Route planning algorithms: Planific@ Project

Carlos Martín García and Gonzalo Martín Ortega
OpenLab

***Abstract* — Planific@ is a route planning project for the city of Madrid (Spain). Its main aim is to develop an intelligence system capable of routing people from one place in the city to any other using the public transport. In order to do this, it is necessary to take into account such things as: time, traffic, user preferences, etc. Before beginning to design the project is necessary to make a comprehensive study of the variety of main known route planning algorithms suitable to be used in this project.**

***Keywords*— ADL, HTN, JSHOP2, JSHOP2GUI, Method, Moviliz@ , Operator, PDDL, Planific@, SHOP, STRIPS**

I. Introduction

NOWADAYS, more people live in cities and much of this population uses public transport every day to move everywhere. It is noticeable that there is a lack of information for the citizen, who does not know all the options available when making a journey.

This leads to distrust and less use of public transport. Unless citizens are sure to know beforehand how to get to a place, they will never go by bus or subway. Also, the lack of information precludes the option of combining these two ways of transport, losing their effectiveness as a whole. People will only combine bus and subway in everyday situations, which are familiar to them.

In turn, the difficulty of calculating times for journeys using public transportation tend to be a handicap when you have to decide on this type of transportation. Madrid. EMT (Empresa Municipal de Transportes) and Metro de Madrid, the public transport companies operating in Madrid, provide trip estimates on its web pages, however there is no connection between them, nor provided through mobile devices that is where, in most cases, the system will be useful for the traveler.

We find that the optimal implementation of this functionality is in mobile device, since the vast majority of people use to carry one on at all times and it is precisely when you are not at home when this functionality is most times needed. For this reason the development will be oriented towards these devices.

The main problem in this Project is how to calculate the optimal route between two locations within a city, taking into account all potential bus and subways routes.

To resolve this problem we must create an algorithm that takes into account all the possibilities offered by public transport in the city of Madrid. This involves planning routes that may include several stages in different modes of transport (subway, bus and walking). It must also take into account the route preferences set by the user such as prices, maximum number of transfers, shortest, cheapest, etc.

The desired result is a novel and very useful product for the citizens who know well the city as well as for tourist people, as it facilitates and encourages the use of public transport in the city. Equally interesting from the technological point of view to giving added value to mobile devices which could become the best city guide.

II. Motivation

In this section we discuss those factors that have influenced in the choice of the development of this project. We could classify the reasons for our decision opted into two main branches: on the one hand we see great interest and potential in the urban transport sector, on which there are many things to do and improve. On the other hand we are interested in logistics field and the route planning algorithms used in this field.

A. International economic and social situation

The world today is under the influence of globalization. There is no developed country that is not clearly influenced. This influence derives, among other things, in a migration to the cities of the population living in rural areas. Consequently there is a growth in these, and is thus an improvement in public transport services.

B. Environment

Another factor to consider is the environment. For many years man has been aware of the influence that society has on the environment. However, until recently, no one talked about global warming, environmental awareness, etc.. Therefore, we conclude that all those issues that help improve the situation of the environment, while encouraging the use of public transport to replace private transport are of great importance.

C. Status of public transport

In large cities, public transport becomes a daily essential element. From the standpoint of government, public transport has become a very complex system, only manageable with information technology.

From the standpoint of people who decide to commute by such cities in their own car we can see that appears a large number of problems such as traffic, parking problems, fuel costs, etc.. A priori, the most immediate and simple solution that could give the user would be using public transport.

However, as we mentioned a few lines above, we have identified some shortcomings in this system, such as poor or even no information that the user has on rates, commuting times, distances in time and transfers, number of stops to destination, etc. Furthermore, the user does not have a system that will efficiently provide the interconnections between different modes of transport, and more importantly, there is currently no system that combines the different transport networks of a city to offer the user the optimal route.
Such claims may be or not met in more or less degree by the government, but reality is that there are deficiencies. While these deficiencies continue to exist, the current public transport users and potential users will try to improve this situation with alternative solutions such as the application we want to accomplish.

D.Technology

All previous motivations that make us challenge the current system of public transport in big cities need some help. But this aid passes inexorably by the application of new technologies.

Today we can find on the market mobile devices with GPS, wifi and high data processing capability. They have touch screens and highly usable interfaces. Furthermore, the development of such applications has been facilitated by the development of very simple to operate SDKs. Other factors to consider are that people are more and more familiar with this type of device, its affordable price and that there is not any application in the market that offers a system for mobile devices that lets you know what is the best option to reach your destination.

III. OBJETIVE

In the following sections we will concentrate on the scheduling algorithm necessary to carry out the project. In particular, we will perform a comprehensive analysis of planning in Artificial Intelligence, the main existing planning algorithms and their characteristics. With this study we will be able to choose the algorithm that best suits our problem. Once located, dig into their characteristics, both technical and operational, with the aim of acquiring the knowledge needed to solve the problems mentioned above.

IV. SPECIFICATIONS LANGUAGES

To achieve efficient planning is so important to have good modeling languages, with good algorithms. The language of STRIPS [1] has conditioned the vast majority of planning work since the early '70s, due to its effective solution to the problem context [1] and his support for the strategies of divide - and - conquer. This section briefly describes this language as well as its two major extensions: ADL [2] and PDDL [3].

