This search is neither complete nor optimal, but it obtains good quality (short makespan) plans in reasonable times of execution.

- SAPA [52]: is a domain-independent heuristic forward chaining planner that can handle durative actions, metric resource constraints, and deadline goals. It employs a set of distance based heuristics to control its search. It uses admissible heuristics for objective functions based on makespan and slack to consider optimisation factors. The heuristics are derived from the *relaxed temporal planning graph structure*, which is a generalization of planning graphs to temporal domains. The temporal graph is built by increasing incrementally the time (makespan value) of the graph.

- STAN [67]: is a domain-independent planner that exploits a representational structure for the core plan-graph and uses a wave-front mechanism to reduce search and graph construction. It uses a domain-independent automatic domain analysis tool, TIM [65, 104], that automatically extracts type information from a planning domain based on identification of simple finite-state machines which characterise the state-transitions possible for objects of different types within a planning domain.

- See also LPG [74] in section 2.2.5.

2.2.4 Heuristic Search Planners

Heuristic Search Planners (HSP) transform planning problems into problems of heuristic search by automatically extracting heuristics functions h from STRIPS encoding instead of introducing them manually [18]. It uses a declarative language for stating problems and a general mechanism for extracting heuristics from these representations. The same code is then able to process problems from different domains.

Considering a relaxed problem in which all delete lists are ignored, from any state *s*, the optimal cost *h´(s)* for solving the relaxed problem can be shown to be a lower bound on the optimal cost *h*(s)* for solving the original problem. As a result, the heuristic function *h´(s)* can be used as an admissible heuristic for solving the original problem. Because obtaining *h´(s)* is a NP-problem, the approximation *h(s)* is set to an estimate of the optimal value function *h´(s)* of the relaxed problem.

Some heuristics that can be used to estimate the cost of achieving an atom from a given state are:

- To calculate the cost of a set of atoms assuming that subgoals are independent. This heuristic is called the additive heuristic h_{add}. This is not true in general, although it is quite useful in planning.

- Combine the cost of atoms by a max operation. This is called the max heuristic, i.e., replacing sums by maximisation. This heuristic is admissible unlike the additive heuristic, as the cost of achieving a set of atoms cannot be lower than the cost of achieving the most expensive atoms in the set.

The HSP2 [18, 19] planner performs a forward state-space search guided by the additive h_{add} heuristic in a best-first search that keeps an Open and Closed list of nodes as A*, but weights nodes by an evaluation function equal to the accumulated cost plus the estimated cost to the goal multiplied by the constant $W = 5$ (if W=1 the algorithm is A*). This heuristic has to be evaluated from scratch every time a new state is generated.

The bottleneck in HSP is the computation in every new state of the heuristic from scratch. This could be solved by performing the search backwards from the goal. In that case, the estimated costs derived from the initial state could be used without recomputation for defining the heuristic of any state arising in the backward search.

For tasks that need to reason about time and resources, the HSP planner can be extended to handle concurrent actions, time, resources and minimises makespan. Some admissible heuristic estimator for time and resources can also be used (see [84] for further details).

We can mention as descendants the following planners:

- The FF [86] planner is a forward chaining state space planner that uses heuristic techniques to guide its search. Instead of obtaining an estimation of the solution length as in HSP, it extracts an explicit solution using a GRAPHPLAN style algorithm. The length of the relaxed solutions is used as a goal distance estimate that controls a local search strategy: hill climbing procedure intermixed with breadth first search to find a better successor. It handles classical STRIPS as well as full scale ADL planning tasks to be specified in PDDL. Metric-FF [85] is an extension of the FF planner to support numerical state variables.

- SSPOP [154] is a hybrid planner since it combines state-space search, the reduced search-space feature of the partial-order planners and heuristic search guided by a domain-independent algorithm. Using local search techniques, it discovers groups of plans that are equivalent re-orderings of a set of actions and keeps one plan of each group for further exploration. Moreover, additional checks detect and eliminate plans with redundant actions. It has been enhanced with Greedy Regression Tables (GRT), a domain-independent heuristic algorithm that estimates distances between intermediate states and the goal [153].

2.2.5 SAT-Based Planners

A SAT-based planner takes a planning problem as an input, guesses a plan length and generates a set of propositional clauses that are checked for satisfiability. After the translation is performed, fast simplification algorithms as unit propagation and pure literal elimination are used to shrink the CNF formula. For satisfying assignment, a SAT solver can use the following methods:

- Systematic: the DPLL [102] algorithm is the most representative of the systematic methods. It performs a backtracking depth-first search through the space of partial truth assignments using unit clause and pure literal heuristics. The order in which variables are chosen strongly influences the satisfying assignment.

- Stochastic: the GSAT [162] solver performs a hill-climbing search with a random-restart through the space of complete truth assignments. WALKSAT [130, 161] improves GSAT adding additional randomness related to simulated annealing. Although this method is incomplete, it presents a high performance.

- Incremental: the more representative is LMTS-style truth maintenance [107]. It differs from the other two methods in the way it finds a truth assignment: instead of having two possible values (true, false) it also has the unknown value. Not only it finds a mapping (using unit propagation), but it maintains it doing incremental changes.

If the formulae are unsatisfiable, the length is increased, and the process is repeated. A symbol table records the correspondence between propositional variables and the planning instance.

In this category one can mention:

- In SATPLAN [92, 93] axioms involve general state-based encoding that incorporate general domain knowledge. This encoding was hand-tuned due to the difficulty to derive them from the STRIPS input. This encoding also includes state invariants (such as the fact that a truck is only at one location at a time) that boost the performance of problem solvers.

- MEDIC [59] generates the encoding directly for the axiom instead of taking advantage of an intermediate plan graph representation, despite the fact that they are logically equivalent.

- BLACKBOX [94, 160] is a system that combines the best features of GRAPHPLAN and SATPLAN. The plan graph-based wffs[7] contain fewer variables and more clauses, so it is easier to solve. It works as follows:
  - The planning problem, specified in the PDDL notation, is converted to a plan graph of length l, and the mutex computation is based on an algorithm that determines the pairs of actions or pairs of facts that are mutually exclusive (i.e., negative binary clauses).
  - The plan graph is converted to a CNF well defined formula.
  - The wff is simplified by a general CNF simplification algorithm (it uses the failed literal rule instead of unit propagation).
  - The wff is solved by any of a variety of fast SAT engines.
  - If a model of the wff is found, then the model is converted to the corresponding plan, otherwise the length is incremented and the process repeats.

- LPSAT [194] uses a randomised backtracking algorithm, introducing a new problem formulation, LCNF, which combines the expressive power of propositional logic with that of linear equalities and inequalities. The LNCF solver, is a systematic satisfiability solver integrated with an incremental *Simplex* algorithm. As LPSAT explores the propositional search space, it updates the set of metric requirements managed by the linear program solver, after *Simplex* notifies the propositional solver if these requirements become unsatisfiable. It is able to solve large resource planning problems, encoded in a variant of the PDDL language based on the metric constructs used by IPP. It supports equality, quantified goals and effects, disjunctive preconditions and conditional effects. LPSAT´s depth-first search of the propositional search space creates a partial assignment to the Boolean variables. When the search fails, it is because the partial assignment is inconsistent with the LNCF problem. This inconsistent subset of the truth assignments in the partial assignment is used by LPSAT to learn clauses that disallow those particular assignments.

- LPG [74] is a planner based on local search and planning graphs that handles PDDL2.1 domains involving numerical quantities and durations. The system can solve both plan generation and plan adaptation problems. The basic search scheme of LPG was inspired by WALKSAT [130, 161].

  The search space consists of action graphs, particularly subgraphs of the planning graph representing partial plans. The search steps are certain graph modifications transforming an action graph into another one. LPG exploits a compact representation of the planning graph to define the search neighbourhood and to evaluate its elements using a parametrized function. The evaluation function uses some heuristics for achieving a (possibly numeric) precondition. Action durations and numerical quantities are represented in the actions

[7]Well Formed Formulae.

graphs, and are modeled in the evaluation function. In temporal domains, actions are ordered using a *precedence graph* that is maintained during search, and that takes into account the mutex relations of the planning graph.

2.2.6 Knowledge-Based Planning Systems and HTN Planners

Knowledge-based planning systems try to use the knowledge available of the domain to solve the planning problem. This knowledge can be obtained using additional task and goal structures, search control techniques, interaction with humans or different type of constraints.

Detecting interactions and solving conflicts is one of the key issues for planning systems. The introduction of tasks networks and tasks decomposition provided an even richer set of interactions and resolution methods. The system NOAH was designed for handling these interactions using the critic mechanism [157]. The power of this mechanism was quickly adapted by Hierarchical Tasks Network (HTN) planning systems as: NONLIN [178], O-PLAN [177, 47], DEVISER [182], SIPE [191] or NMRA (New Millenium Remote Agent) [121].

Actions, which in HTN planning are usually called tasks, correspond to state transitions. A task network is a collection of tasks that need to be carried out, together with constraints on the order in which tasks can be performed, the way variables are instantiated, and what literals must be true before or after each task is performed. A task network that contains only primitive tasks is called a primitive tasks network. In the more general case, it can contain non-primitive tasks, which the planner needs to figure out how to accomplish. Ways of accomplishing these tasks are represented using constructs called methods. An HTN problem [60] is represented as a triple P = $<$ d, $\mathcal{I}$, $\mathcal{D}$$>$, where d is the goals we need to plan for, $\mathcal{I}$ is the initial state, and $\mathcal{D}$ is the set of operators and methods associated with the planning domain.

A method maps a task into a partially ordered network of tasks, together with a set of constraints. The basic algorithm is to expand tasks and resolve conflicts iteratively, until a conflict-free plan can be found that consists only of primitive tasks. The task network may contain conflicts caused by the interaction among tasks: after each reduction, a set of critics is checked so as to recognise and solve interactions between this and any other reductions. Thus critics provide a general mechanism for detecting interactions early, so as to reduce the amount of backtracking.

The primary difference between HTN planners and STRIPS-style planners is in what they plan for, and how they plan for it. In the second one, the objective is to find an ordered set of actions that will bring the world to a state that satisfies certain conditions or attainments goals. Planning proceeds by finding operators that have the desired effects and by making the preconditions of those operators into subgoals. In contrast the first one searches for plans that accomplish task networks, which can include things other than just attainment goals; and they plan via task decomposition and conflict resolution. But methods must contain all possible ways tasks can be achieved, so this is usually a tedious tasks. Unlike STRIPS, the constraints may or not contain conditions on what must be true in the final state.

Extending HTN systems to handle time and resources is a relatively easy task: constraints can be specified within the methods and can be checked for consistency along orderings and constraints.

Some Knowledge-based Planner systems apart from the mentioned above:

- ALPINE [23] is one of the modules that composes the PRODIGY architecture (The Abstraction Planning Module). It produces abstraction hierarchies to help control the problem solver's search. This module employs an algorithm that takes a set of operators and a problem domain as input and then partitions the literals in the operators' preconditions, add lists, and delete lists to form a hierarchy of abstraction spaces.

  The higher levels represent more abstract descriptions of the domain while the lower levels represent the domain in more detail. The problem solver can plan at a high level of abstraction, so the most important preconditions and effects are considered. The plan is later refined as needed by planning at lower levels.

- SHOP (Simple Hierarchical Ordered Planner) [124] is a domain-independent HTN planning system that plans for tasks in the same order that they will be later executed. This feature avoids some goal interaction issues that arise in other HTN planners. But the total-ordering requirement for the subtasks of its methods is more restrictive than it needs to be. To soften this constraint, it has been extended to allow the subtasks of each method to be partially ordered [125]. Since it knows the complete world-state at each step of the planning process, it can use highly expressive domain representations.

- TALplanner (Temporal Action Logic planner) [106] is inspired by TLplan [10]. TLplan consists of: a search engine, a goal tester to determine if the goal has been reached, a state expander to find the successors of a world, a formula progressor and a formula evaluator. It converts operators into formulae universally quantified by the parameters so there is no need to do unification. It can plan with conditional actions expressed using the full ADL language and can handle certain types of resource constraints. However TLplan has been extended to generate plans satisfying timing deadlines. The language used for expressing search control knowledge is a first-order version of linear temporal logic [58].

  TALplanner is a forward chaining logic-based planner where domain knowledge is expressed as logical formulae, controlling the search for solution plans. The planning algorithm is a forward chaining depth-first search algorithm that allows easy world state representation and the use of complex operators. The biggest disadvantage is that no goal directedness exists and consequently quite elaborate control has to be imposed on the search to make the planning process efficient.

  Analysing the control rules, it creates pruning constraints, that is, logic formulae that must hold in any partial plan. If the pruning constraint does not hold, the state node can be pruned, and with it, the entire branch of the search tree that would stem from that node. The result is a drastic reduction of the search space enabling to find a solution more efficiently.

- HYBIS [24, 25] is a hybrid planning system successfully applied to the control software design for manufacturing systems. A planning domain is described as a compositional hierarchy of agents, where agents are described at different levels of abstraction: the leaf nodes of the hierarchy are primitive agents, that represent real world entities to act while higher levels are constituted by compound agents.

  Every agent has attached a set of actions that it can execute, representing its behaviour, described as a finite automaton. Every action can be either primitive or compound (depending on which agent it is attached to), and they share the same structure at every level of abstraction. The goal of the hybrid hierarchical planning process is to obtain a hierarchical and modular plan.

2.2.7 Planning in Non-Deterministic Domains

In real problems as the satellite domain, uncertainty plays an important role. When a planning problem includes uncertainty, classical plans cannot be reliable executed. But if the domain is predictable to some degree, classical planning can be adapted to produce plans with run-time sensing and conditional steps. The only problem to deal with is the increased search space. Now, we will describe some common solutions to handling uncertainty.

2.2.7.1 Markov Decision Process

A Markov Decision Process (MDP) is defined by an initial distribution $\mathcal{P}$ over the states, the dynamics of the system with states annotated by different possible actions, the probabilistic state transitions (s,a) and a reward function to make the transition from the state x_t to state x_{t+1}, R(s,a). An MDP is solved specifying actions to perform at each state, such that the application of the action will maximise the expected cumulative reward of the system over time. Techniques based on policy iteration or value iteration [141] have been traditionally used. These techniques that belong to dynamic programming require fully enumeration of all possible states. Again, this can be intractable in most of the planning systems, so most of the work in this area has focused on using only a subset (the most probable state space) or abstractions of the state space.

Similar work has been done with Partially Observable Markov Decision Processes (POMDP) in which the assumption of complete observability is relaxed. Although work in this area is promising, it has only been used to solve small POMDP problems.

An important area based on MDP has been the control learning task. Watkins [186] introduced the method of reinforcement learning called Q-learning. The agent exists within a world that can be modeled as a MDP. It observes discrete states of the world and can execute discrete actions. At each discrete time step, the agent observes states, takes actions, observes new states and receives immediate reward.

2.2.7.2 Probabilistic Planers

Probabilistic planners use probabilities to represent and reason about uncertainty in the planning domain, that is, the probabilities of the possible uncertain outcomes to construct plans that are likely to succeed. They have typically a larger search space than their classical counterparts so heuristics to reduce the search effectively are even more important to the planners belonging to this category. To learn more about these techniques in planning as well as their limitation see [17].

Among the planners of this type, one can mention:

- C-BURIDAN [54] is a PO planner based on CNLP [135]. The actions can have sensing results as well as effects in the world. It allows more that one causal link for each condition in the plan. Under different execution scenarios, different actions may link for each condition to be true, so the links combine to increase support for the condition. It creates plans that meet a threshold probability of success when actions are non-deterministic.

- WEAVER [16] is based on PRODIGY [22] and as C-BURIDAN has no explicit model of observability but assumes the world is completely observable at plan execution time. However, it has an explicit representation for uncertain exogenous events. Search heuristics for planning under uncertainty were used to reduce the search space.

- In DRIPS [80] ranges of utility values are computed for partial plans, calculating the best and the worst expected utility value for all possible partial plans. If the lowest value in some partial plan exceed the highest value of another plan, the first one can be dropped. This can lead to a lot of savings. This is accomplish by refinement planning based on an abstraction hierarchy of operators. The representation in DRIPS allows for deadlines and maintenance goals.

- C-MAXPLAN [103] compiles a probabilistic planning problem into a SAT problem. It uses the same philosophy as in SATPLAN.

- See also HYBIS [24, 25] in section 2.2.6.

2.2.7.3 Model Checking Techniques

Model Checking [43] is a method for formally verifying finite-state concurrent systems. Specifications about the system are expressed as temporal logic formulas, and efficient symbolic algorithms are used to traverse the model defined by the system and check if the specification holds or not.

Symbolic Model Checking [42] is a particular form of Model Checking that allows to analyse extremely large finite-state systems by means of symbolic representation techniques (e.g., Binary Decision Diagrams and propositional satisfiability). (Symbolic) Model Checking is the core technique for several industrial verification tools and is the main research topic in the area of hardware and software verification.

In the last few years, Model Checking has been shown to be a very effective approach to planning. The Planning as Model Checking formulates a planning problem in a logical context, while the symbolic representation techniques allow for handling complex domains.

The key idea underlying the Planning as Model Checking paradigm is that planning problems should be solved model-theoretically. A planning domain is formalized as a specification of possible models for plans. Goals are formalized as logical requirements about the desired behaviour for plans. The planning problem (i.e., finding a plan that achieves the goal) is solved by searching through the state space, checking that there exists a plan that satisfies the goal. The same framework can be used to tackle different planning problems, e.g., planning under uncertainty, conditional and iterative planning, and planning under partial observability. The approach is practical: planning algorithms based on Symbolic Model Checking techniques can deal with large size problems.

As examples of planners that integrate Model Checking techniques one can mention:

- MIPS [56] is a planning system that applies binary decision diagrams to compactly represent world states in a planning problem and efficiently explore the underlying state space. It was the first general planning system based on Model Checking methods. It has been extended for the AIPS'02 competition to handle numerical quantities and durations of PDDL2.1 (Level 2-3 in PDDL+).

- CIRCA [57] uses Model Checking with timed automata to verify plans that should meet timing constraints. The CIRCA architecture allows real-time control system as in Uninhabited Aerial Vehicles and deep space probes.

- See also TLplan [10] and TALplanner [106] on section 2.2.6.

2.3 Scheduling Techniques

Scheduling has to face the problem of organising tasks in time. The problem is to locate a set of tasks in time, each task needing one or several resources during its execution. Many techniques used in this area come from the Operational Research (OR) area (branch and bound, simulated annealing, lagrangian relaxation, ...). Lately, Constraint Programming CP has been applied to the *Job-shop* scheduling problems with very good results.

2.3.1 Classical Operational Research Technology

OR also known as Management Science (MS), is the scientific approach to solving management problems.

Typical OR/MS problems involve deciding how to make the most cost-effective use of limited resources such as people, machines, money and time. Using observation, data, and analysis, the OR practitioner builds up quantitative relationships, called models, which may be used to help management make informed decisions.

OR has been used intensively in business, industry and government. Many new analytical methods have evolved, such as: mathematical programming, simulation, game theory, queuing theory, network analysis, decision analysis, multicriteria analysis, etc., which have powerful application to practical problems with the appropriate logical structure.

Within the OR, the following techniques are the most well-known used.

2.3.1.1 Linear Programming

A Linear Programming problem [41] is a special case of a Mathematical Programming problem [99].

From an analytical perspective, a mathematical program tries to identify an extreme (i.e., minimum or maximum) point of a function $f(x_1, x_2, x_3, ..., x_n)$, which furthermore satisfies a set of constraints, e.g., $g_i\ (x_1, x_2, x_3, ..., x_n) \geq b_i$.

Linear Programming is the specialization of mathematical programming to the case where both, function f - to be called the objective function - and the problem constraints are linear.

From an applications perspective, mathematical (and therefore, linear) programming is an optimization tool, which allows the rationalization of many managerial and/or technological decisions required by contemporary techno-socio-economic applications.

For Linear Programming problems, the *Simplex* algorithm provides a powerful computational tool, able to provide fast solutions to very large-scale applications, sometimes including hundreds of thousands of variables (i.e., decision factors).

It is a systematic approach that starts with an initial basic feasible solution (bfs) and tests its optimality. If some optimality condition is verified, then the algorithm terminates. Otherwise, the algorithm identifies an adjacent bfs, with a better objective value. The optimality of this new solution is tested again, and the entire scheme is repeated, until an optimal bfs is found. Since every time a new bfs is identified the objective value is improved (except from a certain pathological case that we shall see later), and the set of bfs's is finite, it follows that the algorithm will terminate in a finite number of steps (iterations).

2.3.1.2 Sensitivity Analysis

Finding the optimal solution to a linear programming model is important, but it is not the only information available. There is a tremendous amount of *sensitivity* information, or information about what happens when data values are changed.

While problems may be modeled using deterministic objective functions, in the real world there is variation. A Sensitivity Analysis can be performed to determine the sensitivity of the solution to changes in parameters.

Sensitivity Analysis [88] (or post-optimality analysis) concerns with how the solution changes if some changes are made in either the data or in some of the solution values. It is used to determine how the optimal solution is affected by changes, within specified ranges.

Marginal Analysis is concerned with the effects of small perturbations, maybe measurable by derivatives. And parametric analysis is concerned with larger changes in parameter values that affect the data in the mathematical program.

2.3.1.3 Integer Linear Programs in Planning

ILP-PLAN [95] solves AI planning problems represented as Integer Linear Programs. Extends the planning problems to handle plans with resources, action costs and complex objective functions as Linear Integer Programming.

They use a commercial modeling STRIPS-style language called AMPL and a mix integer programming package, ILOG Cplex. They argue that specifying a planning problem by a combination of STRIPS operators and linear inequalities is better and more natural than using only extend STRIPS operators or only linear inequalities.

The classical AI planning domain allows quantitative resources (integer or real-valued) with a minimum and maximum value and set an optimisation criteria. A resource precondition is a linear inequality that must hold in any state at which the action is applicable. There are several types of action effects: effects that consume (decrease), provide (set to a value) or produce (increase) a resource.

2.3.2 Constraint Satisfaction Technology

Constraint Satisfaction (CS) has been the technique most widely used for solving scheduling problems. Most of the algorithms used for resolving CS fall into these two categories:

- Refinement or Constructive Search Methods: a CSP that progresses *incrementally* assigning values to variables, checking the constraints and backtracking when violations appear. The evaluation of global criteria can only be approximated given that there is only a partial schedule.
- Iterative Repair or Local Search Methods: they start with a complete assignment of values to variables and then reassign new values to variables to resolve violated constraints. The evaluation of global criteria is evaluated with low cost. However, one of the main disadvantages is that they can suffer from local minima and they are often incomplete.

2.3.2.1 Modeling a Problem as a CSP

A Constraint Satisfaction Problem (CSP) prescribes some requirements for a finite number of variables in the form of constraints. The set of possible values, that is, the domain for each variable is finite. A CSP problem can be described by a triple <X, D, C>, where:

i) $X = \{x_1, ..., x_n\}$ is the set of variables.

ii) $D = \{D_1, ..., D_n\}$ is the set of domains. Each domain is a finite set containing the possible values for the corresponding variable x_i: $1 \leq i \leq n$.

iii) C = $\{C_1,..., C_n\}$ is the set of constraints. A constraint C is a relation defined on a subset $\{x_1, ..., x_n\}$ of all the variables.

The size of this set is known as the arity of the constraint. A solution to a CSP is an assignment of a value $a_1 \in D$ to x_i, $1 \leq i \leq n$ that satisfies all the constraints. If all the variables have a value, then we talk about complete instantiation of the variables. Otherwise, the variables are partially instantiated. If we assign a value to an uninstantiated variable in a partial instantiation, then we extend the partial instantiation.

We will refer to an instantiation of a variable as <x, v>. The projection of a constraint C on the variables $x_1, ..., x_n$, denoted by P (C, $x_1, ..., x_n$), is the set of all ($v_1, ..., v_n$) values for which it holds that the (<x_1, v_1>, ... <x_n, v_n>) partial instantiation can be extended to hold the constraints.

A CSP is usually formulated in such a way that the domain D contains other elements than the ones in P(C, x). Removing those elements from D has no effect on the solution set. However, it decreases the size of the solution space, which is advantageous when searching for a solution. The same can be told of constraints.

The task of producing the minimal equivalent of a CSP is also NP-hard, so in general it is not a feasible scenario to reduce the problem to its minimum and then solve it.

The structure of a constraint problem is a graph whose nodes represent the variables, and any two nodes are connected if the corresponding variables participate in the same constraint scope.

The main techniques to solve CSP are:

- Search algorithms: cross the space of partial instantiations, building a complete instantiation that satisfies all the constraints, or determine that the problem is inconsistent.

- Propagation algorithms: infer new constraints that are added to the problem without loosing any information. The worst case time complexity is polynomial in the size of the problem. They are fast in discovering local inconsistencies.

- Structure-driven algorithms: is a mix of the two above techniques.

All the algorithms from the above three classes investigate the solution space systematically.

2.3.2.2 General Search Techniques

As general search techniques in scheduling we can mention systematic and local search methods.

Systematic Search. The most known algorithm for performing systematic search for a solution of a constraint problem is the backtracking search algorithm. The algorithm explores the space in a depth-first manner, and at each step it extends a partial solution assigning a value to one more variable. When a variable value is not consistent with the current partial solution (that is, it leads to a dead-end), the algorithm reconsiders one of the previous assignments: *backtracks*. The complexity for the best case is linear to the size of the problem and occurs when a value is assigned to each variable without backtracking. In the worst case, the time complexity of this algorithm is exponential.

In order to improve the search, special attention has been dedicated to the two phases of the algorithm: moving forward to decide the next variable, usually the most constrained (look-ahead methods) and moving backward to a previous assignment, usually the least constraining value (look-back) when a dead-end is found.

The most well-known methods in this category are:

- Forward-checking: a look-ahead scheme that performs arc-consistency at each step, ruling out some values that would lead to a dead-end.
- Backjumping: a look-back scheme that decides how far to backtrack, by analysing the reasons for the current dead-end.
- Conflict-directed backjumping: is an improved variant of the above algorithm, which jumps back to a potential cause of a conflict whenever a variable instantiation has to be undone, not only when a dead-end is detected.
- Constraint learning: a look-back method that centres on the reasons of the dead-end in the form of new constraints so that the same constraint will not arise again.

Local Search. Local Search methods are also used to solve CSPs. These methods try to find a solution by stepping from one complete instantiation to another one in its neighbourhood. This is achieved by changing the value of a few variables at a time. The algorithms differ in their policy of selecting and changing the variables to be altered:

- In a deterministic way: the neighbourhood of the current element consists of all instantiations differing from the current one in at most one variable. The improvement can be maximised, by selecting the value producing the least constraint violations. However, there is the danger of arriving at a *peak* or a *plateau*, so in order to avoid it, the algorithm should be extended with some explicit stopping criterion and mechanism to force a *jump out*.
- Incorporating random elements in the choices: it will follow different paths and terminate with different results.
- Neural networks methods: work on a network with vertices representing all the possible <variable, value> pairs of the CSP to be solved, and the edges have weights. A vertex is either live (on) or dead (off). The live vertices provide input for all their neighbour vertices. Each vertex evaluates the weighted effect of the inputs, and preserves its state or changes it to the opposite. This process keeps repeating until an equilibrium state is reached. If the reached

state is not acceptable, that is, many constraints are violated, then some of the weights assigned to the edges are changed, which may start another sequence of changes.

2.3.2.3 Deduction Rules: Propagators

It transforms the initially given CSP step by step to an equivalent, but smaller problem. The most basic and well-known propagation algorithms are:

- Arc-consistency or 2-consistency: ensures that any value in the domain of a single variable has a legal match in the domain of any other selected variable. The complexity is $O(n^2)$.
- Path-consistency or 3-consistency: if for any path in its constraint graph it holds that if the assignments of the starting and ending variables are consistent, then this can be extended to a consistent partial instantiation by assigning values to the remaining variables along the path. The path-consistency extends the arc-consistency algorithm to a third variable. It is more powerful than arc-consistency, but is complexity is also higher, $O(n^3)$.

These algorithms can be extended to ensure i-consistency, in that case the complexity will be $O(n^i)$.

2.3.2.4 Specialized Heuristics for Classes of Problems

In between search and constraint propagation algorithms, one can identify the structure driven algorithms. These techniques emerged from an attempt to identify constraint problems that are polynominally solvable. They enforce low-level consistency in polynomial time and for some problems guarantee global consistency.

They rely on the fact that binary tree-structured problems can be solved efficiently. The applicability of the methods requires some features of the primal or dual graph of the problem, which can be checked by well-known algorithms of graph theory [62]. Among the methods used, one can mention:

- Decomposition into subproblems: if a CSP can be decomposed into independent subproblems, these subproblems should be solved independently. The complexity of each independent problem is O(n).
- Problems with tree dual graph: the dual problem of any CSP is always a binary CSP. The solution of the dual problem requires that the vertices representing constraints get instantiated in such a way that the *join* along the edges representing shared variables. If the dual problem is a CSP with tree structure then it can be solved easily. There are efficient graph algorithms to decide if the dual graph can be reduced to a join tree. If the original CSP has m constraints, each of them with maximum r solutions, then the dual of the problem can be solved in maximum O(mr) steps (or less, O(mlog r) [51]).

- Decomposition into nonseparable subproblems: the decomposition into nonseparable subproblems method also produces a tree graph, by identifying such subgraphs of the primal graph which, if contracted to a point, the resulting graph is a tree. This method is worth using if the nonseparable components are all small.

- Cycle cutset: is used for binary CSPs. It identifies nodes the removal of which would turn the constraint graph into a cycle-free one. Once the variables in a cycle cutset are instantiated in a consistent way, the remaining tree-structured problem can be solved or unsolvability can be proven without backtracking. The complexity of this algorithm is exponential in the size of the cycle cutset.

2.3.2.5 Specialized Constraint Reasoners

One can distinguish the following CP types depending on the problem one want to solve:

Constraint Optimization Problems (COP). Constraint Satisfaction can be easily extended to optimisation problems [81, 117]. Constraint Optimisation seeks the best solution relative to one or more criteria, usually represented by a cost or an objective function. Frequently, cost functions are specified additively, as a sum of cost components, each defined on subset of variables. The technique most extended is Branch and Bound.

Dynamic Constraint Satisfaction Problem (DCSP). Also sometimes called a conditional CSP problem, is a generalisation of the CSP that is specified by a set of variables, activity flags for the variables, the domains of the variables, and the constraints on the legal variable-value combinations. In a DCSP [13, 172], initially only a subset of the variables is active and the objective is to find assignments for all active variables that are consistent with the constraints among those variables. In addition, the DCSP specification also contains a set of activity constraints. An activity constraint is of the form *if x takes on the value v, then the variables y,z,w ... become active*.

Soft Constraint Satisfaction Problem (SCSP). Many real problems, even when modeled correctly, are often over-constrained. The Soft Constraint Satisfaction Problem [46, 155] allows a faithful modeling of many applications and can accommodate user preferences (Fuzzy-CSP), probabilities (Probabilistic-CSP), costs (Weighted-CSP), and uncertainties within the constraint formalism. One way to specify preferences between solutions is to attach a level of importance to each constraint. There are several frameworks for soft constraints, such as the semi-ring based formalism, where each tuple in each constraint has an associated element taken from a partially ordered set (a semi-ring) and the valued constraint formalism, where each constraint is associated with an element from a totally ordered set.

Temporal Constraint Satisfaction Problem (TCSP). A Temporal CSP [159] is a particular class of CSP where variables represent times and constraints represent sets of allowed temporal relationships between them. Different TCSP are used depending on the representation of the time entity: Time Points (TP), time intervals, durations (distances between TP) and the type of constraints (qualitative, metric or both).

TIMEGRAPH I-II [75] are two temporal reasoning systems that manage large sets of relations in the Point Algebra, in the Interval Algebra and metric information such as absolute time and durations. Temporal relations are represented through graphs, called timegraphs, whose vertices represent points and edges represent temporal relations. The edges are either directed and labelled $\leq$ or $<$ or undirected and labelled $\neq$.

A temporally labelled graph (TL-graph) is a graph with at least one vertex and a set of labelled edges where each edge connects a pair of distinct vertices. Each vertex (TP) in a timegraph has a pseudotime (or rank) associated with it, which is arbitrary except that it respects the ordering relationship between it and other vertices on the same chain.

Temporal distances (durations) between TPs can be specified on each edge of the time-graph through a pair of numbers indicating the upper and lower bounds on the elapsed times between the TP represented by the vertices connected by the edge. TL-graph allows metric information such as absolute times (dates) represented by storing with each vertex 2 six-tuples of the form (year month day hour minute second) corresponding to the earliest and latest time or global optimal bounds on the absolute times of TPs associated with vertices. This is computed by constraint propagation without introduction of new edges. In essence, upper bounds are propagated backward and lower bounds forward. This ensures that each point has the best absolute TP information possible.

A particular case of the TCSP is the **Simple Temporal Problem** (STP) [28, 50]. It has been widely used due to its polynomial time in constraint propagation. It specifies a single interval per constraint since the domain of their variables is convex and coincides with the continuos interval [lb_i, ub_i].

Usually a special variable X_0 is added to represent the origin of the time and its domain is fixed to [0,0]. An STP can be associated with a so called distance graph $G_d(V, E)$. In such a graph the set of vertices V represents the set of variables $\{x_1,...x_n\}$ and the set of edges E represents the set of constraints $a_{ij} \leq X_j - X_i \leq b_{ij}$. Each edge $i \rightarrow j$ is labelled by a weight w_{ij} representing the constraint $X_j - X_i \leq w_{ij}$.

A STP is consistent iff G_d does not contain negative cycles and the minimal network in STP corresponds to the minimal distances in G_d.

2.3.2.6 Constraint Programming in Planning

As an example of a system that solves AI planning problems using constraint programming we can mention CPLAN [179]. It is an approach that performs generalised arc consistency propagation and conflict-directed backjumping. Its CSP model is a purely declarative representation of domain knowledge and is thus independent of any algorithm. It can easily represent incomplete initial states or partial information about intermediate states as well as resource and capacity constraints. The main disadvantage of this approach is that for each new domain, a robust CSP model must be developed.

To solve an instance of a planning problem with a particular initial and goal states, it starts with some lower bound on the length of an optimal plan, generate a CSP model with that many steps in it, instantiate the variables in the initial and goal states and passes the model to the algorithm. The process is repeated incrementing the number of steps until a solution is found or the upper bound on the length of an optimal plan is exceeded. The algorithm uses a dynamic variable ordering that selects as the next variable to instantiate the variable with the smallest domain.

2.4 Integration of Planning and Scheduling

Both fields, as pointed before, have strong and weak points. From one hand we have the planning systems with a rich representation of the problem description, but one of the most important problems that planners have is its way of deal with time and resources. Usually, they use a discrete model of time in which all actions are assumed to be instantaneous and uninterruptible, and also there is not an explicit language that allows to represent issues, such as the basic Allen primitive relations between temporal intervals. About resources there is not a way to reason about cumulative resources.

On the other hand, the scheduling systems can perfectly handle temporal reasoning and resources consumption but there is a lack of expressive language to represent the problems. From this perspective, by combining scheduling and planning systems synergistically these weaknesses could potentially be solved. The planning systems could deal with time and resources thanks to the scheduling search procedure that interleaves refinement solution (assign values to variables) and constraint propagation that computes implications of the assignments to others variables and eliminates inconsistent values, and the planning system can supply to the scheduler the language and the precedence constraints for the activities.

Planning and scheduling system designers [167] have to answer important questions related to:

- The design of methods and algorithms that are able to deal with planning and scheduling as a dynamic problem.
- The definition, the combination, and the optimization of criteria like plan and schedule quality, robustness, and stability.
- The definition and the use of various abstraction levels and of various temporal horizons.

The following section describes the main systems that have addressed the problem of time and resources. Since DEVISER [182], one of the first systems that handled the problem of time used in the Voyager spacecraft, to ASPEN [152], that includes reasoning about time, resources, preferences, and integration of execution during P&S. They show that AI & OR techniques are becoming successful and reliable for complex domains as the satellite or workflow domains are. We have considered two groups: first systems and modern systems.

2.4.1 First Systems that Integrate Planning and Scheduling

We have considered the following systems as first systems that have integrated planning and scheduling:

2.4.1.1 DEVISER

DEVISER [182] was used in the autonomous spacecraft Voyager which photographed Jupiter, Saturn and their satellites in 1979, 1980 and 1981. It is a general purpose automated planner/scheduler implemented on top of NONLIN [178] that generates plans by backward chaining from unordered subgoals.

The program synthesises plans to achieve goals, which may have time restrictions on when sets of goals should be achieved and on how long the goal conditions should be preserved. This system merges planning and deterministic simulation into what might be called goal-directed simulation.

DEVISER adopts the style of a discrete event simulation system, in which all changes can be assumed to occur at specific points in time, rather than continuously as in continuous time simulations. However, it differs in that it is goal directed and generates its activities in reverse time order. For each activity, a duration and a start time *window* are presented. The output is a partially ordered network of activities.

The activities are represented by relational productions, which are related to STRIPS rules. Relational productions have three components (context, antecedent and consequent) consisting of unordered sets of literals. The context consists of literals which are preconditions for the activity, and whose truth is not affected by the activity. The antecedent consists of preconditions which are deleted from the world model when the activity occurs. The consequent consists of literals that are added to the world model when the activity occurs. Changes in the world model occur at the end of the activity. The conflict resolution mechanisms of the core planner ensure that all preconditions are sustained through the duration of an activity. When activities can change immediately, this is modeled by two consecutive activities.

There is an expression attached to the relational production to specify durations, by default in seconds. A window is typically an upper and lower bound on the time when an activity may occur. It may also have an ideal time, indicating an ideal or preferred time of occurrence.

It is defined to be a triple < *Earlier.Start.Time, IDEAL, Latest.Start.Time*> where EST $\leq$ LST and IDEAL is either NIL or a real number such that EST $\leq$ IDEAL $\leq$ LST.

Windows must be explicitly specified for goals. One can specify an optional input to the program that specifies the time when the plan may begin, its default value is zero (TIME_0). All time components must be numeric constants, expressed either in seconds or hh:mm:ss.

It becomes possible to specify goals as more than just a set of conditions to achieve. As planning progresses, the EST and the LST bound may be revised to compress the windows. The windows of nodes which are sequential in the plan network are dependent. Whenever nodes are linked, expanded, or ordered to resolve conflicts, the windows of those nodes must be re-examined and compressed as necessary to preserve these conditions.

The final plan is a partially ordered network in which activities have a consistent start time for goal conditions. The planner backtracks to the last choice when it reaches a dead end.

2.4.1.2 SIPE

SIPE [191] is a domain-independent planning that supports both automatic and interactive generation of hierarchical partially ordered plans. It was the first planner to handle consumable and producible resources. Although no constraints were added to the plan to prevent potential resource conflicts, the search space was pruned to avoid some unuseful exploration of over-consuming plans. It assumes discrete time, states and operators.

In SIPE a plan is a set of partially ordered goals and actions. The system has critics that find potential problems and attempt to avoid them. In particular, most of the reasoning about interactions between parallel actions is done by the critics. The plans are represented in procedural nets.

Invariant properties of the objects are represented in a sort hierarchy, which allows inheritance of properties and the posting of constraints on the values of attributes of these objects. The relationships that change over time are represented in a restricted version of first-order calculus that interacts with the knowledge in the sort hierarchy. Operators allow to post constraints on variables and time need not be represented explicitly for many tasks, since the ordering links in the procedural network provide the necessary ordering information.

SIPE provides different representations (both a frame-based hierarchy and predicate calculus) for different parts of the domain. The system provides representation of domain objects and their invariant properties by nodes linked in a hierarchy. Invariant properties do not change as actions planned by the system are performed. All parts of the system (procedural net nodes, operators or goals) are represented as nodes in the hierarchy. Operators represent actions the system may perform. They contain information about the objects that participate in the actions (represented as resources and arguments of the actions), what actions are attempting to achieve (their goals), the effects of the actions when they are performed and the conditions necessary before the actions can be performed.

SIPE is an extension of STRIPS and follows the closed-world assumption: any negated predicate is true unless the unnegated form of the predicate is explicitly given in the model or in the effects of an action that has been performed. Quantifiers are allowed whenever they can be handled efficiently. Universal quantifiers are always permitted in effects, and over negated predicates in preconditions. Existential quantifiers can occur in the preconditions of operators, but not in the effects. Disjunction is also not allowed in effects. These restriction results from using add lists to solve the frame problem.

It represents knowledge on how to achieve goals, it does not try to achieve preconditions. The preconditions are useful for connecting different levels of detail in the hierarchy. The preconditions of an operator might specify that certain higher-level conditions must be true, while the operator itself specifies goals at a more detailed level.

One of the new ideas provided by SIPE is its ability to construct partial descriptions of unspecified objects. This feature makes the domain representation easier and finding of a solution faster. Constraints can be represented explicitly in the domain (on objects properties or relation between objects) or the system can generate and propagate them during the planning process based on interactions within the plan.

SIPE has specialised knowledge for handling resources. Mechanisms in the planning system as allocate or deallocate resources, automatically check for resource conflicts and ensure that the preconditions will be satisfied. The user does not need to axiomatise in the operator preconditions the availability of resources. It is able to detect conflicts between uninstantiated variables. The system does not allow one branch to use an object that is also a resource in a parallel branch, that is, when an object is listed in an action as a resource, the system then prevents that particular object from being mentioned as either a resource or an argument in any action or goal that is in parallel. However, it is possible to specify shared resources whereby a resource in one branch can be an argument in a parallel branch. If a resource-argument conflict occurs, the SIPE's heuristics orders the branch using the object as a resource before the parallel branch using the same object as an argument.

The search in SIPE is a simple depth-first search to a given depth limit without using knowledge for making choices or backtracking. It uses a constraint satisfaction algorithm to resolve unsatisfied constraints, propagating the effects as soon as the constraints are posted. The algorithm takes advantage of the fact that invariant properties of objects are stored directly in the sort hierarchy. It assumes that it is possible to satisfy any constraint that it cannot prove unsatisfiable.

The system supports interactive planning: it can be guided through problems that would not be solved using automatic search. It also allows to construct partial plans in any level of detail (called islands in [190]) in a plan and then link them together.

SIPE integrates execution and monitoring, understanding the rationale of nodes in the plan. If, at any point, a predicate becomes false when it has thought the opposite, the program will look through the plan and find all the goals that are affected. It can identify what goal nodes are affected or which still serve their purpose.

2.4.1.3 The Open Planner Architecture: O-PLAN

O-PLAN [47] was based on NONLIN [178] and blackboard systems. It is a domain independent, hierarchical tasks network, least commitment architecture that uses heuristic search strategy and keeps many partial plan solutions pawned at each decision point.

The action on plan failure is guided by decision dependency information, recorded at choice points earlier in the plan. The planning will re-start by re-making a choice at an appropriate previous decision point allowing the planning process to continue hopefully to a solution. However, the full design in O-PLAN2 [177] allows for dependency directed search and will at any stage retain only one solution. It integrates planning, scheduling and execution monitoring. It addresses the managing of sharable resources using a criterion based on the optimistic and pessimistic resource profile in order to identify bounds violations.

The main AI planning techniques used in O-PLAN project are:

- It is a hierarchical planning system that can produce plans as partial orders on actions.
- It is an agenda based control architecture in which each control cycle can post pending tasks during plan generation.
- Constraint posting and least commitment on object variables are used.
- Temporal and resource constraint handling has been extended to provide a powerful search space pruning method.
- Repair techniques are used. The O-PLAN temporal constraints propagation algorithms extend those of the DEVISER [182] planner.
- The main date structure in the plan state is a graph of nodes reflecting the activities so far included in the plan, and a set of agendas which hold a list of pending tasks.

The domain representation language in O-PLAN is called Task Formalism (TF). It gives a hierarchical description of a problem specifying the activities needed and how these activities can be expanded into a set of sub-activities with ordering constraints imposed and time windows or resource restrictions. This description of actions is represented in TF by the schema. There are four types of schemas:

- Normal schemas: are actions under the control of the planner. Events or external actions are out of the control of the planner.
- Process schemas: describe an event or set of events.
- Task schema: is a task the planner is being asked to perform.
- Meta-schemas: groups of any of the schemas mentioned above.

Schemas also specify the initial conditions, or state of the world, in the application domain. They are compiled into an internal data structure, which holds information about the way in which information on the requirements of the use of the schema and details the way in which it expands the activity network when used.

TF allows to represent condition types. They allow to keep information about when, how and why a condition has been satisfied and the way to treat them if they cannot be maintained. They help the planner to prune some part of the search space, so bad written type conditions can have negative results.

In the TF, resource usage can be expressed as the overall limitation in the schema or utilisation at points in the expansion of the schema. They can be classified as consumable, producible by an agent or a process or reusable (it also distinguish between non-sharable, sharable independently and sharable synchronously).

At each level, the plan is represented as a network of nodes in a form that allows the use of knowledge about the problem to restrict the search for a solution.

An agenda based architecture has been used as the central feature of the system and the design approach. The main components are:

- Domain information supplied by the TF.
- Plan state: the emerging plan to carry out the identified tasks. It is the dynamic data structure used during planning and builds the emerging plan.
- Knowledge Sources (KS): the processing capabilities of the agent (plan modification operators).
- Support modules: functions that support the processing capabilities of the agent and its components.
- Controller: the decision-maker on the order in which processing is done. It is intended to provide complete information about the question asked of them to the decision making level itself. Some support modules act as constraint managers for a sub-set of the plan state information.

The way O-PLAN works is as follows:

1. An event is received by the Event Manager. This has two guards: one from the task assigner (left input channel) and the other form the execution system (right input channel). The guards verify, and, if necessary, reject events that are not relevant to the system. They use knowledge of the system derived from the Knowledge Source and domain model (TF).
2. If the event is approved by the guard, then it is passed to the Controller/Agenda Manager (AM) which assigns it the necessary triggers and KS activation entry.
3. When triggered, the Trigger Detection informs the Agenda Manager and can cache a copy of the triggered agenda entry in the Agenda Manager. The order of entries in the Agenda is constantly updated as new agendas entries are added or as triggers on waiting agenda entries become invalid.
4. The Controller/AM assigns an available KS Platform (KP) that can run pre-nominated KS on the triggered agenda entry.
5. When a KP has been allocated, if it does not already contain the nominated KS, the Platform can request the body of the KS from the database Manager in order to process the agenda entry.
6. A protocol and an access key are used to control communication between the Controller/AM and a KS running on a Platform. A KS can terminate with none, one or multiple alternative results through interaction with the Controller via the protocol.

2.4.2 Modern Approaches that Integrate Planning and Scheduling

Among the modern system for integrating planning and scheduling, we can mention:

2.4.2.1 Heuristic Scheduling Testbed System

HSTS [122] is an integrated planning and scheduling architecture that has been applied to the problem of generating observation schedules for the Hubble Space Telescope. It addresses the problem of managing sharable resources and goal expansion as complementary aspects of a more general process: the construction of the behaviour of a dynamical system that is consistent with stated goals and constraints.

HSTS decomposes a domain into a vector of state variables continuously evolving over time. Similarly to planning, it provides general devices for representing complex states and causal justifications. The fundamental structuring principle of the input and state description is a finite dimensional vector of values evolving over continuos time. In this representation, the range of vector component values can be wider than binary, allowing the inclusion of more complex state information into the representation of resources and easily represent physic components as resources. The input vector identifies those system properties directly controlled by an external agent and the state vector refers to those that can only be influenced indirectly. It plans using dynamic and temporal CSP.

When the planner commits to an action, new nodes are added to the CSP, constraints are also added between TP to specify duration and to force preconditions and effects constraints. Since the result is an STP, arc-consistency is enough to guarantee consistency. HSTS does not perform any reachiability or mutual exclusion, it must commit to an action or event before any constraint reasoning.

They have developed a heuristic scheduling methodology called Conflict Partition Scheduling (CPS) [120], that operates on a temporally flexible network of constraints under the guidance of statistical estimates of the network's properties.

Initially, the temporal database contains the token network for the abstraction level. None of the tokens are inserted into a state variable time line. The way HSTS works is as follows:

1. Goal selection: it selects some goal tokens.
2. Goal insertion: inserts each token into the state variable time line.
3. Compatibility selection: selects an open compatibility for an inserted token.
4. Compatibility implementation: implements the selected compatibility, that is, it finds a behaviour segment on the time line compatible with the conditions on type and time imposed.

At each step, decision making is needed to choose between alternatives. Heuristics designed for the Hubble Space Telescope are applied in both levels of abstraction.

The two core components of HSTS are:

- Domain Description Language (HSTS-DDL).
- Temporal Date Base (HSTS-TDB): supports the static and dynamic structure of a system thanks to the extension of the time map formalism by connecting the state of the Data Base and the model of a system.

To ensure the satisfaction of the conditions, HSTS posts temporal and type constraints among pairs of tokens. This is done within the token network. To support the satisfaction of constraints intrinsic to the domain, HSTS-TDB automatically associates to each value token an instance of its type's compatibility specification tree. When a compatibility is satisfied, it posts a temporal relation between two tokens and marks as achieved the leaves in the causal justification tree.

HSTS-TDB maintains token network consistency through auxiliary constraints networks. Temporal constraints are organised in a graph, the time point network. Each token start or end constitutes a node in the graph, or TPs; arcs between TPs are metric interval distances derived from the temporal relations posted in the database.

It can support a single capacity or multiple capacity resource (represented by a primitive aggregate state variable), single user or multiple user resource and renewable or not renewable. This is done using compatibilities and duration constraints.

2.4.2.2 IXTET

IXTET [2, 76] is a least commitment planner that uses a graph-based algorithm for detecting resource conflicts between parallel actions. The time logic relies on a restricted interval algebra represented by TPs. TPs are seen as symbolic variables where temporal constraints can be posted. There are two types of variables:

- Temporal variables, handling temporal symbolic constraints (after, before, simultaneity and their disjunction) and numeric constraints expressed by lower and upper bounds on the temporal bounds on the temporal distance between two points [l,u].
- Atemporal variables, ranging over finite domains are managed in a variable constraint network. Constraints are inequalities between variables, propagated by a local path consistency algorithm.

The planning system looks for a solution plan in a space of partial plans. Given a problem and a set of tasks, the system generates a solution by successive refinements of the initial problem. A partial plan is composed of a set of temporal propositions (events, asserts, or resource usage), a temporal map (for the management of the temporal constraints post in each instant) and a table of variables (for the management of constraints post on the plan variables).

Three are the modules in charge of the defaults analysis in partial plans:

- The analysis module of non-satisfied goals.
- The analysis module of threats.

- The analysis module of resources in charge of conflict detection of potential resources.

These modules are managed by a decision or control module that chooses, in every node of the search tree, the inconsistencies to resolve following a least commitment strategy.

The world is described by a set of multi-valued domain attributes temporally qualified into instantaneous events and persistent assertions and a set of resource attributes, where each attribute is a functional term over atemporal variables. IXTET relies on the closed world hypothesis. The initial plan P_{init} describes the problem scenario, that is:

1. Initial values for all instantiated attributes.
2. Expected availability profile of resources.
3. Goals that must be achieved.
4. The expected changes on some state attributes that will not be controlled by the planner.

A complete information on the value of attributes at the initial time point is assumed. Expected contingent events are inserted into P_{init}. They must be consistent, i.e., events on the same domain attribute should be totally ordered with values compatible with this order. The user has the possibility of describing a dynamic domain with contingent events.

Multi-valued attributes belong to the following ontology:

- Rigid attributes: the value does not change over time, they express a structural relationship between their arguments. *Is-key-of(key)* $\in$ {*door1, door2...*}
- Flexible attributes (or fluents) the value may change, they can be:
  - Controllable: the change of the value can be planned for or can also change independently of the planning system: *position(obj)* $\in$ {*room1, room2...*}
  - Contingent: the change is not controllable by the planner. *Sun-light(loc)* $\in$ {*day,night*}

Resources are gathered together into resource types. The type of predicates that manages resources in IXTET [100] are:

- Unsharables resources: are single items with a unit capacity (a key).
- Sharable resources: can be used simultaneously by different actions, not exceeding a total maximal capacity (that is, aggregate resources).

Resource availability profiles and the use of resources are described by the following predicates:

- *Consume(typ(r):q,t)*: consumes a quantity *q* of a resource *r* at time *t*. The availability of *r* will decrease at *t*.
- *Produce(typ(r):q,t)*: produce a quantity *q* of a resource *r* at time *t*.

Planning operators (hierarchical operators) in IxTeT are called tasks. Preconditions and effects do not appear explicitly separated within a task, since the same event predicate can express its two values.

The operators are composed of a set of subtaks, set of events describing the changes of the world induced by the task, set of assertions to express conditions that should prevail during part of the task, set of resource uses and a set of constraints binding the different time-points and atemporal variables in the task.

The possibility of using subtasks within tasks makes the description hierarchical. It does not allow recursion or other complex control structure into a task: the planner itself is not hierarchical. The types of predicates available are:

- *event(att(x_1 , ...) : (v_1, v_2), t)* says that attribute att(x_1,..) changes its value instantaneously from v_1 to v_2 at time t.
- *hold(att1(x_1, ...) : v, (t_1, t_2)* asserts the persistence of the attribute att(x1,...) to a value v for t: $t_1 \leq t < t_2$. It is used in action models but also by the planner to manage causal links.

To maintain the consistency of the time-map, IxTeT has implemented two separate networks for symbolic and numeric constraints. All variables are universally quantified; conjunction of such literals are called indexed time tables. In such a table, temporal constraints are gathered into the time-map (temporal constraint network), instantiation constraints and rigid attributes into a variable binding network, where equalities constraints are propagated by transitive closure and the rest by arc-consistency. The time manager propagates the constraints to ensure the global consistency of the network. Management of atemporal variables is achieved through a variable constraint manager. Constraint propagation on atemporal variables is achieved through classical CSP techniques.

Resource conflicts are detected and solved through Minimal Critical Sets (MCS) on a specific representation of the temporal constraints: the possible intersection graph (PIG). The PIG associated to a resource type on a partial plan P is a non-oriented graph G(U, E) whose vertices U are the resource propositions $\{u_1,...u_n\}$ for resource type, and edges E the pairs $\{u_i, u_j\}$ such that u_i and u_j may possibly intersect in some temporal instantiation of P. To solve a MCS it is sufficient to post one of the constraints.

The resource analysis module is integrated into the planning system. At each node of the global search tree, the resource analysis module selects a certain number of smallest MCS and computes their minimal resolvers, whereas the other analysis modules focus on threats and pending subgoals.

The planning system explores a search tree of partial plans whose root is the initial plan and branches represent some tasks or constraints inserted on the current plan in order to solve one of its flaws. Starting from the initial plan the search of a solution consists in solving incrementally the conflicts and subgoals, by adding tasks, temporal constraints or instantiation constraints. At each step of the search, the consistency of the partial plan is checked. If it is not consistent the planner backtracks according to a best-first strategy. The search stops when a solution plan is reached.

A partial plan P is a solution when all goals and sub-goals are satisfied, and there are no causal links or resource conflicts. The procedure which checks the consistency of a partial plan runs in polynomial time $O(n^3)$. However, it is not complete since the variable binding network is checked by arc-consistency. Indeed, a complete constraint propagation in this network could be more costly than the planning search, which precisely reduces the complexity of this propagation by adding equality constraints (through establishments). A complete propagation is performed at the end, in order to ensure that the found plan is really a solution.

There are two non-deterministic steps in the IXTET control procedure:

- The choice of the next conflict or subgoal to be solved. When all conflicts and subgoals have several possible resolvers, the set of resolvers for each conflict or subgoal is evaluated by a function *K* that can be maximised or minimised.

- The choice of a resolver. The choice of the subgoal with the minimal value of the function *K* corresponds to a resolver with the higher difference of cost between it and the next best.

Variable constraints as well as causal link commitment are estimated through functions since the explicit representation of temporal metric constraints and finite domain variables permits an estimation of the commitment induced by a constraint posting.

2.4.2.3 The New Millenium Remote Agent

The New Millenium Remote Agent (NMRA) [121] was the first time an AI agent controlled for a six days period a NASA spacecraft: the Deep Space One (DS-1). NMRA is a hierarchical reactive system partially based on SAT compilation that integrates real-time monitoring and control with constraints based-planning and scheduling, robust multi-threat execution and model-based diagnosis. The planner searches in the space of partial plans and is concerned with activities at a high level of abstraction that encapsulates a detail sequence of executive level commands. Time and resources are allocated in the activities using an interval time-based representation.

For planning purposes, it does not matter if a time interval belongs to an activity or a state but it does for execution, so a different representation is used for execution and planning. Heterogeneous representations, although present some advantages, duplicate knowledge, and it can make the models to diverge rather than to converge. In order to overcome some of these drawbacks, the unified modeling of the Livingstone architecture [192] has been used, together with code generation techniques to translate some model properties into the different representations used for each computational engine. Another feature is that plans are generated on-board instead of on-ground.

The domain model contains an explicit declaration of the spacecraft subsystems on which an activity or a state will occur. It is a propositional encoding quite similar to the STRIPS domains. In the temporal database (based on the HSTS-TDB [120]) each subsystem has an associated timeline on which the planner inserts activities and states and solves resource allocation conflicts. It also contains the declaration of duration constraints and of templates of temporal constraints between activities and states. The temporal database provides constraints propagation services to verify the global consistency of the constraints posted.

The planning model is a set of compatibilities that must be satisfied in every complete plan. A compatibility is a temporal relation that must hold between a master token and a target token whenever the master token appears in the plan. A master token can have several compatibilities expressed as Boolean expressions of compatibilities called compatibility trees. The initial state of the plan is the state of the spacecraft at the time the plan is executed. As output, it generates a plan that achieves the goals when executed from the initial state.

The high level goals are received from ground as a part of a pre-loaded mission profile or generated on board. In the case conflicts can arise, two strategies are proposed: prioritise the goals or ensure that the goals are flexible enough that there is always a way to satisfy all of them.

The decisions are made by two layers (the reactive and deliberative layer) and they consist of three modules of a domain-independent engine that reasons about and acts upon domain-independent knowledge contained in a model:

- The Executive (EXEC): the basic execution units are tokens and timelines. Each activity is represented by a token, though not every token is an executable activity. A timeline is a sequence of tokens that specifies the evolution of states over time. The plan specifies start and end time windows for each token, and temporal constraints among the tokens. The EXEC guarantees that each activity in the plan will be executed within the temporal constraints specified in the plan. The executive invokes the planner and the Mode Identification components to help it to perform its functions. In case of plan failure, the executive knows how to enter a stable state (standby mode) and translate it into a language understandable by the planner. The basic runtime loop of the executive is as follows: the system maintains an agenda on which all tasks are stored. Tasks are either active or sleeping. On each pass through the loop, it checks any change in the external world that may affect it and responses to these events by updating its internal model, installing new tasks in the agenda or changing affected tasks status. After it selects new tasks, it updates the agenda and the cycle repeats.

