Expert Systems with Applications xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Expert Systems with Applications

journal homepage: www.elsevier.com/locate/eswa

A visual programming system for automated problem solving

Ourania Hatzi [a,*], Dimitris Vrakas [b], Nick Bassiliades [b], Dimosthenis Anagnostopoulos [a], Ioannis Vlahavas [b]

[a] Department of Informatics and Telematics, Harokopio University of Athens, 89, Harokopou str, Kallithea, Athens 17671, Greece
[b] Department of Informatics, Aristotle University of Thessaloniki, Thessaloniki 54124, Greece

ARTICLE INFO

Keywords:
Automated planning
Visual interfaces
Knowledge engineering
Semantic web service composition

ABSTRACT

Although new AI planning algorithms and techniques are being developed and improved rapidly, there is a lack of efficient and easy to use systems able to incorporate and utilize them. Furthermore, while visual representation facilitates design, maintenance and comprehension of planning domains and problems, very few systems incorporate it. This paper presents VLEPPO, an integrated system aiming at visually modeling planning domains and problems through a convenient graphical interface, while maintaining compatibility with the Planning Domain Definition Language (PDDL), with import and export features. Solutions to planning problems can be obtained by invoking different planners employing the web services technology. The demonstration of the system is performed through a case study involving web service composition viewed as a planning problem.

1. Introduction

Planning has been an active research area since the early days of AI (Weld, 1999), and its applications have been extremely useful in cases when agents act in dynamic environments. Several formalisms for the representation of planning domains and problems have emerged over time, among which Planning Domain Definition Language (PDDL) (Ghallab et al., 1998) has dominated and become a standard. Over the same period of time, numerous algorithms, methods and techniques have been proposed (Hendler, Tate, & Drummond, 1990).

Most of the algorithms and techniques developed so far mainly focus on improving the efficiency of planning systems in terms of required time and resources. Although the results are in many cases impressive, there are not many successful examples of planning systems adapting to industrial use. This can be partly attributed to technical reasons such as the tendency of existing planning systems to be bound to a specific algorithm which might be overwhelmed by the amount of data they have to deal with, as it is becoming more and more immense. This brings on the requirement for systems that have the ability to incorporate multiple state-of-the-art algorithms before they are rendered outdated with respect to current problem demands. Therefore, there is a need not only for more efficient and scalable algorithms, but also for systems that, instead of being limited to a single algorithm, are flexile enough to adapt and employ new algorithms and techniques.

Furthermore, another important issue is the lack of systems that facilitate the development and deployment of planning domains and problems. Modelling and encoding them in some definition language such as PDDL, necessary as it is, can be tiresome and error-prone. It should not be assumed that the designer will always be a planning expert, and even in such cases, visual interfaces can save a lot of time and effort required, while reducing the number of errors and prevent inconsistencies. Visual interfaces in planning systems liberate the designers from the requirement to pay attention to syntax and facilitate focusing on the semantics and structure of the problem at hand.

This paper presents an attempt to approach the design and solving of planning problems while taking into account the aforementioned issues. The result of this research was the development of VLEPPO (Visual Language for Enhanced Planning Problem Orchestration), which is an integrated system for visually modeling and solving planning problems that aims at:

- offering a convenient and intuitive graphical interface which simplifies modeling, facilitates maintenance and promotes understanding of planning domains, even for non-expert users;
- conforming to the current standards for domain and problem representation, such as PDDL, thus facilitating communication with planning systems that comply with this standard;
- providing a high degree of flexibility in integration of different planning algorithms on top of the local one by employing the current web services technology.

The VLEPPO system provides interoperability with the PDDL language, as it can import domains and problems from PDDL files

* Corresponding author. Tel.: +30 2109549288; fax: +30 2109549281.
E-mail addresses: raniah@hua.gr (O. Hatzi), dvrakas@csd.auth.gr (D. Vrakas), nbassili@csd.auth.gr (N. Bassiliades), dimosthe@hua.gr (D. Anagnostopoulos), vlahavas@csd.auth.gr (I. Vlahavas).

0957-4174/$ - see front matter © 2009 Elsevier Ltd. All rights reserved.
doi:10.1016/j.eswa.2009.12.047

for visualization, validation and maintenance purposes and at the same time export the designed or modified domains and problems to PDDL. Furthermore, it offers separate but interoperable editors for planning domains and problems facilitating modular design and definition of new problems corresponding to existing planning domains. Moreover, as far as planners are concerned, the web services technology enables the proposed system to communicate with planners implemented or wrapped and deployed as web services; therefore, the user is not limited to a single planning algorithm, while the implementation language and platform of the planning algorithms is not a restriction.