A. STRIPS

In artificial intelligence, STRIPS (Stanford Research Institute Problem Solver) is an automated planner developed by Richard Fikes and Nils Nilsson in 1971. The same name was later used to refer to the formal language of the inputs to this planner. This language is the base for most of the languages for expressing automated planning problem instances in use today. This section only describes the language, not the planner.

An operator o O is defined on STRIPS as a tuple (Name, Pre, Eff). Name is the name of the operator and is represented by a syntactic expression of the form or (X1, X2, ..., Xn) where each Xi is a variable symbol is called a parameter of the operator. Pre and Eff are respectively the preconditions and effects of the operator, and are represented by:

• An atomic formula (predicate_name arg1, ... argn), where the predicate describes the type of fact and arguments are symbolic variables that correspond to the parameters of the operator. An atomic formula can also occur if it appears negated the effects of the operator.

• A conjunction of atomic formulas. An action is obtained after replacing all the parameters of an operator for specific values. An action, therefore, is a specific instance of an operator. Pre (a) is a set of facts that represents the preconditions of the action. The effects of the action Eff (a) are the facts that add and remove action. The positive effects are represented as Add (a), and negative effects such as Del (a). The result, therefore, to apply a sequence of actions on a state can be formalized as shown below:

$$result(S,\{\}) = S$$

$$result(S,\{a\}) = \begin{cases} (S - Del(a)) \cup Add(a) & \text{, si } Pre(a) \subseteq S \\ S & \text{, en caso contrario} \end{cases}$$

$$result(S,\{a_1,a_2,\ldots,a_n\}) = result(result(S,\{a_1\}),\{a_2,\ldots a_n\})$$

For convenience, any action can be initiated within a state, but only takes effect if their preconditions are met. If its preconditions are met, the positive effects of the action are added to the state, while the negative effects are eliminated.

A plan P is defined as a sequence of sets of actions applicable (A0, A1, ..., An), and indicates the order in which the actions of these sets will run. If a set of actions Ai contains more than one action, such actions can be executed in parallel. Therefore, if | Ai | = 1 i = 1 ... n says that the plan P is sequential and parallel otherwise. A plan P is a solution to a planning problem if result (I, P) is a state objective, ie, if G ⊆ result (I, P).

B. ADL

Although the STRIPS language is very limited for most complex domains, allows a high degree of enlargement. A major expansion has been carried out is language ADL (Action Description Language [2]). ADL is more expressive than STRIPS and is based on an algebraic model to

characterize the states of the world. The main extensions added are:

• Types: ADL allows assigning types to objects of the problem and the parameters of the operators. This facilitates understanding of problems and reduces the number of predicates (fact types) needed.

• Preconditions and negated goals: ADL can include negated atomic formulas in the preconditions of an operator. Similarly, you can specify negated literals in the goals to represent those who do not want facts in an objective state.

• Disjunctive Preconditions: ADL allows a precondition can be a disjunction of atomic formulas.

• quantified preconditions: preconditions may include quantified formulas, both existential (exists) as universal (forall).

(forall (? v1 • v2 ...) [formula])
(exists (? v1 • v2 ...) [formula])

• Comparisons: ADL introduces a new type of atomic formula in the pre-conditions (= arg1 arg2), which is satisfied when its two arguments are equal. This equality predicate (called Equality) symbols compares variables within an operator.

• Effects conditionals in ADL domain, operators can contain conditional effects (when condition [formula]). Conditional effects have effect only if specified condition is satisfied in the state on implementing the action. Conditional effects are mainly situated in quantified formulas.

These extensions can reduce the number of instantiated actions, because each action is possible to express a wider range of situations. You can take this advantage to improve the efficiency of many planning systems [2].

C. PDDL

ADL has been one of the extensions of STRIPS most used by planners, but not alone. For example, FStrips (Functional STRIPS) is a first-order language, without quantification, working with constants, functions and relational symbols - but not symbols variables - and increases the expressiveness of the language. However, the extent of greatest success has undoubtedly been PDDL (Planning Domain Definition Language [3]). PDDL was developed for the international planning competition in 1998 [McDermott 2000] with the aim of providing a common notation for modeling planning problems and evaluate the performance of the planners. Since its inception, PDDL has been a point of reference as modeling language for the vast majority of planners.

Apart from STRIPS and ADL, PDDL has been influenced by many other formalisms: SIPE-2, Prodigy 4.0, UCMP, Unpop and UCPOP [1]. The PDDL goal is to express the physics of a domain, ie which predicates are, what actions can be performed and what are its effects, without providing any additional knowledge about it. PDDL provides a wide variety of features, among which are:

• Model-based action STRIPS.

• Conditional effects and universal quantification, as proposed in ADL.

• Specification of hierarchical actions. The actions are broken down into sub-actions and sub goals that can contribute to more complex problems.

• Definition of domain axioms. The axioms are logical formulas that establish relationships between things that are satisfied in a state (as opposed to equity, which define relations between successive states).

• Specification of security restrictions. These restrictions allow to define a set of objectives to be met throughout the planning process.

Given the large number of features that PDDL can express virtually any existing planner is able to handle them all. PDDL brings these features into a set of requirements. In this way, planners can quickly check if they can handle a particular domain.

D. PDDL Extensions

One of the main contributions of the competition in 2002 was planning a new version of PDDL language: v2.1 PDDL [4]. The most important characteristics are incorporating the ability to define actions with duration and to describe the effects that time has on stocks. It also modifies the treatment of numeric expressions and to specify, as part of the problem itself, an objective function (called metrics) that establishes the criteria for optimizing the plan.