- The Planner/Scheduler (PS): given a set of high-level goals from the ground, the PS generates an ordered set of actions that achieves the goals while obeying resources and time constraints. It is a heuristic search engine operating on a temporal database. The search engine posts constraints based on external goals or constraints templates stored in a model of the spacecraft. Using iterative sampling it tries to improve quality metrics.

- The Mode Identification and Reconfiguration (MIR) is called when the commanded state differs from the observed state. It uses its model of the device to infer the most likely failure mode using a conflict directed best-first search. Then, it suggests a repair activity to the executive. If the failure cannot be solved EXEC aborts, and a new plan is requested to the PS. The MIR extends the basic ideas of model-based diagnosis by modeling each component as a finite state machine, and the whole spacecraft as a set of concurrent synchronous state machines. For modeling the behaviour of each component, it uses abstracts or qualitative models encoded as propositional clauses.

For testing the autonomous software [166], a scenario-based verification augmented with model-based verification and validation is followed. The universe of possible inputs is partitioned into a manageable number of scenarios. The RA is probed on each scenario and its behaviour verified against the specifications.

2.4.2.4 The Automated Scheduling and Planning ENvironment

The Automated Scheduling and Planning ENvironment (ASPEN) is an object oriented system, based on AI techniques. It is a modular, reconfigurable application framework that is capable of supporting a variety of planning and scheduling applications. It allows that the Earth Orbiting (EO-1) spacecraft can be controllable by a small on-ground team or to supply manual and automatic scheduling capabilities for the Citizen Explorer (CX-1) and for the 2^{nd} Antarctic Mapping Mission (AMM-2). It has been used for automatic commands generation and planning on-board a rover and it is being evaluated for operational use in the rover mission Mars01 Marie Curie.

The main features of ASPEN [152] are:

- A hierarchical representation of activities.
- Iterative search approach to perform planning, scheduling and optimisation.
- Heuristics to prune the search to find solutions in less time or of higher quality.
- A local algorithm to try many plan modifications, although it may not guarantee that it will find a solution.
- An early-commitment strategy for parameter values.

The spacecraft models are represented in AML (ASPEN Modeling Language). It is a descriptive language that it can model a physical spacecraft system directly: activities, resources, states and any constraints among them. These models are translated into data structures that provide efficient reasoning capabilities for planning and scheduling. The central data structure is the activity. It represents an action or step in a plan/schedule. An activity has a duration, start and end times and resources. The language also allows to represent state variables and constraints between activities.

The current model of resource usage is discrete. There are four types of resources in the language: atomic (only used by an activity), concurrent (used by an activity and only available before it is reserved), non-depletable (used by more than one activity and no need to be replenished) and depletable (the capacity decreases after use).

The main algorithm used in ASPEN is based on iterative repair. During iterative repair, the conflicts are detected and addressed one by one until no more conflicts exits or the time introduced by the user has expired. To resolve the conflict, they have implemented several repair methods as moving, adding, deleting, abstracting an activity, disconnecting constraints, changing parameters value, etc. (to know more about this technique, one can refer to [152]).

Another feature in ASPEN is its ability of considering plan quality. For that, it uses a local early-commitment approach. They define preferences for variables in the plan to define variables relevant to the plan quality. Preferences can be made in the number of activities, goals or resources and in the duration of a state or state variable. Each preference includes a weight for specifying the relative importance to the global plan. The score of a plan is computed as the weighted average of scores for plan variables with preferences. Aggregate preferences (many plan variables) are also allowed. The plan is scored for each variable independently (in that case they are all weighted equally and averaged) or applying a function (as minimum, sum, maximum or average) to all the variables.

The Continuous Activity Scheduling Planning Execution and Replanning system (CASPER) [36] was designed to help ASPEN in updating the plan when necessary during execution, that is, a continuous planning approach. The main difference with traditional planning relates to the initial state and goals. In this approach, instead of waiting until something does not go as expected and replan, at anytime there is an incremental update of the current state, goals or planning horizon. The planner cycle is as follows:

- New goals and initial states are updated.
- Propagation of the changes and conflicts through the plan projection.
- The repair methods are invoked in order to resolve conflicts and find a solution to the new situation.

As a result, the plan is created by using incremental replanning from the portion of the old plan. This approach provides fast replanning during plan execution.

2.5 Summary

Intelligent planning comprises the representation and solution of planning and scheduling problems. Traditionally, both areas have evolved separately of each other but nowadays applications require more communication between them.

By scheduling, we mean the problem of assigning resources and times for activities, obeying the temporal restrictions of activities and the capacity limitations of shared resources. By planning, we mean the problem of selecting and ordered set of activities such that they achieve one or more goals and satisfy a set of domain constraints. The main techniques in both areas have been described.

Total Orders planners have the disadvantage of generating a plan that is a sequence of actions, so in real domains when minimising time and resource usage is important, they do not provide good solutions, although in the last five years efficient and fast algorithms have been created. This disadvantage is surpassed by Partial Order planners, graph-based planners and SAT-planners. The first ones, had been abandoned due to their slowness compared to the current approaches, although it has been recently revived by the work of Nguyen and Kambhampati [126] (most of the integration architectures are based on this approach).

Graph-based planners need to search for plans at every encoding length until a solution is found what leads to large spaces and time requirements and in some cases intractability in real problems when time and resources get involved.

SAT-engines present a huge formulae instantiation (there might be huge redundancy). This is specially important with time and resource reasoning. Also it can be difficult to develop automatic STRIPS translations (getting worse for real domains) but they have the advantage of saving time by avoiding variable binding during search.

In the HTN approach, the task networks allow to plan easily in real domains, but they have as a disadvantage the effort to code the domain decompositions in the formalism.

We have also summarised the more representative systems (classical and modern) that have faced the problem of planning and scheduling integration. The classical systems share the problem of being restrictive and having poor modeling languages. In some cases, they need a lot of coding effort and in other cases they are slow. The modern approaches vary from systems oriented towards the physic nature of the domain (HSTS), followed by a more planning oriented approach (IxTeT and O-PLAN), or the more integrated *hierarchical* approach for planning and scheduling in real domains (ASPEN). One of the inconvenients of these modern systems is that none of them are free distributions.

Languages to easily model the problems and with a rich representation are needed in both fields. We have revised the main languages used to represent problems focused on planning needs as STRIPS, ADL, or the most recent version of PDDL, PDDL2.1, and some other languages focused on scheduling such as OZONE or DDL. But what it is still missing in these modeling languages is the possibility of representing resources explicitly or imposing a time horizon (in the case of planning languages) or an easier way to represent activity constraints (in the case of scheduling languages). Also, languages that allow non expert users to model real domains as satellites or workflow domains where time and resources are still missing.

For these reasons, we want to explore the possibility of using existing rich modeling user oriented tools for planning and scheduling systems that allow to represent real domains. This type of tools need that planning and scheduling are integrated: we will use for this task specific techniques of planning and scheduling studied in this chapter. However our objective is not only to present an integration model and an richer language, but also to present a general integration model that can be extended to most of the state of the art planners.

Chapter 3

Basic Ingredients

Because one of our goals in this work is to try to integrate planning and scheduling, our start point will be explain in detail the features and the functionality of two existing problem solvers: the PRODIGY planner and its extension, QPRODIGY, and the O-OSCAR scheduler.

3.1 The planner: PRODIGY and QPRODIGY

PRODIGY [180] is an integrated architecture that has been used in a wide variety of domains. The problem solver is a non-linear planner that uses a backward chaining means-ends analysis search procedure with full subgoal interleaving [22]. The planning process starts from the goals and adds operators to the plan until all goals are satisfied. Although in the planner there is not a language that allows an explicit representation of resources or temporal information, it is able to handle some scheduling reasoning thanks to its capability to represent infinite types (numeric variables) which are needed to represent information about time and resources; define functions that obtain variables values in preconditions of operators so that values can be constrained; and use of control rules to prune the search which allows to make the overall process efficient. PRODIGY is implemented in Common Lisp.

QPRODIGY is a version of the planner that is able to reason about different quality metrics, allowing to declaratively define these quality metrics in the operators, and then performing a branch-and-bound search for *good* solutions [20]. This search can be enhanced by using pre-defined or learned quality-based control knowledge.

3.1.1 Inputs and Outputs

The inputs of the planner are:

1. The domain theory (D) that contains all the actions represented by the operators. The PRODIGY domain theory is based on the STRIPS representation originally proposed by Fikes and Nilsson [63]. Since this representation is quite restrictive, it has been extended to allow disjunctive preconditions, conditional effects and universally-quantified preconditions and effects [22]. The

language also allows two useful features: variable values can be constrained using any user-defined function; and different quality metrics can be defined in each operator (only for QPRODIGY). Figure 3.1 shows an example of a QPRODIGY operator with *time* as the only quality metric defined, whose value depends on the agent that will perform the activity. It belongs to the ROBOCARE Time domain (go to Appendix D to see this domain coded in PDDL2.1 syntax). The ROBOCARE domain [33] is a multi-agent system which generates user services for human assistance. The system is to be implemented on a distributed and heterogeneous platform, consisting of a hardware and software prototype. It intends to give a user service in a closed environment such as a health-care institution or a domestic environment. Basically, the domain consists of robots that have to perform tasks as clear beds, serve meals, make beds or move people from one room to another.[1] The operator in the figure has as preconditions that the bed is unmade and clear and the agent in charge of making the bed must be in the same room where the bed is, and as effect the bed will be made. Also, QPRODIGY allows to represent different cost metrics. In the example, we have defined the *time* metric as the duration that the agent takes to perform the activity.

```
(OPERATOR MAKE_BED
 (params <a0> <b0> <r0>)
  (preconds
   ((<a0> AGENT)
    (<b0> BED)
    (<r0> ROOM))
   (and (unmade <b0>)
        (clear <b0>)
        (is_in_bed_room <b0> <r0>)
        (is_in_agent_room <a0> <r0>)))
  (effects ()
    ((del (unmade <b0>))
     (add (made <b0>))))
  (costs ((<duration> (and DURATION
                         (cost-from-pred
                              (make_bed <a0> <duration>)))))
         ((TIME <duration>))))
```

Figure 3.1: A ROBOCARE operator example in QPRODIGY.

With respect to types, in PRODIGY/QPRODIGY types can be finite (structured in a hierarchy as in PDDL2.1 [68]) or infinite (corresponding to continuous variables, such as time, or fuel). Since variables cannot be given infinite possible values, values for those infinite-typed variables should be constrained using functions that should return a list of possible values at planning time. To do so, they are allowed to inspect the current meta-state of the search. Therefore, one

[1]This domain will be used in Chapter 8 for experimental evaluation.

can handle continuous variables within operators, but at instantiation time, those variables will only be potentially given a finite set of alternatives. In Figure 3.1 the function that generates all the values that the numeric variable *<duration>* can have is the QPRODIGY *cost-from-pred* function.

2. The second input to the planner is the problem, described in terms of an initial state ($\mathcal{S}$) and a set of goals ($\mathcal{G}$) to be achieved. Figure 3.2 shows an example of a problem in the ROBOCARE domain. This problem has two goals: to make two beds, using for this task any of the eight agents available.

```
(setf (current-problem)
 (create-problem (name Example)
  (objects (person0 person1 person2 person)
   (agent0 agent1 agent2 agent3 agent4 agent5 agent6 agent7 agent)
   (room0 room1 room)
   (bed0 bed1 bed2 bed3 bed4 bed))
  (state
   (and (is_door room0 room1) (is_door room1 room0)
    (is_in_person_room person0 room0)
    (needs_to_walk person0)
    (is_in_person_room person1 room0)
    (needs_to_walk person1)
    (is_in_person_room person2 room0)
    (needs_to_walk person2)
    (is_in_agent_room agent0 room0)
    (is_in_agent_room agent1 room0)
    (is_in_agent_room agent2 room0)
    (is_in_agent_room agent3 room0)
    (is_in_agent_room agent4 room0)
    (is_in_agent_room agent5 room1)
    (is_in_agent_room agent6 room0)
    (is_in_agent_room agent7 room1)
    (is_in_bed_room bed0 room1)
    (unmade bed0) (clear bed0)
    (is_in_bed_room bed1 room0)
    (unmade bed1) (not-clear bed1)
    (is_in_bed_room bed2 room1)
    (unmade bed2) (clear bed2)
    (is_in_bed_room bed3 room0)
    (unmade bed3) (not-clear bed3)
    (is_in_bed_room bed4 room0)
    (unmade bed4) (not-clear bed4)))
  (goal (and (made bed2)
             (made bed1)))))
```

Figure 3.2: A ROBOCARE problem example in PRODIGY/QPRODIGY.

3. When there is more than one decision to be made at decision points, the third input to the planner, the control knowledge (declaratively expressed as con-

trol rules, $\mathcal{C}$) could guide the problem solver to the right alternatives of the search tree potentially avoiding backtracking. There are three types of rules: selection, preference or rejection. One can use them to choose an operator, an instantiation of an operator (binding), a goal, or deciding whether to apply an operator or continue sub-goaling. Figure 3.3 shows one of the control rules coded by a PRODIGY expert for this particular domain. If the agent needs to be in a room, but it is on another room that is connected by a door to a third room that is also connected to the room that the agent needs to be, for that case the control rule decides to select the operator *GOTO_ROOM* (to learn more about the PRODIGY and QPRODIGY representation language and their similarities/differences with respect to PDDL2.1, go to Appendix F).

```
(control-rule SELECT-GOTO_ROOM
  (if (and (current-goal (is_in_agent_room <agent-1> <room-2>))
           (true-in-state (is_in_agent_room <agent-1> <room-5>))
           (true-in-state (is_door <room-5> <room-4>))
           (true-in-state (is_door <room-4> <room-2>))
           (some-candidate-goals nil)
           (type-of-object <room-5> room)
           (type-of-object <room-4> room)
           (type-of-object <room-2> room)
           (type-of-object <agent-1> agent)))
  (then select operators goto_room))
```

Figure 3.3: A ROBOCARE control rule to choose an operator.

As a result, PRODIGY/QPRODIGY generates a plan with the sequence of operators that achieves a state (from the initial state) that satisfies the goals, that is, a Total Order Plan $\mathcal{P}$ is generated. Figure 3.4 shows the inputs and output of PRODIGY. The obtained plan, if we use PRODIGY, does not consider any optimisation with respect to resource use, availability or time. The only quality criteria is the length of the solution. If we want a plan according to some quality criteria, besides the length solution, QPRODIGY can reason about optimality as described in [20].

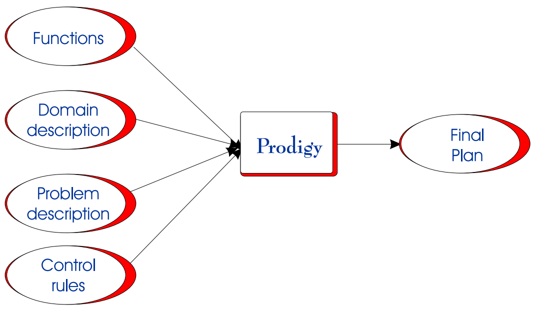


Figure 3.4: PRODIGY/QPRODIGY inputs and output.

The solution given by QPRODIGY to the problem shown in Figure 3.2 is depicted in Figure 3.5.

```
1 (make-bed agent7 bed2 room1)
2 (clear-bed agent6 bed1 room0)
3 (make-bed agent6 bed1 room0)
```

Figure 3.5: Plan solution to the problem of Figure 3.2.

3.1.2 PRODIGY algorithm

PRODIGY is an integrated non-linear, domain-independent planner and learning system. It uses means-ends analysis to reason about multiple goals and multiple alternative plans achieving the goals. The dynamic goal selection enables the planner to interleave plans and exploit common subgoals [180].

PRODIGY explores the search space building a node tree where each node belongs to a decision point type. The tree of Figure 3.6 shows the relationship among the four types of nodes in the search tree. The algorithm involves the following decision points as Figure 3.7 sketches:

1. The *goal* to select from the set of pending goals and subgoals.
2. The *operator* to choose to achieve a particular goal.
3. The *bindings* to choose to instantiate the chosen operator.
4. To *apply* an operator whose preconditions are satisfied or continue *subgoaling* on an unsolved goal.

The algorithm starts selecting a goal from the set of goals in the problem. Once a goal is selected, it has to choose an operator whose preconditions match the goal selected. Then, it binds all the possible variables in the operator. If the operator can be applied, PRODIGY can decide to apply it, in that case it will be part of the solution plan, otherwise it can pay attention to another goal still not achieved. This process repeats until it finds a solution, or returns no solution if the time given to solve the problem has expired, the maximum number of nodes has exceeded or the search tree has been exhausted.

Another way to see PRODIGY's planning algorithm is by considering two searches that are constantly interchanging information. On one side the *backward-chaining planning search* and on the other hand the *simulator search* of the plan execution. The execution-simulator search calls the backward-chainer search to select operators relevant to the goal and simulates the application of these operators. The execution-simulator is managed by a Total Order head-plan and a tree-structured tail-plan. The root of the tail-plan's tree is the goal statement, the nodes derived from the root

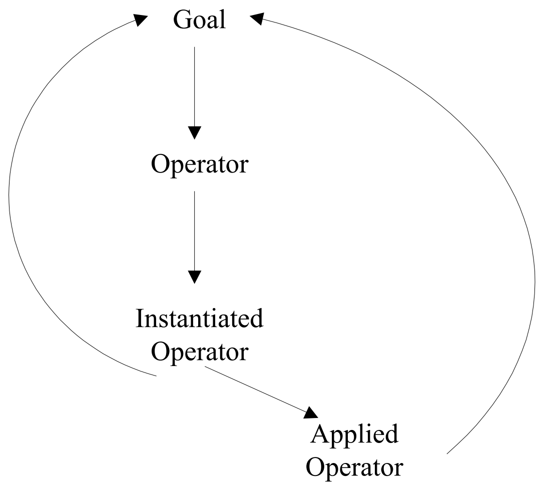


Figure 3.6: Relationship among the nodes in the search tree.

are operators, and the edges are ordering constraints. The head-plan is a sequence of completely instantiated operators that can be executed from the initial state.

Given an initial state and a goal statement, PRODIGY starts with the empty plan and modifies it, step by step, until a correct solution plan is found. The empty plan is the root node in PRODIGY's search space. At this point, the head and tail of this plan are empty, and the current state is the same as the initial state.

At each step, PRODIGY can modify the current incomplete plan in one of two ways. First, it can add an operator to the tail-plan. Modifications of the tail are handled by the backward-chaining planning algorithm. Second, it can move an operator from the tail to the head plan. This is called the *application* of an operator and changes the initial state. The backward-chainer views the current state after applying the operator as the initial state.

PRODIGY allows the user to code functions to handle different situations during planning and calls them as often as once every time a new node is generated in the search space. These functions are called handlers. In order to allow QPRODIGY consider quality measures two handlers have been implemented:

1. *cost-handler-applied* is called every time an operator is applied except in the last operator. The handler calculates the cost of applying an operator, given that depending on the defined metric, the operator could have different costs (by default its value is 1). If a cost bound is defined and in the actual path this value is surpassed, the search path is abandoned, the problem solver backtracks to the last node with some alternatives left, and tries another one.

2. *finish-handler* calculates the cost of the last applied operator with the same functionality as the *cost-handler-applied*. It is used when more than one solution is calculated.

```
Function Prodigy4.0 ($\mathcal{D}, \mathcal{C}, \mathcal{S}, \mathcal{G}$)

$\mathcal{S}$ the state of the problem
$\mathcal{G}$ the set of goals to be achieved, also called pending goals
$\mathcal{D}$ the domain description: operators and objects hierarchy
$\mathcal{C}$ the set of control rules (control knowledge)
$\mathcal{P}$ the plan, initially empty
$O$ an instantiated operator (from the set of operators in $\mathcal{D}$)
$\mathcal{B}$ a substitution (bindings) of the variables of an operator
$O_B$ the operator $O$ instantiated with bindings $B$
$\mathcal{O}$ the set of chosen instantiated operators not yet in $\mathcal{P}$, initially empty
$\mathcal{A}$ the set of applicable operators (a subset of $O_B$)
$\mathcal{G}_o$ the set of preconditions of all operators in $\mathcal{O}$
ST the planning search tree

While $\mathcal{G} \not\subseteq \mathcal{S}$ AND search tree and planning resources (time or nodes) not exhausted do
1.      $\mathcal{G} = \mathcal{G}_o - \mathcal{S}$
2.      $\mathcal{A} \leftarrow \emptyset$
3.      Forall $O_B \in \mathcal{O} \mid$ preconditions($O_B$) $\subseteq \mathcal{S}$ do $\mathcal{A} \leftarrow \mathcal{A} \cup \{O_B\}$
4.      If select-subgoal-or-apply ($\mathcal{G}, \mathcal{A}, \mathcal{S}, \mathcal{C}$) = subgoal
5.      Then $\mathcal{G} \leftarrow$ select-goal($\mathcal{G}, \mathcal{S}, \mathcal{C}$)
6.           $O \leftarrow$ select-relevant-operator ($\mathcal{G}$,$\mathcal{C}$)
7.           $\mathcal{B} \leftarrow$ select-bindings($O, \mathcal{S}, \mathcal{C}$)
8.           $\mathcal{O} \leftarrow \mathcal{O} \cup \{O_B\}$
9.      Else $O_B \leftarrow$ select-applicable-operator ($\mathcal{A}, \mathcal{S}, \mathcal{C}$)
10.          $\mathcal{S} \leftarrow$ apply ($O_B, \mathcal{S}$)
11.          $\mathcal{O} \leftarrow \mathcal{O} - O_B$
12.          $\mathcal{P} \leftarrow$ enqueue-at-end ($\mathcal{P}, O_B$)
13.     If there is a reason to suspend the current search path Then Backtrack
Return plan $\mathcal{P}$, ST and measures (total time, expanded nodes and quality)
```

Figure 3.7: PRODIGY4.0 algorithm.

3.2 The scheduler: O-OSCAR

O-OSCAR is a scheduler that follows a Constraint Satisfaction approach [119] to solve complex multi-capacity temporal problems [27, 32]. It is implemented in C++ under Windows. The release used in this work is able to solve project scheduling problems with time windows by using the ISES algorithm [30]. Its basic solving method uses a two layered problem representation:

1. The ground layer, or Ground-CSP, that only represents the temporal aspects of the problem in the form of a quantitative temporal constraints network [50].
2. The higher layer is a Meta-CSP, where resource conflicts are represented and reasoned about. In this layer resource constraints are super-imposed, giving rise to a set of resource conflicts that are solved with a number of sequencing decisions [29, 30].

The search proceeds by iteratively performing the following steps:

- Propagation of temporal consequences through the Ground-CSP to compute currents bounds for all temporal variables.
- Computation of the Meta-CSP identifying the set of resource capacity violations implied by the Temporal Network.
- Selection of a conflict in the Meta-CSP according to the variable ordering heuristic.
- Resolution of the conflict by imposing a new precedence constraint in the Ground-CSP. This is done using a value ordering heuristic that preserves temporal flexibility.

The result is an efficient greedy procedure for generating feasible solutions to the Resource Constrained Project Scheduling Problem with time windows (RCPSP$_{max}$). This basic one-pass procedure is then inserted in a random restart optimization cycle that incrementally searches for better solutions by sampling new solutions on problems with reduced temporal horizons.

A RCPSP$_{max}$ consists of a set of activities where two kinds of constraints can be interrelated [98]:

- Precedence constraints that force that an activity cannot start before its predecessor activities.
- Resource constraints among activities that consume the same resource due to the limited capacity.

The objective is to find precedence and resource assignments for all the activities in the horizon imposed.

Before getting into details of each layer, we need to introduce the concept of Temporal Network as we will be referring to it along this dissertation.

A **Temporal Network** (TN) is defined in [26] as a directed graph whose nodes represent Time Points (TPs) and whose arcs represent distance constraints between TPs. Nodes represent points on a time line where temporal changes happen, and arcs (they must be directed arcs) represent activities duration and distance constraints between activities and events. In this framework:

- Nodes are labelled with the pair *[lb, ub]* where *lb* represents the lower bound, that is, the earliest possible start time for the TP satisfying the constraints in the network, and the *ub* represents the upper bound, that is, the latest possible start time.
- Arcs are labelled with the pair *[d, D]* where *d* and *D* represent the minimal and maximal duration constraint.

There is a particular node in the graph that in [26] is called **reference-point**. This node specify the beginning of the temporal horizon and has the constant label *[0, 0]*. The possible values for that pair varies from 0 to *h*, where *h* is the length of the temporal horizon, that is, the make span of the solution.

3.2.1 The Ground-CSP layer

The Temporal Constraint Network is a specialised Constraint Satisfaction Problem where variables represent TPs (or temporal events), their values are time instants, and constraints are binary inequalities that bound the distance between two TPs. Temporal constraints are mapped to distance constraints between TPs and an additional constraint is posted to bound the maximum temporal horizon. A time feasible solution is extracted from the TN choosing consistent values for each TP; for example the earliest possible start time in each activity. Path consistency allows to detect in constant time the set of feasible ordering associate to a decision variable.

3.2.2 The Meta-CSP layer

To handle cumulative resources, special attention has to be paid to the Meta-CSP due to exponential time to complete the computation. To address this problem two main ideas have been integrated:

1. After propagation in the Ground-CSP, the earliest start time of all temporal variables are extracted; this is used as a basis for computing the Minimal Critical Set (MCS) (i.e., the Meta-CSP).

2. A polynomial MCS sampling method is introduced to extract a subset of MCSs and overcomes the exponential worst case of complete enumeration.

If the Meta-CSP has no assigned variables, there are no remaining resource conflicts and a solution has been found. But if the set of possible orderings of a conflict is empty then there is no feasible solution.

The philosophy under the Meta-CSP is based on isolating MCSs, so a single precedence relation between any pair of activities in the MCS eliminates the resource conflict. The Meta-CSP is defined by taking the current set of MCS as variables and the set of precedence constraints that can be feasibly posted between some pair of activities in a given MCS as the domain of the corresponding variable. In general a peak is eliminated by *leveling* it, that is, by posting one or more precedence constraints between pairs of activities contributing to the peak. The choice of which pair of activities to order is made after a MCS analysis. To limit the number of MCS, only those MCS closest to an unresolvable state are chosen. In general, an MCS's distance from an unresolvable state depends on the number of different precedence constraints that could be posted to solve it and the temporal flexibility that is in the solution after it is solved. So, to collect the MCS, first, peaks in resource usage must be detected and then MCSs within these detected peaks are sampled. To promote diversity in the set of MCSs that are generated, sampling peaks which are either subsets of others peaks or have a large percentage of activities in common are avoided.

Having generated a MCS choice set, the next step is to identify one particular MCS for resolution. If at least one resolvable MCS remains outstanding, the selected MCS is removed by selecting and posting an additional precedence constraint between two of the competing activities in the MCS.

To select and solve a conflict the heuristic followed is to choose the most constrained variable and set the value that is least constraining. This is measured by calculating the temporal flexibility, that is, the number of temporal positions that the time variables in a solution may be assigned with respect to each other. The less flexibility a MCS has, the more critical it is to solve first.

3.2.3 Inputs and Outputs

To show what the inputs and the outputs of O-OSCAR are, let us consider the problem of Figure 3.2. The solution to this problem using a PO planner or any other planner[2] that works in parallel is shown in Figure 3.8. The number in the left of each operator represents the time instant when the activity is executed. We call *Activity 1* to the first activity, *Activity 2* to the second, and so on. As it can be seen in the solution, there are no interactions between the activities; that is, *make-bed* and *clear-bed* cannot be executed by the same agent, but the domain does not consider the availability of the agents/robots. It is the scheduler that will be in charge of constraining the resources used.

```
1 (make-bed agent7 bed0 room1) → Activity 1
1 (clear-bed agent7 bed1 room1)→ Activity 2
1 (goto-room agent1 room0 room1)→ Activity 3
2 (make-bed agent1 bed1 room1)→ Activity 4
```

Figure 3.8: Input plan to O-OSCAR.

Figure 3.9 shows the O-OSCAR input file to the solution of Figure 3.8.[3] For a detailed description of the input file, the reader can go to Appendix B.

Basically, the file is divided in two groups:

1. The fist part concerns the number of activity constraints of each activity and the temporal distance among them. For instance, in the example file of Figure 3.9, Activity 2 (row 4), has two activities that must be executed after it (column 3): Activity 4 (column 4) and the end point (column 5), each one to a unit temporal distance (columns 6 and 7 between brackets).

2. The second part concerns the duration and the capacity of each resource consumed by each activity. For example, Activity 2 (row 10) has duration one (column three) and it consumes one unit (column four) of resource one (agent1) and zero units (column five) of resource two (agent7).

[2]In this case we have used BLACKBOX [94].

[3]For simplicity, we have omitted the representation of the eight agents, reducing to two agents the number of resources in the file.

```
    4    2    0    0
    0    1    3    1    2    3    [1]     [1]     [1]
    1    1    1    5    [1]
    2    1    2    4    5   [1]    [1]
    3    1    1    4    [1]
    4    1    1    5    [1]
    5    1    0
    0    1    0    0    0
    1    1    1    1    0
    2    1    1    1    0
    3    1    1    0    1
    4    1    1    0    1
    5    1    0    0    0
    1    1
```

Figure 3.9: O-OSCAR Input File.

The solution that O-OSCAR generates to this problem is shown in Figure 3.10, and Figure 3.11 shows the graphical interface. The top window represents only the activity constraints, that is, the original plan (Figure 3.8) and the bottom window, the solution to the problem when considering time and resources (Figure 3.10).

```
  Activity    Start Time    End Time
     1            0            1
     2            1            2
     3            0            1
     4            2            3
```

Figure 3.10: Solution to the problem of Figure 3.9.

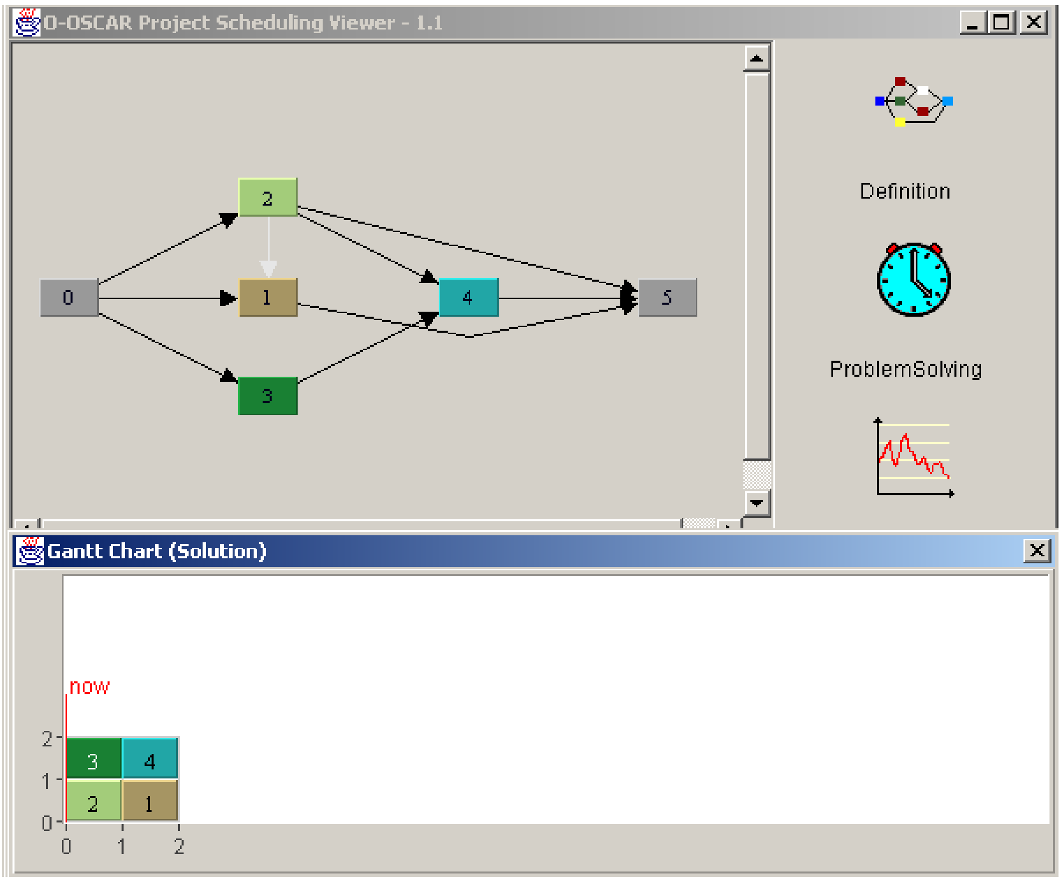


Figure 3.11: Graphical solution to Figure 3.10.

Part II

Modeling for Planning and Scheduling

Chapter 4

From Planning Competition to Workflow Domains

In this part of the dissertation we will focus on the representation problems for real domains as workflow or satellites domains. One of the main obstacles in applying AI planning techniques to real problems is the difficulty to model the domains. Usually, this requires that the people that have developed the planning system carry out the modeling phase since the representation depends very much on a deep knowledge of the internal working of the planning tools. Since, in Business Process Reengineering (BPR), there has already been some work on the definition of languages that allow non-experts to enter knowledge on processes into the tools, we propose here the use of one of such languages to enter knowledge on the organisation processes.

As instances of this domain, we will use two workflow tools: the Workflow System at BRITISH TELECOM (BT), where we have proposed improvements in the template generation thanks to the integration of AI contingent planners; and the workflow modeling tool SHAMASH, where we have exploded its object oriented structure to introduce the knowledge through its user-friendly interface and, using a translator transform it into predicate logic terms. After this conversion, real models can be automatically generated using the planner explained in Chapter 3.

4.1 Features of Workflow Systems Domains

Every organisation tries to shape its processes to optimally suit the market and offer the best service to the customer. When an organisation is analysed with the purpose of identifying possibilities for optimising its routines and procedures, three basic facets are outlined:

- A task or activity describes what should be done.
- An organisation model describes who is able to do something, and under what roles.

- An information model describes which information is needed to perform an activity.

From a historical perspective, the first issue that companies focused on was the design of organisational units. Then, control logic (when should something be done) was set to play a central role in connection with optimisation of business processes. Numerous issues need to be considered when designing business processes [48, 82] and implementing them in IT systems. These include: reusability of past processes, accessibility from the different agents in the organisation, consistency of usage, and selection of the right model.

Nowadays the efficiency of the internal workings of organisations strongly depends on the company processes. Over the past few decades, BPR has become fashionable among medium and big enterprises due to its capability to present solutions that improve their processes. Although there have already been many approaches to the computer-aided design of processes, very few have focused on the automatic generation of process models.

Due to the competitive nature of businesses and the strong pressure of the market, quality initiatives and the gradual improvement of processes are not enough for the continuous and dynamic changes that current organisations need. Nowadays organisations do not look for incremental percentage of improvement of say 10%, but multiplicative improvements of, for example, 15*X or more. Levels of changes so radical need new and powerful tools that can allow the redesign of the work. This is the main objective of BPR [82].

These business processes are represented as workflow, that is, computerised models within which all the parameters needed for the completion of the processes can be defined: orders, tasks, conditions, the information flow to support the tasks, resources involved, etc. Workflow Management Systems (WFMSs) [101, 114, 118] have been widely deployed in sectors like insurance, banking, accounting, manufacturing, telecommunications, administration and customer service [105]. A WfMS can provide active support to a business process by controlling the routing of work around the organisation automatically. This is done based on input describing the flow, the decisions, the exceptions, the resources to be used, etc. It co-ordinates user and system participants, together with the appropriate data resources, which may be accessible directly by the system or off-line to achieve defined goals by setting deadlines. The co-ordination involves passing tasks to participants' agents in correct sequence, and ensuring that all complete their tasks successfully. In case of exceptions, actions to solve the problem can be triggered, or human operators alerted.

In the last few years an increasing development of documentation tools and/or process modeling techniques have emerged to capture the knowledge of the current processes. Usually, the task of defining those models is performed with the aid of a set of tools that allow the graphical representation of them, together with the relations among the activities that occur within the processes, by a human who is an expert in the processes that take place in the organisation.

Prior to WfMS, many enterprises created special-purpose bespoke applications to support their processes. The advantage of WfMS-based solutions is that the workflow representation is explicit, and separated from the application code. This means that a WfMS can be customised quickly to support a new business or process, and that workflows are relatively easy to modify when a process changes. Current WfMS do not address all aspects of the problem, however. In general, they do not deal with scheduling or resource management/allocation. Similarly, while they provide means of generating exception events when things go wrong, they do not have a built-in re-planning function or automatic methods for adapting the workflow model according to those exceptions. They do, however, provide interfaces so that application-specific modules performing these functions can be integrated.

Workflow systems hold the promise of facilitating the everyday operation of many enterprises and work environments. Despite the popularity of these products, there is still a lack of maturity in some respect, that could be overcome using techniques coming from other fields such as AI.

Recently, there has been considerable interest in the application of Artificial Intelligence techniques to Workflow Management systems. Some researchers have seen the advantages of the integration of this approach, as shown by the existence of a Technical Co-ordination Unit of the European research network on planning and scheduling, PLANET [136], on applications of planning and scheduling to workflow. This has lead to some exploratory work reflected in a Roadmap [138] and some published papers [83, 96, 123, 149].

4.2 Workflow Management and AI Planning and Scheduling

To provide a frame of reference, in [96, 138] four stages in Workflow Systems were identified, although some authors only identify three since the Monitoring phase is included in the Enactment phase [79, 89]. We follow the first classification:

- Process Modeling: is the stage where the user designs, models, optimises, simulates the organisation's processes. We include in this stage the design of the process templates that can be instantiated and enacted by a workflow system.
- Process Planning: is the stage where the activities required to achieve some user goals are instantiated in an order, resources assigned, and preliminary scheduling performed.
- Enactment/Execution: in this stage the agents (software or humans) carry out the activities, with the workflow system co-ordinating execution.
- Monitoring: this is conducted concurrently with Enactment/Execution. The system enacting the workflow is monitored, with status information being made available to human operators. Exceptions, such as deviation from the plan, and subsidiary processes are initiated to rectify problems.

In AI Planning systems the following phases can be identified:

- Domain modeling: in this phase the user introduces the knowledge to the system, that is, the operators, the initial conditions and goals. Each planner has its own syntax although lately there has been an effort to unify the syntax in a unique language: the Planning Domain Definition Language, now in the 2.1 version (PDDL2.1) [68].
- Process Planning: the plan is outlined as a set of instantiated actions in a given order. Commonly, plans do not contain information about resources, so in some problems planning and scheduling can be separated. In other cases, this idea has to be abandoned and mechanisms to treat resources through constraining equations must be integrated to solve the problem.
- Execution: this stage relates to the actions' execution.
- Monitoring: the results of the actions execution can differ from the actions expected results, so monitoring must take place to anticipate events or re-plan if the initial plan cannot be achieved.

Table 4.1 outlines at a high level the concepts that AI planning & scheduling share with the Workflow community (for a more detailed description we refer to the Workflow Management PLANET TCU [138]).

Table 4.1: Concepts mapping between AI P&S and workflow.

AI P&S	Workflow
Modeling+Planning and Scheduling	Modeling and Scheduling
Execution	Enactment
Operators	Activities, tasks, ...
Initial State	Organisation, resources
Goals	Business goals, service provision, ...

4.3 The SHAMASH Tool and the AI planner QPRODIGY

In this section we describe the features of the the SHAMASH modeling tool and then, we make the connection between AI planning and workflow tools more precise, by describing first how to translate an organisation model described in terms of SHAMASH into a planning domain model for PRODIGY, how PRODIGY can produce the desired plan (model), and how this is translated back into SHAMASH.

4.3.1 The SHAMASH System

SHAMASH [5, 164], a R&D project funded by the IV Esprit Program, generated a process modeling tool that allows simulation, modeling and optimisation processes taking into account a realistic model of the organisation, including interaction models with the users. This aspect combined with the features that the tool embodies, make SHAMASH a powerful tool for the BPR. Very few tools allow more complex behaviour to be defined, such as: more detailed modeling than just having activities and attributes of those activities, automatic validation and optimisation of the models, or generation of HTML output, that allows people from the organisation to freely browse the processes in which they intervene.

Basically the SHAMASH tool consists of four subsystems:

- Author subsystem: this module provides a user-friendly interface that enables definitions of three types of knowledge: on standards, on processes and on the organisation. The standards, or norms, constrain how processes should behave. The interface allows the user to create their structure, related standards or processes, or organisation resources to be used. Processes are computational *units* within the organisations. The user can describe the sequence of steps to be completed to accomplish some goal. Each step or activity has pre- and post-conditions that define its behaviour. Once the user has defined his/her processeses, the tool allows connection of the related process so they can be simulated and optimised. SHAMASH also allows the user to define libraries of processes to be used in other modeling applications.

- Simulation and optimisation subsystem: this is an important part of the modeling tool. The simulator can check the behaviour of the process that the user has created. SHAMASH also helps in detecting static problems (using validation expert heuristics) and spotting problems that can only be detected when the model is running (as underused resources or bottlenecks). Also, the tool is able to automatically perform an optimisation phase by which new optimised models are generated. The user can decide whether to adopt the new models, or to continue with the old ones.

- Text generation subsystem: this subsystem is responsible for maintaining coherence between the graphical (defined by specific people in the organisation) and the text versions (the ones seen by most people in the organisation). Sometimes we could need a graphical representation of the processes, sometimes we will need plain text, and in other cases we could need it in a mixed format. If there is a change in the graphical representation, SHAMASH allows to automatically generate a new HTML version in accordance with the changes made.

- Workflow interface subsystem: SHAMASH cannot directly be used as a workflow engine. An interface is needed to automatically translate the defined process models into the input of a workflow engine. As a result, a WPDL (Workflow Process Description Language, the standard generated by the Workflow Management Coalition (WFMC) [195]) can be generated as output of the process representation.

The tool also allows the user to create and maintain knowledge about the organisation through a friendly interface. The behaviour of the model is defined through objects and rules with a well-defined and easy-to-use syntax. This allows detailed specification of how activities are performed, what inputs they take and in what format, how the outputs are generated from the inputs, etc.

The SHAMASH ontology on processes adopts an activity-centred modeling viewpoint [123]. The activities use resources to satisfy the process requirements defined by the user. Every activity has a set of preconditions that must be true to execute an activity or a process.

Rules can be used to define these conditions and to verify if an activity can be executed or not, or if it has achieved what it was supposed to. Rules are composed of two parts:

- The left part of the rule: defines the preconditions of the rule, i.e. when the right part of the rule can be executed.

- The right part of the rule: defines the actions to execute when preconditions are met.

An example of rule syntax in pseudocode is shown in Figure 4.1. This rule says that if there is a Consultation with attributes *Processing* equal to *Manual*, *FoundResponse* equal to *No* and *SentToUser* equal to *No*, and exist an Employee with *hours-left* greater or equal to three, then after the execution of the activity the field *FoundResponse* and *SentToUser* will be modified to *Yes* and the employee will have in its activities list to send the Consultation to the user.

```
<Rule Consultation-activity>
  <Left-part>
    Check if there is a Consultation with the values
      Processing = Manual
      FoundResponse = No
      SentToUser = No
      Resource = (Exists Employee with hours-left >= 3)
  <Right-part>
    Assign the new values to the Consultation
      FoundResponse = Yes
      SentToUser = Yes
    Assign the new values to the resource Employee
      assigning to list to-do-activities = Consultation
```

Figure 4.1: Example of SHAMASH Rule.

The activities need objects (e.g., data or information) that are created and modified within a process, through a series of activities. In Figure 4.1, *Processing*, *FoundResponse* and *SentToUser* are attributes of the data *Consultation*. The data *Consultation* will be modified by different activities depending on its initial values. The user will specify in each activity the input and output data. The activities also need resources to be performed and those are assigned according to the organisation structure and the activity rules of behaviour. In the example, we need an employee with at least three hours available; if there is any, the consultation task will be assigned to the list of tasks that the resource (employee) must perform. SHAMASH allows the user to define the type of resource that will be associated to each activity during the definition stage.

Once the user has defined all the data, activities and organisation structure that will be part of his/her process, he/she should specify the flow of control, i.e., the order in which activities are executed, and how they are linked. In the case of big processes, this stage is usually quite tedious for the user: draw all the activities, decision points and all the control connectors. The focus of our work is to automatically generate the model, avoiding going through all the drawing process, and making sure that the established connections among activities conform to a valid sequence of activities. After the model has been automatically generated, the user can simulate and optimise the process using SHAMASH, as it had been defined by him/her. If the model does not satisfy his/her expectations, he/she could modify it as required.

4.3.2 PRODIGY and SHAMASH

The translation presented in this section [142, 143, 146, 148] would serve to any planner that uses logical formulae.

We have used the non-linear planner PRODIGY [22] but other planners such O-PLAN [47] or CNLP [135] could also have been used. The automatic process generation was sketched in [142] as follows: first, the system translates the SHAMASH objects and rules into the PRODIGY language. Then, PRODIGY produces an ordered sequence of activities. And later, this sequence is translated back into SHAMASH to be presented graphically to the user. This process is shown in Figure 4.2. The translation has in mind the different semantics of BPR and AI planning concepts as well as their similarity (see Appendix A for a deeper model of the translation among the different elements that compose SHAMASH and QPRODIGY):

Predicates: PRODIGY domain theory is an augmentation of the STRIPS representation. In this representation, a world state is represented by a set of grounded literals, the conjunction of which is intended to describe the given state. The states in SHAMASH are represented as C++ objects (that is, instances, attributes and their values). To represent them in PRODIGY we have generated, for each tuple <instance, attribute, value> in the SHAMASH, a binary predicate whose name is the attribute name. Its two arguments correspond to the identifier of the instance that it belongs to, and the value of the attribute. For instance, one of the attributes of the order document element in the billing process of any company is the type of payment method (Visa, Check, Cash...). If there is an instance of an order document, *order-Client123*, whose payment method attribute has *Check* as value, then this is translated into the predicate:

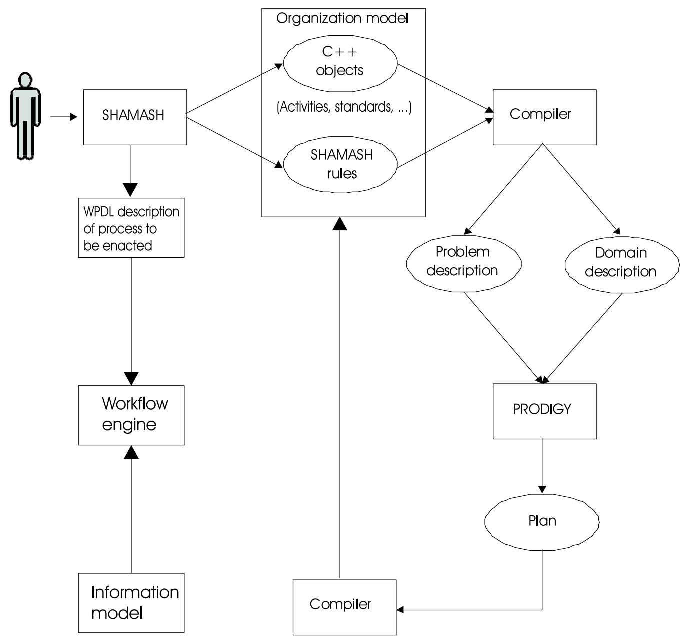


Figure 4.2: SHAMASH and PRODIGY integration.

```
(Payment order-Client123 Check)
```

Operators: task dependency is represented in BPR as activities with pre- and post-conditions (following the WFMC standard [195]). In SHAMASH preconditions are represented as rules that have to be fired before the activity can be executed. The activity post-conditions have to be true after the activity execution. On the other hand, PRODIGY has operators with preconditions and effects. Therefore, each activity name is translated into an operator name, the left-hand side of the preconditions of the activity into the operator preconditions and the right-hand side of the post-condition rules of the activity into the operator effects. Pre and postconditions refer to questions about the contents of the process at a given moment. Most of these questions refer to knowing the value of a given attribute of an instance, or setting it. In order to translate those questions and settings, the translation of attribute values into literals in the previous paragraph is taken into account.

Also, to establish the precedence between activities in a process, they must have the same input/output elements. For instance, in a billing process (explained in more detail in the next section) the activity Bill precedes the activity Pay, so the output element of the Bill activity (order document element) must be the same as the input element of the activity Pay. In PRODIGY, input/output elements are operator parameters (*params* in Figure 4.4), that is, the variables that will appear with the operator name when it is instantiated.

Types: PRODIGY requires that each variable in the operators belongs to a user-defined type. Since SHAMASH follows an object-oriented approach, it represents all static knowledge as classes and instances. Therefore, every class in SHAMASH is translated into a PRODIGY type. Thus, the Element class will be translated as:

```
(ptype-of Element:  top-type)
```

and the classes that belong to the Element class (as the order document element) as:

```
(ptype-of Order Element)
```

SHAMASH has also five basic types that are translated into PRODIGY user-defined types. For example, SHAMASH represents integer numbers as type integer. Given that PRODIGY has the capability of allowing to define variables with continuous values, called infinite types, this SHAMASH type is defined in PRODIGY as such.

Problem: apart from the domain theory, PRODIGY also needs the initial state and the goals to achieve. In SHAMASH the initial state is represented as the instances of the organisation units, roles, resources and elements defined by the user for each BPR domain (activities and organisation). For example, the instance that represents a human resource (e.g. `Mary`) can have its availability as one of its attributes. If it is considered a boolean attribute, this information is translated as:

```
(Availability Mary TRUE)
```

It could also be measured (represented as such as in SHAMASH) as a numeric value (e.g. `(Availability Mary 0.7)`). This could also be translated into PRODIGY.

In relation to PRODIGY goals, we translate the user goals defined in the SHAMASH process into the PRODIGY problem goals. As an example, in a Billing process, the aim of the process might be defined as client satisfaction. If we define it as when the client receives the product, it could be represented in PRODIGY as:

```
(Product-Sent order-Client123 TRUE)
```

If there were quality metrics in SHAMASH for the process, they could also be translated into PRODIGY. For instance, if we want to define as a metric the process duration, we should use the extension of PRODIGY to work with costs. In each operator it is added the field *cost* as Figure 4.4 shows and the user must specify in the activity an attribute that represents its duration.

Plan: the planner generates a sequence of instantiated operators (activities). This sequence or plan represents the activities instances or tasks dependency in a process model. A plan in PRODIGY for a simplified process would look like:

```
(Receive-Client-Order order-Client123)
(Bill order-Client123 Mary-Shefield)
(Pay order-Client123 Check)
```

If any metric is defined, QPRODIGY will try to find a plan minimising that metric. Then, the plan is translated back into SHAMASH, where it is graphically presented. Once the activities are represented and linked in the interface, the user can change its order, add/delete new activities if the results are not as expected or improve the model obtained by simulating and optimising it.

An Example: An Internet Billing Process

Let us consider a simple scenario in order to clarify the concepts presented until now. The billing process could be considered as a process that every company has to face. The following is a simplified description of it. Suppose a client contacts the company (for instance, via the Internet) to buy some product. We identify the element order with the attributes: `ClientName` (type string), `Product-Number` (type integer), `Payment` (type string with two values: "Check" or "Visa"), `Paid` (type boolean) and `Product-Sent` (type boolean).

Once the client has specified the products that s/he needs, the business process starts. A delivery note is generated with the cost of the products, and, at the same time, the goods are sent to the customer. If the customer has received the goods, it is time to pay by one of the payment types specified in the order. In this process three are the activities to perform: input client data (we call the activity Receive-Client-Order), generate the bill (Bill) and the payment of the products (Pay). The order document element is the input/output element of the three activities. As an example of a possible behaviour of the Bill activity, the rule describing its behaviour is shown in Figure 4.3. Figure 4.4 shows its corresponding QPRODIGY operator translated by the system automatically from the rule.

```
Rule Bill with properties ruleset behaviour
  If Exists a COrder of type MetaClassElement named var order1
    with Product-Number > ( 0 )
    with Product-Sent = ( FALSE )
    with Payment = ( ''Visa'' )
  Then Modify object order1
    with Product-Sent is ( TRUE )
```

Figure 4.3: Example of a Bill activity rule in SHAMASH.

The precondition of the Billing activity checks if there is an element of the type Element class (represented by the variable order1) with attribute values such that: the number of products that the client has bought (`Product-Number`) is greater than zero, the product has not yet been sent (`Product-Sent`) and the method chosen to pay (`Payment`) is Visa. As a post-condition, the execution of the activity will cause that the products bought are sent to the customer.

The variable `order1` is used to represent the element instantiated by QPRODIGY among the orders that the problem might have (order-Client123, in the plan instantiated in the last section). The QPRODIGY function *gen-from-pred* generates a list of values to be possible bindings for the corresponding variable by using the information of the current state. This function is used with infinite-type (numbers), as is the case of the integer type, and in this case generates all the numeric values of the predicate `Product-Number` in the problem with a value greater than zero.

With respect to the problem definition, Figure 4.5 shows the initial conditions generated automatically from the organisation information. This includes issues such as who works in each unit (`Assignedto Financial-Unit Mary`), the cost per hour of the employees belonging to a given category (`Cost Administrative 548Euro`) and other information related to the details of a specific order (in this

```
(OPERATOR  Bill
 (params <order1>)
  (preconds
   (<order1> COrder)
   (<cct> (and integer (gen-from-pred (Product-Number <order1> <cct>))
                       (> <cct> 0))))
   (and (Product-Sent <order1> FALSE)
        (Payment <order1> Visa)))
  (effects
   ()
   ((add (Product-Sent <order1> TRUE))
    (del (Product-Sent <order1> FALSE))))
  (costs
   ((<duration> (and (DURATION
                     (cost-from-pred (billing-time <duration>)))))
   ((TIME <duration>))))
```

Figure 4.4: QPRODIGY operator corresponding to the Bill rule of Figure 4.3.

example the order number 123). The Figure also shows the goals that PRODIGY needs to accomplish. In this case, the only goal is to send the products to the client whose order document is 123. With these conditions PRODIGY generates the plan described before, with resources, units and roles in that process.

```
(state
   (and (Availability Mary TRUE)
        (Cost Administrative 548Euro)
        (Id Financial-Unit 123A)
        (Assignedto Financial-Unit Mary)
        (ClientName order-Client123 SmithD.)
        (Product-Number order-Client123 2334)
        (Payment order-Client123 Visa)
        (Paid order-Client123 YES)
        (Product-Sent order-Client123 FALSE) ...  ))
(goal (Product-Sent order-Client123 TRUE))
```

Figure 4.5: PRODIGY problem for the Billing example.

4.4 A Legacy Workflow System: COSMOSS and the CASSANDRA planner

In this section, we describe the automatic generation that can be done in the template creation of the COSMOSS system using the CASSANDRA planner.

4.4.1 COSMOSS

The Customer Orientated System for the Management Of Special Services (COSMOSS) [34] provides support for progressing orders concerning provision of private lines at BRITISH TELECOM. It was built at the beginning of 90's and it handles about a dozen main product families divided into digital and analogue categories.

The business processes start with a customer contacting a call centre to place an order. The representative in the call centre gathers information from the caller about the customer and the service required in the form of a Service Order. This is passed automatically to COSMOSS for completion where it becomes a job. The job is decomposed into activities by matching against a set of job templates. Target times are then derived for these activities based on information stored in the job and activity templates and the customer requirements. These activities, with target start and completion dates, are passed to other OSSs where they are allocated and enacted. Progress is reported back to COSMOSS.

The main COSMOSS modules are:

- Order taking module.
- Product Register: interfaces to the portfolio database which holds information on 90% of BT's product range. This is also used by other systems.
- Customer and site database: holds information on customers and their premises, and is basically specific to the COSMOSS system.
- An engineering database.
- Job Management module.

All the information that COSMOSS can handle is organised in templates. Each time new products are introduced, new templates are created. The Process Modeling stage corresponds to design and creation of the Job and Activity Templates. In AI Planning systems, each activity template corresponds to an operator.

A service order (with its parameters) is used to create a job, which is decomposed into activities linked by dependencies. Activity templates may have conditions to be met for them to complete. Activities also have *input criteria*: these are similar to preconditions. They can be used to prune branches of the process tree that are no longer potentially relevant. When a process is initially instantiated, it generally is under-specified, including alternative branches, only some of which will be used.

Conditional processing controls which activities actually become part of the job by prompting the users with questions that they must answer. Different answers will cause different activities to be created in the job. Sometimes the same question may be repeated in some activities; the reason to repeat the questions is to avoid redoing all the design if the user answers the question incorrectly by an oversight. However, the system allows an Automatic conditional response, that is, the system will assume that the question is correct and will jump it.

This corresponds to the Process Planning phase, that is, the appropriate template is identified, instantiated, and the instance is elaborated in sufficient detail to be executed. For Process Planning in the AI sense, an instantiated plan without having in mind temporal/resources constraints, just activities linked by dependencies is obtained. As we describe below, the resource assignment is done separately.

After template selection, the software constructs a schedule for a job by trying to meet either the Customer Requirement by Date (CRD) or the Target Completion Date, whichever is the later. COSMOSS uses a common algorithm critical path analysis to apply date rules to a template to adjust the overall dates of the order and the window lengths of the activities within the template, to ensure the Job is completed by the CRD. It is worth noting that when the job is created it will be scheduled according to the contents of the whole template. Once the job and the corresponding activities have been selected, they are assigned to *owners* (or rather queues) for its completion. Usually an owner is an organisational unit, but it could be a queue for another system.

At this point, the activities are assigned to agents (bearing in mind the computer programs and the human and material resources available in the system as well as the roles they can hold), and the Execution phase begins. During execution, completion of activities, and delays and other problems must be detected and reported; this is the Monitoring phase. In COSMOSS this information is sent to a human manager in the customer service centre. This manager can take the appropriate measures to rectify the situation, or at least to try to ensure that the situation does not recur: in some cases, the problem can be solved adding some templates. In other more drastic cases, a new job template must be required (that is, a new plan).

Readers knowledgeable about AI planning will recognise many of the issues addressed here, though the terminology may be unfamiliar. The designers of COSMOSS developed an ad hoc and domain specific solution without knowledge of the great body of planning research that might have enabled a more elegant and manageable solution. Improvements can be achieved if one can avoid the *oversight* problems in the template answers and speed up all the design templates.

4.4.2 CASSANDRA and COSMOSS

To automate COSMOSS job template design we need a planner that could have different outputs depending on the action that is required to complete the service [148]. Most classical planners use the assumption that there is no uncertainty in the world: every action has a predictable output. A contingent plan is a plan that contains actions that may or may not actually be executed, depending on the circumstances that hold at the time. A contingent planner must be able to produce plans even if the initial conditions and the outcomes of some of the actions are not known. Several contingent planners can be used to automate Activity Template selection in a Job Template. As described in section 2.2.2, CASSANDRA [140] is a partial-order, contingency, domain-independent problem solver architecture based on UCPOP [133]. SGP [189] is an extension of the Planning Graph Analysis algorithm GRAPHPLAN [15] to support uncertainty in initial conditions and actions. CNLP [135] uses the basic SNLP algorithm to construct contingents plans while PLINTH [78] is a total order plan very similar to CNLP in its treatment of contingency plans. SENSp [55] like CASSANDRA is based on UCPOP but differs in the way it represents uncertainty. All have in common the way they represent operators, based on STRIPS representation. Another type of planner that can be considered was the probabilistic contingency planner exemplified by C-BURIDAN [54]. However, the fact that those planners are based on a probabilistic model make them unsuitable for use with COSMOSS because the probabilities are not known and the bigger the domain are, the harder is for these type of planners to find a solution.