The rest of the paper is organized as follows: Section 2 describes some formalisms and standards used for the representation of planning domains and problems, while Section 3 gives an overview of VLEPPO and its architecture. Sections 4–6 elaborate on the features of the system, while a case study involving web service composition is described in Section 7. Related work in the area is presented in Section 8, and finally, Section 9 concludes the paper and poses future directions.

2. Representation of planning domains and problems

This section describes the most prominent formalisms and standards used for the representation of planning domains and problems, namely STRIPS and PDDL. The approach described in this work evolves around these formalisms, and particularly around PDDL, as clear correspondences exist between elements of the system and PDDL elements. In addition, PDDL is used in the VLEPPO system when importing and exporting domains and problems as a means of interoperability with other planning systems.

2.1. The STRIPS formalism

The STRIPS (Stanford Research Institute Planning System) formalism comprises the base for the representation of planning domains and problems in most classical planning systems (Fikes & Nilsson, 1971). A planning problem in STRIPS is a tuple $\langle I, A, G\rangle$, where I is the initial state, A is a set of available actions that can be used to modify states, and G is a set of goals. States are represented as sets of predicates.

Each action A_i has three lists of facts containing:

(a) the preconditions of A_i (noted as $prec(A_i)$);
(b) the facts that are added to the state (noted as $add(A_i)$) and
(c) the facts that are deleted from the state (noted as $del(A_i)$).

The following formulae hold for the states in the STRIPS notation:

- An action A_i is applicable to a state S if $prec(A_i) \subseteq S$.
- If A_i is applied to S, the successor state S' is calculated as: $S' = S/del(A_i) \cup add(A_i)$.

The solution to such a problem is a sequence of actions, which if applied to I leads to a state S' such as $S' \supseteq G$.

Usually, in the description of domains, action schemas (also called operators) are used instead of actions. Action schemas contain variables that can be instantiated using the available objects and this makes the encoding of the domain easier.

2.2. The PDDL definition language

Planning Domain Definition Language (PDDL) was initially designed for providing a standard means of encoding planning domains and problems used as input test sets for planners that took part in planning competitions such as AIPS (AIPS, 2000) and IPC (IPC, 2004; IPC-5). However, it has since been enhanced, extended and become a standard in the planning community for modeling planning domains. The importance of PDDL lies in the fact that it is established as the most widely acceptable common language for the exchange of information for planning domains and problems between researchers, while the majority of the state-of-the-art planners use it as input and output language (Gerevini, Saetti, & Serina, 2005; Hsu, Wah, Huang, & Chen, 2006; PDDL4J, 2008; SATPLAN, 2004).

PDDL (Ghallab et al., 1998) mainly focuses on expressing the physical properties of the domain under consideration in each planning problem. In order to facilitate the development of PDDL-compliant planners which are not obliged to cover the entire language, its features have been formally divided into subsets referred to as *requirements*; therefore, each domain definition declares which requirements will be put into effect, and planners can determine whether the domain can be handled. In the following paragraphs, in order to provide an overview of the language, the most commonly used definition elements are explained briefly.

When types are used, each object that appears in a problem is declared to be of a certain type. Consequently, the arguments of the predicates must also be of a certain type, which permits domain validation. A *type* can be thought of as a timeless unary predicate, and, in cases typing is not used, unary predicates are in fact used to compensate for the lack of types.

Variables have the same semantics as in any other definition language, and are used in conjunction with built-in functions for expression evaluation. In more recent versions of PDDL, *fluents* seem to gain momentum instead of variables when there is a need for values that can change over time, as a result of an action. *Constants* represent objects that do not change values over time and can be used in domain operators definitions or problems associated with a domain.

Relations between objects in the domain are represented by *predicates*. A predicate may have an arbitrary number of arguments, whose ordering is important in PDDL. Predicates are used to describe the state of the world at a specific moment and as preconditions and results of an action. *Timeless predicates* are predicates that are considered to be true at all times, therefore they are cannot be affected by actions.

Actions enable transitions between successive situations. An action declaration mentions the parameters and variables involved, the preconditions that must hold for the action to be applied and the results of an action (effects). The effects, according to the STRIPS formalism, include the predicates that will be added to the world state and the predicates that will be removed from the world state after the application of the action. Predicates that are not mentioned among the action effects are assumed to stay unchanged from this action.