PDDL v2.1 is organized into the following four levels:

• Level 1: includes propositional and ADL levels of the previous version of PDDL.

• Level 2: establishing a definitive syntax for handling numeric expressions. The numerical expressions are constructed by arithmetic operators and numeric functions. These functions associate numerical values to tuples of objects of the problem. The numerical terms the actions are always comparisons between pairs of numeric expressions, while the effects can modify the values of numerical functions.

• Level 3 makes use of discrete durative actions. Thus, it is possible to indicate the moments during and after the implementation of an action effect occurs.

• Level 4: Allows durative actions that have continuing effects. To model this effect, introduce the symbol # t which represents the continuum over time during the execution of a durative action.

More recently, a new extension called PDDL+ a fifth level. This level allows you to model efficiently the occurrence of events during the execution of a plan. PDDL+ also supports modeling of business processes that are activities that, while they last, cause continuous changes in the values of numerical expressions.

V. CHOICE OF THE SHEDULING ALGORITHM

After analyzing the world of planning in Artificial Intelligence, the main existing algorithms and the requirements for carrying out the main project exposed before, we can reach an important conclusion. The most appropriate algorithm for our problem is JSHOP2. The reasons that led us to this choice are detailed below.

The main difference between SHOP2 and most other HTN planners is that SHOP2 plans tasks in the same order will be executed knowing the current status of each step of the planning process. This reduces the reasoning complexity by removing the large uncertainty degree about the domain and allows to easily incorporate power of expression to SHOP2.

Besides the common HTN methods and operators, the description includes SHOP2 domain axioms, mixed symbolic and numerical conditions and external function calls. The planning process is complete according to Turing, consistent and complete for a typology of planning problems.

Like other HTN planning systems, SHOP2 plans decomposing tasks into subtasks. A key idea in the use of any HTN planner is to design a set of methods that encode standard operating procedures catching several passes techniques for refining a task. Some features of the domain are expressed in a much more natural in a notation that HTN in a stock-based notation.

JSHOP2 is SHOP2 Java implementation. From a global standpoint, it is important to consider the programming language being used in the project. We have decided to use J2EE to develop the main solution, so JSHOP2 becomes the most appropriate algorithm.

VI. SHOP2: SIMPLE HIERARCHICAL ORDERED PLANNER 2

A. Introduction

SHOP2, Simple Hierarchical Ordered Planner 2 [5] is a domain-independent planning system based on Hierarchical Task Network (HTN) planning. In the 2002 International Planning Competition, SHOP2 received one of the top four awards, one of the two awards for distinguished performance. This paper describes some of the characteristics of SHOP2 that enabled it to excel in the competition.

Like its predecessor SHOP, SHOP2 generates the steps of each plan in the same order that those steps will later be executed, so it knows the current state at each step of the planning process. This reduces the complexity of reasoning by eliminating a great deal of uncertainty about the world, thereby making it easy to incorporate substantial expressive power into the planning system. Like SHOP, SHOP2 can do axiomatic inference, mixed symbolic/numeric computations, and calls to external programs.

SHOP2 also has capabilities that go significantly beyond those of SHOP:

• SHOP2 allows tasks and subtasks to be partially ordered; thus plans may interleave subtasks from different tasks. This often makes it possible to specify domain knowledge in a more intuitive manner than was possible in SHOP.

• SHOP2 incorporates many features from PDDL, such as quantifiers and conditional effects.

• If there are alternative ways to satisfy a method's precondition, SHOP2 can sort the alternatives according to a criterion specified in the definition of the method. This gives a convenient way for the author of a planning domain to tell SHOP2 which parts of the search space to explore first. In principle, such a technique could be used with any planner that plans forward from the initial state.

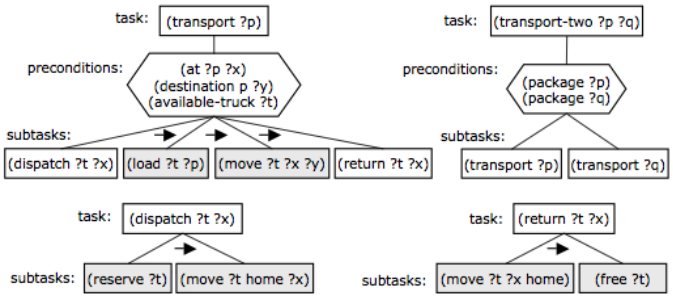
• So that SHOP2 can handle temporal planning domains, we have a way to translate temporal PDDL operators into SHOP2 operators that maintain bookkeeping information for multiple timelines within the current state. In principle, this technique could be used with any non-temporal planner that has sufficient expressive power.

The rest of this paper is organized as follows. Section 2 gives some background on HTN planning, and Section 3 describes SHOP2's features and planning algorithm. Section 4 describes how to write domain descriptions for SHOP2: in particular, Section 4.1 discusses basic problem-solving strategies, and Sections 4.2 and 4.3 describe aspects of SHOP2 that are specific to handling temporal and metric domain features. Section 5 discusses SHOP2's performance in the competition, Section 6 discusses related work, and Section 7 gives a summary and conclusion. Appendix A contains a SHOP2 domain description for one of the problem domains in the planning competition.

B. HTN Planning

HTN planning is like classical AI planning in that each state of the world is represented by a set of atoms, and each action corresponds to a deterministic state transition. However, HTN planners differ from classical AI planners in what they plan for, and how they plan for it.