We are going to use the planner CASSANDRA, but any of the other non probabilistic planners mentioned before could be used. To generate plans in CASSANDRA, all the possible uncertainty outcomes of actions must be known *a priori*, that is, the planner must be able to enumerate these contingencies. Each single operator may introduce any number of sources of uncertainty with mutually exclusive outcomes. Every source of uncertainty is a decision to make.

Figure 4.6 shows an example of a syntax operator of the CASSANDRA planner [140] corresponding to the example introduced in the next subsection. Each Activity in COSMOSS may have conditions that define possible user answers. All the possible answers are known at design time. These answers will cause different Activity templates to become part of a job at execution time. This is also how CASSANDRA works: if a decision cannot be made in advance due to lack of information, the agent will choose which branch to follow when the information becomes available during the execution of the plan.

```
Action (InstallingL ?line)
Preconds:(:and (required-service ?client)
                (not-occupied ?worker))
Effects:(:when (:unknown ?spare-available Yes) ;uncertain effect
                 :effect (connect ?line))
         (:when (:unknown ?spare-available No) ;uncertain effect
                 :effect (built ?line))
```

Figure 4.6: An example of representing the action of installing a line.

In addition to the domain theory, one has to provide the problem description in terms of initial state and goals. Those states are represented by a logical formula that specifies a situation for which one is looking for a solution. The initial state specifies the starting situation of the posed problem. Goals are often viewed as specifications for a plan. They describe the result that successful execution of the plan should produce: what one would like to be true at the end of the solution of the problem. Figure 4.7 shows an example of some initial and goal conditions in CASSANDRA.

```
Initial:  (and (required-service Mary-Th)
               (installed-telephone Mary-Th No)
               (spare-available Yes)
               (card-available No)
               (not-occupied Smith))
Goals:  (and (agree Mary-Th Yes)
             (installed-telephone Mary-Th Yes))
```

Figure 4.7: Initial state and goals for installing a line.

An Example: Installing a New Telephone Line
A customer contacts BT customer service (over the phone, in a BT shop or via the Internet). Let us say Mary Thompson contacts BT to ask for a new telephone line. At this point the business process starts (if the user agrees to the terms and conditions). The customer details are needed to see if she is an existing customer and already has a line with BT (in that case, a discount will be applied to the second line). If the customer is asking a line for the first time, a spare pair of wires must be available from the house to make a connection from the Distribution Point (DP, e.g.: telegraph pole). If no pair is available, then a new cable must be built (and the customer must be notified that the delivery date will be delayed).

Afterwards, it is necessary to check that there is a spare line card available in the exchange (in that case it is reserved/allocated). If none is available installation must be arranged. Installation involves making the connection at the DP (connecting a drop wire to the pair of wires that lead back to the exchange). Then, someone must:

- Contact the customer to arrange a visit to the house to fit new network terminating equipment (NTE, that is, the box on the wall that the phone is plugged into);
- Arrange for an engineer to turn up on the right day/time to test the line end to end and install the NTE;
- Allocate a telephone number to the new line and configure the exchange;
- Update the exchange, line plant and customer records;
- And of course, check with customer that he/she is happy with the service.

Figure 4.8 shows the plan that CASSANDRA builds for this particular example. The decision-steps in the plan and all possible outcomes of uncertainty are represented in the same way that the authors of CASSANDRA use in [140]. This particular

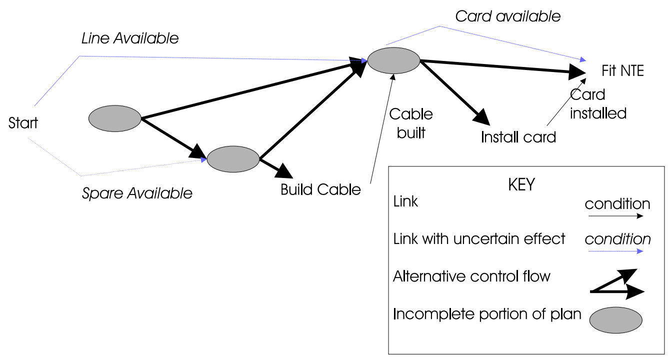


Figure 4.8: A partial plan showing the sources of uncertainty for providing a new line.

plan has three sources of uncertainty: the existence of a line, the existence of spare pair and the availability of a card spare.

We start with an incomplete portion of the plan with two uncertain effects: the availability of the line or of the availability of the spare. If the line is not available, we arrive at another incomplete portion of the plan with two possible situations: the availability or non-availability of the spare. If the spare is not available we just need to build the cable. Once the cable has been built (in the case it is needed) or checked the availability of the line or spare, the third incomplete portion of the plan depends on the card availability. If it is not available, the card must be installed. After this, the rest of the plan does not present any uncertain effect and the control follows a linear flow.

Each of the possible actions/operators in CASSANDRA will be matched to a particular Activity Template in COSMOSS. Figure 4.9[1] shows the Job template with its activities for the example of Figure 4.8. The code represents each possible situation (activity templates) and the arrows show the different paths depending on the user answers in each activity template.

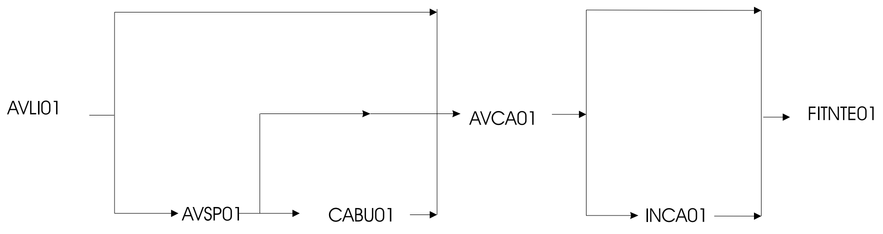


Figure 4.9: A partial plan showing the sources of uncertainty for providing a new line using the template notation at BT.

So LIAV01 represents the template for the AVailable LIne with two possible paths (arrows) for the two user answers: Yes or No. Each answer will add different activity

[1]The template shown has been created for this particular example, and does not exist as such in COSMOSS.

templates to the job template. If the answer is Yes, the AVailable CArd template will be added or AVailable SPare template in the other case.

If we consider the problem of Figure 4.7, CASSANDRA would generate the following plan:

InstallingL line501 → InstallingC card327 → FitNTE Mary-Thompson → Test & Install line501 → AllocateNumber line501 → UpdateL line501 → UpdateC Mary-Thompson CheckCustomerOrder → Mary-Thompson.

4.5 Related Work

Several research groups have integrated AI techniques with Workflow Management Systems (support the modeling and the enactment phase of the software engineering process). Some have applied planning tools in the enactment phase and very few have integrated planning techniques with BPR tools during the modeling phase.

The SWIM system [14] integrates process instantiation, task allocation, execution, monitoring and optimisation for improvement in the schedule. There is a Process Library that provides the correct process definition to the Dynamic Process Manager. New processes can also be added to this library.

The MILOS project [79] is a process modeling and enactment approach that provides project planning and management support like resource allocation and time scheduling for tasks in the project. The MILOS workflow engine allows the model and plan to be changed during project enactment and provides support for process restarts whenever necessary. The library of process models contains descriptions of best practices in software development for a given company. The project manager selects processes and methods from the library creating an initial project plan. The plan is uploaded to standard project management tools as MS-Project.

The TBPM [89] project is based on work carried out in the Enterprise project [175] and centres around an intelligent workflow engine that includes an interactive process planner. The planner uses AI techniques based on O-Plan [47] to assist in task planning, while permitting the user to participate in planning decisions. An agent-based architecture supports the execution and co-ordination of the planned process among multiple participants and distributes processes across computer networks. The user is able to plan a task by assembling fragments and then to refine it, generating a hierarchical model of the process to be performed. For more flexibility, the user is able to edit the process model before and during its enactment, in order to specialise it.

In all these systems the modeling phase is based on a process library, but there is no automatic generation as we have outlined in this integration of COSMOSS-CASSANDRA or SHAMASH-PRODIGY. Each time new model/templates are created, there is no need to create a library; the planner will generate the correct one automatically.

4.6 Summary

In this chapter, the representation problem for workflow domains has been faced. We have used two tools: the COSMOSS WORKFLOW system at BRITISH TELECOM, and the SHAMASH WORKFLOW modeling tool generated in an Esprit Project.

To provide a frame of reference between Workflow Systems and AI planners, the stages that both areas have in common has been described following the classification given by some researchers [96, 138]. The translation between both fields has had in mind the different semantics of BPR and AI planning concepts as well as their similarities.

We have described the advantages of using a classical planner for modeling processes in SHAMASH, a Knowledge Based System that uses an object oriented and rule-based approach. The SHAMASH rules and objects are translated into types and operators to produce a plan that correspond to a process instance (tasks dependency). Each plan will vary depending on the resources available, elements that flow through the process and the different tasks that the organisations can introduce to adapt their processes to the market changes. With this approach the models generated are automatically validated avoiding inconsistencies in linking activities and saving time to the user. Also, we have mentioned the improvements to automate COSMOSS job template design at BT using a contingent planner that can have different outputs depending on the action required.

Finally, we also wanted to study the issues that AI planners can gain with this approach. Generally, to specify the domain theory, a deep understanding of the way AI planners work and its terminology is needed. However, if we use a tool like SHAMASH or COSMOSS, the description language is closer to the user and allows an automatic verification of the syntax through a friendly interface. In this integration we could have also used the PDDL2.1 language instead. Then, any planner that supports PDDL2.1 could exploit the advantages of using a descriptive language as the one used in SHAMASH or COSMOSS.

Chapter 5

From Planning Competition to Satellite Domains

In the previous chapter, we have described some features and requirements when dealing with workflow domains. In this chapter, we provide an overview of the scheduling concepts that we had to face in order to give a solution to the nominal operations of HISPASAT, a Spanish multi-mission system in charge of satisfying national communication needs, using a planner, since we needed to represent time information and constraints through the Allen primitives [7]. We also describe the features of the tool developed for this domain.

5.1 Features of the HISPASAT Domain

Satellite domains are becoming a fashionable area of research within the AI community due to the complexity of the problems that these domains need to solve. With the current US and European focus on launching satellites for communication, broadcasting, or localization tasks, among others, the automatic control of these machines becomes an important problem.

From all possible tasks in this domain for which planning can be used, we will focus on the HISPASAT ground scheduling operations for its four satellites. HISPASAT is a Spanish multi-mission system in charge of satisfying national communication needs. It also supplies capacity for digital TV in Europe, America and North Africa, TV image, radio and other signals, and special communications for defense purposes. It is the first European satellites system to offer transatlantic capacity that allows simultaneous coverage between South and North America.

Every operation in orbit must have explicit engineering instructions. These instructions provide a guide for technicians to consider the work accomplished. The operations engineering group generates this documentation every year by hand and on paper. Later, this documentation is revised and verified. Due to the increasing number of satellites, actually four, 1A, 1B, 1C and 1D[1], and two in the future, 1E and

[1]1D was launched into orbit on September 18, 2002, from Cape Canaveral, Florida.

Amazonas, a program that generates and validates the schedule of operations for engineering support is needed.

Currently, the engineers create two documents: one which provides an overview of all the operations performed each day of the year and another that represents in more detail each operation duration. The weekly representation is generated every week having in mind the annual representation, so the problem of maintaining two documents relates very often to incongruencies between them.

In the HISPASAT domain the maneuver operations play an important role in the schedule of the rest of operations. The maneuver operations are in charge of the correct satellite positioning into orbit. They must follow a rigid time table: every two weeks (on Monday or Tuesday depending if it is summer or winter) in a specific hour given by an external software tool. These operations can only be moved ahead if any interference (by the sun or by the moon) occurs. Interferences cause wrong measures in the satellite orbit so the sensors affected by them must be masked using the adequate operations. Once these operations are scheduled, the operations related to the use of tanks or batteries are set: always in weeks without maneuvers. Finally there is a small set of operations that only depend on the last time they were performed. In these cases just the data of the last operation is needed.

From the features described above, we have identified three categories of planned operations, according to the flexibility to schedule them (hard vs. soft constraints). The following subsections describe them in more detail.

5.1.1 Operations which are driven by external events and shall start or be completed at a fixed time

Some operations depend on external events such as Moon Blindings, Sun Blindings or Eclipses of the Sun by the Earth or by the Moon. These events constraint when other operations can be performed. Given that these are hard constraints, they will be represented as preconditions of the operators. Given that the external (i.e. eclipses and blindings) and seasonal (i.e. solstice and equinox) events are foreseen several weeks before the year starts, all these events are represented in the initial state of the problem. In the case of external events, one needs to include the affected satellite and their start and end times. For the seasonal events, one just needs to represent the start time of the spring and summer equinox and winter and autumn solstice. Examples are the CHANGE-TO-MODE2/4 and the CONF-ADCS operations performed every time a Moon Blinding occurs or during the sun blinding periods. These operations are in charge of setting masks onto the upper or lower position detectors depending where the interference occurs (north or south). The satellite does not consider the value of these detectors to define its position and it does not destabilize the right positioning into its orbit. Or the CPE-MODE operation, performed in spring and autumn equinox, switching on or off level heaters to compensate seasonal evolution. As goals, we define the two possible modes (summer and winter) for the CPE operator, so the planner generates the appropriate satellite operations in each period. The operations in this first category do not force the addition of temporal constraints to the operators effects.

5.1.2 Operations that are part of a given strategy and should be performed at specific dates

Other operations have more flexibility in the date they can be performed. This is the case of Manoeuvres, Localisation Campaigns or Batteries Reconditioning. Given that they are not hard constraints, we have coded functions in the preconditions to guide the planner for preferring some dates over others having in mind the restrictions imposed in some preconditions. This decision restricts the number of possible solutions that the planner will provide, but as we said before, on this domain it is enough to generate a feasible solution. After constraining the set of values the planner is still able to obtain valid plans.

Most of the operations in this category are scheduled having in mind the date when a South Maneuver was performed. This operation is in charge of the correct satellite orbit positioning. They are performed every two weeks at a specific time.[2] In the initial state we have the satellite affected by the secular drift, and the date and time of the year during which the Maneuver is going to be performed. Table 5.1 shows the features of a South Maneuver. Given that the maneuvers are forbidden during moon or sun blindings they have to be moved ahead twenty-four or forty-eight hours (in case two Moon Blindings appear in following days) from the secular drift time. The two maneuvers (West and East Maneuver) that follow the first one must also be moved ahead the same number of hours. The rest of the operations in this category start/finish some hours before/after the start/end of the South Maneuver was performed. Some operations cannot be performed if a Maneuver has been scheduled during that week, while others must be performed during Maneuvers. Others just have to be performed *N* days/weeks/months since its last time performed operation. Therefore, we have different types of temporal relations among operators.

Table 5.1: SOUTH MANEUVER task. This is one of the South Maneuvers that will be performed during year 2002.

Name	South Maneuver
Description	Follows the two weeks keeping cycle, operating the satellite in one of every two weeks
Requirements	On Monday or Tuesday at an hour corresponding to the secular drift. A West and East maneuver must follow it
Constraints	It cannot be performed with Moon or Sun Blindings
Secular drift	18/02/2002 22:47:56
Duration	3 hours

In order to represent this knowledge, we needed a problem solver able to express the fact that if a South Maneuver is performed during one day of a week (having in

[2]The hour corresponding to the secular drift direction, that is, an hour when it is possible to position the satellite, having in mind different parameters.

mind Moon and Sun Blindings). Other operations, such as BOOST HEATING, cannot be performed during the same week, but the following week. This kind of reasoning is hard to represent in the operator preconditions and difficult to solve by some current planners.

5.1.3 Operations that should be performed within a long period of time

A significant flexibility exists for scheduling them as it is the case of the Steerable Antenna Maintenance. This operation is performed four times a year. So, in order to schedule it, we need to know the last date at which the operation was performed. The operations that belong to this group present the softest constraints because they have no other constraints with the rest of the operators; the constraints relate just to the same operation.

For these operations, we only need to represent in the initial state the last time the operation was performed in each satellite during the previous year. As with respect to the goals, we have to introduce the number of executions of each operator during the year (in the above example, four times).

5.2 HISPASAT Domain Theory

In order to define a domain, we have to consider the set of operator descriptions, the initial state and goals of problems, and, possibly some control knowledge. In this section, we describe the different elements that compose the HISPASAT domain.

5.2.1 Operators

The first step for defining the domain consists of identifying the operators (actions that can be performed) and the types of objects that are needed in the domain (given that one needs to declare the type of each operator variable). In PRODIGY, as mentioned in Chapter 3, types can be defined structured in a hierarchy. Infinite types allow to represent continuous valued variables (i.e. DATE), while finite types represent nominal types. In our domain, we have, among others, the following types: SATELLITE, TIME, PERCENTAGE, DIRECTION and DATE. Variables of type SATELLITE instantiate to one of the available satellites. DIRECTION can have the values north or south. TIMES and PERCENTAGE could have been defined as numbers, but we have chosen to declare them as discrete variables given that under our domain formalisation only a finite number of values for them are needed to be considered. Figure 5.1 shows one of the hundred operators that have been implemented in the HISPASAT domain (see Appendix C for the complete domain description). In particular it is the South Maneuver operator, in charge of calculating the date at which this operation can be performed having in mind that any Moon Blinding (represented by the *moon-blinding* predicate) cannot interfere with the expected South Maneuver date within three hours. If this occurs, the operation will be moved twenty-four hours ahead.

```
(OPERATOR SOUTH-MANEUVER
  (preconds
  ((<s> SATELLITE)
   (<t> TIMES)
   (<t1> TIMES)
   (<p> PERCENTAGE)
   (<n> DIRECTION)
   (<d> (and DATE (gen-from-pred
                    (moon-blinding-start <s> <d> <n> <p> <t1>))))
   (<p1> (and PERCENTAGE (over-n <d> greater 40)))
   (<d1> (and DATE (is-south-maneuver <d> 3 hours 3 hours)))
   (<start-time> (and DATE (del-time <d1> 24 hours)))
   (<end-time> (and DATE (end-operation <start-time> 3 hours))))
  (south-maneuver <s> <t> <d1>))
  (effects
   ()
   ((add (south-m <s> <t>))
    (add (south-man <s> <t> <start-time>)))))
```

Figure 5.1: Operator corresponding to the South Maneuver task of Table 5.1. If there is a Moon Blinding within three hours from the expected maneuver with intensity over 40, the maneuver has to be moved ahead twenty-four hours. The end date of the task is calculated by the end-operation function.

In the operator of Figure 5.1 we use the *gen-from-pred* function to generate all the values that the numeric variable DATE can have. DATE represents seconds since 1900 in GMT (we have used Common Lisp as the programming language for functions, given that PRODIGY is written in Lisp and this is the way Common Lisp handles time). The reason to use this format is for efficiency: it is faster to generate the bindings of one date variable instead of generating values for the six usual time dependent variables (corresponding to the year, month, day, hours, minutes and seconds). Also GMT is the reference zone time for HISPASAT. Other similar approaches fix the starting point of the computation, call it time zero, and schedule all the activities from that point [120, 177, 182].

The rest of functions that appear in Figure 5.1 have been coded for this particular domain. However, since some of them are generic for any domain with temporal restrictions they can be re-used in any such domain as it will be described later on. As an example of dependent functions of this domain, *is-south-maneuver* generates the date of the maneuver (if there is any) that overlaps within three-hours any Moon Blinding. If the blinding intensity is over 40%, the maneuver must be moved twenty-four hours ahead (the function *over-n* calculates if the percentage of the Moon Blinding is over forty).

As examples of domain independent functions the *del-time* function subtracts twenty-four hours from the date, in this case the expected maneuver, and *end-operation* calculates the end of the operation from the start time and the duration, three hours in this case. We have defined a similar function *add-time* that adds some time to a date and also helps to define temporal constraints into the operators.

With respect to the preconditions of this operator, it has only one: *(south-maneuver <s> <t> <d1>)*, that is, the date that the South Maneuver is expected to be performed (as shown in Figure 5.2, this is part of the initial state). The operator has two effects that belong to the add list; the predicates *south-m* that is the goal that we want to achieve, and *south-man* that adds to the state the date at which the *south-maneuver* must be performed. In case any interference occurs within three hours, the value of the <start-time> variable matches the value of the expected maneuver, <d1>, moved ahead twenty-four hours.

5.2.2 Initial State and Goals

For each type of satellite control operation we need to define the set of goals that must be achieved and the initial conditions that must be set up in order to apply them. Figure 5.2 shows a small fraction of the initial state, that is, all the events that will occur during the year. For instance, for Moon Blindings we need to know the satellite affected by the blinding (1B), the date in seconds during which it will take place (3219869400 = 13/1/2001 01:10:00), the direction (south) and the intensity (45% represented by p-45). Finally, the Moon Blinding chronological appearance, that is, the first time it appears is represented by t1, the second time t2, etc. With respect to the goals, we need to perform a fixed number of maneuvers during the year, which is represented in the goals section of the problem.

```
(state
   (and
      (moon-blinding-start 1B 3219869400 south p-45 t1)
      (moon-blinding-end 1B 3219873005 south p-45 t1)
      (spring-equinox 3194121600)
      (last-antenna-maintenance 1B 3172893395)
      ...
      (south-maneuver 1B t1 3194035800)
      (south-maneuver 1B t2 3195245400) ...))
(goal
   (and
      (south-m 1B t1) (south-m 1B t2)...
      (south-m 1B t20) ...))
```

Figure 5.2: Some goals and initial conditions for the 1B satellite. For instance, in the goals it is required that the South Maneuver should be performed a number of times a year.

5.2.3 Control Rules

We have coded 10 control rules for this domain. As an example, we can select resources as bindings of some operators. In the HISPASAT domain we identify as resources the tanks and the batteries. The control rules can help to assign a given resource to an operation, for example a tank. Each satellite has four tanks. Any of them could be chosen for swapping, but there is a recommendation from the satellites builder company to use a given tank during each period of the year. The control rule in Figure 5.3 prunes the search tree and chooses the tank for that period of the year. It says that if the date is around autumn equinox, and we can apply the operator TANK-SWAPPING, then we should select the tank NTO1. The same can be said of the batteries: there are two batteries, batt1 and batt2, that must be charged twice a year before every eclipse season. Another control rule helps to choose the battery in the right season.

```
(Control-rule select-tank-NTO1
   (if (and (current-goal (swapped <s> <tank>))
            (current-ops (TANK-SWAPPING))
            (true-in-state (swapped <s> NTO3))
            (true-in-state (equinox-au <d>))))
   (then select bindings ((<tank> .  NTO1))))
```

Figure 5.3: Control rule to select the tank NTO1 during the autumn equinox.

We could have alternatively coded a function to choose the correct tank for the TANK-SWAPPING operation. However, coding this type of preferences as control rules provides more flexibility, given that it is easier to decide which ones to use at run time.

5.3 CONSAT

One of the current problems for the industrial application of planning technology applications is the lack of a easy-to-use front end for users and their integration with the current tools used in organization. This section presents a graphical modeling and validation tool, CONSAT,[3] developed for scheduling the nominal operations for in orbit control of HISPASAT's satellites [147]. The interface has been implemented in Visual Basic 6.0 under Windows. Figure 5.4 shows the main window.

The tool consists of the following subsystems that will be described in the next subsections:

- **The user subsystem:** is in charge of the control access to the tool and the interaction with the user in order to obtain and manipulate all the data needed for planning and scheduling.

[3]It stands for CONtrol de SATélites, that is, satellites control.

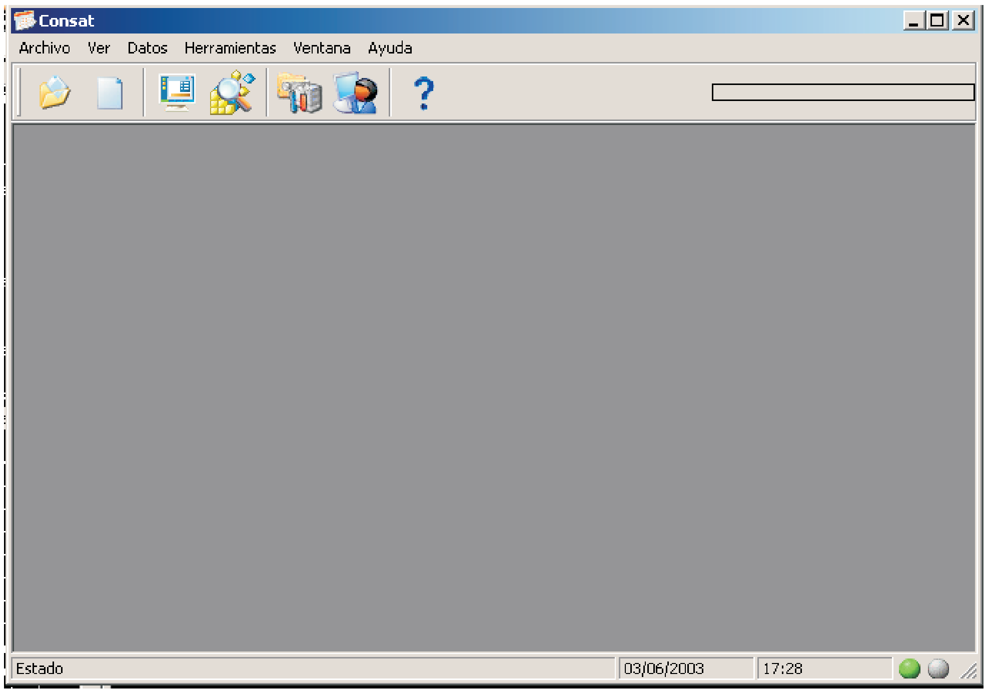


Figure 5.4: Main window of the CONSAT application.

- **The reasoner subsystem:** once the input data is introduced, a domain independent planner is in charge of generating the solution to the problem.

- **The generator susbsystem:** is responsible for maintaining coherence between the two possible representations that the tool offers: *annual* to provide a general overview of the operations and *weekly* for a more detailed view of the hours and resources (if any) involved in the operations. If the user modifies the *weekly* representation, the *annual* representation will be updated automatically. This subsystem also allows to convert the solution to a specific format document type, compare two different solutions, or generate an HTML version.

Figure 5.5 shows a high level view of the architecture, and the modules that it comprises. The first and the third subsystems interact with the user. The second subsystem interacts with the other two and is in charge of the solution generation.

5.3.1 The User Subsystem

The User Subsystem is in charge of data introduction. The control engineers are the ones that interact with this subsystem. As we have seen before, the initial state and goals refer to data about external events and some operations. Some of this data is difficult to provide by a user directly such as the time in seconds that most operators and literals need as arguments. Therefore, the user subsystem allows the user to

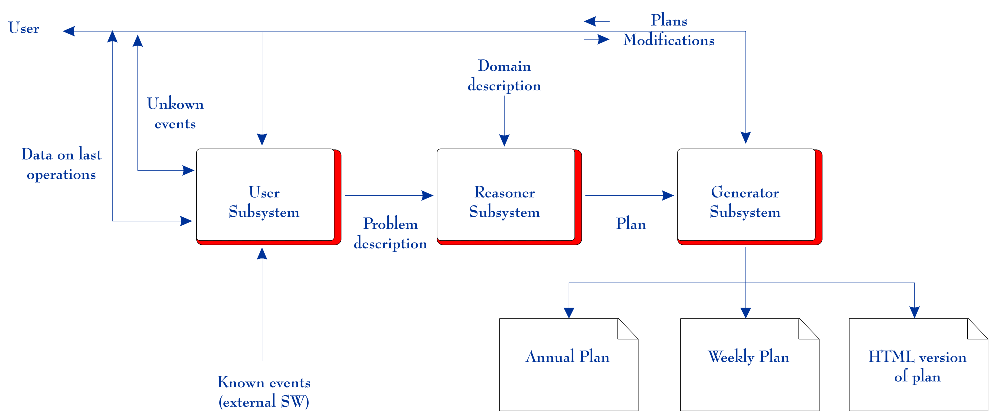


Figure 5.5: Architecture of the planning tool.

specify data in their usual current way and the subsystem automatically translates it into the internal model.

This subsystem provides the following more specific functionality:

- In the case of external events and SOUTH MANEUVERS, the events are known in advance by means of specific supporting engineering tools that predict, according to some parameters, when the events will occur and the hour at which the South Maneuver can be performed every day of the year. These software tools generate the data in a set of ASCII files. The subsystem allows to import these files and detect the correct file format.

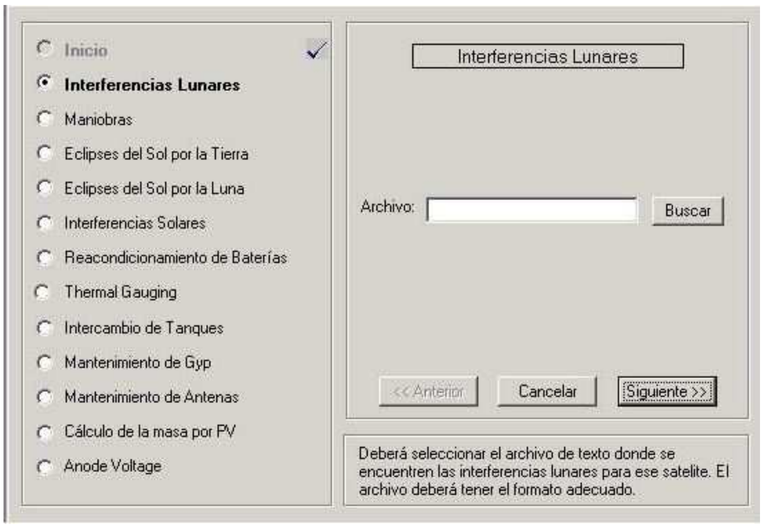
- Nowadays, the engineers represent this data in a specific document, and schedule the operations according to it. Every time a new modification is done, the engineer in charge of it must sign the document. Therefore this subsystem also controls the user access to register who is creating, revising, modifying or verifying the schedule.

- Related to the operations belonging to the third type of operations (section 5.1.3), they require to know the data at which they were performed the last time, so the engineers must find in the last year's document when these operations were performed. If the tool has been previously used, this data is available to the user subsystem and automatically re-used.

- Finally, it assists the user in validating the introduced data. This allows to avoid the possible user inconsistencies, such as trying to swap twice the same tank in the same period of the year or perform an operation in an illegal period.[4] Figure 5.6 shows one of the interface windows (written in Spanish due to engineers requirements) of the User Subsystem. It shows in the left part of the window an ordered list of operations that require user inputs. In the right hand side, the system asks for specific data for each operation, such as a file

[4]A legal period in *tank swapping* is a date around equinox, for instance.

(as shown in the Figure) or the last performed operation if it is not available in the database. The user must complete each step on the left part of the window in order to obtain an annual schedule.



Deberá seleccionar el archivo de texto donde se encuentren las interferencias lunares para ese satelite. El archivo deberá tener el formato adecuado.

Figure 5.6: One of the windows of the User Subsystem interface.

5.3.2 The Reasoner Subsystem

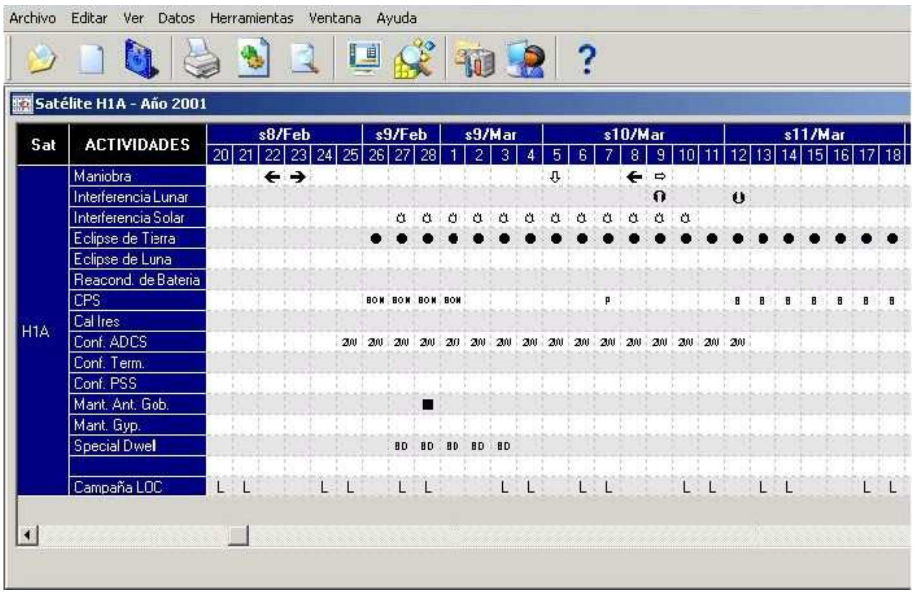
The User Subsystem translates all this information into a suitable format for input to the Reasoner Subsystem. This system is composed of the AI planner explained in Chapter 3. It is in charge of the plan generation, with temporal and resource reasoning. It generates a problem file with all the information introduced by the user. This file joined to the already defined domain and control knowledge files allows to run the planner. The planner output is saved into another ASCII file that will be manipulated to generate the input to the Generator Subsystem.

5.3.3 The Generator Subsystem

Currently, once the engineers know every external event and have represented them in a document, the laborious task of scheduling every operation starts. They generate two types of documents:

- A document which provides an overview of all the operations performed each day of the year; and
- A document that represents in more detail each operation duration, type of manoeuvre, the resources, if any, affected in the operation as batteries or tanks, etc.

The weekly representation is generated every week having in mind the annual representation (shown in Figure 5.7). The problem of maintaining two documents relates to the incongruencies between them. Also, these documents are generated by the engineers of the headquarters at Arganda (Madrid) and sent to other backup centers in Spain and South America. In case any problem arises at the headquarters, one of those other backup centers must continue performing the scheduled operations. Therefore, everytime a change on a schedule occurs, it must be sent to the rest of backup centers.



Sat	ACTIVIDADES	s8/Feb						s9/Feb			s9/Mar				s10/Mar							s11/Mar						
		20	21	22	23	24	25	26	27	28	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18
	Maniobra			←	→										⇩			←	⇨									
	Interferencia Lunar																		∩			∪						
	Interferencia Solar								Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö	Ö								
	Eclipse de Tierra							●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●	●
	Eclipse de Luna																											
	Reacond. de Bateria																											
	CPS							BON	BON	BON	BON						P					B	B	B	B	B	B	B
H1A	Cal Ires																											
	Conf. ADCS						2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0	2/0						
	Conf. Term.																											
	Conf. PSS																											
	Mant. Ant. Gob.									■																		
	Mant. Gyp.																											
	Special Dwell								BD	BD	BD	BD	BD															
	Campaña LOC	L	L			L	L		L	L			L	L		L	L			L	L		L	L			L	L

Figure 5.7: The Generator Subsystem interface: annual representation.

The Generator Subsystem guarantees the consistency of the two representations allowing to easily modify the results obtained by the planner by just dragging and dropping the symbols in the table of the annual or weekly representations. It also allows to: compare two solutions, showing the differences between them; convert the results to the document format that the engineers use in its daily operation; and generate the solution in HTML for visualisation at the other centers.

5.4 Experimental Results

We have done some experiments to show the performance of the tool developed for HISPASAT on real data (we have used the real data of year 2002). We have varied the number of satellites and measured the time performance in seconds, the number of goals, the number of expanded nodes and the number of operations (operator instantiations) in the plan. Tables 5.2, 5.3 and 5.4 show the results obtained increasing the number of the same satellite type (1A, 1B and 1C) from two to five.

Table 5.5 shows the results obtained for each satellite and the three satellites at

Table 5.2: Time in seconds for the 1A type of satellite, varying from one to five satellites.

N. of satellites	Time (sc)	Goals	Initial state	N. of nodes	Ops. in solution
1	548	215	93	1837	411
2	1764	430	186	3990	822
3	4346	645	279	5939	1233
4	9002	860	372	7888	1644
5	16229	1075	465	9831	2055

Table 5.3: Time in seconds for the 1B type of satellite, varying from one to five satellites.

N. of satellites	Time (sc)	Goals	Initial state	N. of nodes	Ops. in solution
1	517	247	91	1947	439
2	1896	494	182	3890	878
3	4527	741	273	5833	1317
4	9194	988	364	7776	1756
5	16349	1235	455	9719	2195

Table 5.4: Time in seconds for the 1C type of satellite, varying from one to five satellites.

N. of satellites	Time (sc)	Goals	Initial state	N. of nodes	Ops. in solution
1	120	221	160	1269	303
2	420	442	320	2567	606
3	1007	663	480	4738	909
4	1945	884	640	7016	1212
5	3849	1105	800	9325	1515

the same time.[5] The big difference in performance between satellite 1C and 1A-1B is due to functions like *is-south-maneuver* (see Figure 5.1) that checks that a maneuver does not interfere with a Moon Blinding. In the 1C satellite operators, these functions are not coded because the software included in the satellite avoids this type

[5]We have not included the results of the 1D satellite because of its recent launching and the different kinds of operations it requires.

of interferences. Also, this difference can be appreciated between satellites 1A and 1B: even if the number of goals for 1B is higher than for 1A, the satellite 1A has two more Moon Blindings (36) that can affect the maneuvers. This is translated into more checking time for each maneuver goal.

Table 5.5: Time in seconds to achieve a plan for each satellite and for the three together

Satellite	Time (sc)	Goals	Initial state	N. of nodes	Ops. in solution
1A	548	215	93	1837	411
1B	517	247	91	1947	439
1C	120	221	160	1269	303
1A-1B-1C	4000	683	344	5045	1153

All these results have been obtained in a Pentium III 800 MHz and 256Mb under Windows. The number of operators in the HISPASAT domain is of one hundred (see appendix C), the number of control rules is ten and a total of fifty coded functions which 25% of these functions are domain independent.

In order to compare it with humans, the time devoted to prepare the annual representation for the three satellites is forty hours a year by one person: basically thirty-five hours to elaborate it and five hours for verification. For the weekly representation, they devote one hour and a half a week and generally not all the changes made in the weekly representation are reflected in the annual representation.

With respect to validation of generated plans, all generated plans are valid. We analyzed differences with the real ones provided by human experts, and the results were that the plans generated by the tool agree in at least 90% with the plans generated by the engineer team. The main differences fall on last time operation changes due to the satellites needs, or coincidence of some operations with holidays which force to move some operations to other dates. This last discrepancy will be considered in future versions of the software.

5.5 Related Work

Several planning systems have addressed applications of planning with limited resources and time considerations. The planner NONLIN+ [178] maintains information about the duration of the activities and represents limited resources. SIPE [191] allows the declaration of the use of resources in the operators. DEVISER [182] could also handle consumable resources. These planners have been designed with the purpose of explicitly handling some types of resources. In the IPP planner [97], based on Graphplan [15], the search algorithm has been modified to combine the ADL search algorithm to handle logical goals together with interval arithmetic to handle resource goals. Realplan [174] also based on Graphplan decouples the logical reasoning from resources reasoning thanks to the CSP formulation that allows helping the planner in the resources allocation.

This is the main difference with PRODIGY that has been designed as a general planning and learning system rather than a scheduler. But this is not a disadvantage for representing this type of knowledge and reasoning about it without modifying the search algorithm.

Other more modern approaches have integrated planning and scheduling in the same system as HSTS [120] by instantiating state-variables into a temporal database. These state-variables are very different to predicate logic formulae as used by PRODIGY, making the representation less powerful. Also the O-PLAN2 [177] integrates planning, scheduling, execution and monitoring within a system that also uses an activity based plan representation. Based on HTN, it requires a lot of hand coding of domain operations and refinement of these operations in user hierarchical levels. Other systems are temporal-based planners, such as IxTeT [76]. It is a least commitment planner that uses a graph-based algorithm for detecting resource conflicts between parallel actions. The time logic relies on a restricted interval algebra represented by Time Points. Time Points are seen as symbolic variables where temporal constraints can be posted. The world is described by a set of multi-valued domain attributes temporally qualified into instantaneous events and persistent assertions and a set of resource attributes. SPIKE [91] was initially used for science operations for the Hubble Space Telescope ground system, but then it was adapted to schedule a variety of astronomical scheduling problems. It has faced the problem using Constraint Satisfaction. MAPGEN [1] is part of the Mars Exploration Rover mission. It is a system that merges ideas from Constraint Satisfaction as propagation and consistency checking and Planning. The planner system is called EUROPA [70]. It is an evolution of the Deep Space One planner [121]. The partial plan that it builds consists of a set of intervals, connected by constraints. This plan is modified using search-based methods until the plan is a valid one.

They all differ from our approach in that they had to (re-)implement the planners to handle scheduling information, while we have used a standard planner and defined a set of simple time handling functions that allow to introduce time management in an intuitive way almost equivalent to the expressive power of other tools.

Also ASPEN [152] uses an AI planner with a richer representation in activities that easily allows the introduction of start times, end times, duration and use of one or more resources. Some algorithms that this planner uses to solve the problem belong basically to the scheduling area (i.e., the schedule iterative repair algorithm). It is part of CASPER (Continuous Activity Scheduling Planning Execution and Replanning) [37] that allows working in dynamic environments as most of NASA projects [38, 39, 40, 61, 193] require. On the other hand, our planner uses specific algorithms of the planning area that can handle a subset of the scheduling problem that is enough for solving tasks in the HISPASAT domain. We do not need replanning, operations monitoring or execution as the operations are not as critical as on-board missions and the planning execution time is not a high priority issue. The flexibility to schedule the operations is one of the main features of this domain as opposite to on-board missions where maximizing the weighted sum of the scheduled observations is one of the main issues.

A quite similar domain as the one presented in this article is shown in the Ground Processing Scheduling System (GPSS) [49]. The way they face the problem differs from ours in several aspects. First, GPSS takes as inputs the set of precedence relation between tasks. It allows four types of temporal constraints with intervals between two tasks: activity-A *finishes* before the *start* of activity-B, the *start* time of activity-A is equal to the *start* time of activity-B, the *finish* time of activity-A coincides to the *finish* time of activity-B and the *start* time of activity-A coincides to the *finish* time of activity-B. Second, resources are modelled as classes separately with an initial capacity. Third, to handle changes produced by tasks, are represented as attributes instead of predicate logic as in our approach. Fourth, a solution is a schedule where each task has a start and end time, a resource assignment for every resource that the activity consumes where all temporal constraints are satisfied. When looking for solutions GPSS searches through the space of all possible schedules, looking for better schedules thanks to the defined cost functions based on expert heuristics.

In our approach we have as inputs the domain theory and the problem definition. For our domain it is not enough the four precedence relations between tasks as we need other Allen primitives as *during* or *overlaps* and we found it easier to model these relationships between activities inside the domain theory. We could also add intervals between tasks thanks to the coded functions. About the resources, these are part of the causal reasoning so PRODIGY as most of the planners consider discrete resources like robots, fuel or batteries as logical predicates. This causes the search space to become huge when the number of resources increases. As experimental results showed in [174] this strategy severely curtails the scale-up of existing planners. Given that the HISPASAT domain does not deal with a high number of resources, it does not affect PRODIGY performance. With respect to how to represent changes, in PRODIGY as any other classical planners, an operator consists of preconditions (conditions that must be true to allow the action execution), and post-conditions or effects (usually composed of an add list and a delete list). The add list specifies the set of formulae that become true in the resulting state, while the delete list specifies the set of formulae that are no longer true and must be deleted from the description of the state. A transition function *f* maps states s into states s´. As a result, the planner generates an ordered set of actions that, when executed in any world satisfying the initial state description, will achieve the goals. Actions have well defined their start and end times and resource assignments for each activity. The obtained plan does not consider any optimisation with respect to resource use or availability, given that for HISPASAT it is enough to find a plan. However, as mentioned before we could also reason about quality-oriented plans according to some criteria using QPRODIGY [20].

5.6 Summary

We have provided an overview of some planning and scheduling problems that we had to face in order to provide a solution to the nominal operations of HIS-

PASAT using a planner since we needed to represent time information and constraints through the Allen primitives, and handle resources.

We have developed a tool, CONSAT to represent through an user-friendly interface the data obtained by the planner when scheduling the operations of its three satellites. Among its features, we can mention that it can:

- Import internal working files and detect the correct file format.
- Control the user access to the tool and register who is creating, revising, modifying or verifying the schedule.
- Save data previously used in other years and make it available to the user and automatically re-used if needed.
- Avoid incongruencies between the two type of representations used in HISPASAT: weekly and annual views. This were generated by hand and in paper until now.

Finally, we have done some experiments to show the performance on real data in the HISPASAT domain, showing that it is a viable solution for the needs of the company.

Chapter 6

Representing Time and Resources

From the experience gained in the modeling of two real domains, we want to study in this chapter, several ways to represent time and resources using different representations. First, we describe the time aspect of the scheduling using as an example the HISPASAT domain. Then, we describe some issues about resource usage [144, 145].

6.1 Representation of Time Constraints

Among the features that makes real domain complex, we will focus on time and resources. In this section we will explain the way to represent time in PRODIGY (by means of the PDL4.0 syntax) and how this is done using a scheduler, formulated as a Constraint Satisfaction Problem (CSP). In some cases, we will also use the PDDL2.1 syntax.

6.1.1 Time Models

The time representation of PRODIGY is a discrete model of time, in which all actions are assumed to be instantaneous and uninterruptible. There is no provision for allowing parallel actions. However, the functions that can be called within the assignment of values to variables in the preconditions of the operators allow to add constraints among and within operators. Using them, PRODIGY can handle the seven Allen primitive relations between temporal intervals [7] and some quantitative relations[1] [50, 115].

PDDL2.1 is also a discrete model of time in the levels used in the competition that considers time representation at level 3 by means of temporally conditions and effects in a durative action. The specification of pre- and postconditions, and the fact that invariant conditions can be identified, means that it allows concurrent behaviour if it is avoided that another action accesses a variable at the exact point at which it is updated by another action. So conflicts over variables can only occur at

[1]Due to the fact that it cannot return infinite possible values for variables and it does not reason about variables that represent intervals, it cannot handle all quantitative relations as CSP does.

the start and end points of actions. In the preconditions, the propositions can be asserted at the start of the interval (the point at which the action is applied), at the end of the interval (the point at which the final effects of the action are asserted) or over the interval from the start to the end (invariant over the duration of the action). In the effects, the proposition can be immediately applied (it happens at the start of the interval) or delayed (it happens at the end of the interval).

On one hand, this representation provides more expressivity to the domain modeler than the PDL4.0 syntax does, but a durative action with the features mentioned above can be subdivided into two STRIPS actions, one for each of the end-points of the durative action in case there are no invariant conditions. If the action specifies invariant conditions it is necessary to guarantee the truth over the interval to avoid conflicts. In PRODIGY this is not a problem due to its serial plan nature but in Partial Order Plans it must be had in mind as shown in [44]. On the other hand, this language presents much more restrictions to represent the Allen primitives than PDL4.0, because it is not able to assign values to variables through coded functions, nor to return a set of values through an interval.

In scheduling systems there is not a standard language, but the wide extended representation as a CSP makes it very easy to handle time and resources. Each operator is represented with two time points: one time point represents its start time and the second the end time. Each time point is represented as an interval of possible values so all quantitative and qualitative relations between them can be perfectly managed.

In spite of the expressivity of the PDL4.0 language, it is not enough to represent the temporal and resources aspects needed for the integration of planning and scheduling. For this reason we propose an extension of the PDL4.0 language, valid also for the PDDL2.1 syntax. Other authors have proposed [12, 69] alternative solutions. This extension is presented in Appendix G.

6.1.2 Allen Primitives

In order to compare the representations mentioned above, we have grouped the seven Allen primitive relations in the following types, and show how they will be done for each one, (for simplicity, in PRODIGY we have reduced the syntax). We have used the HISPASAT domain independent functions that Table 6.1 shows. There are some functions that return a value and others that return a finite number of possible values in an interval, so these type of functions are obviously discrete. But, the values returned can be as many as one needs so, at the end, they can be thought as equivalent in some practical sense to a continuous representation. Also, while in the planning notation, a value is assigned and it is possible to establish constraints, with infinite quantities it is hard to assign a value (commitment) while in the CSP representation constraints are established much easier.

- OperatorA has a *duration*: although this is not an Allen primitive, we have included here because PDDL has added this field in the operators (*durative actions*). The duration is the time required for OperatorA to be executed. This

Table 6.1: PRODIGY domain independent coded funtions to handle time.

Name	Meaning
add-time-in-interval <d> ELAPSED^1 TIME^1 ELAPSED^2 TIME^2	returns a finite set of possible values for the variable <d>in the interval ELAPSED^1 and ELAPSED^2 placed to the right of the value of variable <d>. TIME^1 and TIME^2 represent time units, that is: hours, minutes, seconds, etc. ELAPSED^1 and ELAPSED^1 must be positive numbers.
del-time-in-interval <d> ELAPSED^1 TIME^1 ELAPSED^2 TIME^2	returns a finite set of possible values for the variable <d>in the interval ELAPSED^1 and ELAPSED^2 placed to the left of the value of variable <d>. TIME^1 and TIME^2 represent time units, that is: hours, minutes, seconds, etc. ELAPSED^1 and ELAPSED^1 must be positive numbers.
add-time <st> DUR TIME	returns a value that is the sum of the variable <st> plus DUR. TIME specifies the time unit, so DUR will be converted to seconds according to the time unit passed as a parameter.
del-time <st> DUR TIME	returns a value that is the subtraction of the variable <st> minus DUR. TIME specifies the time unit, so DUR will be converted to seconds according to the time unit passed as a parameter.

value in PRODIGY is used to calculate the end/start time of OperatorA. This can be represented using the function *add-time* (DUR is a number, integer or not, that represents the duration, and TIME represents the time units) as shown in Figure 6.1. In this example, the operator constrains the value of variable *<start-time>* by querying the current state where the literal *start-process-op-A* will be grounded. The function *add-time* adds the *duration* of the operation to the start time of OperatorA, obtaining its end time. In QPRODIGY [20], duration can be defined as a quality metric, and hence, it can reason explicitly about it. To see the details of representing duration as a metric go to Appendix F. In PDDL2.1 the durative actions have a field to represent durations as Figure 6.2 shows.

In the case we use the CS representation, each activity/ operator is represented by two time points: the *earliest start time* (*est*) and the *latest finish time* (*lft*) of the activity. If we want to say that such activity has a duration equal to DUR, we just need to impose between *lft* and *est* that value as shown in Figure 6.3.

- The end time of OperatorA occurs before the start time of OperatorB within a range of time in the interval [a, b]. The following Allen relations belong to this type: OperatorA *before* OperatorB and OperatorA *meets* OperatorB (where a=b=0). The elapsed time from the end of OperatorA can be a value that can

```
OperatorA
preconds:  (<start-time> (start-process-op-A <start-time>))
           (<end-time> (add-time <start-time> DUR TIME))
effects:  (add (started-A <start-time>))
          (add (finished-A <end-time>))
```

Figure 6.1: A representation of the Allen duration primitive in PRODIGY.

```
(:durative-action OperatorA
 ...
 :duration (= ?duration (+ (start-process-op-A) DUR))
 ...
)
```

Figure 6.2: A representation of the Allen duration primitive in PDDL.

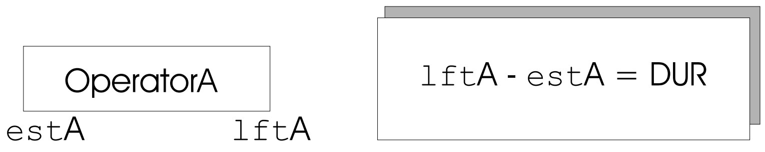


Figure 6.3: The Allen duration primitive using the CS representation.

be constrained, in the general case, by an interval [a, b]. The limits can be zero or positive numbers. The way this can be represented in PRODIGY is shown in Figure 6.4 (variables DUR and DUR^0 represent the duration, $ELAPSED^1$ and $ELAPSED^2$ represent the minimal and maximal values of the interval [a, b] and TIME, $TIME^0$, $TIME^1$ and $TIME^2$ represent time units). The function *add-time-in-interval* (see Table 6.1) returns a finite possible values in that interval. For instance, if we want to represent that OperatorB is between four and five minutes *after* OperatorA, the call to function *add-time-in-interval* should be: *(add-time-in-interval <d> 4 minutes 5 minutes)*, where <d> represents the end time of OperatorA. For simplicity we suppose that the start time of OperatorA is given explicitly in the initial conditions of the problem by the predicate/function *start-process-op*.

The representation using the CS notation is shown in Figure 6.5.

```
OperatorA
preconds:  (<start-time> (start-process-op-A
                                       <start-time>))
           (<end-time> (add-time <start-time> DUR TIME))
effects:  (add (finished-A <end-time>))
OperatorB
preconds:  (<d> (finished-A <d>))
           (<start-time> (add-time-in-interval <d>
                          ELAPSED¹ TIME¹ ELAPSED² TIME²))
           (<end-time> (add-time <start-time> DUR⁰ TIME⁰))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 6.4: A PRODIGY representation of the A *before* B and A *meets* B Allen primitives.

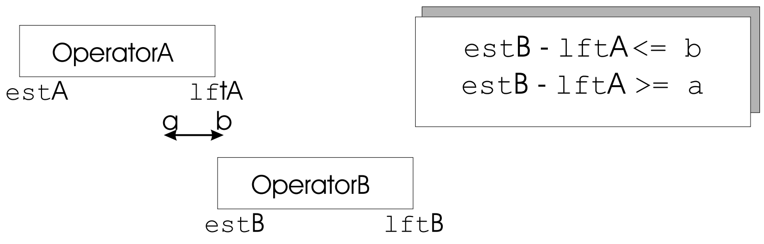


Figure 6.5: Graphical and CS representation of A *before* B and A *meets* B.

In the case of the PDDL2.1 syntax, we need to declare four functions that represent the start and end time of OperatorA and OperatorB. By defining them as functions, we can modify their values in the effects and do arithmetic and logical operations in the preconditions. Figure 6.6 shows how to represent these primitives in PDDL2.1.

```
(:functions(start-A)
           (end-B)
           (start-B)
           (end-B))
```

```
(:durative-action OperatorA
 :duration (= ?duration DUR)
 :condition (at start (start-process-op-A))
 :effect (and (at end (assign (end-A)
                              (+ (start-process-op-A) DUR)))))

(:durative-action OperatorB
 :duration (= ?duration DUR0)
 :condition (at start (and (>= (+ (end-A) ELAPSED1)
                                start-process-op-B)
                           (<= (+ (end-A) ELAPSED2)
                                start-process-op-B)))
 :effect ...  )
```

Figure 6.6: A PDDL2.1 representation of the A *before* B and A *meets* B Allen primitives using only the lower bound of the interval.

- The start time of OperatorA is the start time of OperatorB. The following Allen relations belong to this type: OperatorA *starts* OperatorB and OperatorA *equals* OperatorB. In the case we would like to represent the *equals* relationship we should also constraint in PRODIGY the end time of the operations. Then, the elapsed time from the start of OperatorA to the end of the operation can be a value that can be constrained by an interval [a, b] as in the previous case. The way to represent this is depicted in Figure 6.7, while Figure 6.8 shows the CS representation.

  In PDDL2.1 we can impose the same start time in both operations and the duration of the OperatorB can be greater than the end time of the OperatorA minus ELAPSED^2 and less than the end time of the OperatorA plus ELAPSED^1. If ELAPSED^2 and ELAPSED^1 are equal to zero, the duration of OperatorB is equal to the duration of OperatorA and then it represents the OperatorA *equals* OperatorB primitive (Figure 6.9).

- The end time of OperatorA is the end time of OperatorB. The OperatorA *finishes* OperatorB belongs to this category. The case OperatorA *equals* OperatorB could have also been in this category (where a=b=0). In PRODIGY the start time of OperatorB computed from the end of OperatorA can be a value that can be constrained, in the general case, by an interval [a, b]. The limits can be zero or positive numbers. The function *del-time-in-interval* (see Table 6.1) returns all possible values in that interval. This is represented in Figure 6.10 and its corresponding to CS is shown in Figure 6.11.

```
OperatorA
preconds:  (<start-time> (start-process-op-A
                                   <start-time>))
           (<end-time> (add-time <start-time> DUR TIME))
effects:  (add (started-A <start-time>))
          (add (finished-A <end-time>))

OperatorB
preconds:  (<start-time> (started-A <start-time>))
           (<end-time> (add-time-in-interval <start-time>
                        ELAPSED1 TIME1 ELAPSED2 TIME2))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 6.7: A representation of the A *starts* B and A *equals* B Allen primitives in PDL4.0.

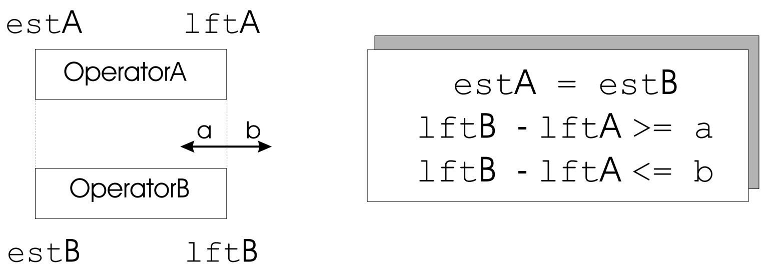


Figure 6.8: Graphical and CS representation of Figure 6.7.

```
(:durative-action OperatorA
 :duration (= ?duration DUR)
 :condition (at start (start-process-op-A))
 :effect (and (at end (assign (end-A)
                              (+ (start-process-op-A) DUR)))

(:durative-action OperatorB
 :duration (and (>= ?duration (- (end-A) ELAPSED1))
                (<= ?duration (+ (end-A) ELAPSED2))
 :condition (at start (start-process-op-A))
 :effect ...  )
```

Figure 6.9: A representation of the A *starts* B and A *equals* B Allen primitives in PDDL2.1.

```
OperatorA
preconds:  (<start-time> (start-process-op-A
                                       <start-time>))
           (<end-time> (add-time <start-time> DUR TIME))
effects:  (add (finished-A <end-time>))
OperatorB
preconds:  (<end-time> (finished-A <end-time>))
           (<start-time> (del-time-in-interval <end-time>
                         ELAPSED1 TIME1 ELAPSED2 TIME2))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 6.10: A representation of the A *finishes* B and A *equals* B Allen primitives in PRODIGY.

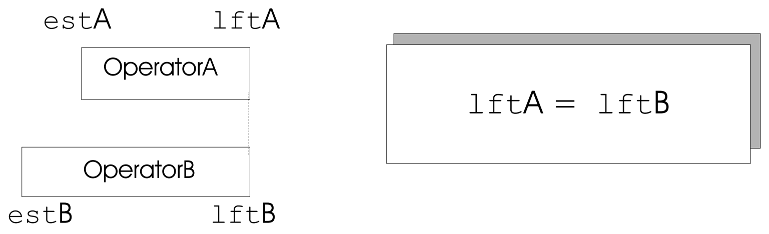
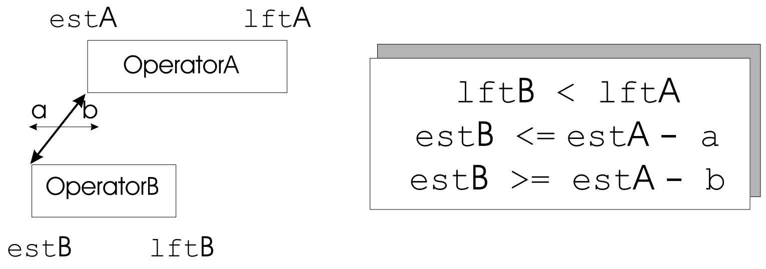


Figure 6.11: CS representation of the A *finishes* B and A *equals* B Allen primitives.

- The start time of OperatorB is in a given range from the start of OperatorA, that is, OperatorA *overlaps* OperatorB. The PRODIGY representation is shown in Figure 6.12 and the CS representation in Figure 6.13.