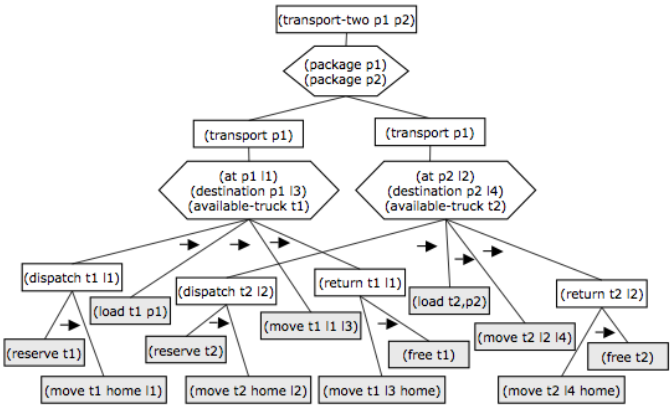
The objective of an HTN planner is to produce a sequence of actions that perform some activity or task. The description of a planning domain includes a set of operators similar to those of classical planning, and also a set of methods, each of which is a prescription for how to decompose a task into subtasks (smaller tasks). Figure below gives a simple example.



Given a planning domain, the description of a planning problem will contain an initial state like that of classical planning—but instead of a goal formula, the problem

specification will contain a partially ordered set of tasks to accomplish.

Planning proceeds by using the methods to decompose tasks recursively into smaller and smaller subtasks, until the planner reaches primitive tasks that can be performed directly using the planning operators. For each nonprimitive task, the planner chooses an applicable method, instantiates it to decompose the task into subtasks, and then chooses and instantiates methods to decompose the subtasks even further, as illustrated in figure below.



If the plan later turns out to be infeasible, the planning system will need to backtrack and try other methods.

HTN methods generally describe the "standard operating procedures" that one would normally use to perform tasks in some domain (e.g., see Figure 1) Most HTN practitioners would argue that such representations are more appropriate for many real-world domains than are classical planning operators, as they better characterize the way that users think about problems.

Like most other HTN planners, SHOP2 is "hantailorable:" its planning engine is domain-independent, but the HTN methods may be domain-specific, and the planner can be customized to work in different problem domains by giving it different sets of HTN methods. The ability to use domain-specific problem-solving knowledge can dramatically improve a planner's performance, and sometimes make the difference between solving a problem in exponential time and solving it in polynomial time. In experimental studies, handtailorable planners have quickly solved planning problems orders of magnitude more complicated than those typically solved by "fully automated" planning systems in which the domain-specific knowledge consists only of the planning operators.

C. JSHOP Characteristics

This section describes SHOP2's planning algorithm and some of SHOP2's distinctive features. The Basic Elements of a Domain Description are:

1) Tasks: A task represents an activity to perform. Syntactically, a task consists of a task symbol followed by a list of arguments. A task may be either primitive or compound. A primitive task is one that is supposed to be accomplished by a planning operator: the task symbol is the name of the planning operator to use, and the task's arguments are the parameters for the operator. A compound task is one that needs to be decomposed into smaller tasks using a method; any method whose head unifies with the task symbol and its arguments may potentially be applicable for decomposing the task. The details are discussed in the following subsections.

2) Operators: Each operator indicates how a primitive task can be performed. The operators are very similar to PDDL operators: each operator o has a head head(o) consisting of the operator's name and a list of parameters, a precondition expression pre(o) indicating what should be true in the current state in order for the operator to be applicable, and a delete list del(o) and add list add(o) giving the operator's negative and positive effects. Like in PDDL, the preconditions and effects may include logical connectives and quantifiers. The operators also can do numeric computations and assignments to local variables. Just as in PDDL, no two operators can have the same name; thus for each primitive task, all applicable actions are instances of the same operator. Each operator also has an optional cost expression (the default value is 1). This expression can be arbitrarily complicated and can use any of the variables that appear in the operator's head and precondition. The cost of a plan is the sum of the costs of the operator instances.

3) Methods: Each method indicates how to decompose a compound task into a partially ordered set of subtasks, each of which can be compound or primitive. The simplest version of a method has three parts: the task for which the method is to be used, the precondition that the current state must satisfy in order for the method to be applicable, and the subtasks that need to be accomplished in order to accomplish that task.

4) Axioms: The precondition of each method or operator may include conjunctions, disjunctions, negations, universal and existential quantifiers, implications, numerical computations, and external function calls. Furthermore, axioms can be used to infer preconditions that are not explicitly asserted in the current state. The axioms are generalized versions of Horn clauses, written in a Lisp-like syntax: for example, (:- head tail) says that head is true if tail is true. The tail of the clause may contain anything that may appear in the precondition of an operator or method.

VII. PLANNING TOOLS: JSHOP2

As we said before, planning was conducted using JSHOP2. JSHOP2, is a planning system based on HTN (Hierarchical Task Network).

A. JSHOP2: Design and Implementation Details

As specified in previous paragraphs, JSHOP2 is based on PDDL. However, it does not recognize the PDDL literally. Therefore, it is used an equivalent PDDL language written in LISP.

The domain is composed of operators, methods, and axioms. Domain components (operators, methods, and axioms) are logical expressions. These logical atoms combine logical expressions in a variety of forms (conjunctions, disjunctions, etc.). The atoms incorporated symbols of predicate logic plus a list of terms. Task lists the problems are composed of atoms of tasks.[6]

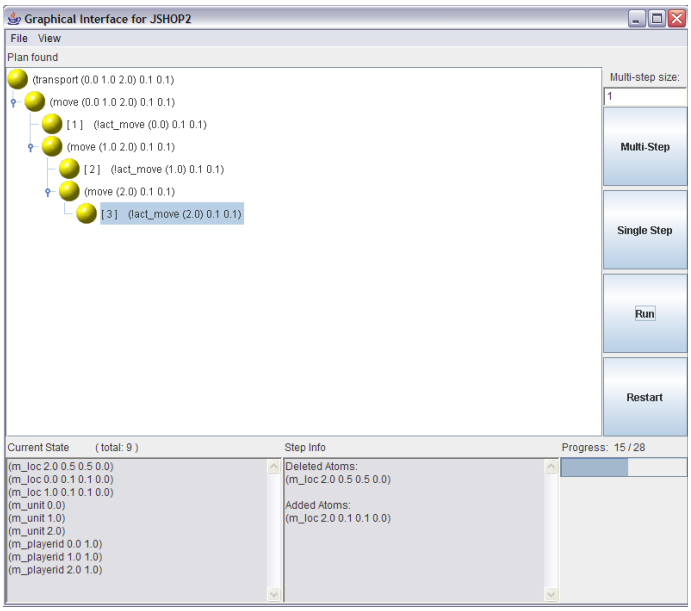
The problem consists of logical atoms (initial state) and a list of tasks (high-level actions).

B. JSHOP2GUI 1.0.1

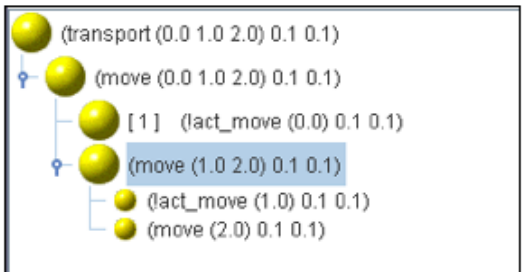
Although the JSHOP2 command line is enough to obtain a planning, it is difficult to analyze the different steps that the planner performs during the planning process. The Graphical User Interface (GUI) for JSHOP2 solves this problem by offering the user a way to analyze the task decomposition tree.

The goal of any GUI is to bring users a fast, easy and intuitive way to work with a program. In this case, the GUI allows you to see graphically the steps that planning is composed of. Its main advantage is that it facilitates analysis of the results and finds possible errors in the algorithm. [7]

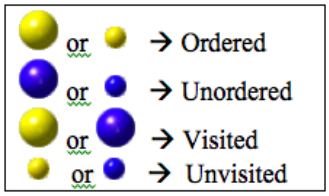
The actions performed by JSHOP2 to achieve a planning are represented by the GUI as a series of steps. These steps can be traversed forwards using the corresponding buttons. The main window of the GUI shows the task decomposition tree. In the information window shows data on the state of the world domain and the action undertaken.



The main window of the GUI shows the task decomposition tree where each node represents a task atom. The task atoms that appear in the tree are always the initial stages of the most high that are used to find the current plan. Nodes can vary in size.

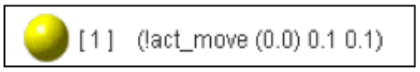


A large node is a visited node, while small ones correspond to those for which has not been yet visited. Furthermore, nodes may be of a different color. Yellow indicates a node that is part of a total order among their brothers. A blue node indicates a unordered task. If the cursor is on any node of the tree, and ordered states visited node will be shown in a popup window.



Leaf nodes have a number surrounded by brackets preceding the name of the atom of its task. Indicates the position of the primitive action in the sequence of actions that create the planning.

Initially it is assumed that each node is a leaf node when it first reached. If the task represented by that node becomes a complex task, the number of the sheet is removed from the task and is decomposed into subtasks.



VIII. PLANIFIC@ PROJECT

So far we have focused on testing the different algorithms and techniques in the route planning field and optimal paths. Once you have decided which suites best to the needs imposed by our project, we will use JSHOP2 based on PDDL. The application that we want to develop is called Planific@ and will be integrated into a bigger and more ambitious project called Moviliz@. The project Moviliz@ is a web and mobile application that guides people through Madrid (Spain) using the public transport of the city.

All HTN-PDDL project consists of two files. One defines the domain and the other the problem.

• In Domain file are defined the methods and operators (or actions) that call the methods.

• In Problem file it could be found objects that are going to take part in the problem, the initial state of all components, as well as the objectives.

A. Domain definition

At the beginning of this document we describes the motivations and features of the project Moviliz@. Now, we are going to define the elements involved in the design of the planner and their interdependencies.

The vehicles needed to carry out the project are bus, subway and on foot.

• **Bus**. The application will consist of a series of buses and lines. Each line consists of several buses that cannot get out of that line. A bus belongs to a single line and a line can have multiple buses.

• **Subway.** The application will consist of a series of trains and lines. Each line consists of several trains that cannot get out of that line. A train is in a single line and a line can take several trains.

• **Foot**. Another option is to travel on foot. A user can move between two points in the city on foot. In most cases where we plan a route, the user's point of origin will not match the bus or subway station near you. It will therefore be necessary to plan a path to walk to the bus stop. Another possible option is to get off at any intermediate stops of the tour, either subway or bus, and walk to another stop to get on other transportation.