```
OperatorA
preconds:  (<start-time> (start-process-op-A
                                    <start-time>))
           (<end-time> (end-operation <start-time> DUR TIME))
effects:  (add (started-A <start-time>))
          (add (finished-A <end-time>))
OperatorB
preconds:  (<d> (started-A <d>))
           (<start-time> (add-time-in-interval <d>
                          ELAPSED$^1$ TIME$^1$ ELAPSED$^2$ TIME$^2$))
           (<end-time> (add-time <start-time> DUR$^0$ TIME$^0$))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 6.12: A representation of the A *overlaps* B Allen primitive in PRODIGY.



```
lftB < lftA
estB <= estA - a
estB >= estA - b
```

Figure 6.13: A *overlaps* B Allen primitive representation using CS.

- The start time of OperatorB occurs after the start time of OperatorA and its end time occurs before the end time of OperatorA. The Allen relation that belongs to this type is OperatorA *during* OperatorB. The way to represent this last category is depited in Figure 6.14 where the variables ELAPSED^3 and ELAPSED^4 represent the minimal and maximal values of the interval [a, b] and TIME^3 and TIME^4 represent time units. The CS representation of this primitive is shown in Figure 6.15.

```
OperatorA
preconds:  (<start-time> (start-process-op-A
                                        <start-time>))
           (<end-time> (end-operation <start-time>
                                     DUR TIME))
effects:  (add (started-A <start-time>))
          (add (finished-A <end-time>))
OperatorB
preconds:  (<d> (started-A <d>))
           (<d1> (finished-A <d1>))
           (<start-time> (add-time-in-interval <d>
                         ELAPSED¹ TIME¹ ELAPSED² TIME²))
           (<end-time> (del-time-in-interval <d1>
                       ELAPSED³ TIME³ ELAPSED⁴ TIME⁴))
effects:  (add (started-B <start-time>))
          (add (finished-B <end-time>))
```

Figure 6.14: A representation of the A *during* B Allen primitive.

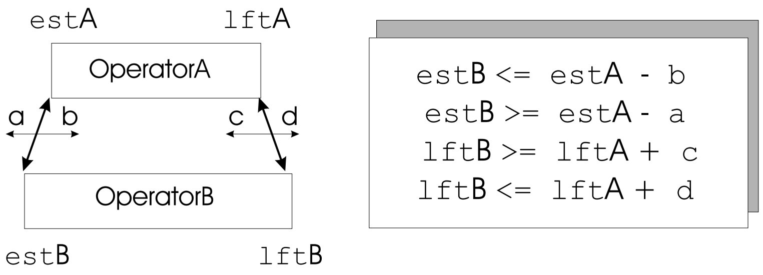


Figure 6.15: A *during* B primitive using CS.

6.1.3 Enriching Time Constraints

These relations between operators described in the previous section, are defined within a time window and considering that there is one instance of these relations in that window. Because in HISPASAT (as well as in some other real domains) there is more than one time window defined with several instances of the same type on it, we need to define two predicates for each operator to control the instantiated relations:

- *index-operatorX*: a pointer to the particular instance of each operator.
- *last-index-operatorX*: the index of the last instantiated operator.

Also we need to define one more predicate for each temporal relation between two operators:

- *used-index-operatorX-for-operatorY*: an index related to an instantiated operatorX in relation to another operatorY.

So for each of the Allen primitives described in this section we should add the predicates explained above as Figure 6.16 shows.

```
                  Initial State
∀ OperatorA ⟶ (last-index-opA 0)
                 (used-index-opA-for-opB 0)
OperatorA
preconds:  (<last-index> (last-index-opA <last-index>))
           (<index> (+ 1 <last-index>)
           ...
effects:  (add (index-opA <index>))
          (del (last-index-opA <last-index>))
          (add (last-index-opA <index>))
OperatorB
preconds:  (<index> (index-opA <index>))
           ( not (used-index-opA-for-opB <last-index>))
           ...
effects:  (add (used-index-opA-for-opB <index>))
```

Figure 6.16: A way of handling instances for any Allen primitives.

Given this solution we could build an interface that defines automatically these high level relations between operators and translates these relations into instances following the notation described above.

A quite similar problem happens in CSP, because we need to generate all the activities (by hand or using a program) that need to be scheduled in the plan.

6.2 Representation of Resources Constraints

A resource is a description of a profile of a physical device over time. There are basically three types of resources as the ASPEN creators [163] describe, although the name given to each type varies from one author to another:

- Type 1: it is available when not in use (one user at a time). Examples are: physical devices such as a robot arm or a CPU. In ASPEN [163] another type is the *concurrency* resource that is similar to Type 1 except that they must be made available to the activity before they are reserved. An example would be a tele-communications downlink pass. The telecommunications station must be available before the spacecraft could initiate a downlink.
- Type 2: it can be used for more than one activity, so a capacity should be defined. It is always available when not in use and many users can use different quantities of it. It does not need to be replenished as, for example, solar array power.
- Type 3: the capacity is diminished after its use, so a capacity should be defined. It may or not be replenished by another activity. Examples are: battery energy, memory capacity (can be replenished) and fuel (it cannot for satellite missions).

In PRODIGY, as in PDDL2.1, there is no provision for specifying resource requirements or consumption. But resources can be seen as variables that can have associated values through literals that refer to them, that is as logical formula. This way of resource representation has the disadvantage of making the search extremely intractable when the number of resources increases as the results in [174] and in Chapter 8 of this book show.

For example, if tank T1 of a satellite has 30 litres of fuel in a given instant, we could represent it as *(has-fuel T1 30)*. Then, in the operators, we can restrict the set of values that can be assigned to the variable that represents the resource, consume it, or refuel it if necessary and possible.

To represent capacity, we can use the scalar quantity model. The capacity constraints of a resource with uniform capacity can be calculated in PRODIGY through a functional expression. For instance, we can define a function (i.e. *compute-consumed-fuel*) that calculates the fuel consumed in each maneuver as a function of the available fuel and the angle of the satellite with respect to the sun as shown in Figure 6.17.

The value of the angle is obtained from the instantiation of the literal *(position <sat> <angle>)* that must be greater that the value four (4 grades), with respect to the current state. Once it sets the value of variable <sat>, it will access the state and set the value of the second argument as the value for variable <angle>. Also, the value of the available fuel is obtained from the instantiation of the literal *(available-fuel <sat> <available>)* with respect to the state, for the instantiated variable <sat>.

In the effects, the operator adds the grounded literals *(performed-maneuver <sat>)* and *(available-fuel <sat> <new-fuel>)* to the state, i.e. it asserts them as true, and deletes *(available-fuel <sat> <available>)* from the state.

```
OperatorManeuver-S
preconds:  (<angle> (position <sat> <angle>)
                     (> <angle> 4))
           (<available> (available-fuel <sat> <available>))
           (<fuel-consumed> (compute-consumed-fuel
                                <available> <angle>))
           (<new-fuel> (- <available> <fuel-consumed>))
effects:  (add (performed-maneuver <sat>))
          (add (available-fuel <sat> <new-fuel>)))
          (del (available-fuel <sat> <available>)))
```

Figure 6.17: How to represent a resource capacity and consumption such as fuel in PRODIGY for the satellite domain.

In PDDL2.1, the functions *decrease/increase* present a compact way of handling numeric fluents, more compact than PRODIGY, but the semantics and the way to use resources consumption is the same, as Figure 6.18 shows.

```
(:action Maneuver-S
 :parameters (?sat - satellite)
 :precondition (> (position ?sat) 4)
 :effect (at end (performed-maneuver ?sat))
         (at end (decrease (available-fuel ?sat)
                           (/(available-fuel ?sat) (position ?sat)))))
```

Figure 6.18: PDDL2.1 representation of Figure 6.17 operator.

In the case of binary resources, in PRODIGY and PDDL2.1 we can model them as predicates that must be available at the beginning of the operation. The operation will remove its availability from the state (we can do that because of the instantaneous representation of actions) so other operations cannot use it, but we need to add an operation that sets the resource free, so it can be used again. Figures 6.19 and 6.20 show an example of a binary resource as it can be an instrument inside a satellite. This example belongs to the satellite domain of the planning competition [87]. In this case, scientific instruments in satellites must collect some data as images. In order to observe some data, the instrument inside the satellite must be free. The operation adds to the state that the instrument is busy. In order to use it again, the Free-Resource operation will add to the state its availability for other operations (the *(del (busy <instrument>))* or *(not (busy ?i))* grounded literals).

```
Take-Data
preconds:  (belong <instrument> <sat>)
           (not (busy <instrument>))
effects:  (add (busy <instrument>))
          (add (have-image <mode> <direction>))
Free-Resource
preconds:  (busy <instrument>)
effects:  (del (busy <instrument>))
```

Figure 6.19: An example in PRODIGY of how to represent a binary resource.

```
(:action Take-Data
 :parameters (?sat - satellite ?d - direction
              ?i - instrument ?m - mode)
 :precondition (and (belong ?i ?sat)
                    (not (busy ?i))
 :effect:  (and (busy ?i)
                (have-image ?m ?d)))
(:action Free-Resource
 :parameters (?i - instrument)
 :precondition (busy ?i)
 :effect:  (not (busy ?i)))
```

Figure 6.20: The example of Figure 6.19 in PDDL2.1 syntax.

In CS scheduling, there are many algorithms and heuristics to handle from binary to multicapacity resources easily as the ones proposed in [28, 31, 169].

6.3 Summary

Real domains have many features that make them complex. Among others we can mention time and resources. In this chapter we have described the way to represent time in PRODIGY (by means of the PDL4.0 syntax) and how this is done using a scheduler, formulated as a Constraint Satisfaction Problem (CSP). We have also used the PDDL2.1 syntax.

Because resources are usually also required, we have described some issues about resource usage and how to represent multicapacity as well as binary resources. PDL4.0 although is not a standard language, allows functions to be coded and called within the assignment of values to variables in the preconditions of the operators so adding constraints among and within operators is quite easy.

PDDL presents more restrictions to represent the Allen primitives than PDL4.0, because it is not able to assign values to variables through coded functions, nor to return a set of values through an interval. But it is a richer expressive modeling language as to identify invariant conditions what means that it allows concurrent behaviour if it is avoided that another action accesses a variable at the exact point at which it is updated by another action. Then, conflicts over variables can only occur at the start and end points of actions.

CSP has not a standard language, because actions are completely instantiated but constraints over time and on resource consumption are easily handled.

To conclude, real domains need techniques from both fields not only from the knowledge representation point of view but also from the methodology used to solve the problems.

Part III

Integrating Planning and Scheduling

Chapter 7

Architectures for the integration

In the previous part of the dissertation we have focused on KR aspects for real world tasks. These type of domains require the combination or integration of tools and techniques from planning and scheduling. Examples of such domains were presented in chapters 4 and 5. Workflow applications require to generate an ordered set of activities that define a process in an organisation and the assignment of resources (human or material) to these activities [123, 138, 142, 143, 146, 148]. Space applications [1, 40, 61, 90, 147, 150, 151, 193] due to the number of operations that have to be performed at specific instants on time, they must try to optimise the total usage of the resources used in the available time.

Integrating planning and scheduling is becoming an increasingly interesting topic in AI due to real application problems. Since some problems allow a strict separation between planning and scheduling, the usual approach to solve the complete planning problem (a valid plan with temporal and resource information) consists on assigning time and resources (using a scheduler) to the selected and ordered activities[1] (generated by a planner). But, in other cases, time and resource assignments affect the selection and ordering of activities. Since there is usually no feedback from the scheduler to the planner about the temporal-resource invalid plans, these problems have either no solution, or become non-optimal solutions. In this chapter, we present three approaches to integrate a planning system and a constraint-based scheduling system. We vary from a pipe-line of a planner and a scheduler towards the more integrated approach of interleaving planning and scheduling.

7.1 Introduction

From the experience gained in HISPASAT, SHAMASH and COSMOSS, we explored the possibility of using PRODIGY directly for integrating planning and scheduling as described on section 5.2.1. We realised that QPRODIGY was not enough for the needs of these real domains. As we have pointed out, this approach has some disadvantages as domain dependent coded functions, no explicit language to represent time

[1]Given the different terminology used in both fields, we will use the terms operator and activity indistinctly.

or resources, or the solution given as a total order sequence of activities.

It was then, when we started to think on other ways to integrate planning and scheduling. A second approach we have explored consists on using the O-OSCAR scheduler system [27, 32] after the QPRODIGY planner. For a given problem, QPRODIGY generates a plan as a sequence of activities that can minimise some quality criteria. Then, the O-OSCAR scheduler obtains a viable temporal and resource compliant solution. We provide two different solutions within this approach. In the first approach, the total-order plan that QPRODIGY generates is given directly to O-OSCAR. In the second approach, the total-order plan is translated into a partial-order plan to be given to O-OSCAR. A similar approach can be found in [173], though they did not use a planner that is able to handle by itself some types of temporal information as in the case of QPRODIGY. Therefore, their planner cannot avoid generating some types of temporal invalid plans.

We considered the second approach quite naive although valid. Then, we thought about integrating the temporal and resource reasoning of the scheduler in an interleaved execution with the planner, by exchanging relevant information. This approach is quite similar to the one reported in [71]. One difference relies on the fact that our planner can handle some time-resource related information, and it can also reason about some other optimisation criteria. Therefore, we can control/vary the amount of reasoning that each component (planner/scheduler) will perform. Also the efficiency of our system is greater than the one presented in [71].

7.2 Using a planner and a scheduler

In this section we present two approaches that explicitly use a scheduler after a planner and return a plan to perform the same integration. In this case, we leave time-resource reasoning to the scheduler, or can even generate different configurations by varying the amount of time-resource management that we allow to the planner.

7.2.1 QPRODIGY and O-OSCAR

An approach to solve the complete planning and scheduling problem consists of integrating in sequence the planner QPRODIGY and the scheduler O-OSCAR. Figure 7.1 shows the inputs and outputs of this approach. In this case, we remove from the domain all time-related information, letting O-OSCAR take advantage of handling the temporal information. QPRODIGY generates a sequence of atemporal operations that describes the precedence relations among activities. Since QPRODIGY generates a total-order (TO) plan, the precedences that QPRODIGY provides to O-OSCAR are the strict order of the TO plan.

Then, the plan is given to O-OSCAR that assigns a start and end time for each operator once the file is created. The duration of each operator is extracted from the quality metric of each operator when QPRODIGY runs (the details of how to obtain the duration in QPRODIGY are explained in Appendix F). Next, we have called the parser that we have built to translate the QPRODIGY TO plan (without temporal-resource information) into an O-OSCAR input file. It will be in charge of assigning to each activity of the solution its duration and resource consumption. By default, it sets the minimal and maximal distance between activities to zero.

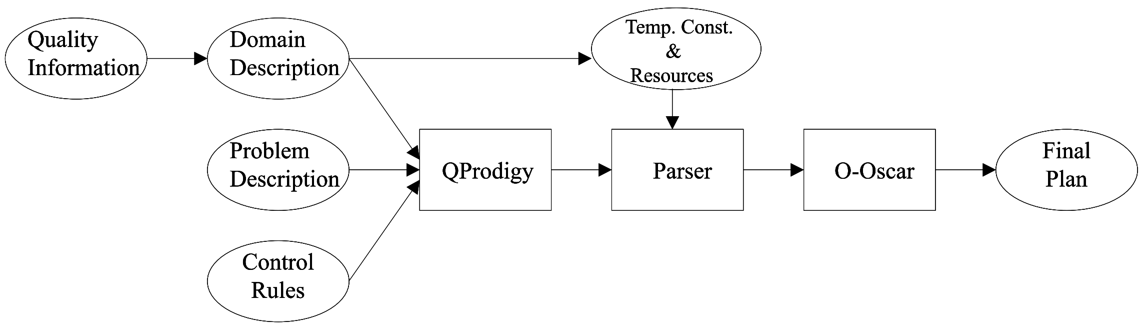


Figure 7.1: Sequential approach of PRODIGY and O-OSCAR.

As an example, in the satellite domain described on section 5.1, one of the operations that is less efficiently solved by QPRODIGY is performing Moon Blinding checks within manoeuvre operations, which requires a coded function to check moon-blindings within two or three hours the expected time of the manoeuver. If there is any, the function should check again if twenty-four hours before and in two or three hours period could exist again a moon-blinding. Using this second approach, we can now handle it by eliminating the function that calculates the blindings from the operators descriptions, and modeling them as binary resources, i.e. sun availability, with two values: 0 if it is available and 1 when it is not available due to any type of blindings. Figure 7.2 shows how the SOUTH-MANEUVER operator of Figure 5.1 would look like when using it in combination with O-OSCAR.

```
(OPERATOR SOUTH-MANEUVER
  (preconds
  ((<s> SATELLITE)
   (<t> TIMES)
   (<t1> TIMES)
   (<p> PERCENTAGE)
   (<n> DIRECTION)
   (<d> (and DATE (gen-from-pred
                  (moon-blinding-start <s> <d> <n> <p> <t1>))))
   (<p1> (and PERCENTAGE (over-n <d> greater 40)))
  (south-maneuver <s> <t> <d1>))
  (effects
   ()
   ((add (south-m <s> <t>))
    (add (south-man <s> <t> <start-time>)))))
```

Figure 7.2: Operator of Figure 5.1 to adapt it to O-OSCAR.

As a result, O-OSCAR tries to locate the maneuver operations where the sun resource is available. The number of total activities in the plan is equal to the number of South Maneuvers that has to be performed in the year plus a number of fictitious activities that need as resource the Moon Blinding availability (that is, the number of Moon Blindings that occur during that year). For each activity, we need to impose the activities constraints of each activity (fictitious or not) and the distance constraints between each pair of activities. Because some data are fixed, we need to impose a fixed distance between the origin and the activity under consideration. Then, the duration of each activity is needed, obtained from the time quality metric defined in QPRODIGY and the resource that each activity consumes. In this case all activities consume the same binary resource, that is, the sun availability. When two activities require the sun availability at the same time (Moon Blinding and South Maneuver), but one of the activities has a fixed start time (Moon Blinding), O-OSCAR will move the south manoeuvre 24 or 48 hours before depending of the location of the Moon Blindings. In section 7.4 we show some results of this architecture in the HISPASAT domain.

The disadvantages of this approach are:

- Not every precedence ordering between plan steps in a TO plan is necessary for maintaining its consistency, given that two activities could be executed in parallel. One way of avoiding this, would be directly using a partial-order planner, such as UCPOP [133]. However, we wanted to maintain the richer representation language of PRODIGY (as shown in section 5.2.1) together with its flexibility to easily inspect the search trees, define control knowledge, reason about quality metrics, or define new alternative control strategies (through the use of functions called handlers) [22].
- Inefficient communication between the two systems: in case of a failure in the scheduler, a new solution has to be generated so the computational cost might be high. It is also difficult to ask for a different solution from the last one.
- There is no way to represent explicitly in the PRODIGY language, or in PDDL2.1, resources, some temporal constraints between activities or define a makespan in the solution.
- We cannot take advantage of the scheduler capabilities (as minimising the makespan) since the obtained plan is very restrictive because it is a TO plan.

7.2.2 QPRODIGY, a TO-PO algorithm, and O-OSCAR

To overcome the first drawback of the last approach, we convert the TO plan provided by QPRODIGY into a Partial Order (PO) plan. A PO plan represents a set of TO plans, where each TO is a linear sequence of steps in the PO such that the PO relation in the latter is not violated by the sequence.

We have followed the algorithm defined by Veloso *et al.* in [181] that takes advantage of the given total ordering of the plan, by visiting at each step, only earlier plan steps. The algorithm is sketched in Figure 7.3. This greedy algorithm constructs an entirely new PO, analysing the action conditions, and using the original TO to guide the greedy strategy. It may fail to produce a minimally constrained deordering as explained in [11] because of step 1(a) in the algorithm. From all possible operators in the plan that achieve an effect needed as precondition of another operator op_j later in the plan, it always selects the closest one before op_j. Since there may be other operators earlier in the plan having the same effect and being a better choice, the algorithm is not optimal. However, given that QPRODIGY (as most current planners) is not an optimal planner either (unless all alternatives are searched for, which can be actually done in QPRODIGY), then the non-optimality of the TO-PO algorithm is not a crucial point in the current state of research.

```
Procedure Build Partial Order (TO-Plan)
Input: TO-Plan: A totally ordered plan (being $n$ the number of steps in the
plan) and the start operator with preconditions set to the initial state.
Output: A partially ordered plan shown as a directed graph

  1. for i=n down-to 1 do
     (a) for each precond $\in$ Preconditions ($op_i$) do
         Find an operator $op_j$ in plan that has as an effect precond
         Add directed edge from $op_j$ to $op_i$
     (b) for each del $\in$ Delete-Effects ($op_i$) do
         Find all operators that have as a precondition (some of the, a)
         delete effects of $op_i$
         Add directed edge from these operators to $op_i$
     (c) for each add $\in$ Primary-Adds ($op_i$) (if it appears either in the goal
         or in the subgoaling chain of a goal proposition) do
         Find all operators that delete any of the primary adds of $op_i$
         Add directed edge from these operators to $op_i$
  2. Remove Transitive Edges in the graph
     Every directed edge $e$ connecting $op_i$ and $op_j$ is removed if there is an-
     other path that connects the two vertices
```

Figure 7.3: Algorithm that transforms a TO into a PO described in [181].

QPRODIGY + Veloso algorithm generates a sequence of atemporal operations that describes the precedence relations among activities. After applying the Veloso algorithm, as a solution we have a graph represented as a bi-dimensional array equal to the number of operators in the solution plus two dummy operators that correspond to the origin operator (initial state) and finish operator (goal state). This array (graph) contains the precedence constraints of each operator with respect to the rest of operators. Then, this graph is translated into the format file that O-OSCAR needs to assign a start and end time for each operator. Its structure allows easily to calculate the activity constraints of each activity. The duration of each operator is extracted from the quality metric of each operator when QPRODIGY runs. Then, the parser fills the file with the duration and resource consumption of each activity (as in the last approach).

This approach is shown in Figure 7.4. As it is also the case of the other approach, it shares the high computational cost and the lack of an explicit representation for time and resources.

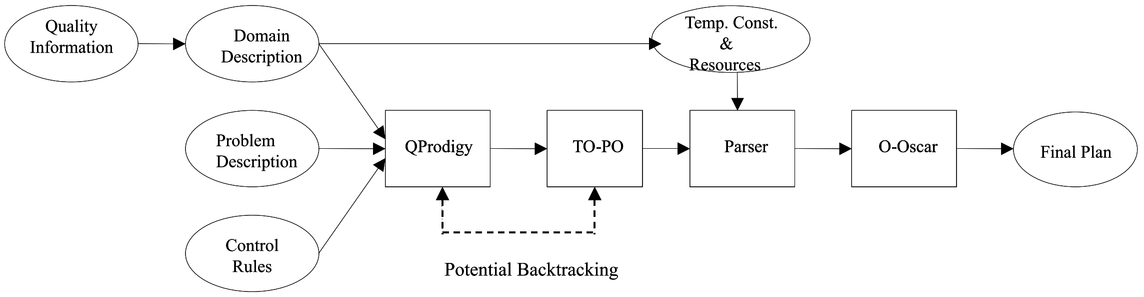


Figure 7.4: Sequential approach of a PO solution of PRODIGY and O-OSCAR.

But there is a little bit more communication between the planner and the scheduler and more flexibility to find an optimal solution since it removes unnecessary ordering constraints from the obtained plan. The algorithm can be extended to look for different deorderings depending on the information that the scheduler gives back, so there are two meta level backtracking points: another deordering from the TO-PO algorithm, or another plan from the planner in case previous alternatives fail to produce a valid schedule.

7.3 IPSS: An Integrated System

The approaches studied above share the disadvantage of the lack of communication between both systems: the planner provides a solution considering a given metric and the scheduler works on that solution trying to obtain an optimal assignment of time and resources. Therefore, we propose the hybrid reasoner IPSS (Integrated Planning and Scheduling System) shown in Figure 7.5. Both systems interleave information during the solving process.

In IPSS the reasoning is subdivided in two levels. The planner focuses on the actions selection (possibly in the optimisation of some quality metric different than time-resource usage), and the scheduler on the time and resource assignments. Figure 7.6 shows in more detail how the different modules (layers) of IPSS interact. During the search process, every time the planner chooses to apply an operator, it consults the scheduler for the time and resource consistency. If the resource-time reasoner finds the plan inconsistent, then the planner backtracks

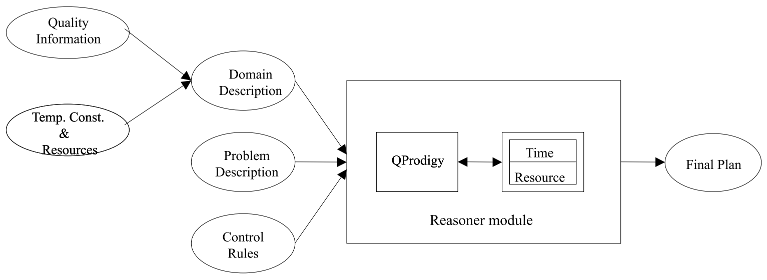


Figure 7.5: Structure of IPSS.

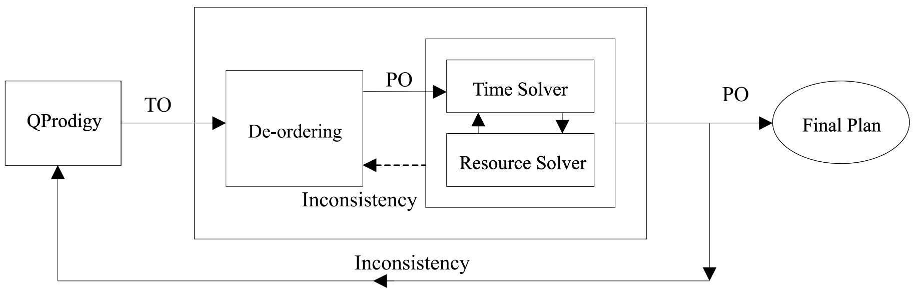


Figure 7.6: Components of the reasoner module.

If not, the operator gets applied, and search continues. The planner allows to choose actions under a pre-defined metric: *actions that consume less quantity of a given resource*, *actions that take less time in being executed*, *actions that maximise the capacity*, etc. The scheduler also helps using other optimisation criteria during the solving process as minimising the makespan or maximising resource usage.

Some advantages that are inherent to the QPRODIGY planner, are the possibility to define metrics different from plan length [20], reason about those multiple criteria [3], prune the search using control rules, or automatically learn them [6, 21].

7.3.1 Considerations on the IPSS algorithm

IPSS as a descendant of QPRODIGY, explores the same space, building a search tree and inheriting the same decision points on that search tree. When deciding which of the decision points (go to section 3.1.2 for a detailed explanation of the QPRODIGY algorithm) would be the best candidate to integrate the CSP solver with, we did the following considerations:

1. Step (1.) is in charge of selecting a *goal* from the set of unresolved goals and subgoals. At this step the CSP solver could not help because we would need to represent the domain knowledge as a CSP model. The disadvantage of applying it at this step is that, for each domain, a new CSP model must be developed.

2. Step (2.) chooses an *operator* to achieve a particular goal. At this step the CSP could give back some information on time and resources. The only disadvantage is that the choice of an operator does not mean that it will be part of the

solution. For example, goal loops (that is, one of the new goals is equal to one of the goals that have appeared before in the current search tree) cannot be detected quickly, so calling the CSP solver would increment considerably the execution time and would not guarantee that after the calculation time the operator will be applied even if it is resource and temporal consistent.

3. Step (3.) is in charge of the instantiation of the chosen operator. The actual algorithm implemented for doing the PRODIGY matching [184] could be changed using CSP techniques, but this would require a lot of modifications inside the PRODIGY code.

4. Step (4.) decides to *apply* an operator whose preconditions are satisfied or continue *subgoaling* on another goal or subgoal. Contrary to what happened on step (2.) if the operator is applied, goal loops or binding inconsistencies have been detected and the CSP solver could help on checking and propagating time and resource constrains.

For these reasons, as a first step to integrate planning and scheduling we have chosen step (4.). If the operator falls into time and resource conflicts, we can quickly detect them and backtrack.

Once we have decided where, the next step would be how we can do the integration. As mentioned on section 3.1.2, QPRODIGY has implemented two handlers that compute the cost of applying an operator. One of these two handlers is called every time an operator is applied, so in the same decision point at which we need to call the CSP-solver.

The handler has been extended and it is in charge of:

1. Creating a ficticious operator, *S-OP* (it corresponds to the source of the Temporal Network (TN) and the initial state in the planning problem) when the first operator is applied.

2. Getting all the information of the applied operator[2] and creating the corresponding time events structure. That is, each operator is subdivided into two Time Points (TPs). The first one corresponds to its start time and the second to its end time. The created structure has the planning information (preconditions, efects and duration), its state after its application, pointers to the two TPs, causal links with other operators in the list of time events and information about the resources that it consumes/produces.

3. Checking time and resource consistency after adding the new operator. If it is not consistent, it closes the actual node and its descendants and saves on it the finish reason (*inconsistent*) and backtracks.

[2]Not all the information is on the node of type *Applied Operator*. We need to go through the search tree and get the needed information from other nodes: from the *Instantiated Operator* node we obtain the preconditions list, from the *Applied Operator* node we get the *add* and *del* lists and from the *Operator* node we obtain the number of add effects.

4. Adds the fictitious operator, *F-OP* (it corresponds to the sink of the TN and the goals in the planning problem) to the time events list.

5. If we are not looking for more than one solution, it visualises the TN and initialises all the variables getting ready for the next run.

The last consideration before explaining each of the IPSS layers is how to obtain the operator **duration**. As mentioned before, QPRODIGY saves in each *Applied Operator* node the sum of the costs of each of the instantiated operators. The type of metric used can be time or any other, but in any case we need to save the duration of each operator for the time events structure. If any time metric is defined, it will be assigned to the duration the default value (that is, one), otherwise we calculate the duration by subtracting the actual cost value to the cost of the previous operator. Just to point out that when more that one cost metric is defined in the operators, we save in the internal node structure the total cost of the time metric in parallel to the chosen cost.

Figure 7.7 shows the three layers that compose IPSS. We have also subdivided it into two groups: IPSS-P that corresponds to the planning reasoner, and IPSS-S that corresponds to the scheduling reasoner. The corresponding functions that implement them can be seen in the IPSS algorithm on Figure 7.8. We have written in bold the functions added to the basic QPRODIGY algorithm: step 10 corresponds to the *Deorder-layer*, step 11 to the *Ground*-CSP layer and step 12 to the *Meta*-CSP layer. The extension of the handler mentioned above corresponds to step 9 of the algorithm (that is, every time QPRODIGY applies an operator).

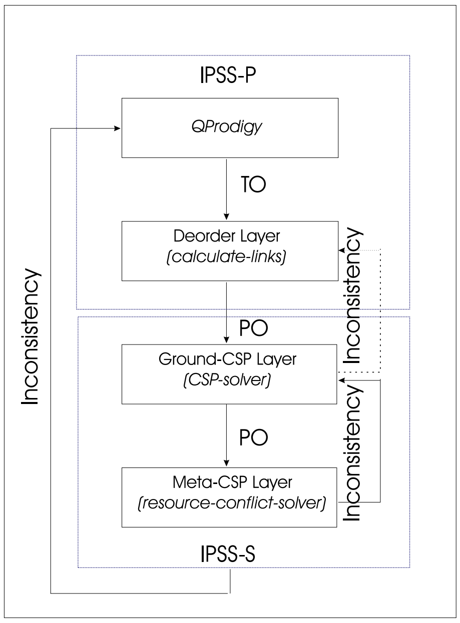


Figure 7.7: IPSS architecture.

Function **IPSS** ($\mathcal{D}, \mathcal{C}, \mathcal{S}, \mathcal{G}$)

$\mathcal{S}$ the state of the problem
$\mathcal{G}$ the set of goals to be achieved, also called *pending goals*
$\mathcal{D}$ the domain description: operators and objects hierarchy
$\mathcal{C}$ the set of control rules (control knowledge)
$\mathcal{P}$ the plan, initially empty
O an instantiated operator (from the set of operators in $\mathcal{D}$)
$\mathcal{B}$ a substitution (bindings) of the variables of an operator
O_B the operator O instantiated with bindings B
$\mathcal{O}$ the set of chosen instantiated operators not yet in $\mathcal{P}$, initially empty
$\mathcal{A}$ the set of applicable operators (a subset of O_B)
$\mathcal{G}_o$ the set of preconditions of all operators in $\mathcal{O}$
ST the planning search tree
$\mathcal{LE}$ the set of Links in $\mathcal{P}$
$\mathcal{L}$ the links for each O_B
$\mathcal{LR}$ the links for each O_B added by resource conflicts
$\mathcal{TN}$ the Temporal Network

```
While $\mathcal{G} \not\subseteq \mathcal{S}$ AND search tree and planning resources (time or nodes) not exhausted do
1.   $\mathcal{G} = \mathcal{G}_o - \mathcal{S}$
2.   $\mathcal{A} \leftarrow \emptyset, \mathcal{LE} \leftarrow \emptyset, \mathcal{L} \leftarrow \emptyset, \mathcal{LR} \leftarrow \emptyset, \mathcal{TN} \leftarrow \emptyset$
3.   Forall $O_B \in \mathcal{O}$ | preconditions ($O_B$) $\subseteq \mathcal{S}$ do $\mathcal{A} \leftarrow \mathcal{A} \cup \{O_B\}$
4.   If select-subgoal-or-apply ($\mathcal{G}, \mathcal{A}, \mathcal{S}, \mathcal{C}$) = subgoal
5.   Then $\mathcal{G} \leftarrow$ select-goal ($\mathcal{G}, \mathcal{S}, \mathcal{C}$)
6.        $O \leftarrow$ select-relevant-operator ($\mathcal{G}, \mathcal{C}$)
7.        $\mathcal{B} \leftarrow$ select-bindings ($O, \mathcal{S}, \mathcal{C}$)
8.        $\mathcal{O} \leftarrow \mathcal{O} \cup \{O_B\}$
9.   Else $O_B \leftarrow$ select-applicable-operator ($\mathcal{A}, \mathcal{S}, \mathcal{C}$)
10.       $\mathcal{L} \leftarrow$ calculate-links ($O_B, \mathcal{A}, \mathcal{LE}$)
11.       If CSP-solver ($O_B, \mathcal{L}, \mathcal{TN}$) = true
12.       Then If ($\mathcal{LR} \leftarrow$ resource-conflict-solver ($\mathcal{TN}$) $\neq \emptyset$)
13.            Then $\mathcal{LE} \leftarrow \mathcal{LE} \cup \mathcal{L} \cup \mathcal{LR}$
14.                 $\mathcal{S} \leftarrow$ apply ($O_B, \mathcal{S}$)
15.                 $\mathcal{O} \leftarrow \mathcal{O} - O_B$
16.                 $\mathcal{P} \leftarrow$ enqueue-at-end ($\mathcal{P}, O_B$)
17.            Else Backtrack
18.       Else Backtrack
19.  If there is a reason to suspend the current search path Then Backtrack
Return plan $\mathcal{P}, \mathcal{LE}, \mathcal{TN}, \mathcal{ST}$ and measures (total time, makespan, quality costs and expanded nodes)
```

Figure 7.8: IPSS algorithm.

7.3.2 The Deorder Layer

Since QPRODIGY is a TO planner, while planning it generates incomplete total ordered solutions. Therefore, at each application of an operator (inclusion in the incomplete plan), we have to check for its consistency with the temporal and resource components. This type of solution is not appropriate if we are looking for time and resource optimisation. That is, some orderings can be removed from the incomplete solution given by QPRODIGY allowing parallel executions of operators. This is referred as *deordering* of the solution.

We have implemented a module that dynamically transforms the incomplete TO plan into an incomplete PO plan. Links and threats solving techniques [139] must be applied in order to obtain a valid incomplete order solution. The deordering algorithm corresponds to the *Deorder-layer* and the step 10 of the algorithm depicted in Figure 7.8. The other two layers will be explained on section 7.3.3.

Figure 7.9 presents a high level definition of the Deordering algorithm. We have implemented a link structure as in UCPOP [133] or any PO algorithm that saves the operator identifier that establishes the condition, the condition and the operator identifier that supports the condition. Two extra additional fields have been also added to save the minimal and maximal time distance between the two operators. The structure of the TN allows to impose constraints between TPs and we want to explote this property for complex problems that could require to define distances between activities. By default these values are set to zero.

The number of operators in the set of applicable operators will always be two or greater than two. The first time that the function `calculate-links` will be called is after applying the first operator, so in that set there will be *S-OP* and the first operator applied.

The function `link-from-state` returns the link that satisfies that the preconditions of O_B are supported by the effects of an operator in the list of applied operators. It also returns the operator identifier that establishes that link. If the third parameter is 1 (the origin of the link is 1), it will start searching from *S-OP* (the initial operator), otherwise it will start searching from the operator passed as a parameter.

The `del-preconds` function finds if the operator O_B deletes any condition established before in any link. It returns a list of safe links in case any link is threatened. The `test-save-links` detects if none of the applied operators deletes any condition established in the links for O_B. If the links for O_B are not safe, new links should be found starting from the next applied operator that has as effects the preconditions of O_B. The cycle is repeated until the links are safe (in the worse case, the link would be established with the last applied operator).

Once the links are computed, the PO Solver exchanges this information with the scheduler module (formulated as a Constraint Satisfaction Problem as it is explained in section 7.3.3), that calculates the resource and/or time consistency of the given solution, serialising (adding links in the incomplete plan) in case of resource conflicts.

Let us clarify how this algorithm works with the example of Figure 3.2. Figure 7.10 shows the same problem in the PDDL2.1 representation. The solution (given by QPRODIGY) is shown in Figure 7.11. We have added on the left part, an operator identifier and on the right part the duration for each operator. The duration values will be calculated internally by QPRODIGY.

Figure 7.12 shows each step when calling the `calculate-links` function. At each step the other two layers will also be called, but we have omitted it here to better understand how this layer works. The next section will continue with the example and will show how the other two layers cooperate with the *Deorder-layer*.

```
Function calculate-links ($O_B$, $\mathcal{A}$, $\mathcal{LE}$)

$O_B$ the operator $O$ instantiated with bindings $B$
$\mathcal{A}$ the set of applicable operators until now
$\mathcal{LE}$ the set of Links in the incomplete plan
$\mathcal{L}$ the links for $O_B$
$\mathcal{L}'$ a list of temporal links
$O_S$ source operator identifier that establishes the link for $O_B$
$O'_S$ temporal source operator identifier for $O_B$

1. $O_S \leftarrow 1$, solved = false
2. While (Cardinality ($\mathcal{A}$) $\neq O_S$) and not (solved) do
2.1.     $\mathcal{L}, O'_S \leftarrow$ link-from-state ($O_B, \mathcal{A}, O_S$)
2.2.     $\mathcal{L}' \leftarrow$ del-preconds ($O_B, \mathcal{L}, \mathcal{LE}$)
2.3.     If $\mathcal{L}' \neq$ empty
         Then $\mathcal{L} \leftarrow \mathcal{L}'$
2.4.     If test-save-links ($\mathcal{A}, \mathcal{L}$) $\neq$ true
         Then $O_S \leftarrow O'_S$
         Else solved = true
3. Return $\mathcal{L}$
```

Figure 7.9: Algorithm to transform a TO into a PO.

The operators whose identifiers are 1 and 2 correspond to the initial and goal states respectively (or *S-OP* and *F-OP* as mentioned above). The first applied operator has the identifier number 3, the second operator 4 and so on. At the first step, `link-from-state` returns the link that satisfies the preconditions of operator 3 (make-bed agent7 bed2 room1), so the link is established between 1 and 3. The other two functions are not called because there are no other operators on the list of applied operators.

When applying the second operator (clear-bed agent6 bed1 room0), the `link-from-state` function returns the link between the origin and the new operator (1 $\rightarrow$ 4). The `del-preconds` function does not return any safe link because operator 3 does not delete any condition established by operator 4 and because the effects of operator 3 do not delete any condition. `test-save-links` does not calculate the next operator that supports the preconditions of operator 4.

In the last operator (make-bed agent6 bed1 room0), the `link-from-state` function starts looking from the operator 1 but it does not support its preconditions. So it looks for operator 3 (it does not support them either) and 4. Finally, it returns the link (4 $\rightarrow$ 5). The other two functions are called, but the link is saved as operators 4 and 5 are in sequence so there are no possible threats between them.

As a last step, links to the goal operator (or operator number 2) are added. The links added are (3 $\rightarrow$ 2 and 5 $\rightarrow$ 2).

The complexity of the deordering algorithm for the worst case is O(n^2), being n the number of applied operators.

```
(define (problem Example)
 (:domain robocare)
  (:objects
     person0 person1 person2 - PERSON
     agent0 agent1 agent2 agent3 agent4 agent5 agent6 agent7 - AGENT
     room0 room1 - ROOM
     bed0 bed1 bed2 bed3 bed4 - BED)
  (:init
     (is_door room0 room1) (is_door room1 room0)
     (is_in_person_room person0 room0)
     (needs_to_walk person0)
     (is_in_person_room person1 room0)
     (needs_to_walk person1)
     (is_in_person_room person2 room0)
     (needs_to_walk person2)
     (is_in_agent_room agent0 room0)
     (is_in_agent_room agent1 room0)
     (is_in_agent_room agent2 room0)
     (is_in_agent_room agent3 room0)
     (is_in_agent_room agent4 room0)
     (is_in_agent_room agent5 room1)
     (is_in_agent_room agent6 room0)
     (is_in_agent_room agent7 room1)
     (is_in_bed_room bed0 room1)
     (unmade bed0) (clear bed0)
     (is_in_bed_room bed1 room0)
     (unmade bed1) (not-clear bed1)
     (is_in_bed_room bed2 room1)
     (unmade bed2) (clear bed2)
     (is_in_bed_room bed3 room0)
     (unmade bed3) (not-clear bed3)
     (is_in_bed_room bed4 room0)
     (unmade bed4) (not-clear bed4))
  (:goal (and (made bed2)
              (made bed1))))
```

Figure 7.10: The example of Figure 3.2 in PDDL2.1 syntax.

```
3:  (make-bed agent7 bed2 room1) --> Dur = 4
4:  (clear-bed agent6 bed1 room0) --> Dur = 5
5:  (make-bed agent6 bed1 room0) --> Dur = 3
```

Figure 7.11: QPRODIGY solution to the problem of Figure 7.10.

7.3.3 The CSP Layers

As described in section 2.3.2.1 an instance of a CSP comprises the tuple <X, D, C>, where: X = $\{x_1, ..., x_n\}$ is the set of variables, D_i is the domain for each variable and

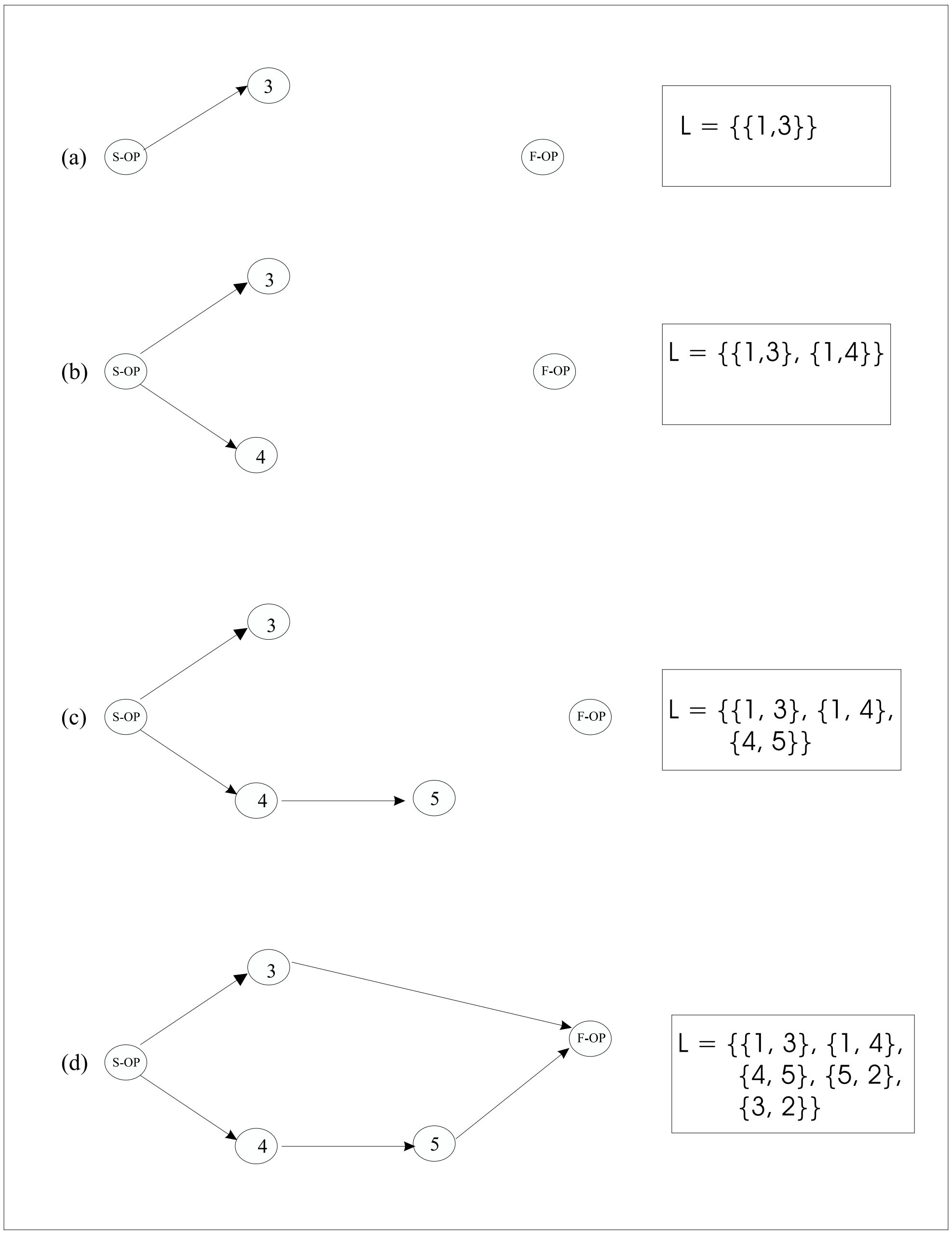


Figure 7.12: The deordered solution of Figure 7.11.

C={C_1,..., C_n} is the set of constraints s.t. $C_i \subseteq D_1$ x D_2 x ... x D_n, which define feasible combinations of domain values. Figure 7.13 shows a schema of a general CSP algorithm. It is an iterative search procedure where the incomplete solution is extended each cycle by assigning a value to a new variable. In step 2.1 of the algorithm a set of *propagation rules* remove elements that are not feasible with the current partial solution [26]. Because propagation alone cannot remove all inconsistent values, steps 2.2 and 2.3 help on the variable and value ordering through *search control* heuristics usually problem domain independent. For example, as an ordering criterion, it uses the *most constrained* variable and for the value selection the *least constrained* value.

From the two most frequently models used in the scheduling literature, the *start time assignment model* [128, 158] (with two types of constraints: binary constraints to designate the start time between activities and constraints to describe the capacity that each resource imposes on the schedule) and the *precedence constraint posting model* [30, 35, 169] (the decision variables correspond to precedence constraints between the set of activities), we have used the second model for our approach.

The *Ground*-CSP layer

Each applied operator is represented by two TPs that designate the early start time (est) and the latest finish time (lft) of the activity respectively. The distance between these two TPs is the duration of the operation. If the duration is a fixed value, for example value eight, we impose as the interval distance between both TPs [8, 8]. Instead, if the duration can have a minimum (4) and maximum (8) value, the interval would be represented as [4, 8]. As we mentioned before, by default the distance between activities is zero, so the constraints between activities will be represented as $[0,\infty]$.

The algorithm of Figure 7.13 works interchanging information between the other two layers: the search proceeds propagating temporal consequences through the *Ground*-CSP, temporal inconsistency could give information back to the deordering algorithm, that is, the *Deorder-layer*. We have omitted this information and used instead an heuristic to avoid searching in the space of possible deorderings. As it will be explained in section 7.3.4 this heuristic makes the *Deorder-layer* be not complete, but we obtain good results as Chapter 8 shows. Otherwise, computation and selection of the resource conflicts in the *Meta*-CSP is done by imposing precedence constraints in the *Ground*-CSP. The CSP algorithm described corresponds to step 11 of the IPSS algorithm (Figure 7.8).

There are two auxiliary functions that manipulate the state of the TN and allow to easily backtrack. One of the functions saves in a stack the actual state of the TN and the other one returns the last consistent state of the TN by poping it from the stack. These operations will be called at steps 11 and 18 repectively on Figure 7.8.

Backtracking to the *Ground*-CSP layer does not mean computing from scratch the new situation, but only eliminating the last applied operator and the links added (due to time and resource constraints) from the new saved state to the inconsistent state and poping the last consistent state from the stack.

Figure 7.14 shows the TN construction at each step when an operator is applied and derordered by the *Deorder-layer* in the example of Figure 7.12. Before any operator is added to the TN, two TPs are created corresponding to the source (initial state) and sink (goal state) and the distance or makespan (if needed) between them (by default the maximum integer value) is established.

When an applied operator is deordered, the links calculated by the *Deorder-layer* and the duration between the two TPs of the operator are added to the TN. To pro-

```
Function CSP-solver ($O_B$, $\mathcal{L}$, $\mathcal{TN}$)

$O_B$ the operator $O$ instantiated with bindings $B$
$\mathcal{L}$ the links for $O_B$
$\mathcal{TN}$ the Temporal Network
Var a variable on the $\mathcal{TN}$
Value a variable value on the $\mathcal{TN}$

1. If not CreatedTN
   Then $\mathcal{TN}$ = CreateTN ($\mathcal{L}$, $O_B$)
   Else $\mathcal{TN}$ = AddtoTN ($\mathcal{L}$, $O_B$, $\mathcal{TN}$)
2. While not solved do
       2.1. $\mathcal{TN}$ = RemoveInconsistentValues ($\mathcal{TN}$)
       2.2. Var = SelectDecisionVariable ($\mathcal{TN}$)
       2.3. Value = SelectValueForVariable ($\mathcal{TN}$, Var)
       2.4. If (unfeasible (Var, Value, $\mathcal{TN}$)) Then Backtrack
3. Return solved
```

Figure 7.13: A skeleton of the CSP algorithm.

pagate the variable and value decisions by the CSP-solver an additional link to the sink operator must be added, in the example [0,∞].

The algorithm used to solve the problem of minimal distance (All Pair Shortest Path - APSP) [9] for the TN is O(n^2), being n the number of TPs. Then, the complexity of the *Ground*-CSP is O($2N^2$), being N the number of operators in the solution.

The *Meta*-CSP layer

For the *Meta*-CSP layer we have implemented the binary resources algorithm of Smith and Cheng [169]. Under this representation model, resource conflicts are computable in polynomial time because conflicts are represented by pairs of temporally overlapping activities that require the same resource. Figure 7.15 sketches the algorithm and in [169] the flowchart is also depicted.

Right now, IPSS only supports binary resources, but the flexibility of the *precedence constraint posting model* makes it easily extendible for multicapacity resources. In that case the total time is exponential in the size of the total resource capacity as described Cesta *et al.* in [30].

Following with the example of Figures 7.12 and 7.14 the `resource-conflict-solver` checks resource conflicts between pairs of activities and impose precedence relations between them, so it is called when more than one operator is on $\mathcal{LO}$. Considering *agent* as a resource, we have two lists: the list of operators that consumes *agent6* and the list that consumes *agent7* (as in the solution plan of Figure 7.11 there are just operators that have as *agent* instances: *agent6* and *agent7*). The only instant at which there can be a resource conflict is when two operators that can be in parallel consumes the same resource. This is the case when the operator (make-bed agent6 bed1 room0) is applied, but although the algorithm returns that Operator 3 should

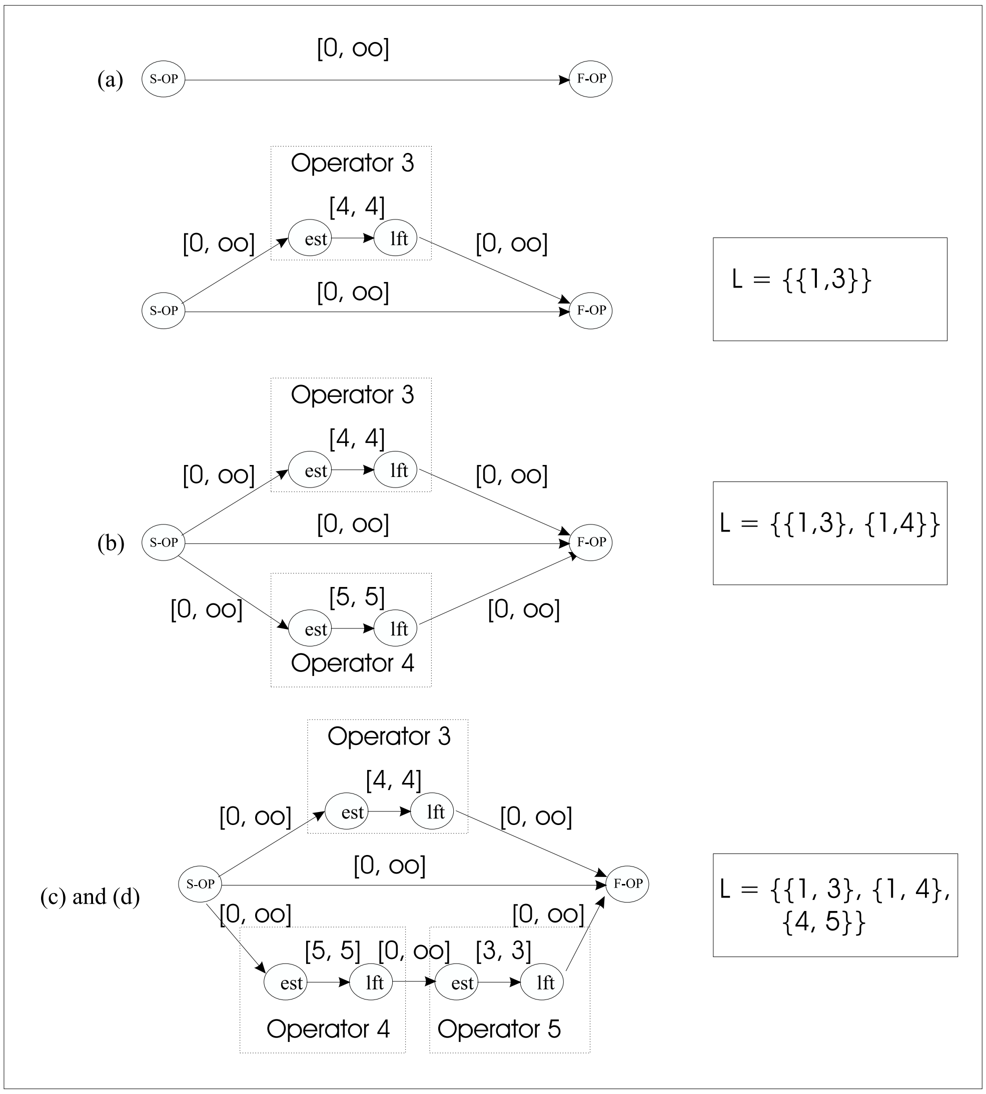


Figure 7.14: The Temporal Network representation of Figure 7.12.

be before Operator 4 (*case 1* on Figure 7.15) it was already a causal link between these two operations.

If the application of the current operators gets into a non solution or inconsistent state, the algorithm backtracks (in the QPRODIGY search tree) and considers an alternative execution sequence. The process is repeated until the head-plan achieves the goal.

```
Function resource-conflict-solver ($\mathcal{TN}$)

$est$ the earliest start time
$lft$ the latest final time
$p$ the duration of the operator
$op_i$ the operator in the incomplete plan $\mathcal{P}$
$\mathcal{LR}$ the links added by resource conflicts
$\mathcal{LO}$ the list of operators that consume the same resource
$\mathcal{TN}$ the Temporal Network

for each $op_i$ in $\mathcal{LO}$ do
    for each $op_j$ in $\mathcal{LO}$ do
        1. If $\mathrm{lft}_i$ - $\mathrm{est}_j$ $<$ $\mathrm{p}_i$ + $\mathrm{p}_j$ $\leq$ $\mathrm{lft}_j$ - $\mathrm{est}_i$
           Then $i$ must be scheduled before $j$ ($\mathcal{LR} \leftarrow (i \rightarrow j)$)
        2. If $\mathrm{lft}_j$ - $\mathrm{est}_i$ $<$ $\mathrm{p}_i$ + $\mathrm{p}_j$ $\leq$ $\mathrm{lft}_i$ - $\mathrm{est}_j$
           Then $j$ must be scheduled before $i$ ($\mathcal{LR} \leftarrow (j \rightarrow i)$)
        3. If $\mathrm{p}_i$ + $\mathrm{p}_j$ $>$ $\mathrm{lft}_j$ - $\mathrm{est}_i$ and $\mathrm{p}_i$ + $\mathrm{p}_j$ $>$ $\mathrm{lft}_i$ - $\mathrm{est}_j$
           Then there is no feasible schedule ($\mathcal{LR} \leftarrow \emptyset$) and return
        4. If $\mathrm{p}_i$ + $\mathrm{p}_j$ $\leq$ $\mathrm{lft}_j$ - $\mathrm{est}_i$ and $\mathrm{p}_i$ + $\mathrm{p}_j$ $\leq$ $\mathrm{lft}_i$ - $\mathrm{est}_j$
           Then any sequence is possible ($\mathcal{LR} \leftarrow (i \rightarrow j)$)
Return $\mathcal{LR}$
```

Figure 7.15: Precedence Constraint Posting algorithm for binary resources conflict.

The feedback from the temporal layer allows to check temporal inconsistencies and to decide which operator to choose in order to reach a specific horizon. The feedback from the resource layer has been implemented in the form of *domain dependent* rules control allowing to choose different resource bindings that minimise the makespan and at the same time avoid resource conflicts. The criteria followed is to try to use when possible and in an efficient way all the available resources.

The same result could be obtained modifying the decision point 3 of the QPRODIGY algorithm to bind the resource (variable *agent*) to the one returned by the *Meta*-CSP layer, but these control rules achieve the same effect without changing the PRODIGY matching algorithm [184]. Figure 7.16 shows an example of a control rule for the Make-Bed operator in the ROBOCARE domain (see Appendix D for details of the domain, also explained in Chapter 3). The **resource-less-used-p** meta-predicate receives the information from the *Meta*-CSP layer, binding the agent that should make the bed with the one that the *Meta*-CSP layer has decided is less used. As the results of next chapter show, these control rules can greatly improve the makespan.

In this new approach, the temporal-resource inconsistencies are discovered as soon as they occur and allow to maximise the resource consumption. This is due to the strong communication between the planner and the scheduler that interchange information at every decision point. Then, the computational cost decreases with respect to the other approaches and the possibility of finding better plans increases thanks to the capability of the CS system to allow different kinds of solution analyses that can be used to guide the planning search.

```
(Control-Rule Bindings_Make-Bed
(IF (and (current-goal (unmade <bb>))
         (current-operator Make-Bed)
         (type-of-object <aa> AGENT)
         (resource-less-used-p <aa>)))
(THEN select bindings ((<a0> . <aa>) (<b0> . <bb>) )))
```

Figure 7.16: A ROBOCARE control rule to bind resources.

Although the QPRODIGY planner (as explained before) can handle time and some type of resources (consumable, producible and resources with single capacity), with this integration we can:

- Reason about activities in parallel. This is an important feature when trying to minimise the makespan.
- Allow different levels of reasoning, that is, we can decide when to reason about resources: during the planning or scheduling process or at both.
- Handle binary resources and easily extend IPSS to multicapacity resources. The modularity of the approach allows us to use any algorithm in the *Meta*-CSP layer without affecting the other two layers or replace the IPSS-P part of IPSS (see Figure 7.7) by any other planning algorithm.

7.3.4 Properties of the IPSS Algorithm

In this subsection we analyse the soundness, completeness and optimality of the IPSS algorithm.

The soundness of the IPSS algorithm
The head plan construction in PRODIGY (see section 3.1.2) always returns a valid plan defined as a sequence of operators that achieves the goals from the initial state. When we are using quality metrics, there is nothing that prevents it from removing soudness given that it only reasons about cost of solutions and not about goal-operators relationships, so QPRODIGY is also sound.

In the case of IPSS, it also constructs a head-plan as its ancestor and it applies a deordering to the solution that it is always a valid one because all the causal links are proven to be safe. Each time a new set of links is computed, the algorithm checks that any of the previous links are threaten. If this occurs, we apply any of the methods used in POP algorithms to solve threats.

Also, the TN provides a correct time and resource assignment to the TPs that compose it. Then, we can conclude that IPSS is sound.

The completeness of the IPSS algorithm
PRODIGY is based on bidirectional planning, that is, a combination of goal-directed backward chaining with simulation of plan execution. Experiments have shown that it is an efficient technique but it is not complete [64].

In IPSS we are going to analyse each of the layers that we have added to the search in QPRODIGY in order to evaluate their degree of completeness.

- The *Meta*-CSP layer: calculates the precedence constraints between activities that consume the same resource. The algorithm has in mind all the activities involved in the consumption of the same resource, but when there is more than one possible order it chooses the link following the logical order, that is, if links $3 \longrightarrow 4$ and $4 \longrightarrow 3$ are possible, it returns the $3 \longrightarrow 4$ link because it always chooses the chronological order imposed by QPRODIGY in the operators. So the *Meta*-CSP layer is not complete.

- The *Ground*-CSP layer: consists of a propagation algorithm and selection of variables and values. In [26] it is shown that the propagation algorithm is correct and complete because a TN is arc-consistent if and only if it is also temporally consistent. This only occurs when assignments of values to the temporal variables exist. Due to the fact that variables and values assignments are embedded within a backtracking framework, the CS algorithm is as a whole complete.

- The *Deorder-layer*: as described in section 7.3.2, the deordering algorithm starts computing the link that satisfies that the preconditions of the last operator added to the head plan is supported by the effects of an operator in the list of applied operators. The links search follows the order of the operators in the head plan, that is, it starts from the origin until the last operator is applied.

  The problem we are interested in is to reduce the computation of all potential deorderings for each applied operator. As a heuristic, we try to place the operators as near to the origin and between them as possible, that is to minimise the makespan. But the use of the heuristic can lead in some cases to incompleteness, so the deordering algorithm is incomplete. We have called this heuristic the *Minimal Link Deordered* heuristic. If we help on the temporal information that the TN returns when we have imposed a temporal horizon, we can see that any deordering different to the one calculated, (let us called it $\mathcal{LR}$) will be inconsistent if $\mathcal{LR}$ is so. The inverse does not have to hold.

  If the *Ground*-CSP returns an inconsistency for $\mathcal{LR}$, we could compute another link farther from the one calculated, but the TN will again return an inconsistency.

  According to what has been stated above, Figure 7.18 and Figure 7.19 show two possible deorderings between activities three and four considering the incomplete serial plan of Figure 7.17. The activity four can be executed in parallel with respect to activity three or after this activity. The algorithm will calculate the *Minimal Deordered Link* $\mathcal{LR}$ that corresponds to the link between the origin and activity four. If we impose a temporal constraint of [10, 20] between the TPs 1 and 2 adding the dc0 arc (that is, a deadline of [10, 20]) on Figure 7.18, it cannot achieve this horizon, and, of course neither the deordering in Figure 7.19 can.[3] So, the last deordering does not have to be computed,

[3]To know if there is an inconsistency in the graph, we just need to sum the lower and upper bounds

and we can go back to the planners search, close the actual node expansion, and consider the next possible applied operator, reducing considerably the execution time.

In all the experiments that will be presented in the next chapter we have followed this heuristic and they show that when QPRODIGY finds a solution so does IPSS with the mentioned heuristic.

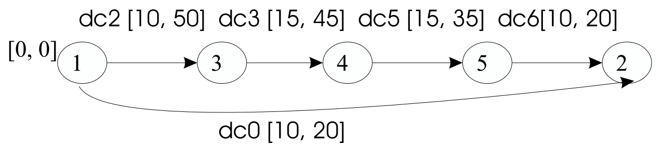


Figure 7.17: The incomplete serial plan.

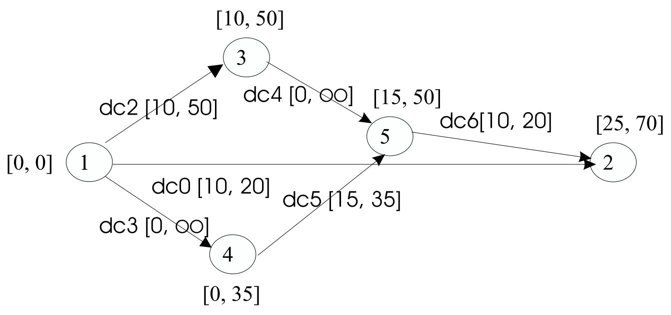


Figure 7.18: The plan of Figure 7.17 using the *Minimal Link Deordered* heuristic.

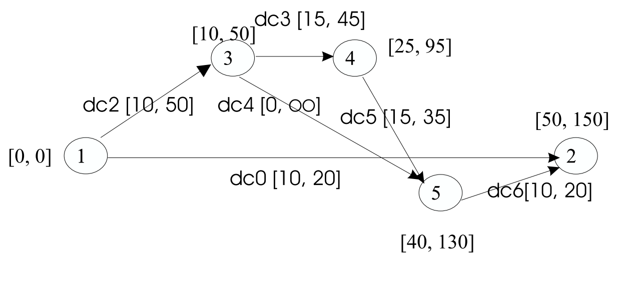


Figure 7.19: The same problem as 7.18 considering another deordering.

of the different arcs in the different paths. If the value obtained in TP 2 is higher than the dc0 arc, then the graph is inconsistent.

The optimality, quality and efficiency of the IPSS algorithm
QPRODIGY algorithm is not optimal given that it is not complete. So IPSS cannot be optimal.

IPSS, as QPRODIGY, can improve the quality of the first solution found, given a time bound, when we set the *multiple solution* mode. In that case, the makespan and the cost of the first solution are used as a bound to the next one and so on. IPSS can also look for solutions with more than one quality metric defined that allows to find plans that try to minimise those metrics. The way it works is as follows: the planner search tree prunes by one metric (generally we set it to a metric different to time and resources) and the TN by time and/or resources.

With respect to the efficiency, the three layers added to the original algorithm usually increase the time to find a solution, given the time needed for computing deorderings and time/resource inconsistencies is O(n^2).

7.4 Experiments

We have used the HISPASAT domain to test the different configurations presented in this chapter, varying the complexity of the problems from 38 literals in the initial state and 6 goals, up to 62 literals in the initial state and 78 goals.

In this domain, the plan obtained does not consider any optimisation with respect to resource use or time. It only requires to obtain a plan, because of the low flexibility to schedule some operations. For example, South-Maneuver operations must be performed on Mondays or Tuesdays every two weeks at an specific hour corresponding to the secular drift. Another example is that specific fuel tanks must be used in specific periods of the year, so even if there is a better schedule that optimises resources and time, this is not feasible due to the domain constrains. For more details about the domain see section 5.1.

We have compared the configurations: QPRODIGY alone considering time and resources, the QPRODIGY+ VELOSO+ O-OSCAR and IPSS in terms of the time in seconds to find a solution. The results show that the three configurations solve all temporal problems. QPRODIGY solves the problems better than any other approach with respect to time to solve. However, it is less general, and applying it to another domain requires a fair amount of knowledge engineering. Also, the plan obtained is a TO and not a PO plan. The other two approaches are simpler to model, given the clear separation of planning and scheduling information, but they are a little bit less efficient. We will improve this through learning control knowledge, by using the feedback of the scheduler to improve the planners decisions. Also Figure 7.20 shows that IPSS is more efficient than the second configuration. In terms of makespan and number of operators the three configurations find the same results.

We have also compared the makespan of QPRODIGY + VELOSO + O-OSCAR and IPSS in the domains that will be presented in the next chapter, obtaining equal results. But as it occurred in HISPASAT domain the time to solve the problems is higher in QPRODIGY + VELOSO + O-OSCAR than IPSS. Note that the deordering algorithm used is different: IPSS starts incrementally from the initial state forward the goals since VELOSO starts backwards once the TO plan is obtained.

South Man.	Moon Blind.	PRODIGY (sc)	PROD+VEL+ OSCAR (sc)	IPSS System (sc)
2	36	0.47	1.11	1.12
5	36	1.05	1.29	1.30
8	26	1.74	1.75	1.80
11	36	3.23	4.59	4.42
14	36	4.02	6.63	5.93
17	36	4.92	10.98	8.77
20	36	6.11	13.92	11.74
23	36	7.67	18.04	14.07
26	36	8.01	23.01	15.31
15	10	5.07	8.18	7.25
26	10	6.05	21.72	13.21
15	20	3.84	7.55	7.19
26	20	6.82	20.57	13.71
26	30	7.85	21.96	15.20
10	36	3.61	8.66	7.11

Figure 7.20: Experimental results on the HISPASAT domain for three configurations.

7.5 Summary

In part II we have used a classical planner to reason about time and resources for workflow and satellite domains. In this chapter we wanted to explore different configurations to integrate planning and scheduling. In the first approaches, the temporal and resource information has been eliminated from the planner reasoning, and the output generated by the planner is used as an input for a scheduler.

To influence the planner choices, the scheduler should provide some information to the planner. Along this line, the second approach, the IPSS (Integrated Planning and Scheduling System), interleaves the refinement solution inside the search tree of QPRODIGY [20]. The idea is to have two systems executing in parallel; a planner and a time-resource reasoner. Information is continuously exchanged between these two representations, and, in particular, it is used to influence the QPRODIGY search to prune choices that lead to either temporal or resource inconsistencies.

The most important features of IPSS are:

- Good balance between efficiency and quality, as we will show in the next chapter.
- The degree of reasoning that can be given to each level can be varied. Since our planner allows to use some type of temporal-resource usage reasoning together with some optimisation capabilities, we can control/vary the amount of

reasoning that the planner performs. For instance, we can decide to solve the temporal aspect of the problem only using the planner, and let the resource assignment to the scheduler or letting the scheduler solve the time and resource assignments.

- It is easy to add control knowledge, metrics and learning to the system. IPSS as a QPRODIGY descendant can use any of the modules implemented for improving its quality or any other learning techniques because the structure of the search tree has not been modified.
- It is implemented in a modular way so we can change any of its components and the system would still work just changing the interface to introduce the knowledge that each part needs.

Chapter 8

Experimental Results

In this chapter an evaluation of the IPSS system with its different configurations is presented. We also compare the results with some state of the art planners in three different domains.

8.1 Introduction

We have presented in the last chapter IPSS, a model that integrates planning and scheduling. We have implemented these ideas on top of a classical planner QPRODIGY that allows the use of control rules to guide the search. The reasons to choose this particular planner are manyfold. Among them, we can highlight definition and handling of quality metrics, explicit definition of control rules, flexibility to define new behaviours through the handlers definition and explicit rationale through the search tree.

One of the domains used, ZENOTRAVEL, belongs to the IPC-2002 [87] competition, the second one is a reduced version of the workflow process of installing a new line at BRITISH TELECOM explained in section E, and the third one is a multi-agent system for the care of the elderly [33]. All of the them have been coded following the PDDL2.1 syntax [68]. Since IPSS uses the PDL4.1 syntax [22] we have implemented an application that translates PDDL2.1 domains and problems automatically into PDL4.1 if the domains do not contain complex durative actions. In that case, it requires some knowledge of Common Lisp and the PDL4.1 syntax in order to translate this knowledge appropriately. For further details of the translation tool see Appendix F.

8.2 IPSS Configurations Used

In this section the different IPSS configurations are explained, and the domains where the IPSS performance is tested are presented, as well as the experimental results of their comparison.

Table 8.1 shows the name of each IPSS configuration and a brief description. For all the problems in each domain, the different IPSS configurations and the planners used to compare to were executed using a time bound of three minutes in the ROBOCARE and BT domains and two minutes in the ZENOTRAVEL domain.

Table 8.1: The different IPSS configurations used in this chapter.

Name	Description
IPSS	Temporal planner with two layers: *Ground*-CSP and *Deorder-layer*.
IPSS-C	Temporal planner with two layers: *Ground*-CSP and *Deorder-layer*. It uses hand-made control rules to guide the search.
IPSS-R	Temporal planner with three separate layers: *Meta*-CSP, *Ground*-CSP and *Deorder-layer*.
IPSS-RC	Temporal planner with three separate layers: *Meta*-CSP, *Ground*-CSP and *Deorder-layer*. It uses hand-made control rules to guide the search.
IPSS-Q/IPSS-R-Q	Allows to run any of the IPSS configurations looking for more than one solution, given a time bound. After the first solution is found, the next solutions will have as a makespan bound the one obtained in the last solution.

- IPSS bases its search in the QPRODIGY search integrated with the algorithm that converts incrementally the partial TO plan generated by QPRODIGY during the planning process into a partial PO plan (see section 7.3.2). Then, the *Ground*-CSP layer checks its consistency and provides the temporal information back to the search. To run this mode, we just need to set the *makespan* flag to true. If it is set to false (by default), we would be running standard QPRODIGY.

- IPSS-C is based on IPSS but includes control rules hand-generated by a QPRODIGY[1] expert based on the rules generated automatically by HAMLET [21] and that help in the selection/rejection of operators and bindings, thus reducing the search. The reason to use this configuration is to show that the system is also integrated in an architecture that allows the use of control rules to improve the solution. It is not the goal of this work to discuss the goodness of those control rules, not the relationship of this control knowledge with the CSP layers.

- IPSS-R bases its search in the QPRODIGY search integrated with the three layers working together. First the TO plan generated into a PO plan by the *Deorder-layer*. Then, the other two layers work in parallel: the Temporal Network that checks the temporal consistency and the resource layer that checks resource consistency, provides temporal and resource information back to the search that helps on next decisions to be made. To run this mode, we need to set

[1]We also follow the same notation for QPRODIGY (QP), that is, QP-C means that it uses control rules.

the *makespan* flag to true as in IPSS and also specify the resources we want to consider by means of the *include-resources* flag (since this first version of IPSS just considers binary resources, we do not need to specify their capacity). Its default value is false.

- IPSS-RC is based on IPSS-R but includes the same control rules as for IPSS-C.
- The -Q extension allows to run QPRODIGY and any of the IPSS configurations mentioned before looking for more than one solution. It is a type of *anytime* search [156]. The planner does not stop when it finds the first plan but after the time bound has exceeded. The first solution is used as a bound for the next solution, so the higher the time bound is, the better the quality of the plans would be. To run this mode, we need to set the *multiple-solutions* flag to true.

The following sections describe in detail each domain used, the results obtained using the different IPSS configurations and comparisons with other state of the art planners. We also describe the type of solution (TO or PO) and the type of problems solved by the planners that have competed in IPC-2002 [87]. This will serve as a baseline to reason about what planners to choose.

8.2.1 Multiagent Domain: ROBOCARE

The ROBOCARE domain [33] has some features that make it specially fit to separate the resource reasoning from the causal reasoning. Most of the planners consider discrete resources like robots or trucks as logical predicates.

This causes the search to become intractable when the number of resources increases. As experimental results showed in [174] this strategy severely curtails the scale-up of existing planners. The ROBOCARE domain is one of these domains that allows to consider the object *robot* as a resource of binary capacity.

8.2.1.1 The Robocare Domain

All the operations in this domain need to be executed by an agent (*robot*), being all agents equal. There are some restrictions to consider: one agent cannot perform two operations at the same time; or for making the bed, is must have been cleaned before. The tasks that robots can accomplish are: cleaning beds, making beds, serve meals and accompany persons to specific rooms. Also, the robot has the skill of moving among the different rooms if there are doors that connect them.

The modeling of the ROBOCARE domain (see Appendix D and section 3.1.1) does not consider if the agent is or not busy doing something else. So, apart from planners that provide serial solutions, as, for example, SERISTAR (that belongs to the HSP family [19]), FF [85] or QPRODIGY [20], the solution given by other planners as LPG [74], MIPS [56] or BLACKBOX [94] would not be valid. This is caused by the fact that they can give as a solution the same agent performing different actions in the same time step, which is impossible.

In order to compare not only TO planners against our approach, we have also coded this domain having in mind the availability of the chosen agent for performing the corresponding action. In this case, each action (operator) requires the agent not to be busy on performing some actions. After the execution of the action, the agent (robot) is freed by a dummy action (the *free-agent* action). This forces the planners that reason about parallel actions not to consider parallelizing two actions that require the same robot. We have called this domain the ROBOBUSY domain (see Appendix D).

We have generated 200 problems, distributed in four groups of fifty problems. Each one of them has a fix number of robots: one (G1), two (G2), five (G5) and ten (G10). Each set of fifty problems is subdivided into five subsets of ten problems each, with the following features:

- Three rooms with seven doors. Each room can contain five beds, and five persons may need to walk from one room to another. The maximum number of goals is three. All the numbers are randomly generated, and there might be problems with just only one room and others with two or three.
- Four rooms with eight doors. Each room can contain eight beds, and eight persons may need to walk from one room to another. The number of goals is between two and four.
- Seven rooms with eleven doors that can contain fourteen beds and fourteen persons may need to walk from one room to another. The number of goals is between five and seven.
- Twelve rooms with sixteen doors. Each room can contain twenty-four beds, and twenty-four persons may need to walk from one room to another. The number of goals is between ten and twelve.
- Seventeen rooms with twenty-one doors. Each room can contain thirty-four beds, and thirty-four persons may need to walk from one room to another. The number of goals is between fifteen and seventeen.

The set of problems for the ROBOBUSY domain is exactly the same as in the ROBOCARE domain with the particularity that in all the problems the availability of the agents must be explicitly set to free (that is, in the initial conditions we add the predicate *(not-busy <robot>)* for each <robot> in the problem).

Table 8.2 shows the number of solved problems by each one of the IPSS configurations for each subset described above in the ROBOCARE domain. As it can be seen, IPSS-R and IPSS solved 73% of the problems while IPSS-C and IPSS-RC solved 72% of the problems. Although the percentage is almost the same, the problems solved by each configuration in each group is not the same. In some cases, the control rules coded for this domain help to solve problems and in other cases the opposite. The control rules generated to handle resources (as explained in section 7.3.3) can improve the efficiency in obtaining the solution (as well as the makespan) but in other cases the time bound given to solve them is not enough as the three layers consume more time.

The total time given to solve each problem was 180 seconds and the number of control rules coded for this domain is 11. They were generated automatically from HAMLET [21]. Then, these control rules were refined by an expert.

Table 8.2: Number of problems solved by each IPSS configuration in the ROBOCARE domain from a total of 200 problems with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
IPSS	38	34	36	38	146	73%
IPSS-R	38	34	37	37	146	73%
IPSS-C	37	33	37	37	144	72%
IPSS-RC	37	34	37	36	144	72%

As quality measures to compare the generated plans one can consider the makespan and the time to generate them. Figure 8.1 shows the makespan for the IPSS configurations in the ROBOCARE domain. We only plot the makespan of problems solved by all configurations in the same graphic. The results show the superiority of using control rules over not using them, and also that in the integration of the planning and scheduling approach, that is, the IPSS-R configurations, the makespan is lower than in IPSS. The makespan improvement obtained using IPSS-R instead of IPSS is around 10% and the use of control rules improves the makespan in 6,5% in IPSS-R and 3% in IPSS.

IPSS-R and IPSS-C obtain almost the same results, which indicate that the coded control rules try to maximise the use of all the resources as the resource layer does, and then reduce the makespan. The worst behaviour, as expected, was obtained by IPSS because there was no information that could guide the search.

With respect to the number of operators in the solution, Figures 8.2 and 8.3 show the results. We do not consider the number of operators as a quality metric because plans with less operators do not guarantee plans with less parallel steps.

As mentioned before, for the IPSS and IPSS-C we have used the *busy* version of the ROBOCARE domain otherwise the solution would not be feasible, that is, that one robot could perform two operations at the same time. For this reason the number of operators in these two configurations is higher because of the dummy operator *free-agent*.

If we focus our attention in the execution time of each configuration (see Figure 8.4), IPSS-R beats the rest. The most surprising result is the time for IPSS: there are some problems in which it takes IPSS much more time to find a solution than it takes the rest. The reason for these results is that IPSS (as QPRODIGY) tries to minimise the length of the solutions by using the less number of robots as possible to perform the activities. Using the same robot to perform almost all the activities is translated in difficulties to find solutions and in worse makespan.

Although control rules help to find and improve the solutions, they increase the total execution time. IPSS-R improves the total time in solving all the problems in 84% with respect to IPSS, while control rules decrease the efficiency in 25% when

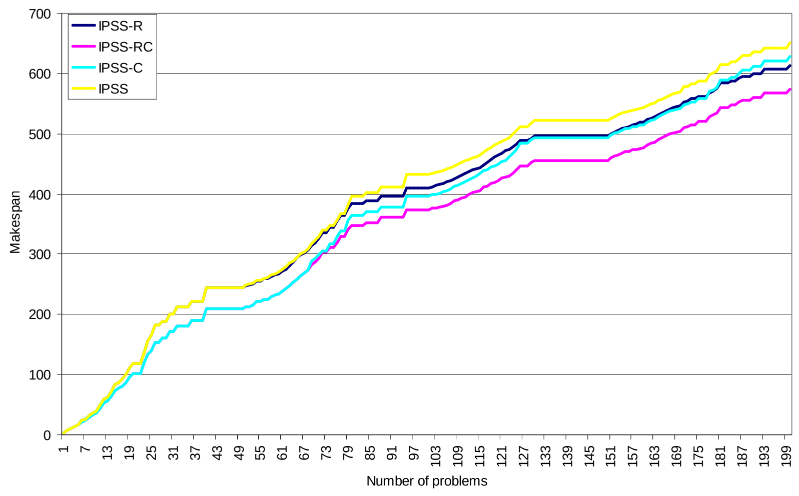


Figure 8.1: Makespan for the IPSS configurations in the ROBOCARE/ROBOBUSY domain.

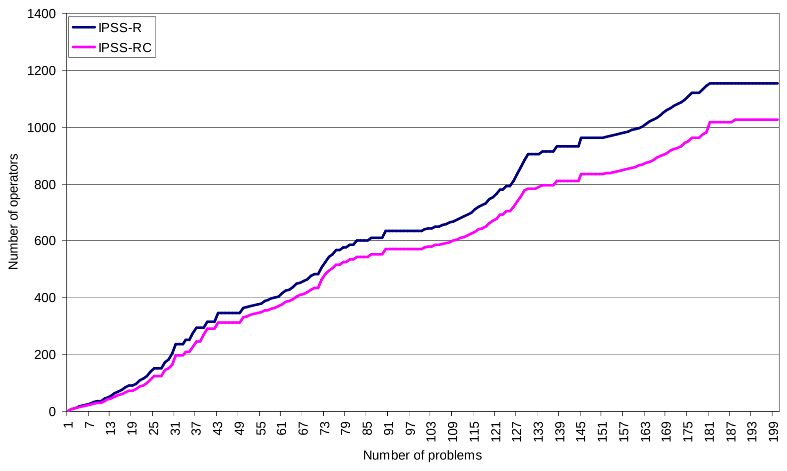


Figure 8.2: Number of operators for the two IPSS-R configurations in the ROBOCARE domain.

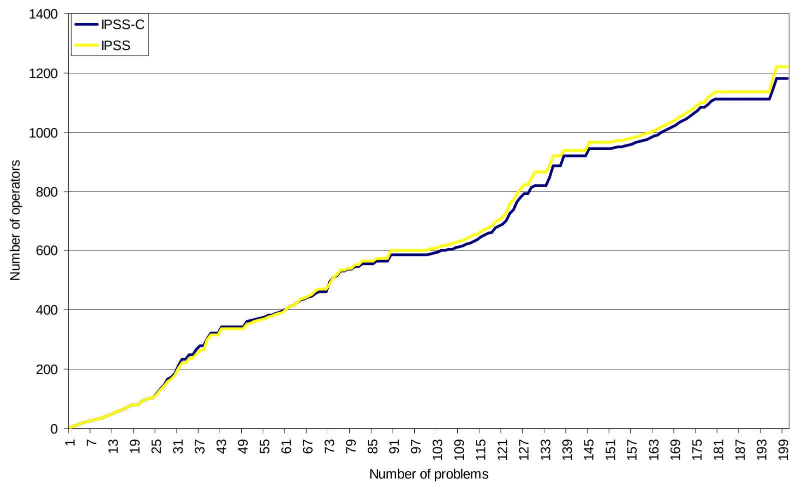


Figure 8.3: Number of operators for the two IPSS configurations in the ROBOBUSY domain.

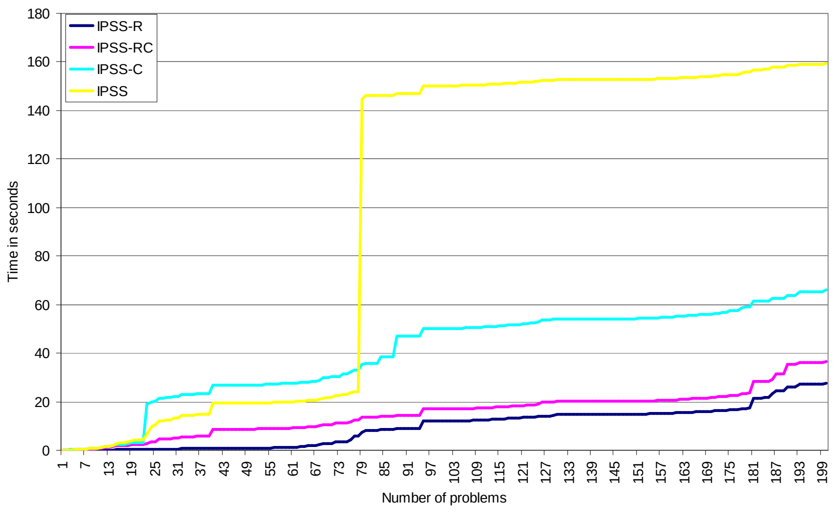


Figure 8.4: Time in seconds for the IPSS configurations in ROBOCARE/ROBOBUSY.

using IPSS-R, but improve it in 58% when using IPSS.

In the light of the experiments, we can say that the IPSS-RC configuration gets the best makespan in a reasonably good execution time.

8.2.1.2 The Robocare Simple Time Domain

As it has been done in the IPC-2002 competition, we have increased the difficulty of the ROBOCARE domain introducing durations in the activities and called this domain ROBOBUSY SimpleTime (in Appendix D the domain has been coded considering that the robots can be busy or not busy). Also, we have defined another version of the domain where durations depend on the initial state of the problem and the speed of the agents and the persons that need to be moved from one room to another and called this domain ROBOBUSY Time (in Appendix D the domain has been coded considering that the robots can be busy or not busy). Note that we have assigned the zero duration to the dummy operator *free-agent*.

For the ROBOBUSY Time domains we have not coded any control rules so we will just consider the IPSS and IPSS-R configurations. As we did for the other ROBOCARE versions, to test IPSS-R we have considered the same domain as in IPSS without the busy and not-busy predicates in all the operators.

We have also generated 200 new problems randomly for the ROBOCARE SimpleTime and ROBOCARE Time, distributed as in the ROBOCARE domain in four groups of fifty problems. Each one of these groups has a fix number of robots: one (G1), two (G2), five (G5) and ten (G10) which each set of fifty problems has the same features described for the ROBOCARE domain.

Table 8.3 shows the number of problems solved in each group, the total number of problems solved and the percentage that it represents. IPSS-R solved 78% of the problems and IPSS 73%. IPSS-R solves more problems when there are more resources, that is, the group with 10 robots.

Table 8.3: Number of problems solved by each IPSS configuration in the ROBOCARE SimpleTime domain from a total of 200 problems with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
IPSS	38	35	37	36	146	73%
IPSS-R	38	38	38	41	155	78%
IPSS-C	38	38	38	41	155	78%
IPSS-RC	38	38	38	41	155	78%

Figure 8.5 shows the makespan for the four configurations. For the makespan we have done one graphic, as we have assigned the duration zero to the dummy operator so it does not increase the makespan value. The improvement of using IPSS-R instead of IPSS is of 18% and the use of control rules increases the value of the makespan in 3% in IPSS and IPSS-R.

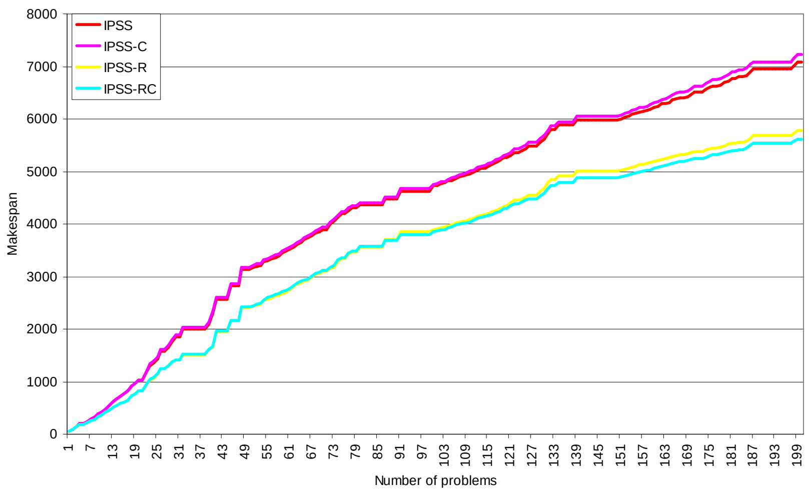


Figure 8.5: Makespan for the IPSS configurations in ROBOCARE/ROBOBUSY SimpleTime.

As we did for the ROBOCARE domain we have plotted only problems solved by all configurations. We have separated the graphics for the number of operators in IPSS-R and IPSS configurations because the first one does not use the *free-agent* operator and obviously is lower. These results are shown in Figures 8.6 and 8.7.

With respect to the efficiency of each configuration, IPSS-R obtains the best time to solve all the problems as Figure 8.8 shows. IPSS-R improves the total time in 17% with respect to IPSS, while control rules worsen the efficiency around 20% in IPSS-R and IPSS. The *levelisation* of resources (that is, trying to use all the available resources) helps to find the solution faster and to minimise the makespan. The control rules used to implement this functionality are domain independent as they just receive information from the *Meta*-CSP layer and they reduce the branching factor. They guarantee to find a solution although not always it is the most efficient one.

8.2.1.3 The Robocare Time Domain

For the ROBOCARE Time domain, Table 8.4 shows the problems solved by IPSS and IPSS-R in each group and the percentage that it represents in the total of 200 problems. Again, IPSS-R solves more problems (76%) and in problems with more agents than IPSS (73%). For this domain, we have not generated any control rule as mentioned before.

Figures 8.9 and 8.10 show the makespan and total time in seconds to solve the 200 problems for the ROBOCARE Time domain. The results for the makespan in

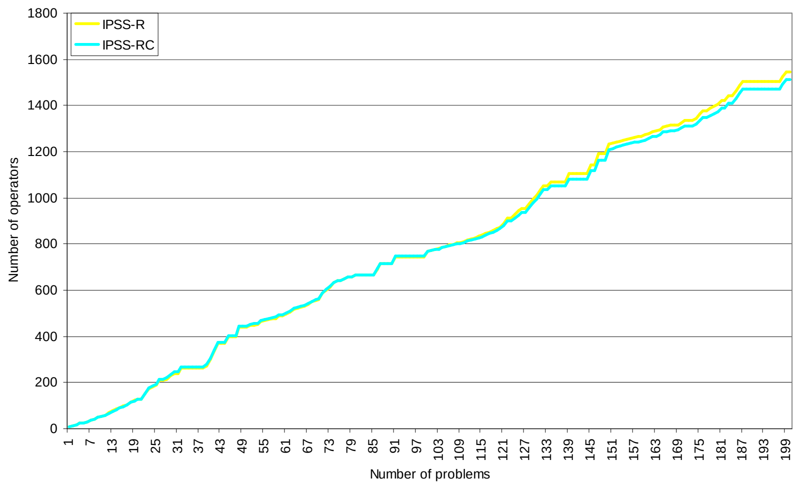


Figure 8.6: Number of operators for the IPSS-R configurations in the ROBOCARE SimpleTime domain.

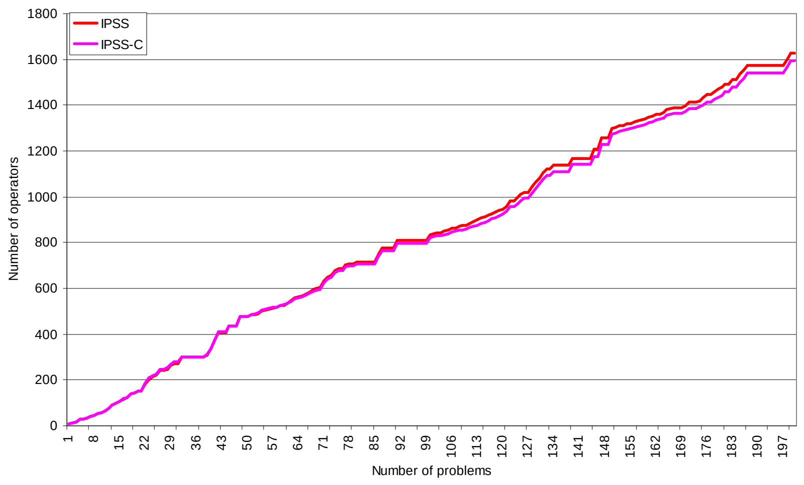


Figure 8.7: Number of operators for the IPSS configurations in the ROBOBUSY SimpleTime domain.

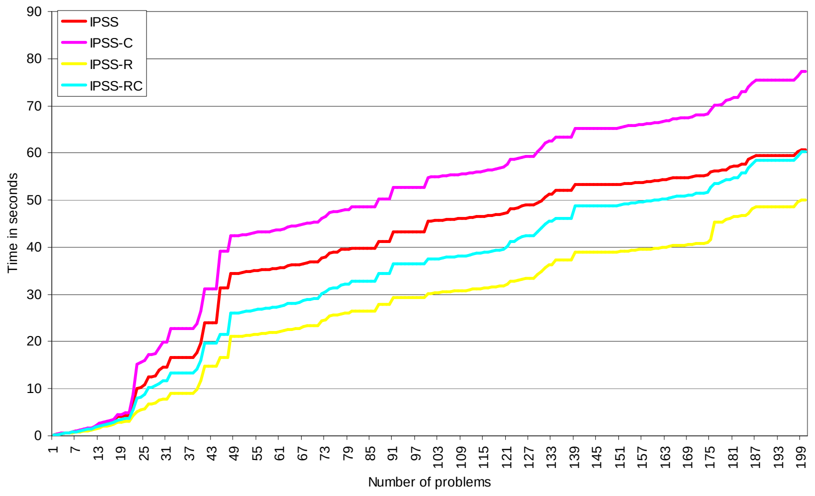


Figure 8.8: Time in seconds for the IPSS configurations in the ROBOCARE/ROBOBUSY SimpleTime domain.

Table 8.4: Number of problems solved by each IPSS configuration in the ROBOCARE Time domain from a total of 200 problems with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
IPSS	38	37	33	38	146	73%
IPSS-R	38	38	35	41	152	76%

IPSS-R are quite similar to IPSS. A potential cause for this behaviour might be due to the robot and person speeds and the numeric distances between rooms that we have added in order to augment the degree of difficulty. The speeds are randomly generated and in same cases the robots speeds are lower than the persons speeds, so some alternative plans are not valid, and there are less possibilities to level the resources used. This is translated into a worse makespan compared to the results in the two domains presented before (the improvement in IPSS-R is of 6%) but the efficiency obtained in the solutions is much better: 62% better in IPSS-R than in IPSS. We have not included the graphics for the number of operators because, as explained before, IPSS-R always uses less number of operators as it does not include the *free-agent* operator.

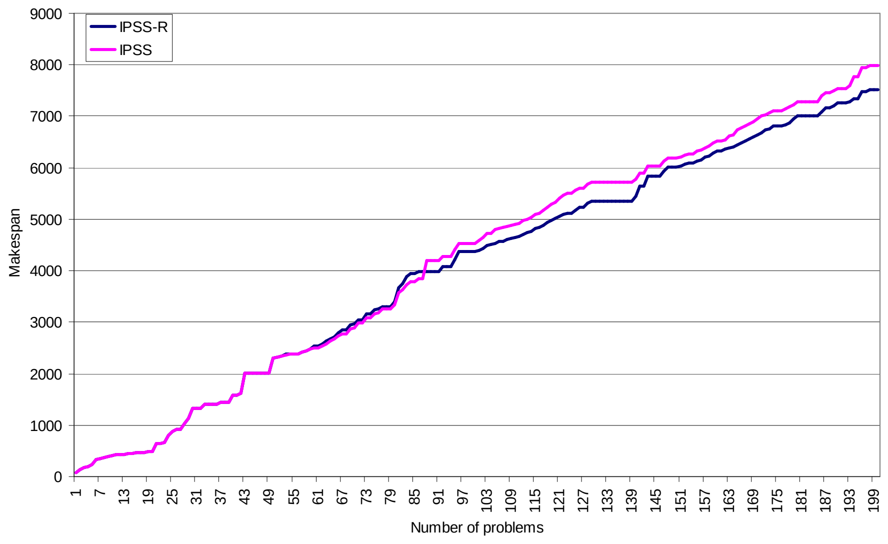


Figure 8.9: Makespan for the IPSS configurations in ROBOCARE/ROBOBUSY Time.

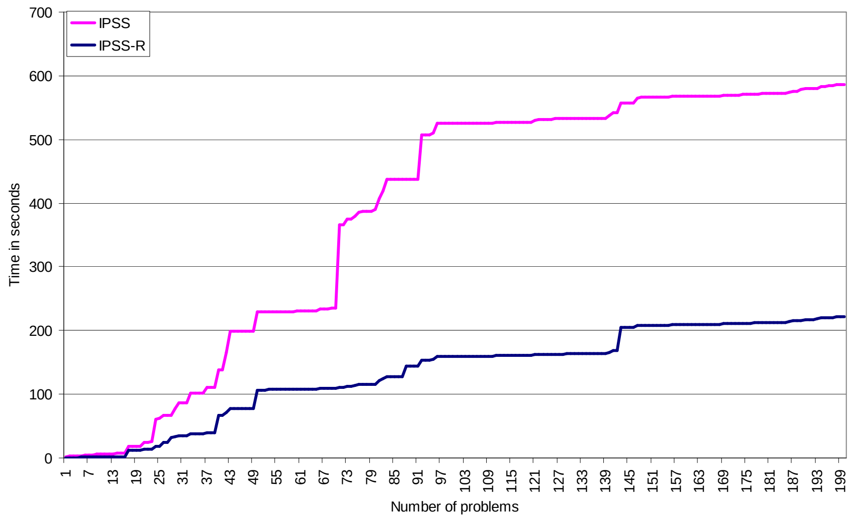


Figure 8.10: Time in seconds for the IPSS configurations in ROBOCARE/ROBOBUSY Time.

8.2.2 A Workflow Domain: BT

The BRITISH TELECOM domain (BT) is an adapted version of the example presented in section 4.4.2, but converted into a non-conditional domain so that it can be used by any planner.

8.2.2.1 The BT Domain

As in the ROBOCARE domain it has also some features that make it specially fitted to separate the resource reasoning from the causal reasoning. All the operations in this domain need to be executed by a human worker, and any worker can perform any action if he/she is not doing something else.

Then, there are some restrictions to consider: one worker cannot perform two operations at the same time; or a cable must be built if the spare card is not available.

We have considered (to make unconditional the example explained in section 4.4.2) that in any case we have to allocate the line, so this can be done in parallel to any other activity instead of waiting that everything is done to allocate the line (go to Appendix E to see the details of this domain). To perform the tasks, the worker should move to the zone where the spares or lines are, in order to accomplish the work that has to be done.

The modeling of the BT domain does not consider if the worker is or not occupied doing something else. So, apart from planners that provide serial solutions, as, for example, SERISTAR (that belongs to the HSP family [19]), FF [85] or QPRODIGY [20], the solution given by other planners as LPG [74] or MIPS [56] is not correct as happens in the ROBOCARE domain.

We have implemented the same solution as in the ROBOCARE domain by defining the *free-worker* action. We have called this domain the BT_OCCU domain (see Appendix E).

We have generated 160 problems, distributed in four groups of forty problems. Each one of this group has a fix number of workers: two (G2), five (G5), eight (G8) and eleven (G11). Each set of forty problems is subdivided in four subsets of ten problems each, with the following features:

- Four lines and spare cards to allocate distributed in four different zones. The workers must move from one zone to another in order to complete the job. All numbers are randomly generated, and the number of goals is between two and four.

- Fourteen lines and spare cards to allocate, distributed in nine different zones. The number of goals is between five and seven.

- Twenty-four lines and spare cards to allocate, distributed in fourteen different zones. The number of goals is between ten and twelve.

- Thirty-four lines and spare cards to allocate, distributed in nineteen different zones. The number of goals is between fifteen and seventeen.

The set of problems for the BT_OCCU domain is exactly the same as in the BT domain with the particularity that in all the problems the availability of the worker must be explicitly set to *not-occupied* (that is, in the initial conditions we add the predicate *(not-occupied <worker>)* for each <worker> in the problem).

Table 8.5 shows the number of solved problems by each one of the IPSS configurations for each subset described above. IPSS-R solved 77% and IPSS 76% of the problems followed by IPSS-C and IPSS-RC with 69% and 67% of solved problems respectively. The total time given to solve each problem is of 180 seconds and the number of control rules coded for this domain is 7. The control rules were generated automatically from HAMLET [21]. For these rules we have not used any human expert help and, as it can be seen, they solve less number of problems in both IPSS configurations. It has been demonstrated that in general, it is quite difficult that a system automatically generates *perfect* control knowledge [4]. They generally increase the efficiency as Minton showed in [116].

Table 8.5: Number of problems solved by each IPSS configuration in the BT domain from a total of 160 problems with a time bound of 180 seconds.

Name	G2	G5	G8	G11	Problems solved	Percentage
IPSS	29	29	33	31	122	76%
IPSS-R	30	30	32	31	123	77%
IPSS-C	26	27	28	30	111	69%
IPSS-RC	26	26	28	27	107	67%

Figure 8.11 shows the makespan for each IPSS configuration. IPSS-R configurations obtain better makespan than IPSS.

The use of control rules improves the makespan in 4% for IPSS and 2% for IPSS-R. But if we compare IPSS-R and IPSS, the improvement obtained by the first one is of 46%. Considering *worker* as a resource allows IPSS-R to obtain better results when the number of *workers* increases. As pointed out, the IPSS philosophy is to minimise plan length, that is translated in over-using some of the resources and do not exploit parallelism. In IPSS-R this behaviour has been modified by allowing that all the resources have the same weight in performing all the activities. The improvement in the makepan is considerable.

With respect to the time taken to produce the plans, IPSS and IPSS-R are more or less equally efficient, but we cannot say the same when we use control rules since the time increases. These results are shown in Figure 8.14 where IPSS gets an efficiency decrease of 12% and IPSS-R of 27%.

We also show the number of operators for each configuration in Figures 8.12 and 8.13. The results highlight that control rules try to minimise the number of operators.

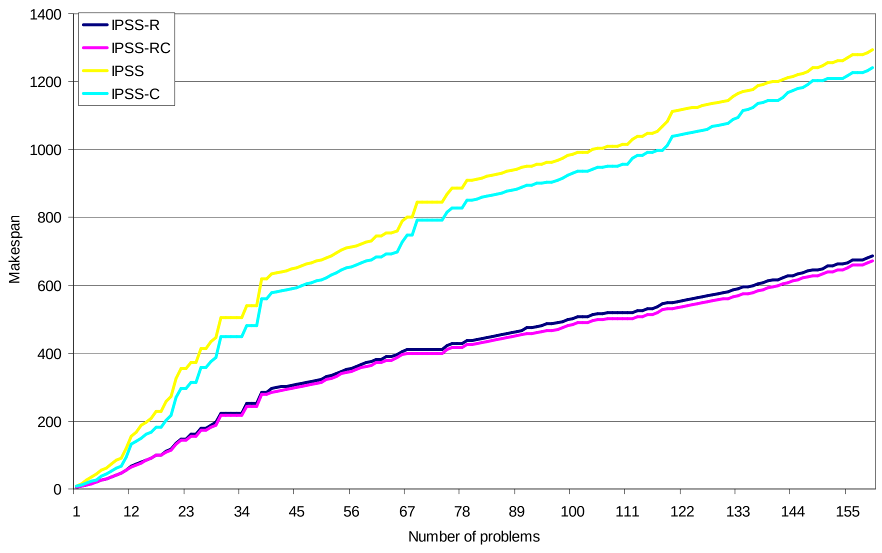


Figure 8.11: Makespan for the IPSS configurations in the BT domain.

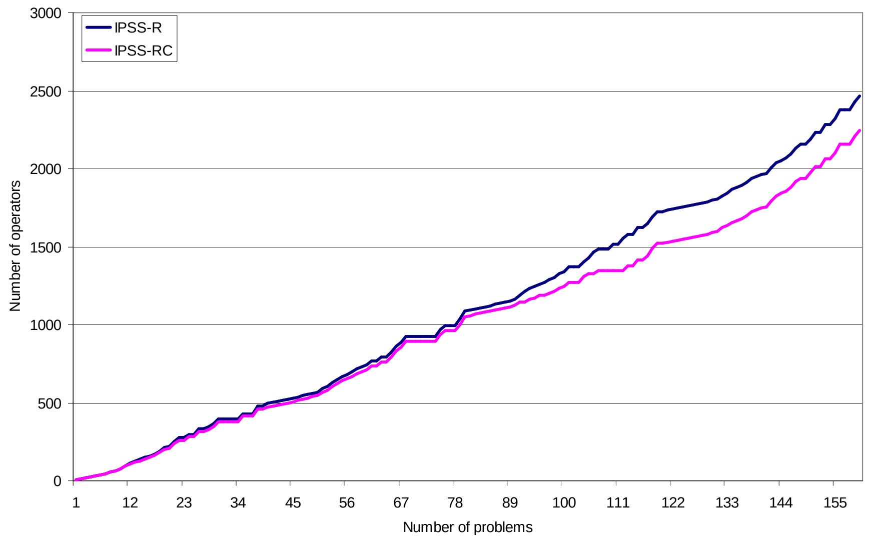


Figure 8.12: Number of operators for IPSS-R configurations in BT.

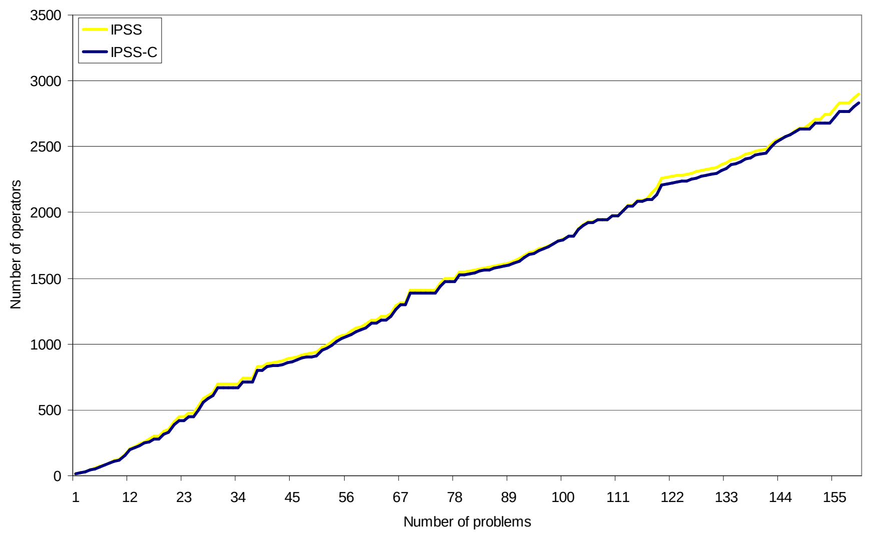


Figure 8.13: Number of operators for IPSS configurations in BT_OCCU.

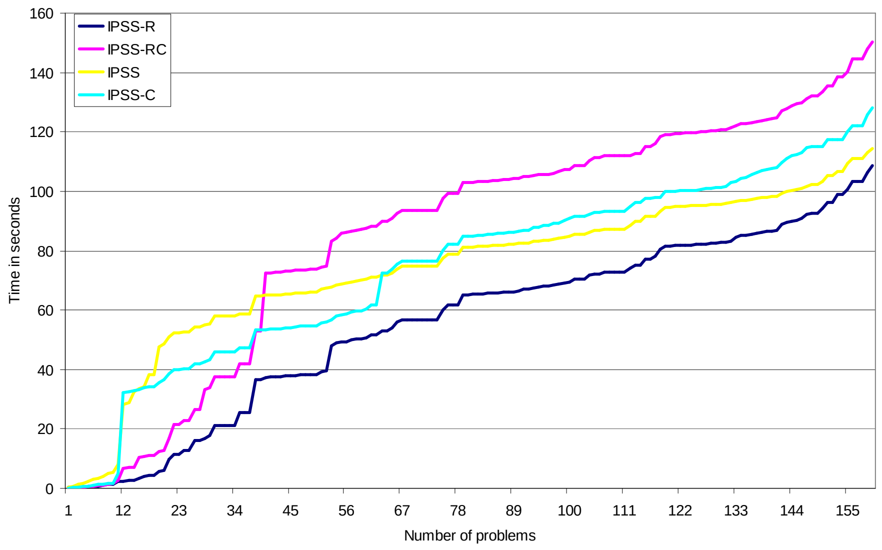


Figure 8.14: Time in seconds for the IPSS-R configurations in the BT/BT_OCCU domain.

8.2.2.2 The BT Simple Time Domain

As we did for the ROBOCARE DOMAIN we have extended the BT domain to consider durations. We have called this domain BT Simple and the one that considers the worker being occupied and not occupied BT_OCCU Simple. In Appendix E a description of this domain in PDDL2.1 syntax is shown.

Table 8.6 shows the number of solved problems by each one of the IPSS configurations for each subset described above. IPSS-R solved 76% of the problems and so does IPSS. They are followed by IPSS-C and IPSS-RC with 69% and 68% of solved problems respectively. The total time given to solve each problem is of 180 seconds and the control rules coded for this domain are the same used for the BT domain.

Table 8.6: Number of problems solved by each IPSS configuration in the BT Simple domain from a total of 160 problems with a time bound of 180 seconds.

Name	G2	G5	G8	G11	Problems solved	Percentage
IPSS	33	29	27	33	122	76%
IPSS-R	36	25	28	33	122	76%
IPSS-C	26	27	28	30	111	69%
IPSS-RC	26	26	28	28	108	68%

Figure 8.15 shows the makespan for each IPSS configuration. IPSS-R with and without control rules obtains better makespan than IPSS. The improvement percentage in the makespan in IPSS-R with respect to IPSS is of 12% and when using the control rules the improvement in both cases is of 2%. Note that in this domain version, the dummy operator *free-worker* has duration zero.

About the time taken to produce the plans, IPSS-R is 38% more efficient than IPSS as Figure 8.18 shows. The control rules increase the total time to produce the plans in 7% in IPSS-R and 12% in IPSS.

We also show the number of operators for each configuration in Figures 8.16 and 8.17. The results highlight that the control rules coded for this domain aim at minimising the number of operators.

As a summary, in this section we have presented two domains: BT and ROBOCARE. They have many things in common, as the separation of resources (*workers* and *robots*) from logical predicates because there are not interaction with goals, and some of the operators share the same abstract model:

moveto_zone $\longrightarrow$ goto_room
built_cable $\longrightarrow$ make_bed
set_spare_cable $\longrightarrow$ clear_bed

However, the ROBOCARE problem is different from the BT one. ROBOCARE domain is more complex as the agents are also in charge of moving people from one room to another. This means that first the agent must find the path where the person is and then find the way to move the person into another place given by the goals.

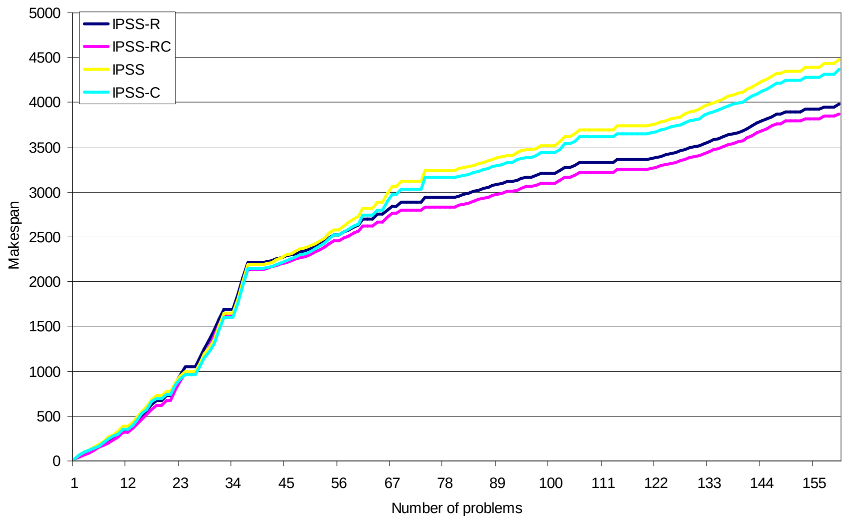


Figure 8.15: Makespan for the IPSS-R configurations in the BT / BT_OCCU Simple domain.

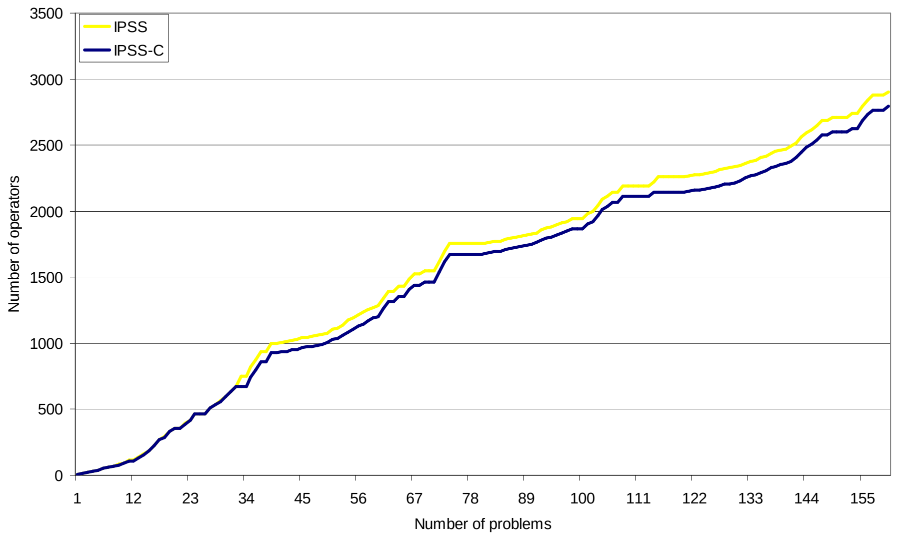


Figure 8.16: Number of operators for the IPSS configurations in BT_OCCU Simple.

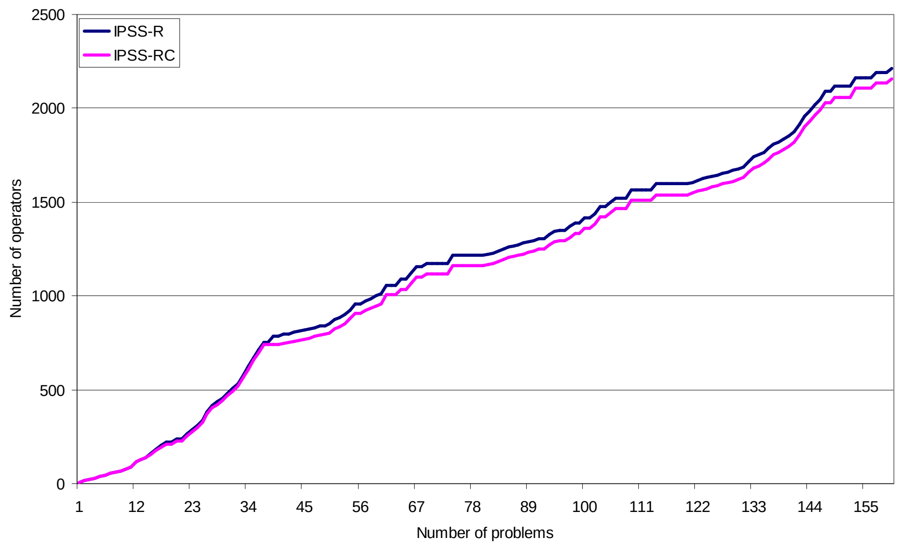


Figure 8.17: Number of operators for the IPSS-R configurations in BT simple.