• **Lines.** The system consists of several lines for different modes of transport, not for on foot routing. The lines are finite, they have several stops, and can be circular or not. The vehicles move along these lines between different stops.

The lines are the sections in charge of merging two stops. When calculating the ideal route these lines are the key decision. More than lines, the weights associated with each of the stages are the key element. You have to mark each section with weight and a long distance. It will take into account the buses or trains timetable, as well as the time it takes from the street to the corresponding platform. This time is really necessary, especially when making changeovers.

• **Stops.** A stop is the point where a line is accessed. The user must get to that station on foot and wait for the appropriate transport vehicle. Once this vehicle has arrived, the user must upload this vehicle, the vehicle moves to the next stop. The operations are equal for both, bus and subway, so we only need to kind of one type of stop. To determine whether the stop belongs to subway or bus, we can see the route through which we reach this stop.

• **Interchange.** It is possible to change the public transport in the middle of planning. This feature implies a change of line. The change can be performed in two ways:

  • Change on foot: in any of the stops the user can move off and walk to a nearby stops if planner decide that.

  • Interchange: some of the stops of the model are matched with other lines. It is in these places where the user can change the line without needing to shift on foot, but with a penalty. You have to get out of your current vehicle and wait for a new one.

B. Problem definition

The problem, as stated in previous paragraphs, is the file that describes the environment that will be applied to the Domain. That is, the problem initializes the variables with which to work and the relationships that exist between different model components.

It would be necessary to define the graph of the planning application. Make an association between stops, both subway and bus, with their respective relationships, indicating the weights as explained in previous paragraphs. It is necessary to define the relationship between stops and lines, that is, to tell which stop belongs to which particular line. This is necessary because, although we have said that transportation and his stops behave the same way, we must always know the line the user is moving on, in order to guide him/her to the destination correctly.

Having defined the graph on the schedules were implemented, we must define the elements within it. First we must define the position of the moving parts. Buses and trains are in different positions each time so the system has to behold it. In addition, the start position from which the user wants to be indicated varies with each plan. The same applies to the destination position.

IX. DIFFERENT SOLUTIONS

This section sets out a series of practical examples that demonstrate the operation of simplified models. These models let analyze the overall operational of the system by a reduced example that gradually approach the desired solution.

First, the route-planning problem was addressed using SHOP, particularly JSHOP2. JSHOP2 is a planner that obtained a certain impact on the 2002 International Planning Competition because it was able to resolve all problems on the proposed domains and it got one of four first prizes.

But beyond this fact, JSHOP2 is not widespread and only exists a few resolved problems running that those which where proposed in the competition discussed above.

After an arduous research work we accomplished a close approximation of the problem solution. Two necessary files, domain and problem, were written in order to calculate how to get from one point to another using various means of public transport.

The conduct and outcome of this research is collected on the research document attached to this article. Despite the

results obtained using JSHOP2 we decided to take a second track because of the problems that by their very nature caused us when we tried to integrate the route reached with this planner into the Web portal and Google Maps Mashup, the interface from which users will interact with the system.

It was therefore decided to initiate a parallel investigation to resolve the problem. The other idea was to use Dijkstra's algorithm for solving the planning, integrate this algorithm into a web portal created to implement and manage route planning, and finally integrate these planning results in a Google Maps Mashup. [8], [9].

Having explained the reason for both approaches, we proceed to detail them.

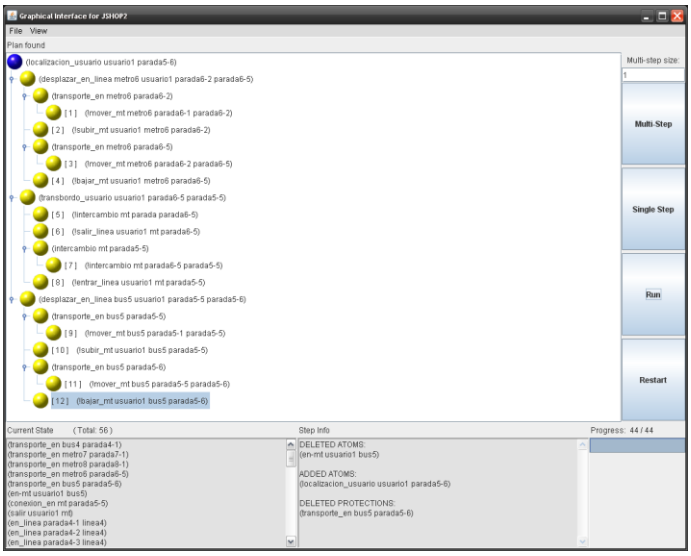
A. JSHOP2 Solution

The definition of the problem domain takes place in the file D_Planifica and comprises methods and operators that specify the tasks and subtasks to perform even a solution to solve a planning problem.

Moreover P_Planifica file contains the specific problem to be solved. In this file were created many problems to solve that will be discussed below. The methods specify tasks and subtasks to be performed. A method in turn can contain several methods to execute if it meets a number of conditions. These methods are included in other subtasks of the task.

The nesting of methods creates a structure of tasks and subtasks that concludes with the execution of primitive operators. These operators, entities of lower levels of abstraction, are responsible for carrying out actions that will resolve the problem. The solution to it is precisely the sequence of all primitive operations are carried out to achieve the objective.

The methods and operators used to define the problem domain that also contain all the search logic of routes are described in the documentation attached to this article. However, below you could find some figures of the execution of and example with some transport lines.