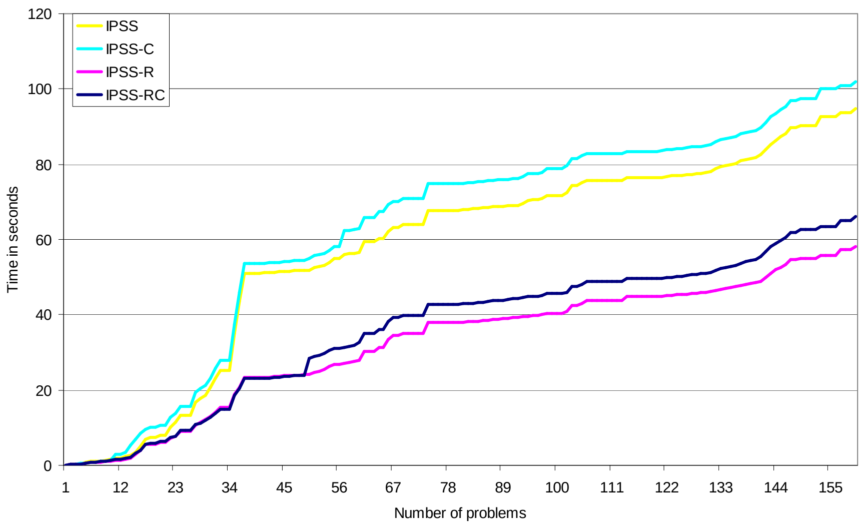


Figure 8.18: Time in seconds for the IPSS configurations in the BT/BT_OCCU simple domain.

But in both of them we see that:

- Consider resources separate from logical predicates and try to maximise its use, minimise the makespan improving the quality of the plans and in the majority of the cases, increasing performance.
- We have modified the default behaviour of IPSS (as well as QPRODIGY) when resources are considered. Instead of looking for plans with less number of operators (and then, maximising the use of few resources), we maximise the use of all the resources available (thus minimising the makespan).
- The quality of the solutions with domain dependent knowledge is better than without knowledge, but it also decreases the time performance.

8.3 IPSS vs. other Planners

In this section we will compare, the different IPSS configurations in three different domains with some state of the art planners and we describe the motivations to choose the planners to compare with IPSS.

8.3.1 The IPC'02 Planners

Table 8.7 shows the planners that have competed in the IPC-2002 [87] competition. The planners that appear with an (*) competed in a previous competition or they have never competed. The ^{dk} symbol that appears in some planners stands for *domain dependent knowledge* that can be hand-coded by the user or automatically learned by the system.

The rows represent each of the planners and the columns the different problems complexity that each domain could have. The last column shows the type of solution that the planner gives: Serial (S) or Parallel (P). The columns are ordered by the difficulty of the problem, and we have divided into two groups: the ones that belong to classical problems (the easier one is *Strips*, then *Numeric* and *Hard Numeric* with functions and different quality metrics) and the problems that support durative actions (first *SimpleTime* where actions have constant durations, then *Time* and *Complex* where the durations depend on the initial conditions of the problems together with arithmetic and/or logical functions and different quality metrics).

The symbols that appear in the table 8.7 have the following meaning:

- X: it means that the planner supports that type of problems.
- –: the planner does not support it.

For the ROBOCARE and BT domains we need to use any planner that gives sequential plans. From all the planners of the competition we have chosen FF [85] for its very good performance. With respect to the type of the domains of the competition, we have focused our attention in Strips, Time and Complex problem types. Taking a look on Table 8.7 the planners that can solve these types of problems are:

Table 8.7: Type of problems solved by the IPC'02 planners.

Name	Strips	Numeric	HNumeric	STime	Time	Complex	Plan
FF	X	X	X	–	–	–	S
IxTeT	–	–	–	X	X	–	P
LPG	X	X	X	X	X	X	P
MIPS	X	X	X	X	X	X	P
Sapa	–	–	–	–	X	X	P
SemSyn	X	X	–	–	–	–	S
SHOP2^{dk}	X	X	X	X	X	X	P
SimPlanner	X	X	–	–	–	–	S
Stella	X	–	–	–	–	–	P
TALPlanner^{dk}	X	–	–	X	X	–	P
TLPlan^{dk}	X	X	X	X	X	X	P
TP4	X	X	–	X	X	X	P
TPSYS	–	–	–	X	–	–	P
VHPOP	X	–	–	X	–	–	P
BB(*)	X	–	–	–	–	–	P
IPSS^{dk}(*)	X	X	X	X	X	X	P
QProdigy^{dk}(*)	X	X	X	X	X	X	S
SS(*)	X	–	X	–	–	–	S

LPG [74], MIPS [56], SHOP2 [125], TALplanner [106], TLplan [10] and TP4 [19]. We have discarded TP4 because it is an optimal planner and ours is not, so the comparison would not be fair. TALplanner and TLplan utilize domain specific search control information to control the search. But any of the other mentioned planners could have been used. We have decided to use LPG, MIPS and FF (when possible) for the three domains.

8.3.2 The Zenotravel Domain

In the previous section we have compared IPSS and their different configurations in two domains (ROBOCARE and BT) that allow the strict separation of resources from the logical predicates. As any of these two domains do not belong to the IPC'02 competition, we wanted to study the IPSS behaviour in one of these domains. We have decided ZENOTRAVEL, in particular the *Time* version as it presents time features. For this domain we cannot use IPSS-R because there are dependencies between the airplanes to use (modeled as binary resources) and the goals. For example, actions as *debark* and *board* passengers in the same airplane can all be done in one time step, but if airplanes are binary resources we will be imposing a precedence link between activities increasing the makespan unnecessary.

We have generated 80 problems, distributed in eight sets of ten problems. Each one of this set has a fix number of goals: one (S1), two (S2), three (S3), five (S5), ten (S10), fifteen (S15), twenty (S20) and fifty (S50); a fix number of five cities, a random number of planes between one to five to fly from one city to another and the following features:

- A fix number of five persons for the first set.
- A fix number of six persons for the second set.
- A fix number of seven persons for the third set.
- A fix number of ten persons for the fourth set.
- A fix number of fifteen persons for the fifth set.
- A fix number of twenty persons for the sixth set.
- A fix number of twenty-five persons for the seventh set.
- A fix number of fifty persons for the eighth set.

We have not used any of the TO planners of Table 8.7 because they do not support durative actions (apart from QPRODIGY). Among the planners that could handle complex metrics are LPG and MIPS. IPSS also allows to minimise the solutions using two metrics: time and a different one.

Table 8.8 shows the number of problems solved by each planner and the percentage that it represents. For IPSS-C we have used 16 control rules coded by a QPRODIGY expert. They help increasing the number of solved problems in almost 40% with respect to QPRODIGY and IPSS. The total time given to solve each problem is 120 seconds. We have included domain dependent knowledge to show that IPSS is an architecture that allows to define metrics different from plan length [20], reason about those multiple criteria [3], prune the search using control rules, or automatically learn them [6, 21] and reason about time and resources (when possible).

Table 8.8: Number of problems solved by each planner in the ZENOTRAVEL *Time* domain from 80 problems with a time bound of 120 seconds.

Name	S1	S2	S3	S5	S10	S15	S20	S50	Problems solved	%
IPSS	9	8	7	5	2	3	1	1	46	57%
IPSS-C	10	10	10	10	10	9	10	7	76	95%
LPG	10	10	10	10	10	10	10	10	80	100%
MIPS	10	10	10	10	10	10	10	10	80	100%

Given that LPG bases it search in a non-deterministic local search, we have run five times each problem and save the best solution, the third best solution found and the worst solution found. This is represented in the graphics by LPG-Min, LPG-Med and LPG-Max respectively.

Figure 8.19 shows the results obtained when the duration of the problems solved is compared using the metric *Time*. IPSS-C finds the solutions with lower duration, followed very closely by LPG-Min. Then MIPS and IPSS and far away LPG-Med. Finally, with a big difference comes LPG-Max.

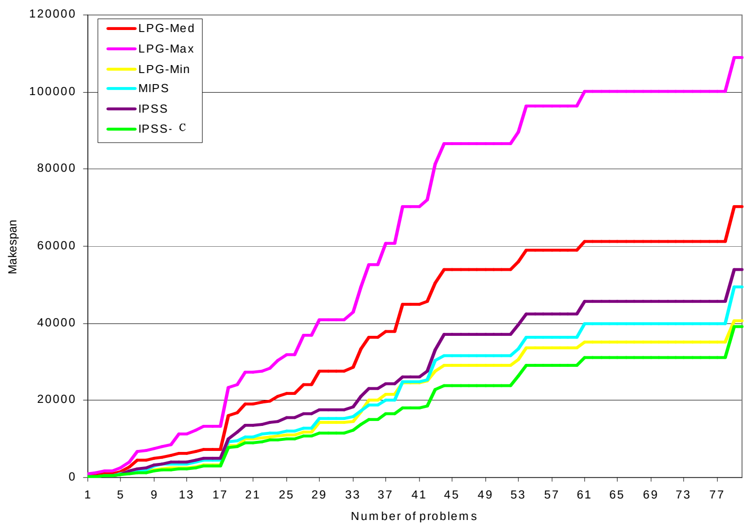


Figure 8.19: Makespan for LPG, MIPS, IPSS and IPSS-C for ZENOTRAVEL.

With respect to the number of steps, MIPS tries to minimise the length of the solutions, followed by LPG-Min, LPG-Med, IPSS-C, LPG-Max and finally IPSS. These results are shown in Figure 8.20 and shows that control rules reduce the number of steps and also plan quality because some control rules tries to maximise as much as possible the resources (airplanes) used.

The best execution time is achieved by LPG (the three runs take almost the same time). Then IPSS and with almost the same execution time come IPSS-C and MIPS (as mentioned before, control rules increase the time to produce the solutions). Figure 8.21 shows the total time in seconds to solve the problems.

But the results in this domain are not exploiting the IPSS features and the power that the *Ground*-CSP layer can offer. In order to understand how IPSS features could be explicited in this domain, we have performed the following experiment. Using IPSS, we can impose a maximum makespan for a given problem since providing this value manually could be difficult, we have devised an automatic way of doing it. We will obtain this value from LPG. We could have used any other planner as MIPS, but it cannot improve the first solution as LPG does.

In this part of the experimentation we will choose randomly one problem of each set that can be solved by IPSS (in total, 8 problems) and make LPG produce a solution. This value will serve as an upper bound for the makespan for IPSS to find a solution. The time (in seconds) that LPG takes to find the solution (will be called t_{lpg}) will be subtracted to the total time of 120 seconds and gives this time bound to

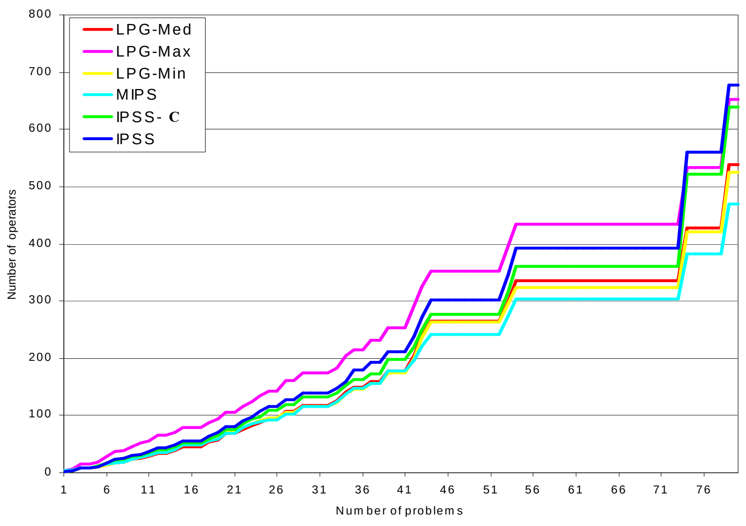


Figure 8.20: Number of operators for LPG, MIPS, IPSS and IPSS-C for the ZENOTRAVEL domain.

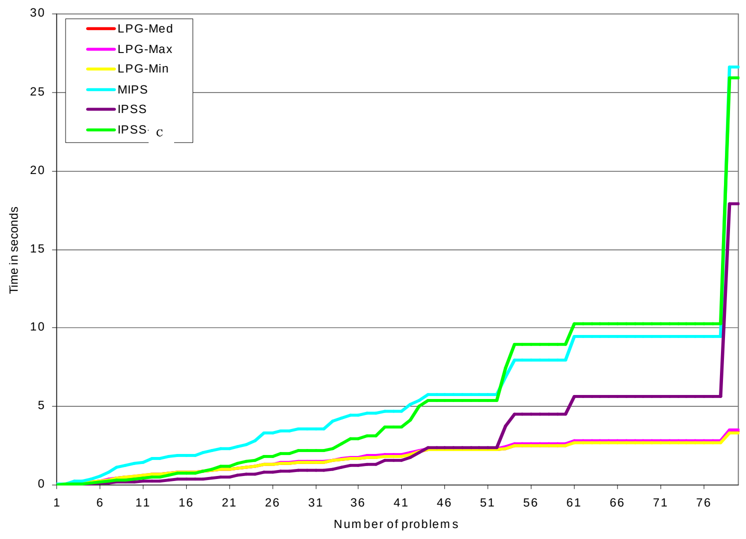


Figure 8.21: Time in seconds for LPG, MIPS, IPSS and IPSS-C for the ZENOTRAVEL domain.

IPSS to find solutions. Then, we run again LPG with a time bound of 120 seconds and compare the last solutions found in both of them after the given time bound.

Table 8.9 shows the results, where the rows represent the problems and the columns: the makespan of the first LPG run, the LPG time to obtain the plan in seconds, the makespan of the plan given by IPSS, the IPSS time to obtain the solution, the LPG makespan with a time limit and the LPG time to find the last solution in the time limit (t_{limit}).

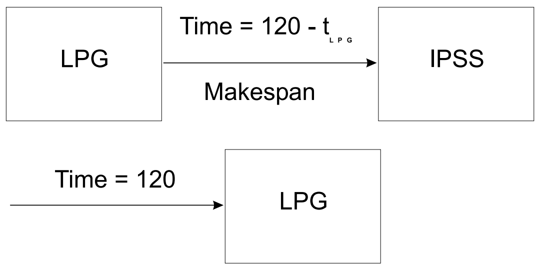


Figure 8.22: Graphical representation for the experiments done in ZENOTRAVEL.

From the table we can see that if the first value obtained by LPG is very close to the *optimal*, IPSS is not able to find a solution to the given makespan bound or if it finds it, the time to find the solution is lower than LPG. But, then it cannot improve it. LPG seems to exploit the total time given to solve the problems. Its local search allows to improve the solutions for this domain. Instead, IPSS keeps searching in a part of the search space that does not allow to improve the solutions for the given time.

Table 8.9: Results when imposing a makespan in 8 problems in the ZENOTRAVEL domain.

Problem	LPG	t_{lpg}	IPSS	t_{ipss}	LPG	t_{limit}
P1	327,6	0,01	151,8	0,02	151,8	0,06
P2	552,3	0,04	521,6	0,04	552,3	119,9
P3	466,8	0,06	435,2	0,18	389,98	0,3
P4	840,1	0,09	759,23	102,34	634,393	1,23
P5	1436	0,13	991,1	0,22	823,45	19,23
P6	9385,7	0,45	6870,7	1,32	5345,23	28,31
P7	2125	0,52	–	–	2045,8	35,24
P8	5953,9	0,57	2838,95	12,32	2520,8	52,11

8.3.3 The Robocare Domain

As mentioned in section 8.2.1.1, we will use two versions of the ROBOCARE domain: the original version as explained in [131] and the free/busy version of the robots called ROBOBUSY domain.

For the first version we have compared the following planners: FF (the metric-FF version)[2], QPRODIGY (QP)[3] that provides solutions as a sequence of actions, and the system used for the original ROBOCARE version, the 42 version of BlackBox (BB) [4] and the O-OSCAR [32] scheduler [131]. Table 8.10 shows the number of problems solved in each group by each planner/system and the total percentage that it represents. The problems have been solved using the versions provided by the authors and setting a total execution time for each problem of 180 seconds.

Due to the low percentage of problems solved by SERISTAR (SS)[5] and that it obtains optimal solutions, we will focus our attention on the other systems as IPSS does not look for optimal solutions. But it will serve us to analyse how affects to find optimal solutions in this domain with respect to the problems solved.

The FF planner solved all the problems. It was followed by QPRODIGY/IPSS, IPSS-RC, BB + O-OSCAR, IPSS-R and IPSS-C with the same number of problems and finally SERISTAR (see Table 8.10).

Table 8.10: Number of problems solved by each planner in the ROBOCARE domain with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
IPSS/QP	38	34	36	38	146	73%
IPSS-R	38	34	37	37	146	73%
IPSS-C/QP-R	37	33	37	37	144	72%
IPSS-RC	37	34	37	36	144	72%
BB + O-OSCAR	31	33	41	39	144	72%
FF	50	50	50	50	200	100%
SS	27	21	23	21	92	46%

Figures 8.23 and 8.24 show the makespan obtained with problems with one robot and problems with more that one robot. The reason to perform this distinction is to appreciate that there is no influence in the search performed by IPSS-R and IPSS with respect to QPRODIGY when there is just one robot. In those cases, it is QPRODIGY who decides the different possibilities for IPSS without any help from the *Deorder-layer*, *Ground*-CSP or *Meta*-CSP layers (there are no possible de-orderings or resource bindings because all the activities are performed by the same robot).

The results are worse because the solutions found when considering one agent are much longer than the ones found by FF or BB. In that case the makespan is equal to the number of operators.

The reason that FF and BB find solutions with fewer operators is that both used a GRAPHPLAN algorithm. In the first case, the explicit solution will serve as a goal distance estimator and in the second case to convert into a CNF well defined formula and incrementing the length of the solution in case of a failure.

[2]http://www.informatik.uni-freiburg.de/~hoffmann/ff.html
[3]http://scalab.uc3m.es/~dborrajo/software.html
[4]http://www.cs.washington.edu/homes/kautz/blackbox/blackbox-download.html
[5]This version is available at: http://www.ida.liu.se/~ pahas/hsps/

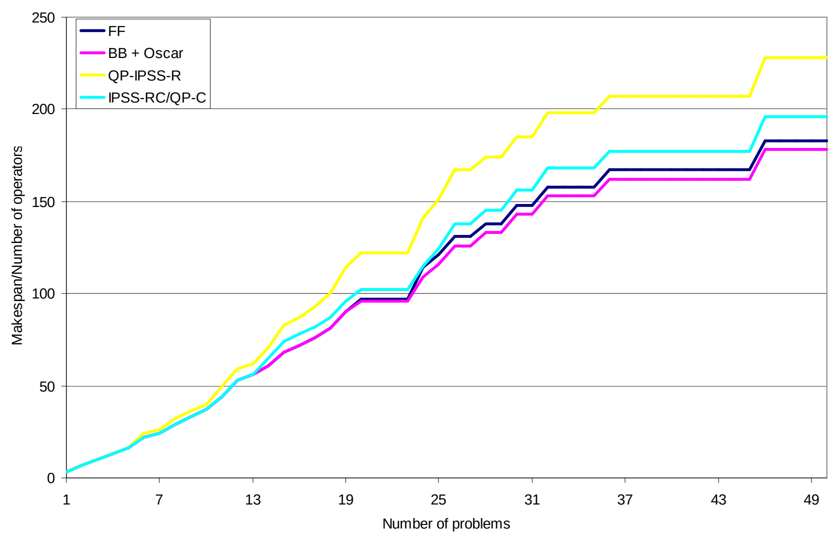


Figure 8.23: Makespan for FF, BB + O-OSCAR, IPSS-R, and QP considering one robot.

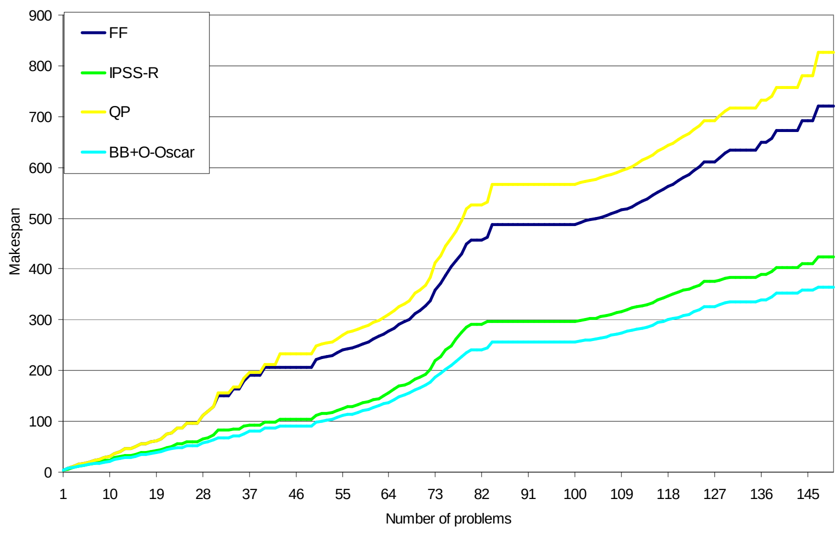


Figure 8.24: Makespan for FF, BB + O-OSCAR, IPSS-R and QP in problems with more than one robot in ROBOCARE.

In both cases, the algorithms used can estimate the length of the solution, but the means-ends analysis of QPRODIGY cannot.

When the problems with one robot are not considered, IPSS-R get closer solutions to the BB + O-OSCAR system. Now, the default QPRODIGY search is being modified to reason in parallel and in some way it works as a GRAPHPLAN algorithm in the way that in each step it tries to maximise the number of activities to be executed in parallel. IPSS-RC gets almost the same global makespan as BB + O-OSCAR when more than one agent is considered.

With regards to the number of operators in the solution, FF finds the shortest solutions as it estimates solutions with the fewer steps as possible, followed by BB + O-OSCAR (the solutions have more operators but can be performed more in parallel), IPSS-R and QP (although it tries to minimise the number of operators, it does not have an initial estimation of the total length, so results are not as good as in FF). Figure 8.25 shows these results.

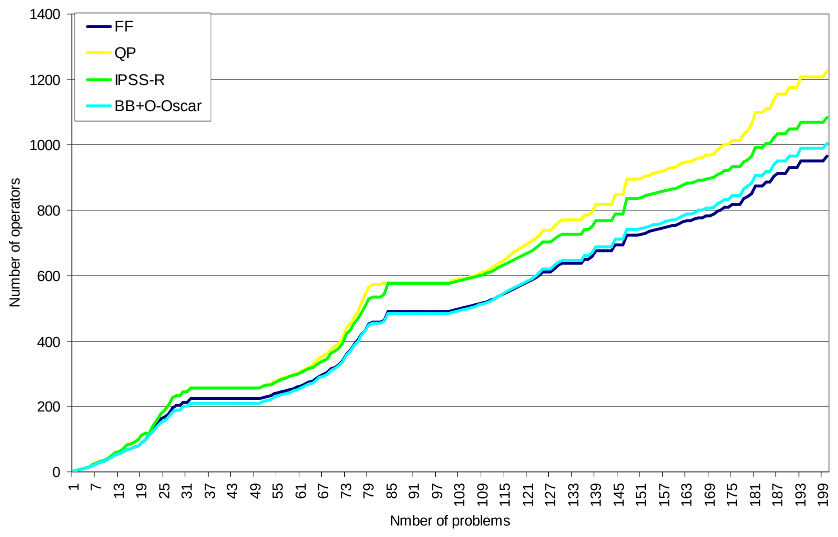


Figure 8.25: Number of operators for FF, BB+O-OSCAR, QP and IPSS-R in ROBOCARE.

With respect to the execution time, FF is amazingly fast compared to rest of planners as shown in Figure 8.26. Instead, BB + O-OSCAR is very slow. We should say that the total time is not increased by the scheduler but by the planner. BB finds good parallel solutions at the cost of not considering the availability of other robots and paying for this with a high execution time.

In the light of the experiments, we can say that IPSS-R provides a good balance between the time to solve the problems (after FF it is the second faster planner) and the makespan found (quality).

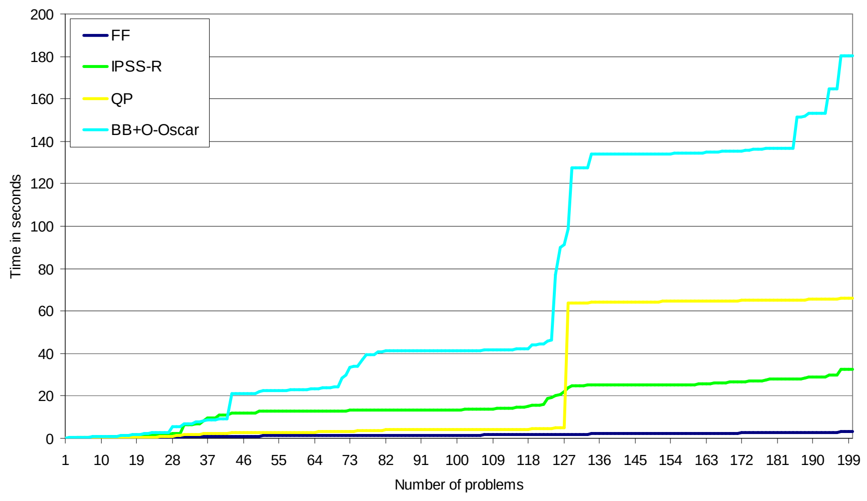


Figure 8.26: Execution time for FF, BB+O-OSCAR, QP, IPSS and IPSS-R in ROBOCARE.

8.3.4 The Robobusy Domain

The second version of the ROBOCARE domain has been tested against the planners MIPS version v3[6], LPG1.1[7] and the 42 version of BLACKBOX. The problems have been solved using the versions provided by the authors.

With respect to the total number of problems, Table 8.11 shows the number of problems solved by each one and the percentage that it represents. As it might have been expected, BLACKBOX solves 10% less problems than for the ROBOCARE domain. Considering resources as predicates within the domain definition makes the search much more difficult. The percentage of solved problems for the MIPS planner is less than 50%. The reason is that in this domain as the number of robots and goals increases, the possible world states increase exponentially.

Given that MIPS applies binary decision diagrams to represent these world states, it becomes in most of the cases intractable.

As explained before, LPG is run five times for each problem and save the best solution (LPG-Min), the third best solution (LPG-Med) and the worst solution found (LPG-Max).

Figure 8.27 shows the results when the makespan is compared. IPSS-R gets close solutions to the best ones that LPG can find in five runs. As LPG search space consists

[6]Available in http://www.informatik.uni-freiburg.de/~mmips/download/
[7]http://prometeo.ing.unibs.it/lpg/download-lpg1.1.html

Table 8.11: Number of problems solved by each planner in the ROBOBUSY domain using the same 200 problems of the ROBOCARE with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
MIPS	15	34	21	23	93	47%
LPG	50	48	47	47	192	96%
BB	30	31	33	30	124	62%
IPSS/QP	38	34	36	38	146	73%
IPSS-R	38	34	37	37	146	73%

of action graphs and the search performed is based on local search, it can find very good results, but it can also find bad results (LPG-Max). The best solutions are found by BLACKBOX. As mentioned in the ROBOCARE experiments, it can exploit parallelism thanks to its graph construction but, if we include resources as part of the logical predicates its performance decreases. MIPS obtains results comparable with LPG-Med and IPSS improves the results of MIPS and LPG-Med. The worse result, as expected, was by LPG-Max.

If we run LPG and IPSS-R during 180 seconds looking for multiple solutions, each one better than the previous, we can see the results in Figure 8.28. IPSS-R-Q finds globally as good solutions as LPG-Q configurations except when there is one agent where the search is not modified by the three layers. As we did for the ROBOCARE domain, the possibility of imposing an horizon constraint on a problem (obtained from another execution) and maximising the resources used in each step allows to use the set of propagation rules (dominance conditions) as GRAPHPLAN based planners do and also the possibility of exploiting an issue strictly correlated to the integration of planning and scheduling. Note that in Figure 8.1 the global makespan is lower than in Figure 8.28. The reason is that in Figure 8.1 there were four configurations to compare and as mentioned before, the four configurations did not solve the same problems. In Figure 8.28 there are just two IPSS configurations so the global makespan is higher because more problems were solved.

Wtih respect to the number of operators, IPSS-R finds the solutions with less number of operators, because as we have mentioned in the last section in the ROBOCARE domain, the *free-agent* operator is not used (see Appendix D), so obviously the length must be shorter. MIPS and LPG provide very similar solutions. The results are shown in Figure 8.29.

The execution time for LPG and its variants is very short in contrast to the long execution time of BLACKBOX and the rest of the planners. IPSS-R and MIPS obtain a medium execution time being shorter in IPSS-R as Figure 8.30 shows.

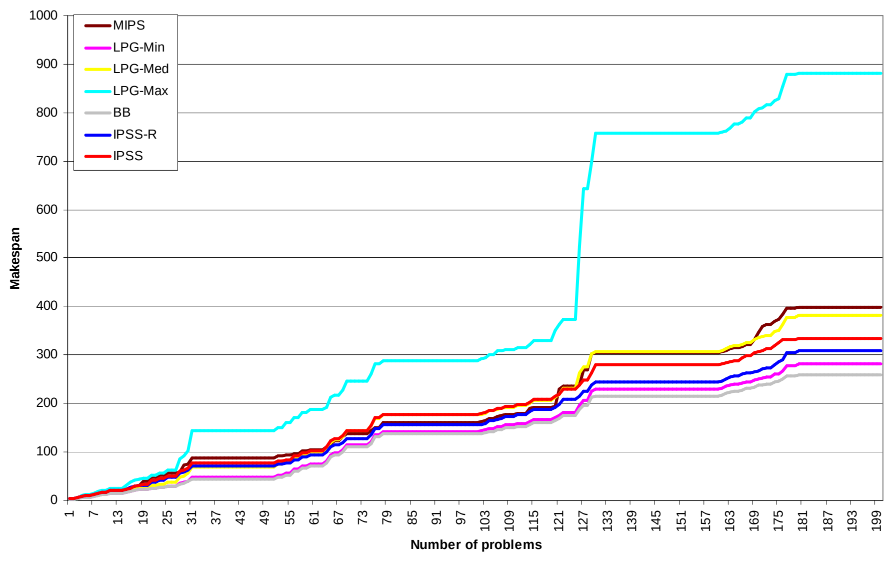


Figure 8.27: Makespan for MIPS, BB, LPG, IPSS and IPSS-R in ROBOBUSY.

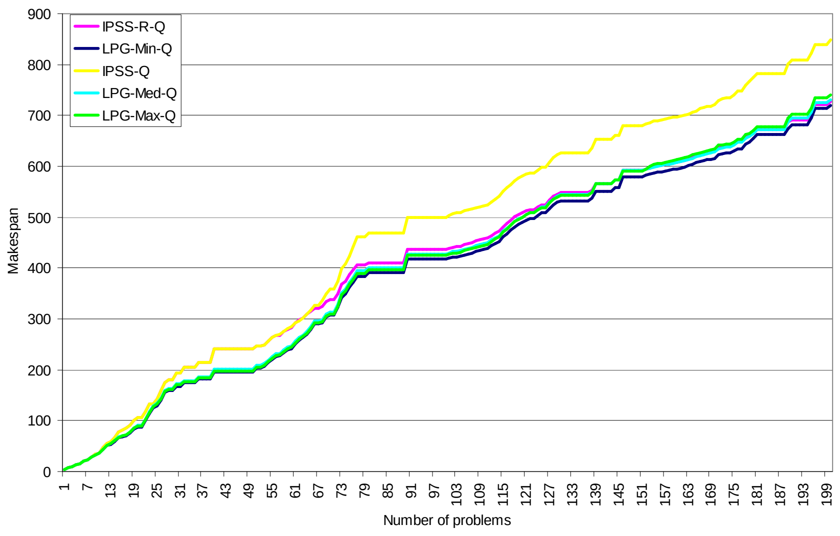


Figure 8.28: Makespan for LPG-Q, IPSS-Q and IPSS-R-Q for ROBOBUSY.

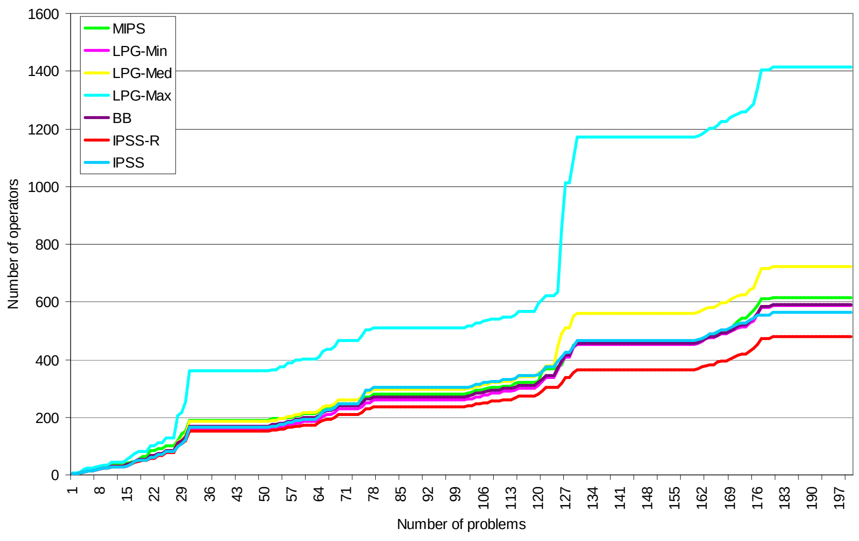


Figure 8.29: Number of operators for MIPS, BB, LPG, IPSS and IPSS-R for ROBOBUSY.

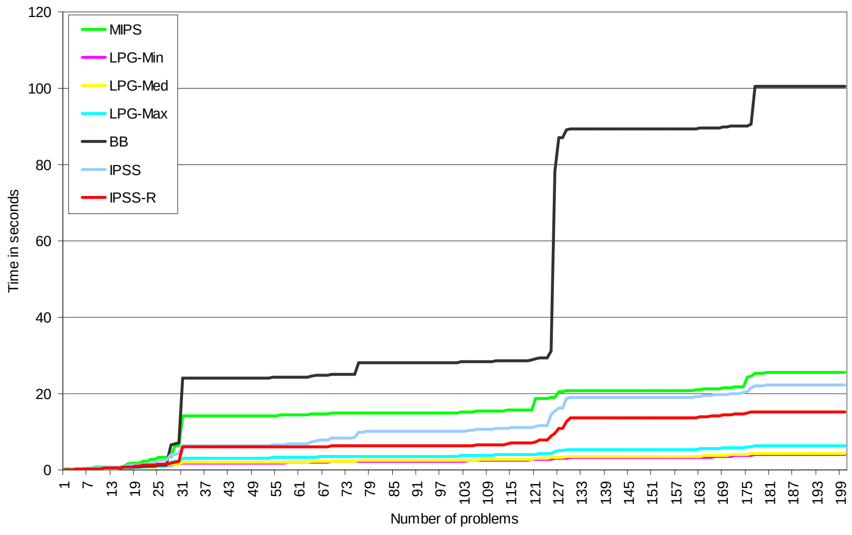


Figure 8.30: Time in seconds for MIPS, BB, LPG, IPSS and IPSS-R for ROBOBUSY.

The results show that the separation of resources from the logical part improves the search and decreases the execution time as IPSS-R shows in the ROBOBUSY domain. The features that ROBOBUSY domain present also make specially fit to GRAPHPLAN based planners, but as resources increase, the problems become more complex. Although this is also true in IPSS-R dividing tasks to different solvers (agents) seems more reasonable than letting this task to a unique solver: we are implicitly using a kind of *divide and conquer* lemma to face the problem.

8.3.4.1 The Robobusy SimpleTime Domain

If we now consider the domain with constant durations, that is, the ROBOBUSY SimpleTime domain, we have the values of Table 8.12 that shows the number of problems solved by each planner in each group. Again, the problems used for this domain are the same as the ROBOCARE SimpleTime domain. LPG solves almost all the problems. It is followed by the IPSS family and finally MIPS with over 50% of the problems.

Table 8.12: Number of problems solved by each planner in the ROBOBUSY Simple Time domain from a total of 200 problems with a time bound of 180 seconds.

Name	G1	G2	G5	G10	Problems solved	Percentage
LPG	50	49	50	48	197	98,5%
MIPS	28	22	30	28	108	54%
IPSS	38	35	37	36	146	73%
IPSS-R	38	38	38	41	155	78%

The makespan results are despicted on Figure 8.31. The best makespan value is achieved as in the ROBOBUSY domain by LPG-Min and almost the same value is obtained by IPSS-R. Then, IPSS and the MIPS makespan is between LPG-Med and LPG-Max.

If we run LPG, IPSS and IPSS-R during 180 seconds to look for quality solutions we have the results of Figure 8.32, where LPG-Q and IPSS-R-Q have globally the same values. These results show again that the possibility of imposing an horizon on a problem, maximising the resources that can be used in order to solve it and the use of propagation rules intrinsic to the integration of planning and scheduling allows to improve the IPSS performance and be as competitive as LPG is.

With respect to the number of operators (see Figure 8.33) MIPS gets the best values followed by IPSS and finally the LPG family. As we have pointed out in the introduction of this chapter, the best makespan values are not obtained by less number of operators, as shown by the LPG results. (We do not mention IPSS-R results as this configuration uses the domain without the *free-agent* action and then the number of operators is less).

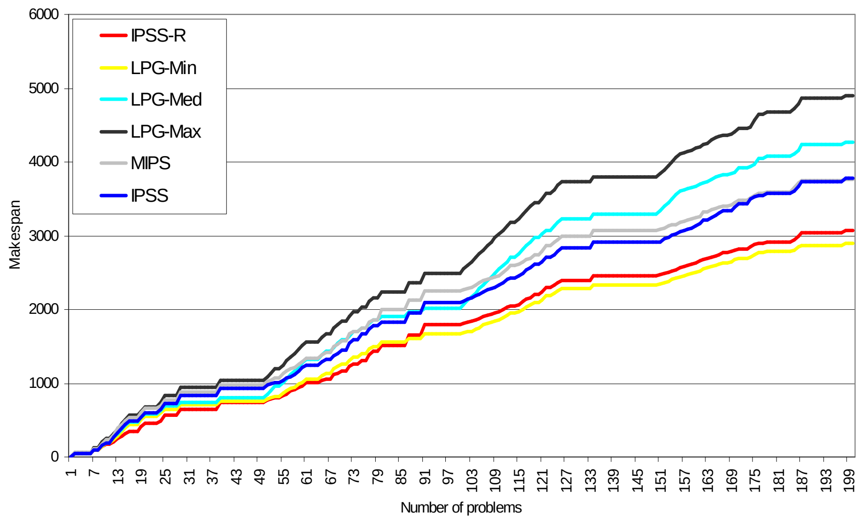


Figure 8.31: Makespan for MIPS, LPG, IPSS and IPSS-R in ROBOBUSY SimpleTime.

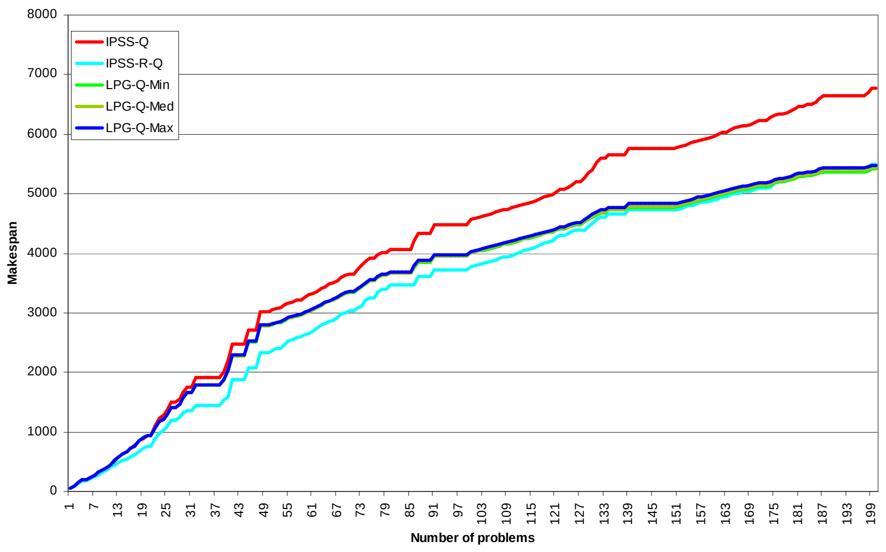


Figure 8.32: Makespan for LPG-Q, IPSS-Q and IPSS-R-Q for ROBOBUSY SimpleTime.

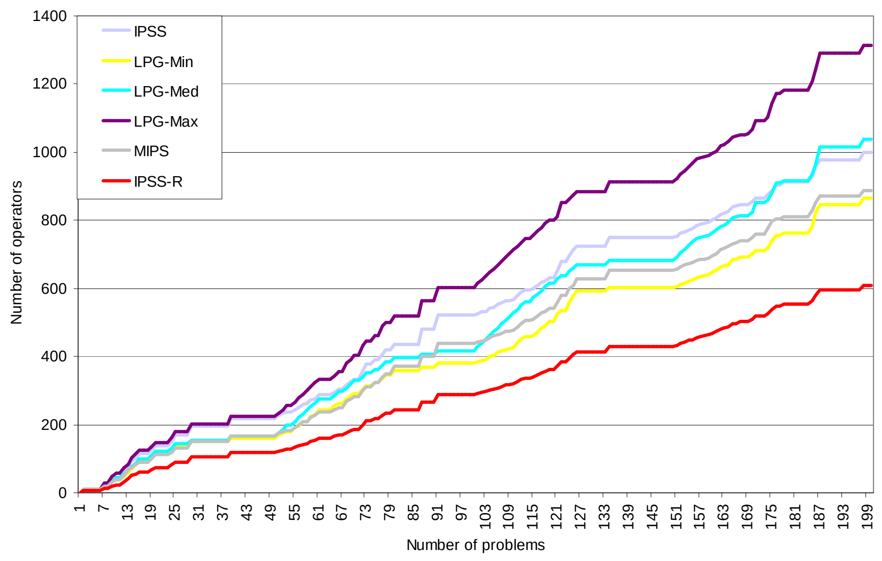


Figure 8.33: Number of operators for MIPS, LPG, IPSS and IPSS-R for ROBOBUSY SimpleTime.

In Figure 8.34 the execution time for each planner to solve the ROBOBUSY Simple Time problems is shown. The worst results during most part of the experiments were by IPSS-R, as the problems that it solved easily are not solved by MIPS and then not considered, worsening its performance related to IPSS (see Figure 8.8 where the different IPSS time configurations were compared). But MIPS have difficulties in solving the problems with higher number of robots as the interactions between goals and world states increase; this is translated into an increasing execution time and, at the end, the worst time.

8.3.4.2 The Robobusy Time Domain

The last version of the ROBOBUSY domain is ROBOBUSY Time domain where the durations are functions depending on the initial conditions of each problem. This is the domain version with less number of problems solved by each planner, but IPSS-R still solves the same percentage. The distribution of the problems solved in each group is shown in Table 8.13. As explained on section 8.2.1.3, we have added the robots and persons speeds and the distance between the different rooms to the original ROBOCARE version, incrementing the problems complexity.

Our purpose when designing this version was as in the IPC-02 competition to augment the difficulty, but we did not realise until we analysed the results that at the same time we were reducing the independence of the resources to choose.

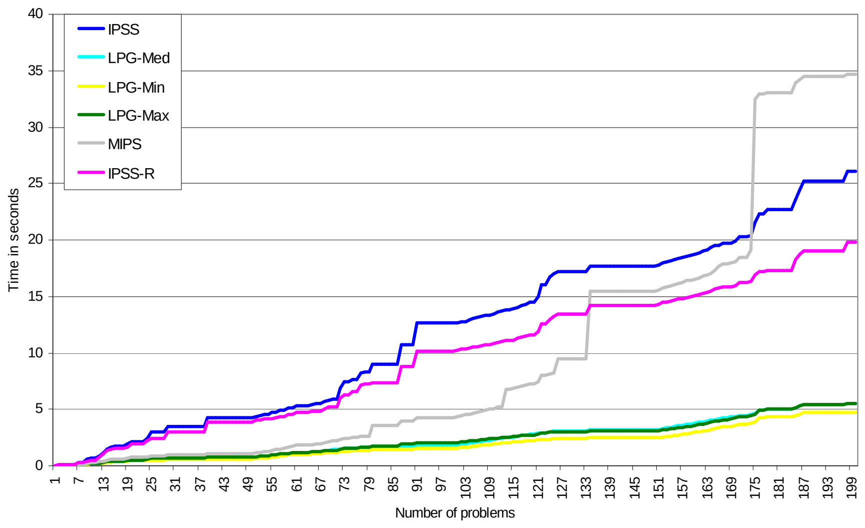


Figure 8.34: Time for solving problems by MIPS, LPG, IPSS and IPSS-R in ROBOBUSY SimpleTime.

Table 8.13: Number of problems solved by each planner in the ROBOBUSY Time domain using the same 200 problems of the ROBOCARE domain.

Name	G1	G2	G5	G10	Problems solved	Percentage
MIPS	23	19	20	29	91	45,5%
LPG	43	43	45	45	176	88%
IPSS	38	37	33	38	146	73%
IPSS-R	38	38	35	41	152	76%

The makespan values are shown in Figure 8.35. The makespan differences between IPSS and IPSS-R are not as significant as in the other two versions explained before. The reason is that the *meta*-CSP layer finds for these problems logical constraints imposed by the domain theory causing interactions with the *robots* (agents) that should perform the activities. The resources cannot be freely chosen as it happened in the other two versions. Then, there are less options for the *meta*-CSP layer in combination with the leverage heuristic to bind the resources.

Figure 8.36 shows the results in the makespan when we let LPG, IPSS and IPSS-R run during 180 seconds looking for more than one solution. As it happened in Figure 8.35, the results are worse in the IPSS configurations (better in IPSS-R than in IPSS) than for LPG.

About the number of operators (see Figure 8.37), as it happened in ROBOBUSY Simple Time, is MIPS that gets the best values followed by IPSS and finally the LPG family. IPSS-R results are the best ones as this configuration does not consider the *free-agent* action.

Figure 8.38 shows the execution time in seconds of each planner. Again, the LPG configurations obtain the faster solutions followed by IPSS-R and MIPS. Far away from the rest of the planner comes IPSS.

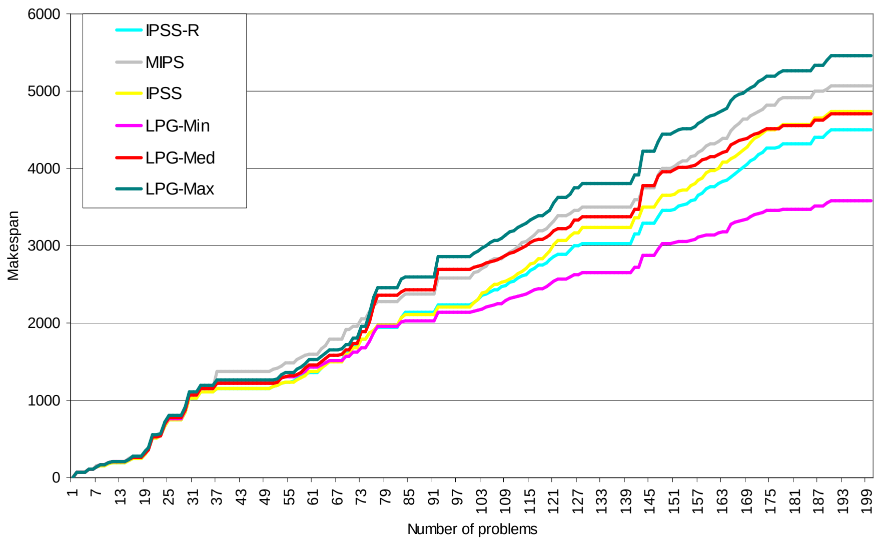


Figure 8.35: Makespan for MIPS, LPG, IPSS and IPSS-R in ROBOBUSY Time.

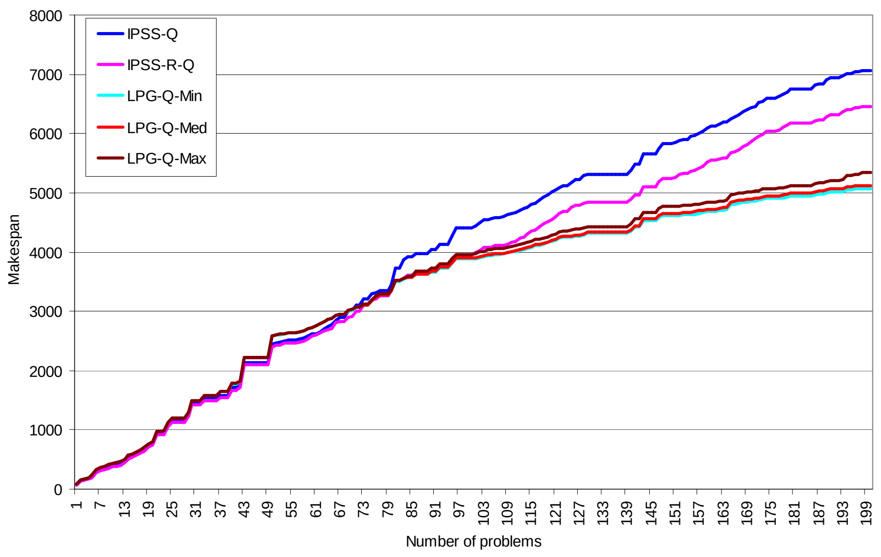


Figure 8.36: Makespan for LPG-Q, IPSS-Q and IPSS-R-Q for ROBOBUSY Time.

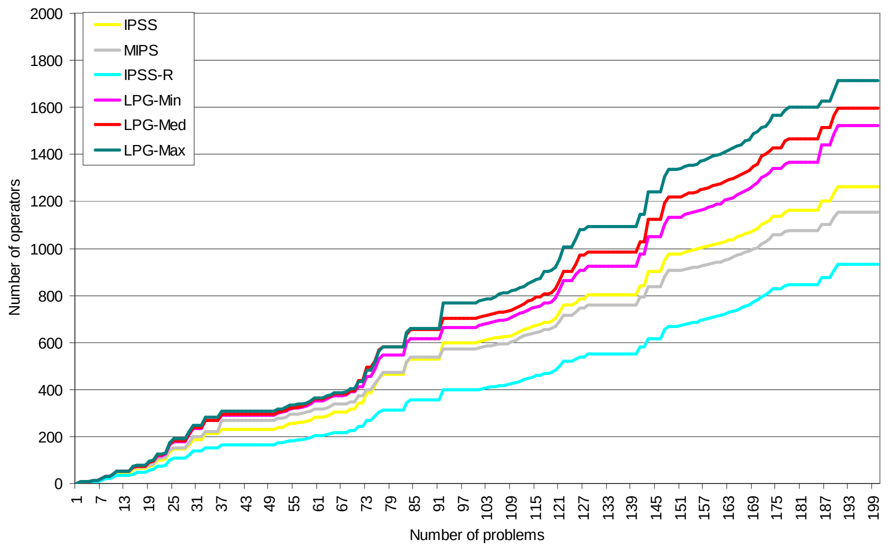


Figure 8.37: Number of operators for MIPS, LPG, IPSS and IPSS-R for ROBOBUSY Time.

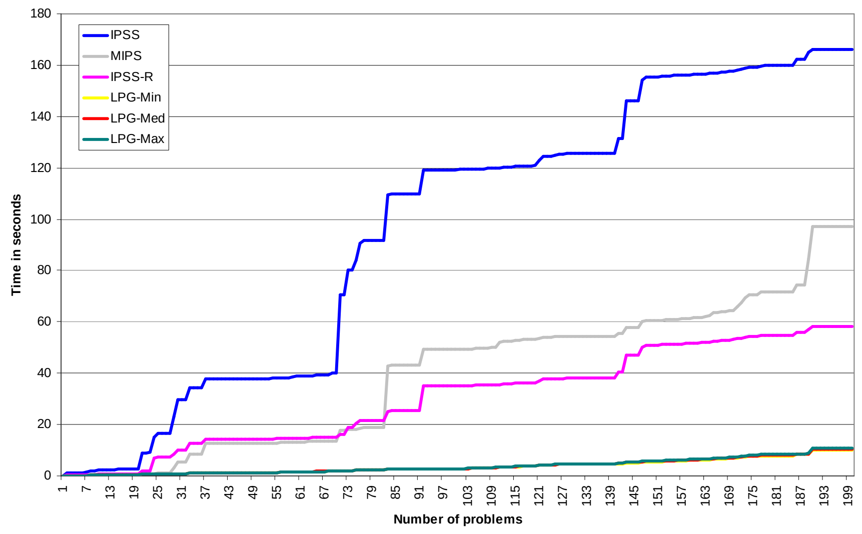


Figure 8.38: Time to solve problems by MIPS, LPG, IPSS and IPSS-R in ROBOBUSY Time.

8.3.5 A Workflow Domain: BT

The third domain is a reduced version of the example of installing a new telephone line in BRITISH TELECOM, explained in section 4.4.2. As it happened in the ROBOCARE domain, we have to consider the predicates *occupied* and *not-occupied* in order to obtain feasible solutions (as one worker cannot perform two activities at the same time). As mentioned before, this version of the BT domain is called BT_OCCU domain and both versions have the same 160 problems distributed in 4 sets of 40 problems. Each set has two, five, eight and eleven workers respectively to perform the activities planned or goals imposed in each problem (see section 8.2.2).

8.3.5.1 The BT Domain

Table 8.14 shows the planners used for the BT domain (we have chosen the same planners as in ROBOCARE, see section 8.3.1 to see why we have chosen those planners). The top table shows the planners that were run using the BT version (FF, and IPSS-R and QP with and without control rules) since the table in the bottom shows the planners that use the BT_OCCU version as they reason about activities in parallel (MIPS, IPSS and LPG). From this table we can see that FF and LPG solve all the problems. Then QP, IPSS-R, IPSS, and finally the configurations with control rules. The control rules as explained on section 8.2.2 were created automatically by HAMLET without any human refinement what explains the decreasing factor in the problems solved.

Table 8.14: Number of problems solved by each planner in the BT domain with a time bound of 180 seconds.

Name	G2	G5	G8	G11	Problems solved	Percentage
IPSS-R	30	30	32	31	123	77%
QP	30	31	33	32	126	79%
IPSS-RC	26	26	28	27	107	67%
QP-C	25	23	29	31	108	68%
FF	40	40	40	40	160	100%
IPSS	29	29	33	31	122	76%
IPSS-C	26	27	28	30	111	69%
MIPS	22	19	20	24	85	53%
LPG	40	40	40	40	160	100%

Figure 8.39 shows the makespan obtained for all the planners in problems solved by all. From the graphic, we conclude that IPSS-R with and without control rules finds the best global makespan for the 160 problems in the BT domain. Then LPG-Min and the IPSS configurations. MIPS finds better solutions than the worse LPG run, and finally, the TO planners, as the makespan coincides with the number of operators: first FF and then the QP configurations.

If we let the planners (just LPG and IPSS are able to) to look for more than one solution during 180 seconds, the results are shown in Figure 8.40. Again as it occurred in ROBOCARE, IPSS-R-Q configuration finds the best solutions.

With regards to the number of operators in the solutions, FF finds the shortest solutions, followed by QP and then the IPSS-R configurations. With respect to the planners that use the BT_OCCU domain, it is MIPS the planner that uses less number of operators, followed by the IPSS configurations and finally LPG runs. Figure 8.41 shows these results that again show that plans with less number of operators do not guarantee plans with less makespan.

With respect to the execution time, FF is faster than any of the LPG runs or the rest of planners as Figure 8.42 shows. Due to the low percentage of problems solved by MIPS, there were some problems (as Figure 8.18 shows) were IPSS-R found the solution faster than IPSS. In this graphic, these problems were not solved by MIPS. Because we have plotted only problems solved by all configurations, they are not considered so that is why IPSS-R obtains worse results than IPSS.

8.3.5.2 The BT Simple Time Domain

As in the competition and in the ROBOCARE domain, we have created a Simple Time version of the BT domain.

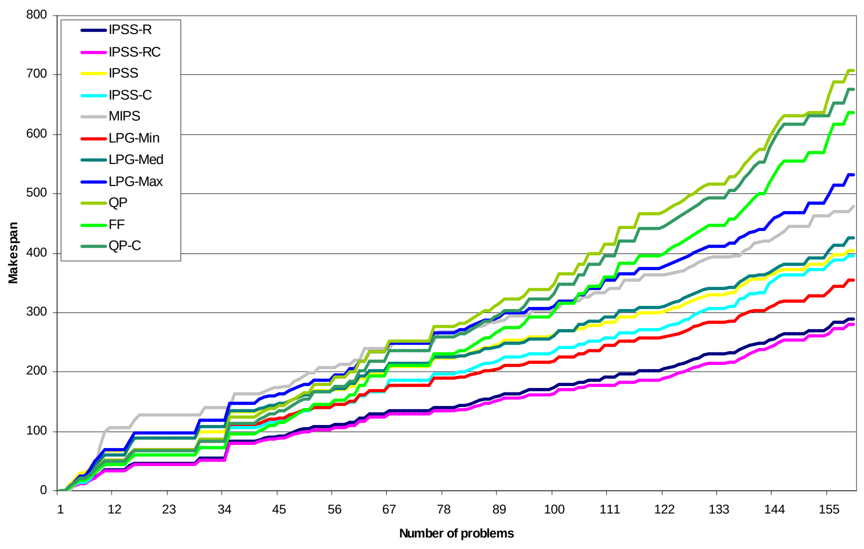


Figure 8.39: Makespan for FF, QP, LPG, MIPS, IPSS and IPSS-R in BT.

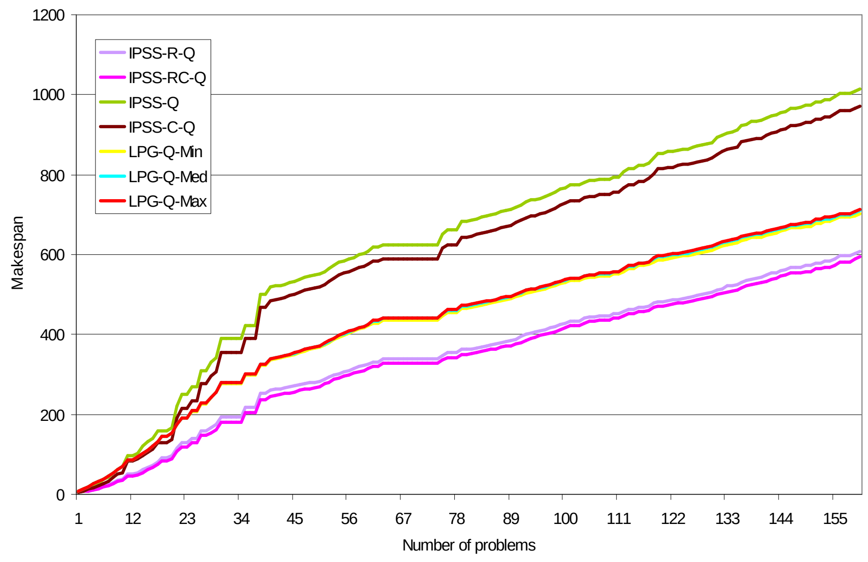


Figure 8.40: Makespan for LPG-Q, IPSS-Q and IPSS-R-Q in BT.