B. Dijkstra – Google Maps Solution

As it was said at the beginning of this section, with the first approach did not obtain the expected results. This is the reason for what we decided to tackle the problem from another point of view. Not the best way to deal with solving a complex problem, as are all combinations of public transportation services at Madrid, but it does provide a good approximation of what would be the end result.

- **Dijkstra**. Dijkstra algorithm is implemented in J2EE, it is integrated into a web portal and the results are displayed in the Google Maps Mashup that is integrated into the web portal.

Dijkstra's algorithm, conceived by Dutch computer scientist Edsger Dijkstra in 1959, is a graph search algorithm that solves the single-source shortest path problem for a graph with nonnegative edge path costs, producing a shortest path tree. This algorithm is often used in routing. An equivalent algorithm was developed by Edward F. Moore in 1957.

For a given source vertex (node) in the graph, the algorithm finds the path with lowest cost (i.e. the shortest path) between that vertex and every other vertex.

It can also be used for finding costs of shortest paths from a single vertex to a single destination vertex by stopping the algorithm once the shortest path to the destination vertex has been determined. For example, if the vertices of the graph represent cities and edge path costs represent driving distances between pairs of cities connected by a direct road, Dijkstra's algorithm can be used to find the shortest route between one city and all other cities. As a result, the shortest path first is widely used in network routing protocols, most notably IS-IS and OSPF (Open Shortest Path First). [8]

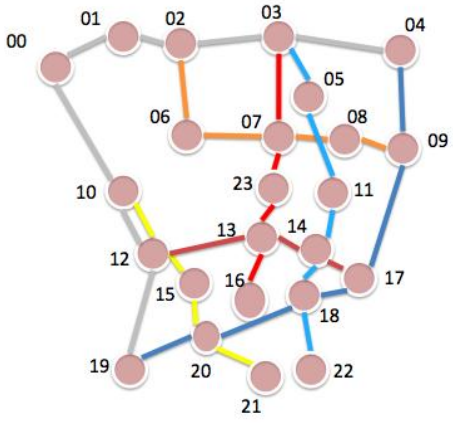
• **Google Maps.** The Google Maps API allows you to embed Google Maps in your own web pages with JavaScript. The API provides several utilities for manipulating maps and adding content to the map using various services, allowing you to create robust maps applications on your site.

Google Maps is a GIS application by Internet company Google, which its potential for this type of project is very high. Therefore, we have decided to show route planning performed by Dijkstra's algorithm using this API. Moreover, this application will be integrated into a web portal as a Mashup. [9]

The possibilities offered by the Google Maps API are many, but not being the main objective of this work we will not deepen them. However, if you want to do more research on the subject, this link may be helpful: http://code.google.com/intl/es-ES/apis/maps/

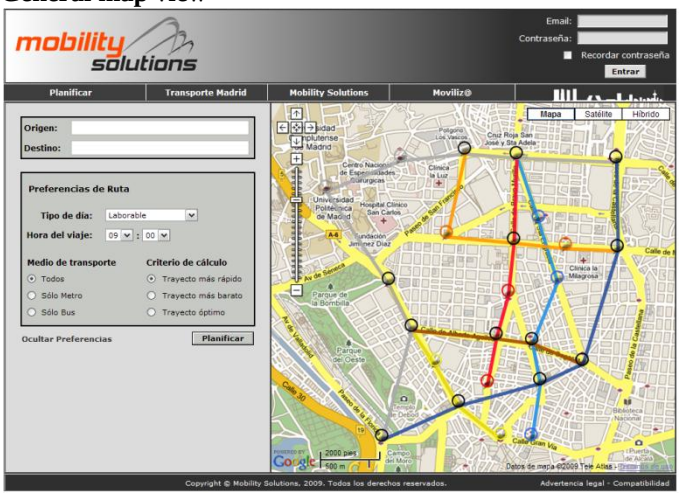
• **Planific@.** In order to simplify the comprehension of the whole project we are going to limit the public transport of Madrid to the following graph.

The same graph in a schematic way will be as follow:

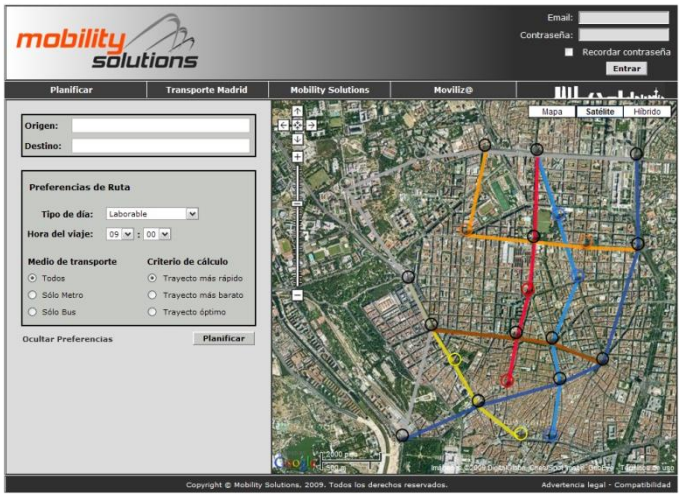


More details about implementation and web portal integration could be found at the attached document. However, below you could find some images of the final solution.