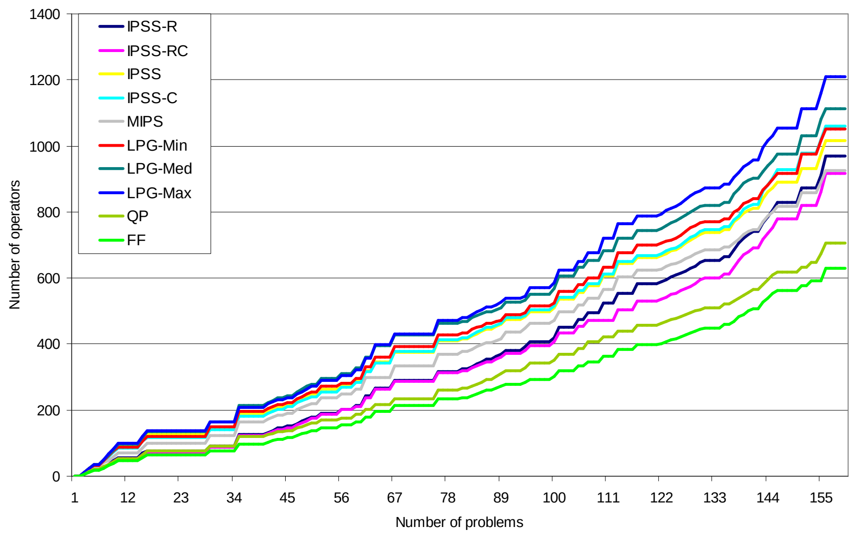


Figure 8.41: Number of operators FF, QP, LPG, MIPS, IPSS and IPSS-R in BT.

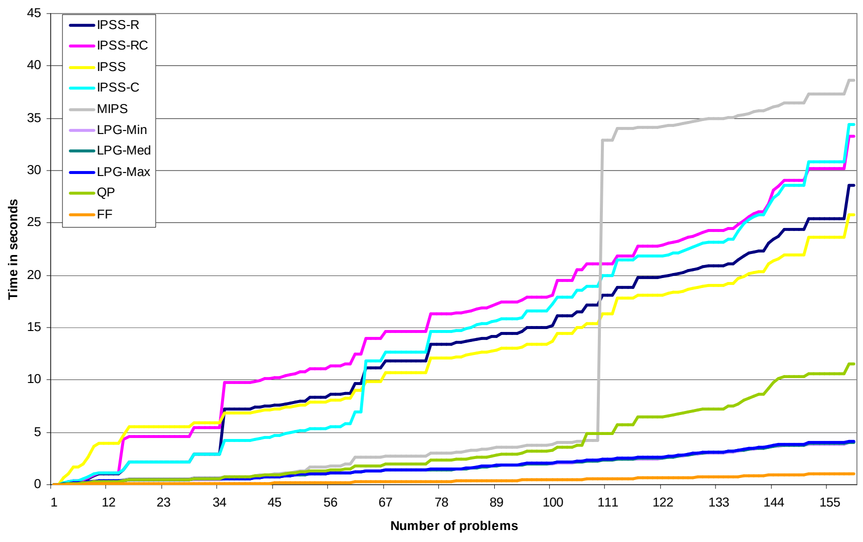


Figure 8.42: Time for FF, QP, LPG, MIPS, IPSS and IPSS-R in BT.

Each operator has a constant duration and we have also coded two versions: the one that considers that the workers are *occupied* and *not-occupied* to use it for planners that reason with activities in parallel (as mentioned before, we have called it the BT_OCCU domain) and the one that does not consider them and let the separation between logical predicates and resources.

We have randomly generated 160 problems for this version distributed in 4 sets with the same features as in the BT domain. Table 8.15 shows the planners chosen for this domain with the number of problems solved in each set and the percentage that it represents. Note that we have used the same planners as in the ROBOBUSY SimpleTime for the same reasons as mentioned on section 8.2.1.1. The first four planners were run using the BT Simple domain, because IPSS-R reasons in parallel but include resource reasoning that allows to add causal constraints between operators that consumes the same resource and QP because it is a TO planner, they are not interactions between activities competing for the same resources. The other four planners obtain parallel plans so they must use the BT_OCCU Simple version in order to get feasible solutions.

Table 8.15: Number of problems solved by each planner in the BT Simple domain.

Name	G2	G5	G8	G11	Problems solved	Percentage
IPSS-R	33	28	25	33	119	74%
IPSS-RC	30	23	25	30	108	68%
QP	33	31	28	34	126	79%
QP-C	31	24	25	31	111	69%
IPSS-C	31	23	25	31	110	69%
IPSS	33	29	27	32	121	76%
MIPS	26	21	17	24	88	55%
LPG	40	40	40	40	160	100%

Figure 8.43 shows the results when the makespan is compared. IPSS-R with and without control rules gets close solutions to the best ones that LPG can find in five runs. MIPS obtains results comparable with LPG-Max and, as expected, QP obtains the worse makespan as the solution is a sequence of operators.

If we run LPG, IPSS and IPSS-R during 180 seconds looking for better solutions, we can see the results in Figure 8.44: IPSS-R-Q finds almost as good solutions as the five LPG-Q runs. IPSS-Q had the worst results as is QPRODIGY who is deciding the operators to apply.

With respect to the number of operators, IPSS-R finds the solutions with less number of operators, because as we have mentioned in the last section in the BT domain the *free-worker* operator is not used (see Appendix E), so obviously that number is lower. For that reason we do not include the graphic with the number of operators for each planner.

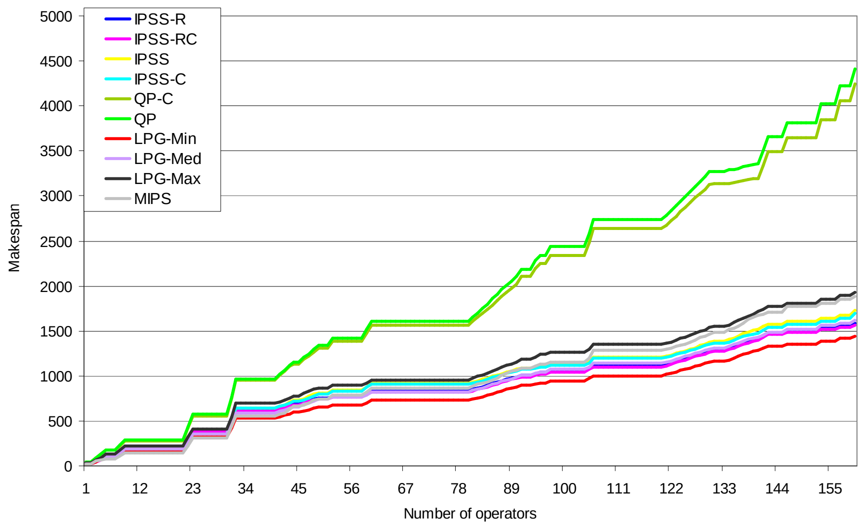


Figure 8.43: Makespan for for MIPS, LPG, IPSS and IPSS-R for BT Simple.

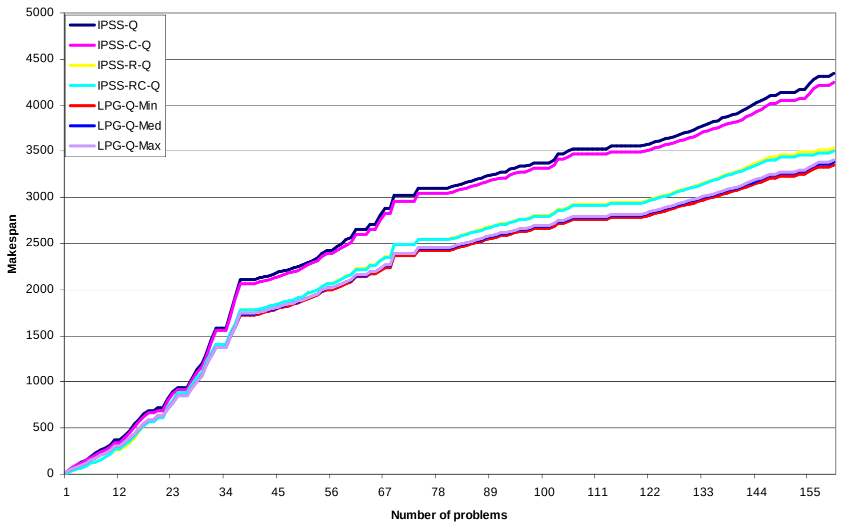


Figure 8.44: Makespan for LPG-Q, IPSS-Q and IPSS-R-Q for BT Simple.

Figure 8.45 shows the execution time. QP and LPG are the most efficient planners. In this set of problems QPRODIGY is able to find solutions faster than IPSS although the quality of the solution found are worse. In IPSS-R the three layers decrease its efficiency respect to QP and LPG. Then, MIPS and the rest of the IPSS configurations.

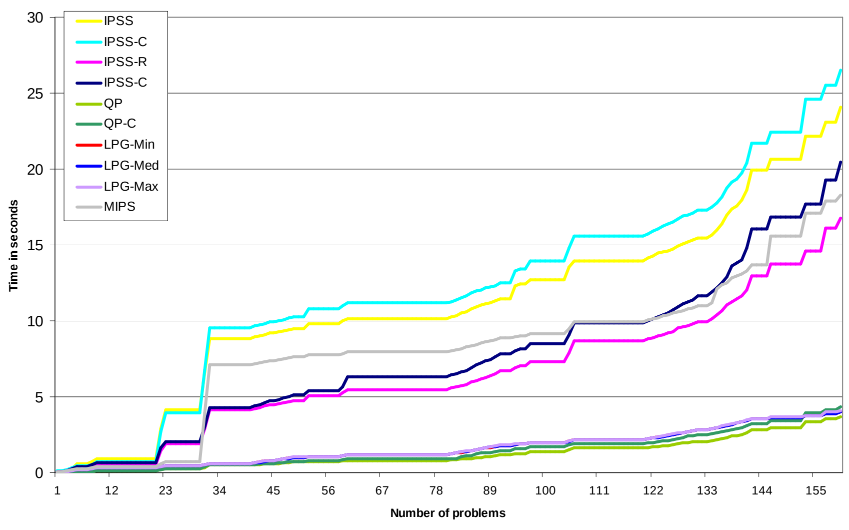


Figure 8.45: Time in seconds for MIPS, LPG, IPSS and IPSS-R for BT simple.

8.4 Summary

In this chapter we have presented the results of the different IPSS configurations for three domains. One of the domains belongs to the IPC-2002 [87], the ZENOTRAVEL domain in its Time version. The other two domains are the ROBOCARE domain [33], a multi-agent system which generates user services for human assistance and the BT domain, a reduced version of how to install a new telephone line at BRITISH TELECOM, explained in section 4.4.2. These two domains have some features that make them suitable for IPSS, i.e., distinction between logical predicates and resources. We can do this distinction because the resources are independent of the goals and the actions to be performed: any action can be executed by any resource. And these are precisely the types of domains (symetric ones) that are usually not appropiate for other planners such as GRAPHPLAN based.

In the previous chapter, we have described the IPSS architecture that consisted on three layers. If we can abstract the resources from the domain and impose a time horizon, we will be able to use the *Ground*-CSP and the *Meta*-CSP layers and then, modify the basic search tree of QPRODIGY; otherwise we will be using QPRODIGY plus a deordering algorithm (basically we have obtained the same results as using QPRODIGY plus the deordering algorithm of Veloso [181]).

In the ROBOCARE and BT domains we can make this separation, but in the ZENO-TRAVEL domain we cannot use the *Meta*-CSP as it does not allow the separation of airplanes (considered as binary resources) from the rest of the logical predicates. For example, actions as *debark* and *board* persons in the same airplane can all be performed in one time step, but if airplanes are binary resources the *Meta*-CSP will impose a precedence link between activities increasing the makespan unnecessarily. This problem will be solved with the multicapacity version of IPSS.

Also, the ROBOCARE and BT domain present some features that make them good domains to exploit parallelism. Planners as BB or LPG that build a graph plan can exploit these features and obtain good solutions. Although IPSS is based on a TO planner that performs mean-end analysis, we have modified it to behave in some how as a GRAPHPLAN based planner. The possibility of imposing an horizon constraint on a problem and maximising the resources used in each step allows to use the set of propagation rules (dominance conditions) as GRAPHPLAN based planners do and also the possibility of exploiting an issue strictly correlated to the integration of planning and scheduling.

If we want to use the *Ground*-CSP layer, for all the domains we can do it in two ways. In the first one, giving a time bound and asking for more than one solution (we have added the extension -Q to IPSS when running in this mode). After the first solution is found, the makespan bound will be set to the value obtained in the first solution and so on. The second way is imposing for the first run a maximum value. This scheme has the disadvantage of giving a value too low for the makespan bound and then not finding a solution.

We have compared the IPSS configurations in terms of makespan, number of operators, time in seconds and number of solved problems with different planners of the IPC-2002 competition.

Part IV

Conclusions and Future Work

Chapter 9

Conclusions and Future Work

In this last chapter the general conclusions of this work will be presented. Although at the end of each chapter some specific conclusions have been described, here we summarise the most important ones. First we summarise the conclusions, then we outline the main directions for Future work and finally we enumerate the most important publications of this PhD. thesis.

9.1 Conclusions

In this dissertation, we have presented our work on developing a framework for fully integrating Workflow and AI planning, represent and solve a real satellite domain (that is, schedule the nominal operations of the Spanish satellite company HISPASAT) and a system that integrates planning and scheduling.

We have described the advantages of using AI planning techniques for modeling processes in SHAMASH, a Knowledge Based System that uses an object oriented and rule-based approach and the potential of applying AI planning techniques within a legacy workflow management system, COSMOSS.

The BPR rules and objects in SHAMASH are translated into AI planning types and operators to produce a plan that correspond to a process instance (tasks dependency). Each plan will vary depending on the resources available, elements that flow through the process and the different tasks that the organisations can introduce to adapt their processes to the market changes. With this approach the models generated are automatically validated avoiding inconsistencies in linking activities and saving time to the user.

We also wanted to study the issues that AI planners can gain with this approach. Generally, to specify the domain theory, a deep understanding of the way AI planners work and its terminology is needed. However, if we use a tool like SHAMASH or COSMOSS, the description language is closer to the user and allows an automatic verification of the syntax through a user friendly interface. The integration described in this work has be done as general that it could have also used the language for the AIPS-2002 Competition Committee (PDDL2.1). Then any planner that supports PDDL2.1 could exploit the advantages of using a descriptive language as the one used in SHAMASH.

Also, satellites provide a challenging domain for applying AI techniques. Due to the use of time and resources, scheduling techniques have been traditionally applied. We have presented how the deliberative planner PRODIGY can solve scheduling problems thanks to its capability of defining numerical variables, coding functions and the use of control rules. The functions allow the user to obtain variables values, calculate the duration, specify resources constraints and relationships among operators. The control rules help to reduce the search and then the time spent obtaining the solution. The planning techniques provide a successful solution to the ground scheduling operations during the year for three satellites in Orbit of the Spanish satellite company HISPASAT.

We have developed a graphical tool to visualise the results obtained by the planner. Before the software development, the engineer group generated by hand and in paper the operations that have to be done along the week and year. There were many incongruencies between the two types of representation used (weekly and annual) and the document modification was a tedious task. So the tool saves a lot of time to the user avoiding these type of errors and guiding him/her through all the process.

But in real domains, as the mentioned above, although PRODIGY is able to generate solutions to problems, some work on domain dependent knowledge is needed in order to solve them when time and resources are considered. For this reason, we have explored some steps to integrate planning and scheduling: from a standalone planning system, to a planning and scheduling pipeline integration, towards the more integrated approach of interleaving planning and scheduling: the IPSS system.

In the case of using the pipeline integration, the temporal and resource information has been eliminated from the planner domain, and the output generated by the planner is used as an input for a scheduler. To influence the planner choices, the scheduler should provide some information to the planner. Along this line, in the last approach, IPSS interleaves the refinement solution inside the search tree. In particular we are using QPRODIGY in the aim of taking advantage of the quality metrics that this version is able to use. The idea is to have two systems executing in parallel; a planner and a time-resource reasoner. Information is continuously exchanged between these two representations and in particular is used to influence the QPRODIGY search to prune choices that lead to either temporal or resource inconsistencies. The main features that difference IPSS from other approaches that integrate planning and scheduling are:

- Good balance between efficiency and quality.
- Flexibility in the information that the planner and the scheduler can handle.
- Easy to add control knowledge, metrics and learning to the system.
- The modularity so any of its components can be changed by modifying the interface to introduce the knowledge that each part needs.

But IPSS has its limitations when we use it in planning domains where it cannot be imposed any time horizon or the resources cannot be separated from the logical predicates. In those cases, QPRODIGY guides the search and the three layers added only worsen the system efficiency.

The experiments show improvements in domains with the features described above when comparing with some state of the art planners and also with respect to the pipeline integration. It also avoids some problems that it can appear when using the pipeline integration.

9.2 Future Work

Among the future work that can be derived from this Ph.D. thesis we can mention:

- Knowledge engineering aspects. From the experience gained in the HISPASAT project and studying the weak points of the integration, we are developing a graphical domain construction and validation tool for PDDL2.1 language as it has being done in the GIPO environment [111]. The language has been extended having in mind resources definition, temporal constraints between activities and the possibility to minimise/maximise the makespan or resources usage. Although the application has an easy-to-use interface, we want to integrate it with the CONSAT application and explore the possibility of adding more satellites allowing to define/delete new operations by the HISPASAT engineer group. This would allow us to study the scalability of the approach of dealing within the planner with temporal and resource reasoning.

- We also want to use SHAMASH that handles time and resources constraints with respect to the IPSS to analyse its behaviour related to the others systems.

- Learning techniques could also be applied, allowing e.g. optimisation of processes over time. As a next step we want to use HAMLET [21] or EVOCK [6] to learn control rules for the correct decisions made by IPSS when solving problem. This has the implicit assumption that the information that is crucial (relevant) for making that decision is available in the state.

- It would be also interesting to integrate a monitoring module to execute the scheduled operations obtained by the planner and to re-plan in case of a failure. Particular areas worthy of attention that can take advantage of AI techniques are: monitoring for (and anticipation of) deviations from plans, and re-planning in the event of such exceptions.

- Augment IPSS to handle multi-capacity resources. The algorithm to handle multi-capacity resources that we have implemented is [30]. We want to compare the results in the satellite and the zenotravel domains of the IPC'02 competition given that those domains required of the multicapacity approach in order to get improvements of the logical and resource separation.

- It will be very interesting to use CSP at the binding level, that is, in the point three of the decision points in QPRODIGY (see section 3.1). This extension can help to discard some bindings when instantiating the chosen operator.

9.3 Publications within this PhD. thesis

The results of this monograph has produced several publications in the field of AI Planning and Scheduling as well as in BPR. Although these publications appear refereed along the dissertation ([142]- [151]) and can be looked up in the bibliography at the end of this book, we can mention among others the following:

- MD. R-Moreno and P. Kearney. Integrating AI Planning with Workflow Management System. *International Journal of Knowledge-Based Systems*, 15, pp. 285-291, 2002. Elsevier.
- MD R-Moreno, P. Kearney and D. Meziat. A Case Study: Using Workflow and AI Planners. *The 19th Workshop of the UK Planning and Scheduling.* PLANSIG2000.
- MD. R-Moreno, D. Borrajo and D. Meziat. Applying an AI Planner to Schedule Ground Satellites Operations. *Working notes of the IJCAI'01 Workshop on Planning with Resources*, pp. 66-67. IJCAI Press, 2001.
- MD. R-Moreno, D. Borrajo and D. Meziat. An Artificial Intelligence Planner for Satellites Operations. *ESA Workshop on On-Board Autonomy*, 2001.
- MD. R-Moreno and P. Kearney. Let's see What Happen if we Integrate AI Planning with Workflow Management System. *21st SGES International Conference on Knowledge Based Systems and Applied Artificial Intelligence*, 2001.
- MD. R-Moreno, M. Prieto, D. Meziat, J. Medina and C. Martín. Controlling and Testing a Space Instrument by an AI planner. Proceedings of the International Conference on Enterprise Information Systems, *ICEIS-2002*, 2002.
- MD. R-Moreno, A. Oddi, D. Borrajo, A. Cesta and D. Meziat. Integrating Hybrid Reasoners for Planning and Scheduling. *The 21th Workshop of the UK Planning and Scheduling.* PLANSIG2002.

Part V

Appendices

Appendix A

The translation from SHAMASH to PRODIGY

In this appendix we will describe all the elements in common between the SHAMASH and PRODIGY syntax. In SHAMASH, elements, resources, units, activities and processes are MetaClasses which derive classes, and from these classes the instances. Tables A.1, A.2 and A.3 show the mapping between both syntax.

A.1 SHAMASH Syntax

The SHAMASH syntax is defined as follows and its semantic is shown in Tables A.4 and A.5.

```
SHAMASH Rule:
   Rule name [with properties [prop value]+]?
     If [[Condition and|not Condition and | Condition or |
          not Condition or]* Condition]?
     Then [Action]+

Condition:
   Exists a class of Meta Class TypeClass [ named id-var-class ]?   ( 1)
      [with  att oprel [id-var-1 | (Condition)] |                   ( 2)
       assigning to id-var1 its att |                               ( 3)
       with id-var in att |                                         ( 4)
       with id-var not in att ]* |                                  ( 5)
   CExpression|
   Assign var the value Expression                                  ( 6)

Action:
   Modify object id-var-class
      [with  att is Expression |                                    ( 7)
       by adding Expression to att |                                ( 8)
       by removing Expression from att]+ |                          ( 9)
   Create a class of type TypeClass [named id-var-class1 ]?
      [with [att is value]+ ]?                                      (10)
   Delete id-var                                                    (11)
   Send message  message to id-var [([Expression,]* Expression)]?
   Call funtion func [([Expression,]* Expression)]?

TypeClass:
   Activity
   Resource
   Role
   Element
   Unit

CExpression:
   Expression =  Expression
   Expression <  Expression
   Expression <= Expression
   Expression >= Expression
   Expression >  Expression
   Expression <> Expression

Expression:
   Expression = Expression
   Expression < Expression
   Expression <= Expression
   Expression >= Expression
   Expression > Expression
   Expression <> Expression
   Add att of all class
   Max att of all class
   Min att of all class
   Expression + Expression |
   Expression - Expression |
   Expression / Expression |
   Expression * Expression |
   (Expression) |
   id-var |
   Call funtion func [([ Expression,]* Expression)]? |
   Send message  message to id-var [([Expression,]* Expression)]?
```

Table A.1: Mapping between SHAMASH and PRODIGY Domain Terminology.

SHAMASH **Elements**	PRODIGY **Domain Elements**
PROCESS Proc-Name	 (create-problem-space 'Proc-Name :current t)
ELEMENT Elem-Name	(ptype-of element :top-type) (ptype-of Elem-Name element)
ROLE UnitRole-N HumanRole-N	(ptype-of role :top-type) (ptype-of UNIT-ROLE role) (ptype-of UnitRole-N UNIT-ROLE) (ptype-of HUMAN-ROLE role) (ptype-of HumanRole-N HUMAN-ROLE)
RESOURCE ResourceHuman-N ResourceMaterialAvailable-N ResourceMaterialConsumable-N	(ptype-of resource :top-type) (ptype-of HUMAN-RES resource) (ptype-of MATERIAL-RES resource) (ptype-of ResourceHuman-N HUMAN-RES) (ptype-of AVAILABLE-RES MATERIAL-RES) (ptype-of ResourceMaterialAvailable-N AVAILABLE-RES) (ptype-of CONSUMABLE-RES MATERIAL-RES) (ptype-of ResourceMaterialConsumable-N CONSUMABLE-RES)
UNIT Unit-Name	(ptype-of Unit :top-type) (ptype-of Unit-Name Unit)
ACTIVITY Activ-Name ActivityInputs Preconditions Rule Postconditions Rule	 (Operator Activ-Name (params <ActivityInput1> <ActivityInput2> ...) (preconds (.....)) (effects (add/del....))

Table A.2: Mapping between SHAMASH and PRODIGY Types.

SHAMASH Types	PRODIGY Types
string	(ptype-of string: top-type) (pinstance-of string-Name string)
UINT	(infinite-type UINT #'integerp)
double	(infinite-type double #'floatp)
BOOL	(ptype-of BOOL: top-type) (pinstance-of TRUE FALSE BOOL)
sreference (reference to)	It is a reference to a SHAMASH element (unit, element, role, resource, process, standard, activity) we do not need to create a new type.
stringlist (List of References to)	We treat this type as a reference type, but we create a predicate for each value of the list.

Table A.3: Mapping between SHAMASH and PRODIGY Problem Terminology.

SHAMASH **Elements**	PRODIGY **Initial states/Goals**
ELEMENT Elem-Name	 (Attribute-Name Elem-Name value)
ROLE Role-Name	 (Attribute-Name Role-Name value)
RESOURCE (Human/Material) Resource-Name	 (Attribute-Name Resource-Name value)
UNIT Unit-Name	 (Attribute-Name Unit-Name value)

Table A.4: SHAMASH Semantic I.

Syntaxis Element	**Terminology meaning**
Name ::= Rule item Properties	The name of a rule is composed of the word Rule, a name, and a set of properties.
Properties ::= with properties Properties′	The rule could have no properties.
Properties′ ::= item item Properties′ | item item	The properties of a rule can be predefined, as ruleset, and priority, or defined by the user. Usually, Shamash will force the user to define a rule in a given ruleset, so that it knows when to use it.
Condition ::= Condition′ and Condition | Condition′ or Condition | Condition′ | not Condition′ and Condition | not Condition′ or Condition | not Condition′	A rule can have one or more conditions. These conditions can be joined using ands or nots.
Condition′ ::= Exists a item of type TypeClass Attributes | Exists a item of type TypeClass named id-var Attributes | CExpression | Call to item Parameters returns true | Assign var item the value Expression | Send message Message to id-var Parameters returns true	An elementary condition can be either a condition on the attributes of an instance of a given class, calls to functions, expressions, or assignment of value to a given variable.
CExpression ::= Expression RelOperation Expression	A logical expression is a logical operation between two expressions.
Actions ::= Actions′ | Actions′ Actions	There are one o more actions in the 'Then' side of the rulw.
Actions′ ::= Modify object id-var changes | Create a item of type TypeClass Attributes′ | Create instance of item of type TypeClass named id-var Attributes′ | Delete id-var | Send message Message to id-var Parameters | Call function item Parameters	An action can be either to create, modify, or delete an instance of a given class, send a message, or call any function.

Table A.5: SHAMASH Semantic II.

Attributes ::= with Attributes″ Attributes | assigning to id-var its item |	It allows to refer to attributes of an instance by preceding them with keyword with.
Attributes′ ::= with Attribute″″ Attributes′ | Attribute″″	The same as above, but to be used in the right hand side of the rules.
Attributes″ ::= id-var in item | id-var not in item | item RelOperation (Condition)	After the with, one has to write the name of the attribute. One can compare the attribute's value with another value, or with another condition. Also, one can ask if the value is or not in a list.
Attributes‴ ::= item is Expression |	One can assign a value to an attribute.
Changes ::= Changes′ | Changes′ Changes	One can modify one or more attributes of an object.
Changes ::= with item is Expression | by adding Expression to item | by deleting Expression from item	An object can be modified by changing the value of an attribute, or by adding/removing a value to/from an attribute.
Parameters ::= (Expression Expression′) |	There can be none, one or more parameters in a call to a function.
Expression ::= (Expression) | Expression RelOperation Expression | Expression NumOperation Expression | Add item of all item | Max item of all item | Min item of all item id-var | Call funtion func [([Expression ,]* Expression)]? | Send message message to id-var [([Expression ,]* Expression)]?	An expression can be an operation (logical or arithmetic), or the computation of the addition, maximum, or minimum of a set of attributes. Also can be a call to a function or a message to other class.
Expression′ ::= ,Expression Expression′ |	There can be one or more expressions joined by a comma.
NumOperation ::= + | - | * | /|	Differents operations on expressions.
RelOperation ::= <> | < | <= | = | > | >= |	Differents operations on expressions.

A.2 PRODIGY Syntax

The PRODIGY syntax is defined as follows and its semantic is shown in Table A.6.

```
PRODIGY Operator_Name
  (OPERATOR Name_Act
    (params params-list)
    (preconds var-spec* pdl-expression)
    (effects var-spec* effects))

params-list
    <ActivityInputs>+
pdl-expression
    (AND pdl-expression*)|(OR pdl-expression*)|(˜ pdl-expression*)

var-spec
    (<ActivityInputs>  Name_Elem)                                     ( 1)
    (<id_var> class)                                                  ( 1)
    (att <id_var> value)                                              ( 2)
    (att <id_var> <id_var1>))                                         ( 3)
    (att <id_var> <value>)                                            ( 4)
    (˜ att <id_var> <value>)                                          ( 5)
    ˆ                                                                 ( 6)

effects
    (ADD predicate)|(DEL predicate)|conditional-effect
    predicate          = literal
    literal            = (att  <id_var> <id-var-class>)
    conditional-effect = (IF pdl-expression effects)

    (add(att <id_var> <id-var-class>))                                ( 7)
    (add(att <id_var> <id-var-class>))                                ( 8)
    (del(att <id_var> <id-var-class>))                                ( 9)
    (del(att <id_var> <id-var-class>))                                (10)

ˆ In case the Expression were of boolean or arithmetic type, a function
  must be defined:
     Called_function (params | var | */- + | TRUE/FALSE)
```

Table A.6: PRODIGY Semantic.

Syntaxis Element	**Terminology meaning**
params-list ::= ActivityInputs	The parameters list is composed by all the activity inputs defined in the activity.
preconds ::= var-spec* | pdl-expression	In the operator preconditions, the type of the activity inputs and the literal that must be true before applying the operator are defined.
var-spec ::= id-var	The variable type must be declared as one of the SHAMASH types or elements.
Expression ::= value | Expression	In case the Expression is boolean or arithmetic, a CLisp function must be defined: Called-function (params var | */- + | TRUE/FALSE)
pdl-expression ::= predicate	It is a predicate composed of three literals: attribute, variable identifier and an Expression or a value.
effects ::= var-spec* | effects	In the operator effects can be declared new variables of one of the SHAMASH types but number values. Effects are composed of the tuple: attribute variable-identifier and a value or an Expression.

A.3 An Example

Let us clarify all the concepts explained here by an example.

```
Rule Change-Resource with properties ruleset optimisation-rules
  If Exists a signature of Meta Class activity named  acty
       with resource = (Exists ballpen of Meta Class resource named r1
                          assigning to c1 its cost
                          assigning to activities-r1 its resource-used-by)
     and
     Exists a ballpen of Meta Class resource named var r2
       assigning to c2 its cost
       assigning to activities-r2 its resource-used-by
     and
     c2 < c1
     Exists a Cconsulta of Meta Class element named e1
       with procesamiento = (’’Manual’’)
       with Entrada = (FALSE)
       with Encontrado = (FALSE)
  Then
     Modify object acty with resource is  r2
     Modify object  r1 by removing acty from activities-r1
     Modify object r2 by adding acty to activities-r2
     Modify object c1 with Encontrado is  (TRUE)
```

This SHAMASH RULE would be translated in the following PRODIGY operator:

```
(OPERATOR Change-Resource
 (params <acty>)
  (preconds
  ((<r1> BALLPEN)
   (<r2> BALLPEN)
   (<e1> CcONSULTA)
   (<acty> SIGNATURE)
   (<v1> resource)
   (<v2> resource)
   (<activities-r1> activity)
   (<activities-r2> activity)

   ;; Exists ballpen of type resource named r1  assigning to c1 its cost
   (<c1> (and UINT(gen-from-pred (cost <r1> <c1>))))

   ;; Exists ballpen of type resource named r2 assigning to c2 its cost
   ;; c2 < c1
   (<c2> (and UINT(gen-from-pred (cost <r2> <c2>))
                (> <c2> <c1>))))
   (and (procesamiento <e1> Manual)
        (Entrada <e1> FALSE)
        (Encontrado <e1> FALSE)

        ;;assigning to activities-r1 its resource-used-by
        (resource-used-by <r1> <activities-r1>)
```

```
      ;; assigning to activities-r2 its resource-used-by
      (resource-used-by <r2> <activities-r2>)
      (resource <acty> <r1>)))
(effects
 ()
 ((del(Encontrado <e1> FALSE))
  (add(Encontrado <e1> TRUE))

   ;; Modify object acty with resource is r2
  (add(resource <acty> <r2>))
  (del(resource <acty> <r1>))

  ;; Modify object r1 by deleting acty from activities-r1
  (del(resource-used-by <r1> <acty>))

  ;; Modify object r2 by adding acty to activities-r2
  (add(resource-used-by <r2> <acty>)))))
```

Appendix B

O-OSCAR Input File

```
n rho   ny delta

0  1    s(0) s(0,1) ... s(0,s(0)) [0] ... [0]

1  m(1) s(1) s(1,1) ... s(1,s(1)) [t(1,s(1,1),1) ... t(1,s(1,1),m(1))]
... [t(1,s(1,s(1)),1) ... t(1,s(1,s(1)),m(1))]

n m(n) s(n) s(n,1) ... s(n,s(n)) [t(n,s(n,1),1) ... t(n,s(n,1),m(n))]
... [t(n,s(1,s(n)),1) ... t(n,s(1,s(n)),m(n))]

n+1  1  0

0 1 d(0,1) rr(1,0,1)  ... rr(rho,0,1)  rn(1,0,1)  ... rn(ny,0,1)  rdr(1,0,1) rdn(1,0,1)
... rdr(delta,0,1) rdn(delta,0,1)

1 1 d(1,1) rr(1,1,1)  ... rr(rho,1,1)  rn(1,1,1)  ... rn(ny,1,1)  rdr(1,1,1) rdn(1,1,1)
... rdr(delta,1,1) rdn(delta,1,1)

m(1) d(1,m(1)) rr(1,1,m(1)) ... rr(rho,1,m(1))  rn(1,1,m(1)) ... rn(ny,1,m(1)) rdr(1,1,m(1))
rdn(1,1,m(1)) ... rdr(delta,1,m(1)) rdn(delta,1,m(1))

n+1 1 0 0 ... 0 0 ... 0 0 0 ... 0 0

Rr(1) ... Rr(rho) Rn(1) ... Rn(ny) Rdr(1) Rdn(1) ... Rdr(delta) Rdn(delta)


Using the following notation:

n:             number of activities (without supersource and supersink)
0:             supersource
n+1:           supersink
rho:           number of renewable resources
ny:            number of nonrenewable resources
delta:         number of doubly-constrained resources
m(i):          number of execution modes of activity i
s(i):          number of direct successors of activity i
s(i,j):        the j-th direct successor of activity i
t(i,s(i,j),m): minimal/negative maximal time lag between the starts of activities i and
               s(i,j) if activity i is performed in mode m
d(i,m):        duration of activity i if performed in mode m
rr(k,i,m):     resource requirement of renewable resource k for the processing of
               activity i in  mode m
rn(k,i,m):     resource requirement of nonrenewable resource k for the processing of
               activity i  in mode m
```

```
rdr(k,i,m):    renewable resource requirement of doubly-constrained resource k
               for the processing of activity i in mode m
rdn(k,i,m):    nonrenewable resource requirement of doubly-constrained resource k
               for the processing of activity i in mode m
Rr(k):         availability of renewable resource k
Rn(k):         availability of nonrenewable resource k
Rdr(k):        renewable availability of doubly-constrained resource k
Rdn(k):        nonrenewable availability of doubly-constrained resource k
```

Appendix C

Hispasat Domain

The HISPASAT domain and the problems corresponding to each satellite can be found in the following Web Page:
http://atc1.aut.uah.es/∼mdolores/hispasat/

Appendix D

Robocare

In this appendix, we present the different Robocare domain versions used in the dissertation. The problems that we have generated for this domain can be downloaded at:

http://atc1.aut.uah.es/∼mdolores/robocare/

D.1 Robocare Domain

```
(define (domain robocare)
(:requirements :strips :typing :equality)
(:types agent object person room door bed)

(:predicates
 (unmade ?b - bed)
 (clear ?b - bed)
 (is_in_person_room ?p - person ?r - room)
 (is_in_bed_room ?b - bed ?r - room)
 (is_in_agent_room ?a - agent ?r - room)
 (is_door ?r1 - room ?r2 - room)
 (needs_to_walk ?p - person)
 (has_meal ?a - agent ?p - person)
 (needs_meal ?p - person)
 (is_kitchen ?k - room)
 (made ?b - bed)
 (not-clear ?b - bed))

(:action goto-room
 :parameters (?a - agent ?room_from - room ?room_to - room)
 :precondition (and (not (= ?room_from ?room_to))
                    (is_door ?room_from ?room_to)
                    (is_in_agent_room ?a ?room_from))
 :effect (and (is_in_agent_room ?a ?room_to)
              (not (is_in_agent_room ?a ?room_from))))

(:action walk-person
 :parameters (?a - agent ?p - person ?room_from - room ?room_to - room)
 :precondition (and (needs_to_walk ?p)
                    (is_door ?room_from ?room_to)
                    (is_in_agent_room ?a ?room_from)
                    (is_in_person_room ?p ?room_from))
 :effect (and (is_in_person_room ?p ?room_to)
              (is_in_agent_room ?a ?room_to)
              (not (is_in_agent_room ?a ?room_from))
              (not (is_in_person_room ?p ?room_from))))

(:action make-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :precondition (and (unmade ?b) (clear ?b)
                    (is_in_bed_room ?b ?r)
                    (is_in_agent_room ?a ?r))
 :effect (and (not (unmade ?b)) (made ?b)))

(:action clear-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :precondition (and (is_in_agent_room ?a ?r)
                    (is_in_bed_room ?b ?r)
                    (not-clear ?b))
 :effect (and (clear ?b) (not (not-clear ?b))))

(:action get-meal
 :parameters (?a - agent ?p - person ?r - room)
 :precondition (and (is_kitchen ?r) (is_in_agent_room ?a ?r))
 :effect (has_meal ?a ?p))

(:action serve-meal
 :parameters (?a - agent ?p - person ?room_person - room)
 :precondition (and (is_in_agent_room ?a ?room_person)
                    (is_in_person_room ?p ?room_person)
                    (has_meal ?a ?p)(needs_meal ?p))
 :effect (and (not (needs_meal ?p)) (not (has_meal ?a ?p)))))
```

D.2 Robobusy Domain

```
(define (domain robobusy)
(:requirements :strips :typing :equality)
(:types agent object person room door bed)

(:predicates
 (unmade ?b - bed)
 (clear ?b - bed)
 (is_in_person_room ?p - person ?r - room)
 (is_in_bed_room ?b - bed ?r - room)
 (is_in_agent_room ?a - agent ?r - room)
 (is_door ?r1 - room ?r2 - room)
 (needs_to_walk ?p - person)
 (has_meal ?a - agent ?p - person)
 (needs_meal ?p - person)
 (is_kitchen ?k - room)
 (made ?b - bed)
 (not-clear ?b - bed)
 (busy ?a - agent)
 (not-busy ?a - agent))

(:action goto-room
 :parameters (?a - agent ?room_from - room ?room_to - room)
 :precondition (and (not (= ?room_from ?room_to))
                    (not-busy ?a) (is_door ?room_from ?room_to)
                    (is_in_agent_room ?a ?room_from))
 :effect (and (is_in_agent_room ?a ?room_to) (not (not-busy ?a))
              (busy ?a) (not (is_in_agent_room ?a ?room_from))))

(:action walk-person
 :parameters (?a - agent ?p - person ?room_from - room ?room_to - room)
 :precondition (and (needs_to_walk ?p) (not-busy ?a)
                    (is_door ?room_from ?room_to)
                    (is_in_agent_room ?a ?room_from)
                    (is_in_person_room ?p ?room_from))
 :effect (and (is_in_person_room ?p ?room_to) (not (not-busy ?a))
              (busy ?a) (is_in_agent_room ?a ?room_to)
              (not (is_in_agent_room ?a ?room_from))
              (not (is_in_person_room ?p ?room_from))))

(:action make-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :precondition (and (unmade ?b) (clear ?b) (is_in_bed_room ?b ?r)
                    (is_in_agent_room ?a ?r)(not-busy ?a))
 :effect (and (not (unmade ?b))  (not (not-busy ?a))
              (busy ?a) (made ?b)))

(:action clear-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :precondition (and (is_in_agent_room ?a ?r) (is_in_bed_room ?b ?r)
                    (not-clear ?b)(not-busy ?a))
 :effect (and (clear ?b) (not (not-busy ?a))
              (busy ?a) (not (not-clear ?b))))

(:action get-meal
 :parameters (?a - agent ?p - person ?r - room)
 :precondition (and (is_kitchen ?r) (not-busy ?a)
                    (busy ?a) (is_in_agent_room ?a ?r))
 :effect (has_meal ?a ?p))

(:action serve-meal
 :parameters (?a - agent ?p - person ?room_person - room)
 :precondition (and (is_in_agent_room ?a ?room_person)
                    (is_in_person_room ?p ?room_person)
                    (has_meal ?a ?p) (needs_meal ?p)(not-busy ?a))
 :effect (and (not (needs_meal ?p)) (not (not-busy ?a))
              (busy ?a) (not (has_meal ?a ?p))))

(:action free-agent
 :parameters (?a - agent)
 :precondition  (busy ?a)
 :effect (and (not (busy ?a)) (not-busy ?a))))
```

D.3 Robobusy SimpleTime Domain

```
 (define (domain robobusySimpleTime)
 (:requirements :durative-actions :typing :equality)
 (:types agent
         object
         person
         room
         door
         bed)

 (:predicates
  (unmade ?b - bed)
  (clear ?b - bed)
  (is_in_person_room ?p - person ?r - room)
  (is_in_bed_room ?b - bed ?r - room)
  (is_in_agent_room ?a - agent ?r - room)
  (is_door ?r1 - room ?r2 - room)
  (needs_to_walk ?p - person)
  (has_meal ?a - agent ?p - person)
  (needs_meal ?p - person)
  (is_kitchen ?k - room)
  (made ?b - bed)
  (not-clear ?b - bed)
  (busy ?a - agent)
  (not-busy ?a - agent))

 (:durative-action goto-room
  :parameters (?a - agent ?room_from - room ?room_to - room)
  :duration (= ?duration 3)
  :condition (and (over all (not (= ?room_from ?room_to)))
                  (over all (is_door ?room_from ?room_to))
                  (over all (is_in_agent_room ?a ?room_from))
                  (over all (not-busy ?a)))
  :effect (and (at end (is_in_agent_room ?a ?room_to))
               (at start (busy ?a))
               (at start (not (not-busy ?a)))
               (at start (not (is_in_agent_room ?a ?room_from)))))

 (:durative-action walk-person
  :parameters (?a - agent ?p - person ?room_from - room ?room_to - room)
  :duration (= ?duration 10)
  :condition (and (over all (needs_to_walk ?p))
                  (over all (is_door ?room_from ?room_to))
                  (over all (is_in_agent_room ?a ?room_from))
                  (over all (is_in_person_room ?p ?room_from))
                  (over all (not-busy ?a)))
  :effect (and (at end (is_in_person_room ?p ?room_to))
               (at end (is_in_agent_room ?a ?room_to))
               (at start (not (is_in_agent_room ?a ?room_from)))
               (at start (not (is_in_person_room ?p ?room_from)))
               (at start (busy ?a))
               (at start (not (not-busy ?a)))
               (at start (not (is_in_agent_room ?a ?room_from)))))

 (:durative-action make-bed
  :parameters (?a - agent ?b - bed ?r - room)
  :duration (= ?duration 15)
  :condition (and (over all (unmade ?b))
                  (over all (clear ?b))
                  (over all (is_in_bed_room ?b ?r))
                  (over all (is_in_agent_room ?a ?r))
                  (over all (not-busy ?a)))
  :effect (and (at start (not (unmade ?b)))
               (at end (made ?b))
               (at start (not (not-busy ?a)))
               (at start (busy ?a))))

 (:durative-action clear-bed
  :parameters (?a - agent ?b - bed ?r - room)
  :duration (= ?duration 5)
  :condition (and (over all (is_in_agent_room ?a ?r))
                  (over all (is_in_bed_room ?b ?r))
                  (over all (not-busy ?a))
                  (over all (not-clear ?b)))
  :effect (and (at end (clear ?b))
               (at start (not (not-busy ?a)))
               (at start (busy ?a))
             (at start (not (not-clear ?b)))))

 (:durative-action get-meal
  :parameters (?a - agent ?p - person ?r - room)
  :duration (= ?duration 3)
  :condition (and (over all (is_kitchen ?r))
                 (over all (not-busy ?a))
                 (over all (is_in_agent_room ?a ?r)))
  :effect (and (at end (has_meal ?a ?p))
              (at start (not (not-busy ?a)))
              (at start (busy ?a))))

 (:durative-action serve-meal
  :parameters (?a - agent ?p - person ?room_person - room)
  :duration (= ?duration 3)
  :condition (and (over all (is_in_agent_room ?a ?room_person))
                 (over all (is_in_person_room ?p ?room_person))
                 (over all (has_meal ?a ?p))
                 (over all (not-busy ?a))
                 (over all (needs_meal ?p)))
  :effect (and (at end (not (needs_meal ?p)))
              (at end (not (has_meal ?a ?p)))
              (at start (not (not-busy ?a)))
              (at start (busy ?a))))

 (:durative-action free-agent
  :parameters (?a - agent)
  :duration (= ?duration 0)
  :condition  (over all (busy ?a))
  :effect (and (at start (not (busy ?a)))
              (at start (not-busy ?a)))))
```

D.4 Robobusy Time Domain

```
(define (domain robobusytime)
(:requirements :durative-actions :typing :fluents)
(:types agent
        object
        person
        room
        door
        bed)

(:predicates
 (unmade ?b - bed)
 (clear ?b - bed)
 (is_in_person_room ?p - person ?r - room)
 (is_in_bed_room ?b - bed ?r - room)
 (is_in_agent_room ?a - agent ?r - room)
 (is_door ?r1 - room ?r2 - room)
 (needs_to_walk ?p - person)
 (has_meal ?a - agent ?p - person)
 (needs_meal ?p - person)
 (is_kitchen ?k - room)
 (made ?b - bed)
 (not-clear ?b - bed)
 (busy ?a - agent)
 (not-busy ?a - agent))

(:functions (speed_robot ?a - agent)
            (speed_person ?p - person)
            (make_bed ?a - agent)
            (clear_bed ?a - agent)
            (get_meal ?a - agent)
            (serve_meal ?a - agent)
            (distance ?r - room ?rt - room))
```

```
(:durative-action goto-room
 :parameters (?a - agent ?room_from - room ?room_to - room)
 :duration (= ?duration (/ (distance ?room_from ?room_to) (speed_robot ?a)))
 :condition (and (over all (not (= ?room_from ?room_to)))
                 (over all (is_door ?room_from ?room_to))
                 (over all (is_in_agent_room ?a ?room_from))
                 (over all (not-busy ?a)))
 :effect (and (at end (is_in_agent_room ?a ?room_to))
              (at start (busy ?a))
              (at start (not (not-busy ?a)))
              (at start (not (is_in_agent_room ?a ?room_from)))))

(:durative-action walk-person
 :parameters (?a - agent ?p - person ?room_from - room ?room_to - room)
 :duration (= ?duration (/ (distance ?room_from ?room_to) (speed_person ?p)))
 :condition (and (over all (needs_to_walk ?p))
                 (over all (is_door ?room_from ?room_to))
                 (over all (is_in_agent_room ?a ?room_from))
                 (over all (is_in_person_room ?p ?room_from))
                 (over all (< (speed_person ?p) (speed_robot ?a)))
                 (over all (not-busy ?a)))
 :effect (and (at end (is_in_person_room ?p ?room_to))
              (at end (is_in_agent_room ?a ?room_to))
              (at start (not (is_in_agent_room ?a ?room_from)))
              (at start (not (is_in_person_room ?p ?room_from)))
              (at start (busy ?a))
              (at start (not (not-busy ?a)))
              (at start (not (is_in_agent_room ?a ?room_from)))))

(:durative-action make-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :duration (= ?duration (make_bed ?a ))
 :condition (and (over all (unmade ?b))
                 (over all (clear ?b))
                 (over all (is_in_bed_room ?b ?r))
                 (over all (is_in_agent_room ?a ?r))
                 (over all (not-busy ?a)))
 :effect (and (at start (not (unmade ?b)))
              (at end (made ?b))
              (at start (not (not-busy ?a)))
              (at start (busy ?a))))

(:durative-action clear-bed
 :parameters (?a - agent ?b - bed ?r - room)
 :duration (= ?duration (clear_bed ?a))
 :condition (and (over all (is_in_agent_room ?a ?r))
                 (over all (is_in_bed_room ?b ?r))
                 (over all (not-busy ?a))
                 (over all (not-clear ?b)))
 :effect (and (at end (clear ?b))
              (at start (not (not-busy ?a)))
              (at start(busy ?a))
              (at start (not (not-clear ?b)))))

(:durative-action get-meal
 :parameters (?a - agent ?p - person ?r - room)
 :duration (= ?duration (get_meal ?a))
 :condition (and (over all (is_kitchen ?r))
                 (over all (not-busy ?a))
                 (over all (is_in_agent_room ?a ?r)))
 :effect (and (at end (has_meal ?a ?p))
              (at start (not (not-busy ?a)))
              (at start (busy ?a))))
```

```
(:durative-action serve-meal
 :parameters (?a - agent ?p - person ?room_person - room)
 :duration (= ?duration (serve_meal ?a))
 :condition (and (over all (is_in_agent_room ?a ?room_person))
                 (over all (is_in_person_room ?p ?room_person))
                 (over all (has_meal ?a ?p))
                 (over all (not-busy ?a))
                 (over all (needs_meal ?p)))
 :effect (and (at end (not (needs_meal ?p)))
              (at end (not (has_meal ?a ?p)))
              (at start (not (not-busy ?a)))
              (at start (busy ?a))))

(:durative-action free-agent
 :parameters (?a - agent)
 :duration (= ?duration 0)
 :condition  (over all (busy ?a))
 :effect (and (at start (not (busy ?a)))
              (at start (not-busy ?a)))))
```

Appendix E

BT

In this appendix, we present the different BT domain versions used in Chapter 8. The problems that we have generated for this domain can be downloaded at:
http://atc1.aut.uah.es/∼mdolores/bt/

E.1 BT Domain

```
(define(domain bt)
 (:requirements :strips :typing :equality)
 (:types worker
         object
         line
         zone
         door
         spare)

 (:predicates
  (not-built-cable-for ?s-spare)
  (available ?s-spare)
  (not-available ?s-spare)
  (at_line ?l-line ?z-zone)
  (at_spare ?s-spare ?z-zone)
  (at_worker ?w-worker ?z-zone)
  (next ?z1-zone ?z2-zone)
  (available-card ?l-line)
  (built-cable-for ?s-spare)
  (allocated ?l-line))

 (:action moveto_zone
  :parameters (?w-worker ?zone_from-zone ?zone_to-zone)
  :precondition(and(not(= ?zone_from ?zone_to))(at_worker ?w ?zone_from)
                 (next ?zone_from ?zone_to))
  :effect(and(at_worker ?w ?zone_to)(not(at_worker ?w ?zone_from))))

 (:action build_cable
  :parameters (?w-worker ?s-spare ?z-zone)
  :precondition (and (not-built-cable-for ?s)(available ?s)(at_spare ?s ?z)(at_worker ?w ?z))
  :effect (and (not (not-built-cable-for ?s))(built-cable-for ?s)))

 (:action set_spare_available
  :parameters (?w-worker ?s-spare ?z-zone)
  :precondition (and (not-available ?s)(at_spare ?s ?z)(at_worker ?w ?z))
  :effect (and (available ?s)(not (not-available ?s))))

 (:action check_line
  :parameters (?w-worker ?l-line ?z-zone)
  :precondition (and (at_worker ?w ?z)(at_line ?l ?z)(available-card ?l))
  :effect (and (not (available-card ?l))(allocated ?l))))
```

E.2 BT Occupied Domain

```
(define (domain BT_Occu)
 (:requirements :strips :typing :equality)
 (:types worker
         object
         line
         zone
         door
         spare)

 (:predicates
  (not-built-cable-for ?s - spare)
  (available ?s - spare)
  (not-available ?s - spare)
  (at_line ?l - line ?z - zone)
  (at_spare ?s - spare ?z - zone)
  (at_worker ?w - worker ?z - zone)
  (next ?z1 - zone ?z2 - zone)
  (available-card ?l - line)
  (built-cable-for ?s - spare)
  (occupied ?w - worker)
  (not-occupied ?w - worker)
  (allocated ?l - line))

 (:action moveto_zone
  :parameters (?w - worker ?zone_from - zone ?zone_to - zone)
  :precondition (and (not (= ?zone_from ?zone_to)) (not-occupied ?w)
                     (next ?zone_from ?zone_to) (at_worker ?w ?zone_from))
  :effect (and (at_worker ?w ?zone_to) (not (not-occupied ?w))
               (occupied ?w) (not (at_worker ?w ?zone_from))))

 (:action build_cable
  :parameters (?w - worker ?s - spare ?z - zone)
  :precondition (and (not-occupied ?w)(not-built-cable-for ?s)
                     (available ?s) (at_spare ?s ?z) (at_worker ?w ?z))
  :effect (and (not (not-occupied ?w)) (occupied ?w)
               (not (not-built-cable-for ?s)) (built-cable-for ?s)))

 (:action set_spare_available
  :parameters (?w - worker ?s - spare ?z - zone)
  :precondition (and (not-occupied ?w)(not-available ?s)
                     (at_spare ?s ?z)(at_worker ?w ?z))
  :effect (and (not (not-occupied ?w))(occupied ?w)
               (available ?s)(not (not-available ?s))))

 (:action check_line
  :parameters (?w - worker ?l - line ?z - zone)
  :precondition (and (at_worker ?w ?z)(at_line ?l ?z)
                     (available-card ?l) (not-occupied ?w))
  :effect (and (not (not-occupied ?w)) (occupied ?w)
               (not (available-card ?l)) (allocated ?l)))

 (:action free_worker
  :parameters (?w - worker)
  :precondition (occupied ?w)
  :effect (and (not (occupied ?w)) (not-occupied ?w))))
```

E.3 BT Occupied SimpleTime Domain

```
(define (domain BT_OccuSimple)
 (:requirements :durative-actions :strips :typing :equality)
 (:types worker
         object
         line
         zone
         door
         spare)

 (:predicates
  (not-built-cable-for ?s - spare)
  (available ?s - spare)
  (not-available ?s - spare)
  (at_line ?l - line ?z - zone)
  (at_spare ?s - spare ?z - zone)
  (at_worker ?w - worker ?z - zone)
  (next ?z1 - zone ?z2 - zone)
  (available-card ?l - line)
  (built-cable-for ?s - spare)
  (occupied ?w - worker)
  (not-occupied ?w - worker)
  (allocated ?l - line))

 (:durative-action moveto_zone
  :parameters (?w - worker ?zone_from - zone ?zone_to - zone)
  :duration (= ?duration 2)
  :condition (and (over all (not (= ?zone_from ?zone_to)))
                  (over all (not-occupied ?w))
                  (over all (next ?zone_from ?zone_to))
                  (over all (at_worker ?w ?zone_from)))
  :effect (and (at end (at_worker ?w ?zone_to))
               (at start (not (not-occupied ?w)))
               (at start (occupied ?w))
               (at start (not (at_worker ?w ?zone_from)))))

 (:durative-action build_cable
  :parameters (?w - worker ?s - spare ?z - zone)
  :duration (= ?duration 9)
  :condition (and (over all (not-occupied ?w))
                  (over all (not-built-cable-for ?s))
                  (over all (available ?s))
                  (over all (at_spare ?s ?z))
                  (over all (at_worker ?w ?z)))


  :effect (and (at start (not (not-occupied ?w)))
               (at start (occupied ?w))
               (at start (not (not-built-cable-for ?s)))
               (at end (built-cable-for ?s))))

 (:durative-action set_spare_available
  :parameters (?w - worker ?s - spare ?z - zone)
  :duration (= ?duration 6)
  :condition (and (over all (not-occupied ?w))
                  (over all (not-available ?s))
                  (over all (at_spare ?s ?z))
```

```
                 (over all (at_worker ?w ?z)))
 :effect (and (at start (not (not-occupied ?w)))
              (at start (occupied ?w))
              (at end (available ?s))
              (at start (not (not-available ?s)))))

(:durative-action check_line
 :parameters (?w - worker ?l - line ?z - zone)
 :duration (= ?duration 12)
 :condition (and (over all (at_worker ?w ?z))
                 (over all (at_line ?l ?z))
                 (over all (available-card ?l))
                 (over all (not-occupied ?w)))
 :effect (and (at start (not (not-occupied ?w)))
              (at start (occupied ?w))
              (at start (not (available-card ?l)))
              (at end (allocated ?l))))

(:durative-action free_worker
 :parameters (?w - worker)
 :duration (= ?duration 0)
 :condition (over all (occupied ?w))
 :effect (and (at start (not (occupied ?w)))
              (at end (not-occupied ?w)))))
```

Appendix F

The Planners Language Automatic Translator Application: PLATA

The Planners Language Translator Automatic Application (PLATA) is in charge of the automatic translation from PDDL2.1 into PRODIGY (PDL4.0) syntax. In this appendix we will show its functionality and how the translation is done by the tool having in mind each of the elements of both syntax.

The tool consists of two modules. Figure F.1 shows these two modules and how they interact.

- **The PDDL module:** is in charge of the domain and problems creation or edition. This module also allows the user to translate the domain and problems into PDL4.0.

- **The PDL4.0 module:** allows to edit or create a new domain and problems in PDL4.0. From this module the user can run PRODIGY and visualise the solution to the chosen problem.