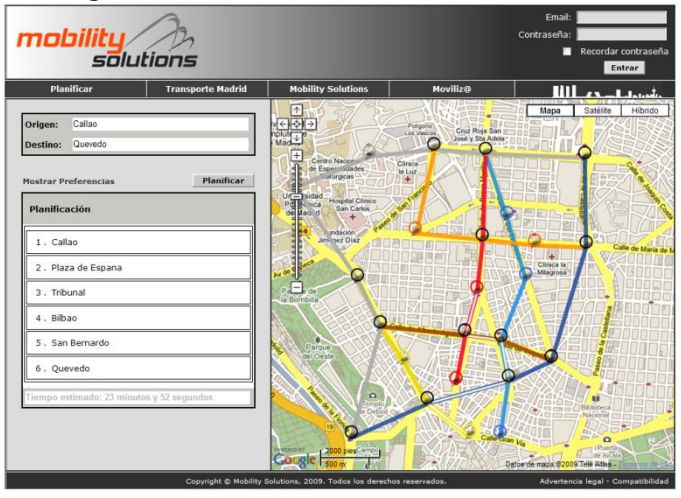
General map view



General Satellite view



Planning view



X. Conclusion

This paper shows the investigation study carried out about the different ways to perform route planning. After searching all possible solutions to resolve this problem we could say that HTN-PDDL and JSHOP2 is apparently the best way to reach the objective proposed based on the studied algorithms. So we started to design the PDDL domain that represented the proposed problem and it was seemingly easy and powerful.

However, it has to be noted that in PDDL there is a clear distinction between the description of parameterized actions that characterize the behavior of the domain and descriptions of specific objects, initial conditions and goals that characterize a particular problem. While, PDDL domain description is robust enough to model the behavior, we think that language has certain shortcomings when is needed to express the initial state and the objectives to be achieved in certain types of problems, for example, PDDL is unable to indicate a multiple initial state that allows not only minimize the objective function.

Moreover the language is not flexible when it necessary to instantiate the predicates and functions describing the initial state of a problem, which is quite tedious and impractical, especially when defining the problems as our case in which initialization involves a large number of nodes. It would be appreciate a reference to instances of more generic predicate to avoid having to make an exhaustive list. As for the tool used, JSHOP2, worth mentioning that the abstraction in which we must write the file and problem domain is based on LISP. This implies that clarity offered by PDDL defining operators, methods and designing the problem is lost, making more complicated to design the solution. We cannot fail to mention the great help that involved the use of graphical user interface used, JSHOP2GUI 1.0.1, which allowed us to do a larger number of tests because of its ease of use and the ease of debugging code.

Another problem with JSHOP2 was the great difficulty that we found to integrate the route reached with this planner into the Web portal and Google Maps Mashup, the interface from which users will interact with the system. After seeing all this difficulties that we find in this approximation to the solution, we tried a second via to solve the problem. We think that the best way to accomplish this is by using the Dijkstra algorithm written in J2EE, the same platform used for the web portal. It was easier to integrate the planning results with the Google Maps Mashup and these results were efficient and accurate enough.

As a final conclusion, we would like to say that throughout this investigation we have noticed that PDDL is a very powerful tool in order to write planning domains, but there is no planner that use it directly, transferring the domain information in the same language. We think that it would be very interesting to research on planners that use all the capacity that it owns. JSHOP2 is an ambitious academic project with great potential, but nowadays it is quite difficult to solve a problem that demands some complexity with this planner.

References

[1] Okhtay Ilghami; Dana S. Nau.. A general approach to synthesize problem- specific planners. Technical Report CS-TR-4597, Department of Computer Science, University of Maryland. USA. Oct 2003.

[2] Dana S. Nau; Muñoz-Avila, Héctor; Cao, Y; Lotem, A; and S. Mitchell. Aug 2001. "Total-order planning with partially ordered subtasks". In Proceedings of the Seventeenth International Joint Conference on Artificial Intel ligence, Seattle, WA, August 2001.

[3] Nau, Dana. "Simple Hierarchical Ordered Planner. SHOP Automated Planning." . University of Maryland. USA. 17 Feb 2005 [Online]. Available: http://www.cs.umd.edu/projects/shop/

[4] D. Nau, T.-C; Au, O; Ilghami, U; Kuter, J. W; Murdock, D; Wu, and F; Yaman. "SHOP2: An HTN planning system. Journal of Artificial Intelligence Research". Dec 2003.

[5] D. S. Nau, Y. Cao, A. Lotem, and H. Muñoz-Avila. "SHOP: Simple hierarchical ordered planner". Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), Morgan Kaufmann Publishers. 06 Aug 1999.

[6] Ilghami, Okhtay. "Documentation for JSHOP2". 24 Aug 2009. [Online]. Available: http:/ourceforge.net/projects/shop

[7] Shin, J. "A Graphical Interface for JSHOP2". Department of Computer Science University of Maryland, College Pak. Jun 2006

[8] Wikipedia. "Dijkstra Algorithm". Aug 2009. [Online]. Available: http://en.wikipedia.org/wiki/Dijkstra's_algorithm

[9] Google Map. "Google Maps API". Aug 2009 [Online]. Available: http://code.google.com/intl/en/apis/maps/

C. Martín García, Madrid, 1984, 5th Year Computing Engineering Student at Universidad Pontificia de Salamanca, Madrid, Spain. e-mail: martin.g.carlos@gmail.com

G. Martín Ortega, Madrid, 1982, 5th Year Computing Engineering Student at Universidad Pontificia de Salamanca, Madrid, Spain. e-mail: gonzalo.martinortega@gmail.com