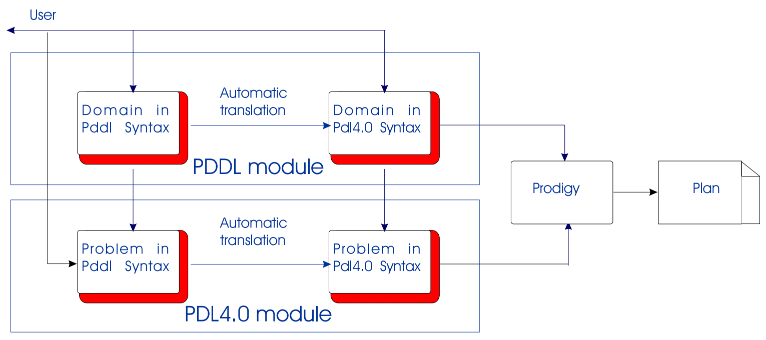


Figure F.1: Architecture of the PLATA tool.

In the following sections we will explain how to translate domains and problems from PDDL2.1 to PDL4.0 using as example the zenotravel domain used in the experiments of section 8.3.2.

F.1 The Domains

The first thing to do when creating a new domain is to introduce its name. The translation is done automatically as shows Figure F.2.

```
      (define (domain zenotravel)
                     ↓
(create-problem-space ’zenotravel :current t)
```

Figure F.2: How to define the domain name in PDDL2.1 and PDL4.0 respectively.

F.1.1 Types

PDL4.0 types and subtypes are also translated directly as we see on Figure F.3.

```
(:types aircraft person city - object)
                 ↓
      (ptype-of object :top-type)
      (ptype-of aircraft object)
      (ptype-of person object)
      (ptype-of city object)
```

Figure F.3: Translation of types and subtypes.

:top-type is used to indicate that a type does not belong to any other type, unlike subtypes where the type they belong to must be specified. Figure F.4 shows how to introduce types through the PLATA interface.

F.1.2 Constants

Any instances of the types defined that are constant in a domain are called *constants*. They are also translated automatically as it is shown in Figure F.5.

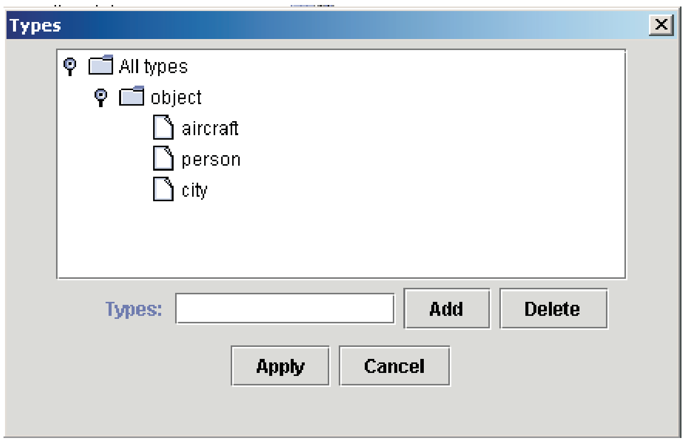


Figure F.4: Window to edit/create types in the PDDL2.1 syntax.

```
(:constants plane - aircraft)
             ↓
(pinstance-of plane aircraft)
```

Figure F.5: Constants translation from PDDL2.1 into PRODIGY syntax.

F.1.3 Predicates and Functions

The predicates (field *:predicates* in Figure F.6) do not have to be declared in PDL4.0 before the operators declaration as in PDDL2.1. They are just translated into the operators body. The problem appears when functions are defined (field *:functions* in Figure F.6) because PDL4.0 does not admit the declaration of functions as such. But functions in PDDL2.1 can be handled using the PDL4.0 *gen-from-pred* function (it assigns a numeric value to a variable as explained in Chapter 5) and converting the PDDL2.1 function to a predicate where the last variable is a number. These variables must be specified and declared as numbers using the type declaration for numbers. In PRODIGY, this is done with the *infinite-type* specification as Figure F.6 shows.

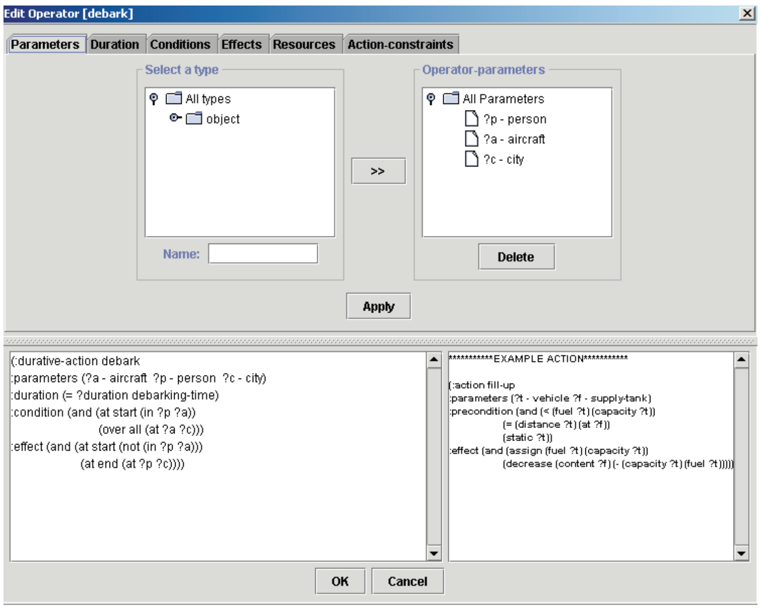
F.1.4 Operators

Figure F.7 shows the PLATA interface to introduce operators in PDDL2.1 syntax and Figure F.8 shows how to translate a *durative-action* operator from PDDL2.1 to PDL4.0. All the data is introduced through the interface to avoid syntax errors. In the case of operators of type *action* (that is, operators without duration) the translation is exactly the same, with the only difference that the field *:duration* does not exist.

The word *durative-action* is changed to *operator*. The fields *:parameters* and *:conditions* (if the operator is of type *action* this field is called *:preconditions*) are translated

```
(:predicates (at ?x - (either person aircraft) ?c - city)
             (in ?p - person ?a - aircraft))
(:functions(fuel ?a - aircraft)
           (distance ?c1 - city ?c2 - city)
           (debarking-time)) ...
                    ↓
...  declaration of infinite types ...
(infinite-type FUEL #'numberp)
(infinite-type DISTANCE #'numberp)
(infinite-type DEBARKING-TIME #'numberp)
...  PDL operator body ...
the predicates are translated inside the operator body
(at <x> <c>)
(in <p> <a>)
...(fuel <a> <FUEL>)
the functions are translated inside the operator body
(distance <c1> <c2> <DISTANCE>)
(debarking-time <DEBARKING-TIME>)
```

Figure F.6: Translation of predicates and functions from PDDL2.1 into PDL4.0.



```
(:durative-action debark
:parameters (?a - aircraft  ?p - person  ?c - city)
:duration (= ?duration debarking-time)
:condition (and (at start (in ?p ?a))
                (over all (at ?a ?c)))
:effect (and (at start (not (in ?p ?a)))
             (at end (at ?p ?c))))
```

```
***********EXAMPLE ACTION***********

(:action fill-up
:parameters (?t - vehicle ?f - supply-tank)
:precondition (and (< (fuel ?t) (capacity ?t))
                   (= (distance ?t) (at ?f))
                   (static ?t))
:effect (and (assign (fuel ?t) (capacity ?t))
                   (decrease (content ?f) (- (capacity ?t) (fuel ?t)))))
```

Figure F.7: Window to edit/create operators in the PDDL2.1 syntax.

into *params* and *preconds*. In this last field, the first thing to do is to declare the variables that will be part of the predicates and the PDDL2.1 'functions'. In the case of Figure F.8, two non-numeric variables of types defined before are declared as well as two new numeric variables <ca> and <f> that belong to the numeric types CAPACITY and FUEL (the tool uses as the name of the numeric variables the same as the function name in upper case). When we need to compare values or do simple arithmetic operations, they can be done in PDL4.0 without using any additional function as it happens with the comparison *(> <ca> <f>)*. Also, the temporal specification that PDDL2.1 uses (*over all, at start* or *at end*) is omitted in PDL4.0 because it does not handle a non-conservative model of actions, although this could be easily managed as [45] shows. Once the variables are declared, the tool automatically translates the predicates. In this case only one: *at*.

With respect to the operator effects, the field *:effect* is translated into *effects*. In PDL4.0 the predicates that are added by the operator (add list, *add*) are considered separately from the ones removed (delete list, *del*). In PDDL2.1 this is done by placing the reserved word *not* in front of the predicate. In the example, the operator just assigns a new value to the function, but in PDL4.0 we must delete the old value and add the new value as Figure F.8 shows.

The QPRODIGY planner supports durations as time metric. The tool translates the field *:duration* into the field *cost* and defining it as the cost metric *TIME* by default. A numeric value or function can be assigned to this variable. In the example, the duration is computed through a function call. Due to the syntactic complexity of the function (it is not a comparison or a simple arithmetic function) the user must code a function because the tool cannot do it automatically. The name we have decided is *new-duration-refuel* and the code implemented is as follows (notice that this function is coded in Common Lisp):

```
(defun new-duration-refuel (capacity fuel rate)
  (list (/ (- capacity fuel) rate)))
```

F.2 The Problems

In relation to the problems, the translation is automatic because functions, as mentioned before, are translated into predicates with the last variable being a number and the predicates remain equal. The only difference falls on the metric; in PDL4.0 it is not specified in the problem, but when setting the values to run the problem. In the problem of Figure F.9 we should set the parameter *cost-type* (this parameter specifies the cost or the metric we want to use) with the value *TIME* since this is the name of the metric specified in the operators (see Figure F.8). In that case, QPRODIGY will try to minimise the total time. Although in PDDL2.1 we cannot specify the maximum value that the duration of the plan can have, in the settings of the running problem in QPRODIGY we can limit it using the sentence *cost-bound*, followed by the value to which we want to limit it. Figure F.11 shows how to edit or create a

```
(:durative-action REFUEL}
 :parameters (?a - aircraft ?c - city)
 :duration (= ?duration  (/ (- (capacity ?a)(fuel ?a))
                            (refuel-rate ?a)))
 :condition (and (at start (> (capacity ?a) (fuel ?a)))
                 (over all (at ?a ?c)))
 :effect (at end (assign (fuel ?a) (capacity ?a))))
```

$$\downarrow$$

```
(OPERATOR REFUEL
 (params <a> <c>)
  (preconds
  ((<a> AIRCRAFT)
   (<c> CITY)
   (<ca> (and CAPACITY (gen-from-pred (capacity <a> <ca>))))
   (<f> (and FUEL (gen-from-pred (fuel <a> <f>))
                  (> <ca> <f>))))
   (at <a> <c>)
  (effects
   ()
   ((del (fuel <a> <f>))
    (add (fuel <a> <ca>))))
  (costs ((<time> (and DURATION
                      (new-duration-refuel <ca> <f> <rate>))))
         ((TIME <time>))))
```

Figure F.8: Translation of the *refuel* operator.

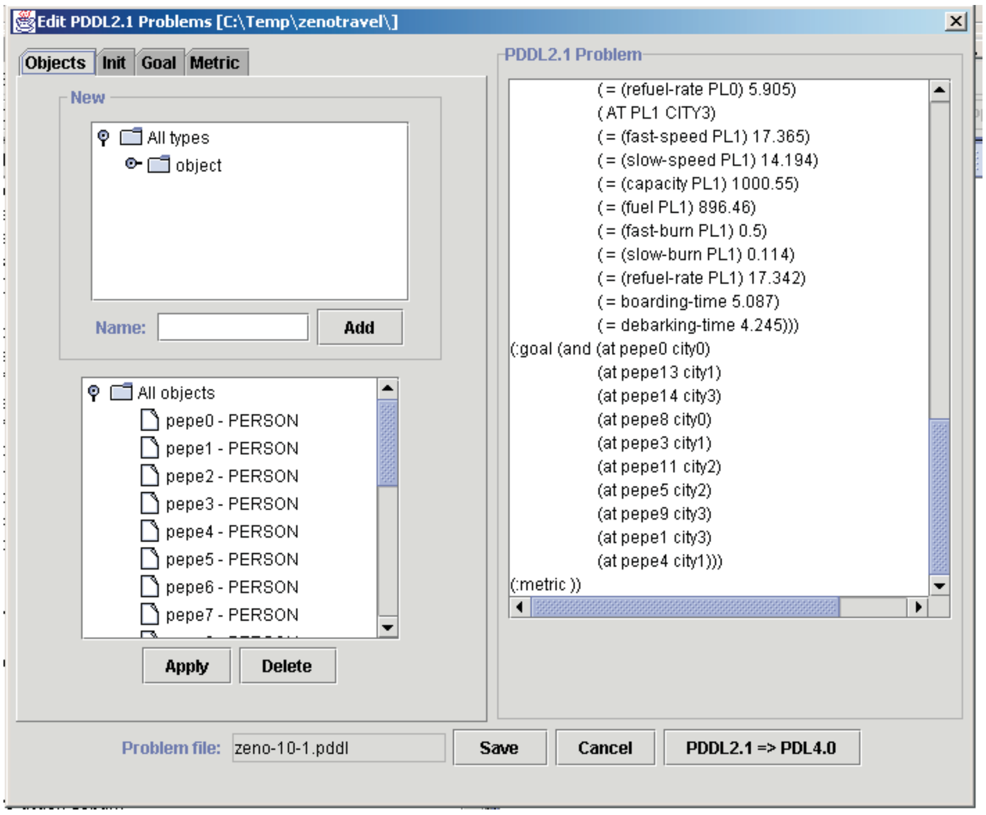
problem in PLATA, notice the button in the bottom of the window to translate the problem automatically into PDL4.0.

```
(define (problem zeno01)
 (:domain zeno-travel)
 (:objects plane - aircraft
           ernie scott dan - person
           city-a city-b city-c city-d - city)
 (:init (and (= total-fuel-used 0)
             (= debarking-time 20)(= boarding-time 30)
             (= (distance city-a city-b) 600)
             (= (distance city-b city-a) 600)
             (= (distance city-b city-c) 800)
             (= (distance city-c city-b) 800)
             (= (distance city-a city-c) 1000)
             (= (distance city-c city-a) 1000)
             (= (distance city-c city-d) 1000)
             (= (fast-speed plane) 10) (= (fuel plane) 750)
             (= (slow-speed plane) 6.67)
             (= (capacity plane) 750)
             (= (fast-burn plane) 0.5)
             (= (slow-burn plane) 0.33)
             (at plane city-c) (at dan city-c)
             (at ernie city-c) (at scott city-a)))
 (:goal (and (at ernie city-d) (at scott city-d)))
 (:metric minimize total-time))
```

Figure F.9: Example of a problem in PDDL2.1.

```
(setf (current-problem)
      (create-problem (name zeno01)
      (objects (plane aircraft)
               (ernie scott dan person)
               (city-a city-b city-c city-d city))
      (state (and (debarking-time 20)(boarding-time 30)
                  (distance city-a city-b 600)
                  (distance city-b city-a 600)
                  (distance city-b city-c 800)
                  (distance city-c city-b 800)
                  (distance city-a city-c 1000)
                  (distance city-c city-a 1000)
                  (distance city-c city-d 1000)
                  (fast-speed plane 10)(fuel plane 750)
                  (slow-speed plane 6.67)
                  (capacity plane 750)
                  (fast-burn plane 0.5)(slow-burn plane 0.33)
                  (at plane city-c) (at ernie city-c)
                  (at dan city-c) (at scott city-a)))
      (goal (and (in ernie city-d) (in scott dity-d)))))
```

Figure F.10: The problem of Figure F.9 translated into PDL4.0.



```
          (= (refuel-rate PL0) 5.905)
          (AT PL1 CITY3)
          (= (fast-speed PL1) 17.365)
          (= (slow-speed PL1) 14.194)
          (= (capacity PL1) 1000.55)
          (= (fuel PL1) 896.46)
          (= (fast-burn PL1) 0.5)
          (= (slow-burn PL1) 0.114)
          (= (refuel-rate PL1) 17.342)
          (= boarding-time 5.087)
          (= debarking-time 4.245)))
(:goal (and (at pepe0 city0)
          (at pepe13 city1)
          (at pepe14 city3)
          (at pepe8 city0)
          (at pepe3 city1)
          (at pepe11 city2)
          (at pepe5 city2)
          (at pepe9 city3)
          (at pepe1 city3)
          (at pepe4 city1)))
(:metric ))
```

Figure F.11: Window to edit/create problems in the PDDL2.1 syntax.

Appendix G

Temporal and Resource Language Extension

This appendix explains the temporal and resource extension proposed to PDDL2.1, valid also for PDL4.0 or any other planning language. This extension as [69] tries to suggest a way to model real domains more easily, although we have not considered uncertainty or sensing extensions.

G.1 Intervals

We can define an interval when an action must be executed. For that we need to define the lower and upper bounds of the interval. Both, upper and lower bounds must be positive numbers. Intervals can be defined between activities (activity constraints) and in the goals. A BNF description of intervals is as follows:

<interval> ::= [<lb>, <up>]   where <lb> $\leq$ <up>
<interval1> ::= [<lb>, <up>]   where <lb> $\leq$ <up> and <lb>, <up> $\neq 0$
<lb> ::= <positive number>
<up> ::= <positive number>
<positive number> ::= integer positive number (included 0)

Example of intervals are:
[3, 6] or [0,0]

G.2 Metric

PDDL2.1 does not allow to define values for the metric defined in the domain. A BNF description of this extension could be:

total-time <binary-comp> <number> (can be considered as the <length-spec>)

<binary-comp> ::= < | <= | = | > | >=

<number> ::= Any numeric literal (integer and greater or equal to zero)

As an example of the metric field:
total-time = 20
total-fuel $\leq$ 20

G.3 Action constraints

Another useful extension would be to use the Allen primitives using the terminology described by the author in [7]. A BNF description of action constraints is as follows:
<Act-const>::= Visibility Window <action-symbol> in <interval>
<Act-const>::= <action-symbol> before <action-symbol> between <interval1>
<Act-const>::= <action-symbol> overlaps <action-symbol> between <interval>
<Act-const>::= <action-symbol> <Temp-Cons> <action-symbol>
<Temp-Cons>::= meets | equals | finishes | starts
<Act-const>::= <action-symbol> during <action-symbol> in <interval> <interval>

Some examples are:
Visibility Window Act_1 in [13, 40]
Act_1 before Act_2 between [3, 4]
Act_1 during Act_2 in [3, 4] [5, 6]

G.4 Goals

Goals can be extended to include the time instant when they must be true.
<GD> ::= <predicate> <op-temp>
<op-temp> ::= <binary-comp> <positive number>
<op-temp> ::= within <interval>
<op-temp> ::= at <positive number> equivalent to (<predicate> t*) in PDDL2.1 syntax
Examples of this syntax are:
(at packet2 office2) at 10
(at packet1 office2) $\leq$ 30
(at packet2 office2) within [40, 50]

G.5 Resources

Instead of considering resources as another predicate, we can consider them separately. For that, we first define the resource type in the domain definition (that is, before the actions specification):

<resource-def> ::= (:resource <resource-squeleton> $^+$)
<resource-squeleton> ::= <res-type> <res-name>
<res-type> ::= :renewable
<res-type> ::= :consumable
<res-type> ::= :producible
<res-name> ::= <name>

As an example of the resource requirements that must declared before the operator definition:

```
(:resource :consumable fuel)
```

Then, in the operator definition, the resource requirements are added (it can not consume more than the total capacity defined in the problem specification). We need to declare one or more variables for the resources total capacity (*total-capacity-1, total-capacity-2, ...*) and one or more resources variables (*resource-1, resource-2, ...*) for each of the resources consumed by the activity (note that it is done in the same way as in the *duration* field). An example of the extension proposed is shown in Figure G.1.

```
(:durative-action board
  :parameters (?p - person ?a - aircraft ?c - city)
  :duration (= ?duration boarding-time)
  :condition (and (at start (at ?p ?c))
                  (over all (at ?a ?c)))
  :effect (and (at start (not (at ?p ?c)))
               (at end (in ?p ?a)))
  :capacity (= ?capacity total-capacity-1 ?a)
  :resource consumes (= (< (?resource-1 fuel ?a) ?capacity)))
```

Figure G.1: Resources extension for PDDL2.1.

In QPRODIGY/IPSS this should be done by functions that we call *capacity-from-pred* and *resource-from-pred*.

```
:capacity (<cap> (and CAPACITY (capacity-from-pred (total-capacity <a> <cap>))))
:consumes (<f> (and FUEL (resource-from-pred (fuel <a> <f>))
                         (> <cap> <f>)))
```

Bibliography

[1] AI CHANG, M., BRESINA, J., CHARESTY, L., HSU, J., JONSSON, A., KANEFSKY, B., MALDAGUEY, P., MORRIS, P., AND RAJAN, K. YGLESIAS, J. MAPGEN Planner: Mixed-initiative Activity Planning for the Mars Exploration Rover Mission. In *Procs. of the International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)* (2003).

[2] ALBERS, P., AND GHALLAB, M. Context Dependent Effects in Temporal Planning. In *Procs. of the 4th European Conference on Planning.* (1997), pp. 1–12.

[3] ALER, R., AND BORRAJO, D. Learning Single-Criteria Control Knowledge for Multi-Criteria Planning. In *Procs. of the AIPS-02 Workshop on Planning and Scheduling with Multiple Criteria* (2002), pp. 35–40.

[4] ALER, R., AND BORRAJO, D. On Control Knowledge Acquisition by Exploiting Human-Computer Interaction. In *Procs. of the Sixth International Conference on Artificial Intelligence Planning Systems (AIPS-02)* (2002), pp. 112–120.

[5] ALER, R., BORRAJO, D., CAMACHO, D., AND SIERRA-ALONSO, A. A knowledge-based approach for business process reengineering, SHAMASH. *Knowledge Based Systems, 15*, 8 (2002), 473–483.

[6] ALER, R., BORRAJO, D., AND ISASI, P. Using Genetic Programming to Learn and Improve Control Knowledge. *Artificial Intelligence, 141* (2002), 29–56.

[7] ALLEN, J. Towards a General Theory of Action and Time. *Artificial Intelligence, 23* (1984), 123–154.

[8] ALLEN, J. F., HENDLER, B., AND TATE, A. *Readings in Planning.* Morgan Kauffmann, 1990.

[9] AUSIELLO, G., ITALIANO, G., SPACCAMELA, A. S., AND NANNI, U. Incremental Algorithms for Minimal Length Paths. *Journal of Algorithms, 12* (1991), 615–638.

[10] BACCHUS, F., AND KABANZA, F. Using Temporal Logics to Express Search Control knowledge for planning. *Artificial Intelligence, 16* (2000), 123–191.

[11] BÄCKSTRÖM, C. Computational Aspects of Reordering Plans. *Journal of Artificial Intelligence Research, 9* (1998), 99–137.

[12] BARTÁK, R. On Modelling Planning and Scheduling Problems with Time and Resources. In *The 21st Workshop of the UK Planning and Scheduling: PLANSIG2002. Delft (The Netherlands)* (2002).

[13] BELLICHA, A. Maintenance of a Solution in a Dynamic Constraint Satisfaction Problem. *Applications of Artificial Intelligence in Engineering,* (1993), 261–274.

[14] BERRY, P. M., AND DRABBLE, B. SWIM: An AI-based System for Workflow Enabled Reactive Control. In *the 16th IJCAI99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business. Mamdouth Ibrahim, Brian Drabble and Peter Jarvis editors* (1999).

[15] BLUM, A., AND FURST, M. Fast Planning Through Planning Graph Analysis. *Artificial Intelligence, 90* (1997), 281–300.

[16] BLYTHE, J. Decompositions of Markov Chains for reasoning about external change in planners. In *Procs. of the 3rd International Conference on AI Planning Systems. AAAI Press.* (1996), pp. 27–34.

[17] BLYTHE, J. Decision Theoretic Planning. *AI Magazine, 20*, 2 (1999), 37–54.

[18] BONET, B., AND GEFFNER, H. Planning as Heuristic Search: New results. In *Procs. of the ECP-99. Springer* (1999).

[19] BONET, B., AND GEFFNER, H. Planning as Heuristic Search. *Artificial Intelligence, 129*, 1-2 (2001), 5–33.

[20] BORRAJO, D., VEGAS, S., AND VELOSO, M. Quality-Based Learning for Planning. In *Working notes of the IJCAI'01 Workshop on Planning with Resources. IJCAI Press. Seattle, WA (USA).* (2001).

[21] BORRAJO, D., AND VELOSO, M. Lazy Incremental Learning of Control Knowledge for Efficiently Obtaining Quality Plans. *AI Review Journal, Special Issue on Lazy Learning, 10* (1996), 371–405.

[22] CARBONELL, J. G., BLYTHE, J., ETZIONI, O., GIL, Y., JOSEPH, R., KAHN, D., KNOBLOCK, C., MINTON, S., PÉREZ, A., REILLY, S., VELOSO, M., AND WANG, X. PRODIGY4.0: The Manual and Tutorial. Tech. Rep. CMU-CS-92-150, Department of Computer Science, 1992.

[23] CARBONELL, J. G., KNOBLOCK, C. A., AND MINTON, S. PRODIGY: An Integrated Architecture for Planning and Learning. *Architectures for Intelligence,* (1991), 241–278.

[24] CASTILLO, L., FDEZ.-OLIVARES, J., AND GONZÁLEZ, A. On the Adequacy of Hierarchical Planning Characteristics for Real-World Problem Solving. In *Procs. of the Sixth European Conference on Planning (ECP'01)* (2001).

[25] CASTILLO, L., FDEZ.-OLIVARES, J., AND GONZÁLEZ, A. A Temporal Constraint Based Network Temporal Planner. In *Procs. of UK PLANSIG02 Workshop, Delft, The Netherlands* (2002).

[26] CERVONI, R., CESTA, A., AND ODDI, A. Managing Dynamic Temporal Constraint Networks. In *Procs. of the Second International Conference of Artificial Intelligence Planning Systems (AIPS94)* (1994).

[27] CESTA, A., CORTELLESSA, G., ODDI, A., POLICELLA, N., SEVERONI, F., AND SUSI, A. Reconfigurable Constraint-Base Architecture as a Support for Automating Mission Planning. In *ESA Workshop on On-Board Autonomy. (AG Noordwijk, The Netherlands).* (2001), pp. 219–226.

[28] CESTA, A., AND ODDI, A. Algorithms for Dynamic Management of Temporal Constraint Networks. Tech. rep., Italian National Research Council (ISTC-CNR), 2002.

[29] CESTA, A., ODDI, A., AND SMITH, S. Profile Based Algorithms to Solve Multiple Capacitated Metric Scheduling Problems. In *Proceedings of the Fourth Int. Conf. on Artificial Intelligence Planning Systems (AIPS-98)* (1998).

[30] CESTA, A., ODDI, A., AND SMITH, S. A Constrained-Based Method for Project Scheduling with Time Windows. *Journal of Heuristics, 8* (2002), 109–136.

[31] CESTA, A., ODDI, A., AND SMITH, S. F. Greedy Algorithms for the Multi-Capacitated Metric Scheduling Problem. In *Proceedings 1999 European Conference on Planning* (1999).

[32] CESTA, A., ODDI, A., AND SUSI, A. O-OSCAR: A Flexible Object-Oriented Architecture for Schedule Management in Space Applications. In *Procs. of the Fifth International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS-99)* (1999).

[33] CESTA, A., AND PECORA, F. The RoboCare Project: Multi-Agent Systems for the Care of the Elderly. In *European Research Consortium for Informatics and Mathematics (ERCIM) News No. 53* (2003).

[34] CHANDLER, J. Management of Special Services: Designing for a Changing World. *BT Technological Journal, 15,* 1 (1997).

[35] CHENG, C., AND SMITH, S. Generating Feasible Schedules under Complex Metric Constraints. In *Procs. 12th National Conference on AI (AAAI-94)* (1994).

[36] CHIEN, S., KNIGHT, R., STECHERT, A., SHERWOOD, R., AND RABIDEAU, G. Integrated Planning and Execution for Autonomous Spacecraft. In *Proceedings of the IEEE Aerospace Conference (IAC), Aspen, CO* (1999).

[37] CHIEN, S., RABIDEAU, G., KNIGHT, R., SHERWOOD, R., ENGELHARDT, B., MUTZ, D., ESTLIN, T., SMITH, B., FISHER, F., BARRETT, T., STEBBINS, G., AND TRAN, D. ASPEN - Automating Space Mission Operations using Automated Planning and Scheduling. In *SpaceOps 2000, Toulouse, France* (2000).

[38] CHIEN, S., RABIDEAU, G., WILLIS, J., AND MANN, T. Automating Planning and Scheduling of Shuttle Payload Operations. *Artificial Intelligence Journal, 114* (1999), 239–255.

[39] CHIEN, S., RABIDEAU, G., WILLIS, J., AND MANN, T. RADARSAT-MAMM Automated Mission Planner. *AI Magazine, 23*, 2 (2002), 25–36.

[40] CHIEN, S., SHERWOOD, R., TRAN, D., CASTANO, R., CICHY, B., DAVIES, A., RABIDEAU, G., TANG, N., BURL, M., MANDL, D., FRYE, S., HENGEMIHLE, J. DAGOSTINO, J., BOTE, R., TROUT, B., SHULMAN, S., UNGAR, S., VAN-GAASBECK, J., BOYER, D., GRIFFIN, M., BURKE, H., GREELEY, R., DOGGETT, T., WILLIAMS, K., BAKER, V., AND DOHM, J. Autonomous Science on the EO-1 Mission. In *Procs. of the International Symposium on AI, Robotics and Automation in Space (i-SAIRAS)* (2003).

[41] CHVÁTAL, V. *Linear Programming.* W.H. Freeman, 1983.

[42] CIMATTI, A., AND ROVERI, M. Conformant Planning via Symbolic Model Checking. *Journal of Artificial Intelligence Research, 13* (2000), 305–338.

[43] CLARKE, M., AND WING, J. M. Formal Methods: State of the Art and Future Directions. *ACM Computing Surveys. 28*, 4 (1996), 626–643.

[44] CODDINGTON, A. A Continuous Planning Framework with Durative Actions. In *9th International Symposium on Temporal Representation and Reasoning, TIME-2002. Manchester, UK. IEEE Computer Society, 2002.* (2002), pp. 108–118.

[45] CODDINGTON, A., AND FOX, M. Handling Durative Actions in Classical Planning Frameworks. In *Procs. of the 20th Workshop of the UK Planning and Scheduling Special Interest Group. PLANSIG2001. The Open University, UK* (2001).

[46] CODOGNET, P., AND ROSSI, F. Solving and Programming with Soft Constraints: Theory and Practice. In *Notes for the ECAI2000 tutorial.* (2000).

[47] CURRIE, K., AND TATE, A. O-Plan: The Open Planning Architecture. *Artificial Intelligence, 52* (1991), 49–86.

[48] DAVENPORT, T., AND SHORT, J. The New Industrial Engineering: Information Technology and Business Process Redesign. In *Sloan Management Review* (1990), pp. 11–27.

[49] DEALE, M., YVANOVICH, M., SCHNITZIUS, D., KAUTZ, D., CARPENTER, M., ZWEBEN, M., DAVIS, G., AND DAUN, B. *Intelligent Scheduling.* Morgan Kauffmann, 1994, ch. The Space Shuttle Ground Processing Scheduling System., pp. 423–449.

[50] DECHTER, R., MEIRI, I., AND PEARL, J. Temporal Constraint Networks. *Artificial Intelligence, 49* (1991), 61–95.

[51] DECHTER, R., AND PEARL, J. Directed Constraint Networks: A Relational Framework for Casual Modelling. In *Procs. of IJCAI' 91.* (1991).

[52] DO, M. B., AND KAMBHAMPATI, S. SAPA: A Scalable Multi-objective Heuristic Metric Temporal Planner. *Journal of Artificial Intelligence Research. To appear.* (2003).

[53] DOU, D., MCDERMOTT, D., AND QI, P. Ontology Translation by Ontology Merging and Automated Reasoning. In *Procs. EKAW Workshop on Ontologies for Multi-Agent Systems.* (2002).

[54] DRAPPER, D., HANKS, S., AND WELD, D. A Probabilistic Model of Action for Least-Commitment Planning with Information Gathering. In *Procs. of the Tenth Conference on Uncertainty in Artificial Intelligence. Morgan Kaufmann.* (1994), pp. 178–186.

[55] DRAPPER, D., HANKS, S., AND WELD, D. Temporal Planning with Continuous Change. In *Hammond, K., ed., Procs. 2nd International Conference on AI Planning Systems. University of Chicago, Illinois. AAAI Press.* (1994).

[56] EDELKAMP, S., AND HELMERT, M. On the Implementation of MIPS. In *AIPS Workshop on Model-Theoretic Approaches to Planning.* (2000).

[57] EDELKAMP, S., AND HELMERT, M. Planning via Model Checking in Dynamic Temporal Domains: Exploting Planning Representations. In *AIPS Workshop on Planning via Model Checking.* (2002).

[58] EMERSON, E. A. Temporal and Modal Logic. *Handbook of Theoretical Computer Science. MIT. B* (1990), 997–1072.

[59] ERNST, M. D., MILLSTEIN, T. D., AND WELD, D. Automatic SAT-Compilation of Planning Problems. *Procs. IJCAI-97.* (1997).

[60] EROL, K. *HTN Planning: Formalisation, Analysis and Implementation.* PhD thesis, Computer Science Dept., University of Maryland, USA, 1995.

[61] ESTLIN, T., RABIDEAU, G., MUTZ, D., AND CHIEN, S. Using Continuous Planning Techniques to Coordinate Multiple Rovers. In *Procs. of the IJCAI99 Workshop on Scheduling and Planning meet Real-time Monitoring in a Dynamic and Uncertain World.* (1999).

[62] EVEN, S. *Graph Algorithms.* Computer Science, Press, 1979.

[63] FIKES, R., AND NILSSON, N. STRIPS: A new Approach to the Application of Theorem Proving to Problem Solving. *Artificial Intelligence, 2* (1971), 189–208.

[64] FINK, E., AND BLYTHE, J. Rasputin: A Complete Bidirectional Planner. In *Procs. of the Fourth Conference on AI Planning Systems.* (1998).

[65] FOX, M., AND LONG, D. The Automatic Inference of State Invariants in TIM. *Journal of Artificial Intelligence Research, 9* (1998), 367–421.

[66] FOX, M., AND LONG, D. PDDL+: Planning with Time and Metric Resources. Tech. Rep. CVC TR-98-003/DCS TR-1165, University of Durham, UK, 2001.

[67] FOX, M., AND LONG, D. STAN4: A Hybrid Planning Strategy Based on Subproblem Abstraction. *AI Magazine 22*, 4 (2001).

[68] FOX, M., AND LONG, D. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *AIPS'02 competition* (2002).

[69] FRANK, J., GOLDEN, K., AND JONSSON, A. The Loyal Opposition Comments on Plan Domain Description Languages. In *Workshop on PDDL in the 13th International Conference on Automated Planning and Scheduling (ICAPS'03)* (2003).

[70] FRANK, J., JÓNSSON, A., AND MORRIS, P. On Reformulating Planning as Dynamic Constraint Satisfaction. In *Procs. of the Symposium on Abstraction, Reformulation and Aproximation (SARA).* (2000).

[71] GARRIDO, A., AND BARBER, F. Integrating Planning and Scheduling. *Applied Atificial Intelligence* (2001).

[72] GARRIDO, A., FOX, M., AND LONG, D. A Temporal Planning System for Durative Actions of PDDL2.1. In *Procs. of the European Conference on AI (ECAI-2002)* (2002), F. V. Harmelen, Ed., pp. 586–590.

[73] GARRIDO, A., ONAINDÍA, E., AND BARBER, F. Time-Optimal Planning in Temporal Problems. In *Procs. of the European Conference on Planning (ECP-2001)* (2001), pp. 397–402.

[74] GEREVINI, A., AND DIMOPOULOS, Y. Temporal Planning through Mixed Integer Programming. In *Procs. of the AIPS'02 Workshop on Planning for Temporal Domains* (2002).

[75] GEREVINI, A. SCHUBERT, L., AND SCHAEFFER, S. The Temporal Reasoning Systems TimeGraph I-II. Tech. Rep. 494, 1993.

[76] GHALLAB, M., AND LARUELLE, H. Representation and Control in IxTeT, a Temporal Planner. In *Procs. of the Second International Conference on AI Planning Systems (AIPS-94)* (1994).

[77] GIL, Y., AND BLYTHE, J. PLANET: A Shareable and Reusable Ontology for Representing Plans. In *Procs. of the AAAI 2000 Workshop on Representational Issues for Real-World Planning Systems* (2000).

[78] GOLDMAN, R. P., AND BODDY, M. S. Conditional Linear Planning. In *Procs. 2nd International Conference AI Planning Systems. Chicago, IL. AAAI Press.* (1994), pp. 80–85.

[79] GOLDMANN, S., MüNCH, J., AND HOLZ, H. Distributed Process Planning Support with MILOS. *Int. Journal of Software Engineering and Knowledge Engineering, 10*, 4 (2000), 511–525.

[80] HADDAWY, P., AND SUWANDI, M. Decision-Theoretic Refinement Planning using Inheritance Abstraction. In *Procs. of the 2nd International Conference on AI Planning Systems. AAAI Press.* (1994).

[81] HAGER, W. W. Dual techniques for constraint optimization. *Journal of Optimization Theory and Applications, 55* (1987), 37–71.

[82] HAMMER, M., AND CHAMPY, J. Reengineering the Corporation. In *Reengineering the Corporation. Harper Business Press, New York.* (1993).

[83] HANNEBAEUR, M. From formal Workflow models to Intelligent Agents. In *AAAI-99 Workshop on agent-Based Systems in The Business Context. Brian Drabble and Peter Jarvis Cochairs.* (1999).

[84] HASLUM, P., AND GEFFNER, H. Heuristic Planning with Time and Resources. *Procs. of the 6th European Conference on Planning ECP01.* (2001).

[85] HOFFMANN, J. The Metric-FF Planning System: Translating Ignoring Delete Lists to Numerical State Variables. *Journal of Artificial Intelligence Research,* (2002). Submitted to the special issue on the 3rd International Planning Competition.

[86] HOFFMANN, J., AND NEBEL, B. The FF Planning System: Fast Plan Generation Through Heuristic Search. *Journal of Artificial Intelligence Research, 14* (2001), 253–302.

[87] IPC-2002. International Planning Competition. In *http://www.dur.ac.uk/d.p.long/IPC* (2002).

[88] ISUKAPALLI, S. S. *Uncertainty Analysis of Transport-Transformation Models.* PhD thesis, The State University of New Jersey, USA, 1999.

[89] JARVIS, P., MOORE, J., STADER, J., MACINTOSH, A., CASSON-DU MONT, A., AND CHUNG, P. What Right Do You Have to Do That?. In *ICEIS, 1st International Conference on Enterprise Information Systems. http://www.aiai.ed.ac.uk/project/tbpm/* (1999).

[90] JÓNSSON, A., MORRIS, P., MUSCETTOLA, N., RAJAN, K., AND SMITH, B. Planning in Interplanetary Space: Theory and Practice. *Artificial Intelligence Planning Systems,* (2000), 177–186.

[91] JOHNSTON, M. D. *Intelligent Scheduling.* Morgan Kauffmann, 1994, ch. SPIKE: Intelligent Scheduling of Hubble Space Telescope Observations., pp. 391–422.

[92] KAUTZ, H., AND SELMAN, B. Planning as Satisfiability. *Procs. of the ECAI-92.* (1992), 359–363.

[93] KAUTZ, H., AND SELMAN, B. Pushing the Envelope: Planning, Propositional Logic and Stochastic Search. *Procs. of the AAAI-96.* (1996).

[94] KAUTZ, H., AND SELMAN, B. BLACKBOX: A New Approach to the Application of Theorem Proving to Problem Solving. *Working notes of the Workshop on Planning as Combinatorial Search.* (1998).

[95] KAUTZ, H., AND WALSER, J. State Space Planning by Integer Optimisation. In *Procs. of the 16th National Conference on Artificial Intelligence.* (1999).

[96] KEARNEY, P., AND BORRAJO, D. An R&D Agenda for AI Planning applied to Workflow. In *Proceedings of the eBusiness and eWork Conference 2000.* (2000).

[97] KOEHLER, J., NEBEL, B., HOFFMANN, J., AND DIMOPOULUS, Y. Extending Planning Graphs to an ADL subset. *Procs. of the ECP97* (1997).

[98] KOLISCH, R., AND HARTMANN, S. Heuristic Algorithms for Solving the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis. *Project scheduling: Recent models, Algorithms and Applications* (1999), 147–178.

[99] KWAK, N. K., AND SCHNIEDERJANS, M. J. *Introduction to Mathematical Programming.* Krieger Publishing Company, 1983.

[100] LABORIE, P., AND GHALLAB, M. Planning with Sharable Resource Constraints. In *Procs. of the International Joint Conference on Artificial Intelligence (IJCAI-95)* (1995).

[101] LEYMANN, F., AND ROLLER, D. Business Process Management with FlowMark. In *Procs. of the IEEE Computer Society International Conference* (1994).

[102] LIBERATORE, P. On the Complexity of Choosing the Branching Literal in DPLL. *Artificial Intelligence, 116*, 1–2 (2000), 315–326.

[103] LITTMAN, M. L., MAJERCIK, S., AND PITASSI, T. Stochastic Boolean satisfiability. *Journal of Automated Reasoning, special issue on Satisfiability.* (2000).

[104] LONG, D., AND FOX, M. Recognizing and Exploiting Generic Types in Planning Domains. In *Procs. of the 6th International Conference on AI Planning and Scheduling, AIPS 2000* (2000).

[105] LYDIARD, T., JARVIS, P., AND DRABBLE, B. Realizing Real Commercial Benefits from Workflow: A Report from the Trenches. In *AAAI-99 Workshop on agent-Based Systems in The Business Context. Brian Drabble and Peter Jarvis Cochairs.* (1999).

[106] MAGNUSSON, M. *Domain Knowledge in TALplanner.* PhD thesis, LiTH-IDA-Ex-02/104, 2003.

[107] MCALLESTER, D. Truth Maintenance. *Procs. of the 8th Nat. Conf. AI,* (1990), 1109–1116.

[108] MCCARTHY, J. Applications of Circumscription to Formalizing Common Sense Knowledge. *Artificial Intelligence, 28* (1986), 89–116.

[109] MCCARTHY, J., AND HAYES, P. J. Some Philosophical Problems from the Standpoint of Artificial Intelligence. *Machine Intelligence, 4* (1969), 463–502.

[110] MCCLUSKEY, L., ALER, R., BORRAJO, D., HASLUM, P., JARVIS, P., AND SCHOLZ, U. Knowledge Engineering for Planning ROADMAP. In *http://scom.hud.ac.uk/planet/roaddir/* (2000).

[111] MCCLUSKEY, T. L., RICHARDSON, N. E., AND SIMPSON, R. M. An Interactive Method for Inducing Operator Descriptions. In *Procs. of AIPS'02 Workshop on Planning for Temporal Domains* (2003), pp. 151–159.

[112] MCCLUSKEY, T. L., SIMPSON, R. M., AND LIU, D. HTN Planning in a Tool-supported Knowledge Engineering Environment. In *Procs. of 13th International Conference on Automated Planning and Scheduling (ICAPS'03)* (2003).

[113] MCDERMOTT, D. PDDL - the planning domain definition language. Tech. rep.

[114] MEDINA-MORA, R. WINOGRAD, T., AND FLORES, P. Action Workflow as the Enterprise Integration Technology. *Bulletin of the Technical Committee on Data Engineering. IEEE Computer Society, 6*, 2 (1993).

[115] MIERI, I. Combining Qualitative and Quantitative Constraints in Temporal Reasoning. *Artificial Intelligence, 87* (1996), 343–385.

[116] MINTON, S. *Learning Search Control Knowledge.* PhD thesis, Carnegie Mellon University, 1988.

[117] MODI, P. J., SHEN, W., AND TAMBE, M. Distributed Constraint Optimization and its Application. Tech. Rep. ISI-TR-509, Information Sciences Institute, 2002.

[118] MOHAN, C. Recent Trends in Workflow Management Products, Standards and Research. In *Procs. NATO Advanced Study Institute (ASI) on Workflow Management Systems and Interoperability* (1997).

[119] MONTANARI, U. Networks of Constraints: Fundamental Properties and Applications to Picture Processing. *Information Sciences, 7* (1974), 95–132.

[120] MUSCETTOLA, N. HSTS: Integrating Planning and Scheduling. In *Intelligent Scheduling*, M. Zweben and M. Fox, Eds. Morgan Kaufmann, 1994.

[121] MUSCETTOLA, N., AND SMITH, B. On-Board Planning for New Millenium Deep Space One Autonomy. In *Procs. of IEEE Aerospace Conference, Snowmass, CO.* (1997).

[122] MUSCETTOLA, N., SMITH, S., CESTA, A., AND D'ALOISI, D. Coordinating Space Telescope Operations in an Integrated Planning and Scheduling Architecture. *IEEE Control Systems 12* (1992), 28–37.

[123] MYERS, K. L., , AND BERRY, P. M. At the Boundary of Workflow and AI. In *AAAI-99 Workshop on agent-Based Systems in The Business Context. Brian Drabble and Peter Jarvis Cochairs.* (1999).

[124] NAU, D., CAO, Y., LOTEM, A., AND MUÑOZ-AVILA, H. SHOP: Simple Hierarchical Ordered Planner. In *Procs. of the IJCAI 1999.* (1999), pp. 968–973.

[125] NAU, D., MUÑOZ-AVILA, H., CAO, Y., LOTEM, A., AND MITCHELL, S. Total-Order Planning with Partially Ordered Subtasks. In *Procs. of the IJCAI' 2001.* (2001).

[126] NGUYEN, X., AND KAMBHAMPATI, S. Reviving Partial Order Planning. In *Procs. of the International Joint Conference on Artificial Intelligence (IJCAI-01).* (2001).

[127] NILSSON, N. *Problem-Solving Methods in Artificial Intelligence.* Ed. Feigenbaum, E. and Feldman, J. McGraw-Hill., 1971.

[128] NUIJTEN, W., AND AARTS, E. A Computational Study of Constraint Satisfaction for Multiple Capacitated Job Shop Scheduling. *European Journal of Operational Research, 90*, 2 (1996), 269–284.

[129] O-PLAN TEAM, . O-Plan: Task Formalism Manual, 1997.

[130] PATTERSON, D. J., AND KAUTZ, H. Auto-Walksat: a Self-Tuning Implementation of Walksat. *Electronic Notes in Discrete Mathematics (ENDM), Elsevier 9* (2001).

[131] PECORA, F., AND CESTA, A. Planning and Scheduling Ingredients for a Multi-Agent System. In *Proceedings of UK PLANSIG02 Workshop, Delft, The Netherlands* (2002).

[132] PEDNAULT, E. ADL: Exploring the Middle Ground between STRIPS and the Situation Calculus. In *Procs. of the 1st Conference on Principles of Knowledge Representation and Reasoning* (1989), pp. 324–332.

[133] PENBERTHY, J. S., AND WELD, D. S. UCPOP: A Sound, Complete, Partial Order Planner for ADL. In *Procs. of the KR-92.* (1992), pp. 103–114.

[134] PENBERTHY, J. S., AND WELD, D. S. Probabilistic Planning with Information Gathering and Contingent Execution. In *Procs of AAAI-94. Seattle, WA* (1994).

[135] PEOT, M. A., AND SMITH, D. E. Conditional non-linear planning. In *Procs. of the 1st International Conference AI Planning Systems. Morgan Kaufmann.* (1992), pp. 189–197.

[136] PLANET. Network home page: European Network of Excellence in AI Planning. In *http://planet.dfki.de/index.html.*

[137] PLANET. Planning and Scheduling Repository. In *http://scom.hud.ac.uk/planet/repository/.*

[138] PLANET. Workflow Management TCU Road Map.

[139] POLLACK, M. E., JOSLIN, D., AND PAOLUCCI, M. Flaw Selection Strategies for Partial-Order Planning. *Journal of Artificial Intelligence Research, 6* (1997), 223–262.

[140] PRYOR, L., AND COLLINS, G. Planning for Contingencies: A Decision-based Approach. In *Journal of Artificial Intelligence Research,* (1996), vol. 4, pp. 287–339.

[141] PUTERMAN, M. *Markov Decision Process: Discrete Stochastic Dynamic Programming.* John Wiley and Sons., 1994.

[142] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. Process Modelling and AI Planning Techniques: A New Approach. In *Second International Workshop on Information Integration and Web-based Applications and Services. IIWAS2000. Yogyakarta.* (2000).

[143] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. Transforming Business Processes Modelling into Planning Problems. *PLANET News Letter. 2* (2000), 10–12.

[144] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. An Artificial Intelligence Planner for Satellites Operations. In *ESA Workshop on On-Board Autonomy. (AG Noordwijk, The Netherlands).* (2001), pp. 233–240.

[145] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. Applying an AI Planner to Schedule Ground Satellites Operations. In *Working notes of the IJCAI'01 Workshop on Planning with Resources. IJCAI Press. Seattle, WA (USA)* (2001), pp. 66–67.

[146] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. Planning Based Generation of Process Models. In *ECP01 Workshop on Planning and Scheduling Technologies in New Methods of Electronic Mobile and Collaborative Work* (2001).

[147] R-MORENO, M. D., BORRAJO, D., AND MEZIAT, D. An AI Tool for Planning Satellite Nominal Operations. In *Workshop on AI Planning and Scheduling for Autonomy in Space Applications, Manchester (U.K.)* (2002).

[148] R-MORENO, M. D., AND KEARNEY, P. Integrating AI Planning with Workflow Management System. *International Journal of Knowledge-Based Systems. Elsevier 15* (2002), 285–291.

[149] R-MORENO, M. D., KEARNEY, P., AND MEZIAT, D. A Case Study: Using Workflow and AI planners. In *Procs. of the 19th Workshop of the UK Planning and Scheduling Special Interest Group. PLANSIG2000. The Open University, UK* (2000).

[150] R-MORENO, M. D., PRIETO, M., , MEZIAT, D., MEDINA, J., AND MARTÍN, C. Control y testeo automático para un instrumento espacial embarcable en satélite. In *CAEPIA01, Oviedo (Spain)* (2001).

[151] R-MORENO, M. D., PRIETO, M., , MEZIAT, D., MEDINA, J., AND MARTÍN, C. Controlling and Testing a Space Instrument by an AI planner. In *Procs. of the International Conference on Enterprise Information Systems (ICEIS-02), Ciudad-Real (Spain)* (2002).

[152] RABIDEAU, G., KNIGHT, R., CHIEN, S., FUKUNAGA, A., AND GOVINDJEE, A. Iterative Repair Planning for Spacecraft Operations in the ASPEN System. In *International Symposium on Artificial Intelligence Robotics and Automation in Space (ISAIRAS), Noordwijk, The Netherlands.* (1999).

[153] REFANIDIS, I., AND VLAHAVAS, I. GRT: A Domain Independent Heuristic for STRIPS Worlds based on Greedy Regression Tables. In *Procs. of the 5th European Conference on Planning (ECP-99)* (1999).

[154] REFANIDIS, I., AND VLAHAVAS, I. SSPOP: A State Space Partial-Order Planner. In *ISAS '99: The 5th International Conference on Information Systems Analysis and Synthesis.* (1999), pp. 240–247.

[155] ROSSI, F., SPERDUTI, A., B., V. K., KHATIB, L., MORRIS, P. H., AND MORRIS, R. A. Learning and Solving Soft Temporal Constraints: An Experimental Study. In *CP 2002.* (2002), pp. 249–263.

[156] RUSSELL, S., AND NORVIG, P. *Artificial Intelligence: A Modern Approach.* Prentice Hall, 1995.

[157] SACERDOTI, E. *A Structure for Plans and Behavior.* American Elsevier, New York, 1977.

[158] SADEH, N. *Look-ahead Techniques for Micro-opportunistic Job-shop Scheduling.* PhD thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1991.

[159] SCHWALB, E., AND VILA, L. Temporal Constraints: A Survey. In *An ICS Technical Report.* (1997).

[160] SELMAN, B., AND KAUTZ, H. Unifying SAT-based and Graph-based Planning. *Procs. of the IJCAI-99.* (1999).

[161] SELMAN, B., KAUTZ, H., AND COHEN, B. Local Search Strategies for Satisfiability Testing. *DIMACS Series in Discrete Mathematics and Theoretical Computer Science, 26* (1996), 521–532.

[162] SELMAN, B., LEVESQUE, H., AND MITCHELL, D. A New Method for Solving Hard Satisfiability Problems. *Procs. of the 10th Nat. Conf. AI* (1996), 440–446.

[163] SHERWOOD, R., ENGELHARDT, B., RABIDEAU, G., CHIEN, S., AND KNIGHT, R. ASPEN User's Guide. Version 2.0. Tech. Rep. D-15482, Jet Propulsion Laboratory, 2000.

[164] SIERRA-ALONSO, A., ALER, R., AND BORRAJO, D. Knowledge-Based Modelling of Processes. In *Working notes of the IJCAI'99 Workshop on Intelligent Workflow and Process Management: The New Frontier for AI in Business.* (1999).

[165] SIMPSON, R. M., MCCLUSKEY, T. L., ZHAO, W., AYLETT, R. S., AND DONIAT, C. GIPO: An Integrated graphical Tool to support Knowledge Engineering in AI Planning. In *Procs. of European Conference on Planning (ECP-2001)* (2001), pp. 445–448.

[166] SMITH, B., MILLAR, W., DUNPHY, J., TUNG, Y. W., NAYAK, P., GAMBLE, E., AND M., C. Validation and Verification of the Remote Agent for Spacecraft Autonomy. In *Procs. of the 1999 IEEE Aerospace Conference.* (1999).

[167] SMITH, D., FRANK, J., AND JÓNSSON, A. Bridging the Gap Between Planning and Scheduling. *Knowledge Engineering Review, 15*, 1 (2000).

[168] SMITH, D., AND WELD, D. Temporal Planning with Mutual Exclusion Reasoning. *IJCAI 1999* (1999).

[169] SMITH, S., AND CHENG, C. Slack-Based Heuristics for Constraint Satisfaction Scheduling. In *Procs. of the 11th National Conference on AI (AAAI-93)* (1993).

[170] SMITH, S. F., AND BECKER, M. An Ontology for Constructing Scheduling Systems. In *Working Notes of 1997 AAAI Symposium on Ontological Engineering. Stanford, CA, (AAAI Press)* (1997).

[171] SMITH, S. F., LASSILA, O., AND BECKER, M. Configurable, Mixed-Initiative Systems for Planning and Scheduling. In *Advanced Planning Technology. Ed. A. Tate. AAAI Press, Menlo Park, CA* (1996).

[172] SOININEN, T., GELLE, E., AND NIEMELA, I. A Fixpoint Definition of Dynamic Constraint Satisfaction. In *Principles and Practice of Constraint Programming* (1999), pp. 419–433.

[173] SRIVASTAVA, B., AND KAMBHAMPATI, R. Scaling up Planning by Teasing out Re-source Scheduling. In *Procs. of the Fifth European Conference on Planning (Durham, United Kingdom)* (1999).

[174] SRIVASTAVA, B., KAMBHAMPATI, R., AND DO, M. B. Planning the Project Management Way: Efficient Planning by Effective Integration of Causal and Resource Reasoning in RealPlan. *Artificial Intelligence, 131* (2001), 73–134.

[175] STADER, J. Results of the Enterprise Project. In *Procs. 16th Int. Conference of the British Computer Society Specialist Group on Expert Systems. Cambridge, UK.* (1996).

[176] TATE, A. INSTERPLAN: A Plan Generation System which can deal with Interactions between Goals. *Machine Intelligence Research Unit, MIP-R-109* (1974).

[177] TATE, A., DRABBLE, B., AND R., K. *O-Plan2: An Open Architecture for Command, Planning, and Control.* Morgan Kauffmann, 1994.

[178] TATE, A., AND WHITER, A. M. Planning with Multiple Resource Constraints and an Application to a Naval Planning Problem. In *First Conference on the Applications of AI* (1984), Denver, Colorado, USA.

[179] VAN BEEK, P., AND CHEN, X. CPLAN: A Constraint Programming Approach to Planning. In *Procs. of the 16th National Conference on Artificial Intelligence,* (1999).

[180] VELOSO, M., CARBONELL, J., PÉREZ, A., BORRAJO, D., FINK, E., AND BLYTHE, J. Integrating Planning and Learning: The PRODIGY Architecture. *Journal of Experimental and Theoretical AI, 7* (1995), 81–120.

[181] VELOSO, M., PÉREZ, A., AND CARBONELL, J. Nonlinear Planning with Parallel Resource Allocation. In *Proceedings of the DARPA Workshop on Innovative Approaches to Planning, Scheduling, and Control* (1990), Morgan Kaufmann, pp. 207–212.

[182] VERE, S. A. Planning in Time: Windows and Durations for Activities and Goals. *IEEE Transactions on Pattern Analysis and Machine Intelligence, pami-5* (1983).

[183] VIDAL, V., AND REGNIER, P. Total Oder Planning is more Efficient than We Thought. In *Procs. of the AAAI-99.* (1999), pp. 591–596.

[184] WANG, X. Constraint-Based Efficient Matching in PRODIGY. Tech. Rep. CMU-CS-92-128, School of Computer Science, 1992.

[185] WARREN, D. H. WARPLAN: A System for Generating Plans. In *Department of Computational Logic Memo No. 76. University of Edinburgh* (1974).

[186] WATKINS, C., AND DAYAN, P. Technical Note: Q-Learning. *Machine Learning, 8* (1992), 279–292.

[187] WELD, D. An Introduction to Least Commitment Planning. *AI Magazine,* (1994).

[188] WELD, D. Recent Advances in AI Planning. *AI Magazine, 20,* 2 (1999), 93–123.

[189] WELD, D., SMITH, D., AND ANDERSON, C. Extending Graphplan to Handle Uncertainty and Sensing Actions. *Procs. of the 15th National Conference on AI* (1998), 897–904.

[190] WILKINS, D. E. Domain Independent Planning Representation and Plan Generation. *Artificial Intelligence, 22* (1984), 269–301.

[191] WILKINS, D. E. *Practical Planning: Extending the Classical AI Planning Paradigm.* Morgan Kaufmann, 1988.

[192] WILLIAMS, B. C., AND NAYAK, P. P. A Model-Based Approach to Reactive Self-Configuring Systems. In *Procs. of AAAI 1996* (1996), pp. 971–978.

[193] WILLIS, J., RABIDEAU, G., AND WILKLOW, C. The Citizen Explorer Scheduling System. In *Procs. of the IEEE Aerospace Conference.* (1999).

[194] WOLFMAN, S., AND WELD, D. The LPSAT Engine and its Application to Resource Planning. *Procs. of the IJCAI-99* (1999).

[195] WORKFLOW MANAGEMENT COALITION.

[196] YANG, Q. *Intelligent in Planning. A Decomposition and Abstraction Based Approach.* Springer, 1997.

[197] YOUNES, H. L., AND SIMMONS, R. G. On the Role of Ground Actions in Refinement Planning. In *Procs. of the Sixth International Conference on Artificial Intelligence Planning and Scheduling Systems,* (2002), pp. 54–61.