Axioms, in contrast to actions, state relationships among propositions that hold within the *same* situation. The necessity of axioms arises from the fact that the action definitions do not mention all the changes in all predicates that might be affected by an action; therefore, additional predicates are concluded by axioms after the application of each action. Axioms were renamed to *derived predicates* in later versions.

Safety constraints in PDDL are background goals which may be broken during the planning process, but ultimately they must be restored. Constraint violations present in the initial situation do not require to be fulfilled by the planner.

After having defined a planning domain, problems can be defined with respect to it. A problem definition in PDDL must state an initial situation and a final situation, referred to as *goal*, by specifying the predicates that comprise them. The closed world assumption holds; therefore, all predicates which are not explicitly defined to be true in the initial state are assumed to be false. The

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

solution given to a problem is a sequence of actions which can be applied to the initial situation, eventually producing the situation stated by the goal description.

PDDL 2.1 (Fox & Long, 2003) was developed by the necessity to express temporal and numeric properties of planning domains. The first of the extensions introduced were *numeric expressions*, which introduced new elements to the language. *Functions* are now part of domain definition, which associate a number of objects with an arithmetic value. Moreover, conditions were introduced, which are actually comparisons between pairs of numeric expressions. Finally, assignment operations are possible, with the use of built-in assignment operators such as *assign*, *increase* and *decrease*. The actual value for each combination of objects is not stated in the domain definition but must be provided to the planner in the problem definition.

Other extensions in this version include *durative actions*, which, in contrast to instantaneous actions considered up to now, have some duration. This duration must be declared at their definition, while temporal annotations are introduced to their conditions and effects.

PDDL 2.2 (Edelkamp & Hoffmann, 2004) introduced *timed initial literals*, which are facts that become true or false at certain points in time known to the planner beforehand, independently of the actions chosen to be carried out by the planner.

In PDDL 3.0 (Gerevini & Long, 2005) the language was enhanced with constructs that increase its expressive power regarding the plan quality specification. The constraints and goals are divided into strong, which must be satisfied by the solution, and soft, which may not be satisfied, but are desired. In addition, the notion of *plan trajectories* is introduced, which allows the specification of intermediate states that a solution has to reach, before it reaches the final state.

3. Overview and architecture of the system

VLEPPO is an integrated system intended to facilitate modeling and solving of planning problems. Among its key features is a convenient, intuitive and easy-to-use graphical interface, which allows design, comprehension and maintenance of planning domains and corresponding problems. The system accommodates for modularity, as domains and problems can be designed separately, and compatibility with standards, as most visual elements present in the system correspond to PDDL elements. Compliance with the PDDL standard is also achieved through the import and export features. Another important characteristic of the system is the increased flexibility in integrating the planning system that will be exploited each time when attempting to acquire a solution to a specific planning problem. This is accomplished by employing the current technology of web services. The system was implemented in Java for portability and interoperability purposes.

The full interface of the system during the design of a domain and a corresponding problem is depicted in Fig. 1. The main window includes the Ontology Editor and the Operators Editor, the combined use of which accomplishes modeling of a planning domain, as well as the Problem Editor where corresponding planning problems are designed.

The architecture of the system is presented in Fig. 2. VLEPPO comprises of four components, namely the Visual, the Planning, the Import/Load and the Export/Save Components.

A planning domain is reflected in the Visual Component using graphical notations corresponding to various PDDL elements, such as predicates and operators. Similarly, planning problems based on this domain can also be visualized. During visual design, the system guides the designer in order to ensure consistency in the resulting domains, as it performs real-time validity checks. The key feature of the Visual Component is its simplicity and convenience; at the same time, a high degree of correspondence to PDDL is achieved (Hatzi, Vrakas, Bassiliades, Anagnostopoulos, & Vlahavas, 2007a). The range of the PDDL elements that can be represented in the system is quite wide, and covers the elements that are used more frequently in contemporary planning domains and problems. Specifically, the PDDL requirements (Gerevini & Long, 2005) covered by the system are the following: :strips, :typing, :negative-preconditions, :fluents, :durative-actions, :derived-predicates and :timed-initial-literals. The Visual Component also serves plan visualization purposes, for plans that comply with the PDDL+ standard (Fox & Long, 2002).

The Import/Load and Export/Save Components provide additional features which ensure interoperability and conformation to the standard. The features include exporting the domains and problems to PDDL, as well as importing existing domains and problems from PDDL for visualization, modification and maintenance purposes. Therefore, the designer is enabled to manipulate existing domains and problems in an intuitive way, even if they are not familiar with the language syntax. At the same time, save and load

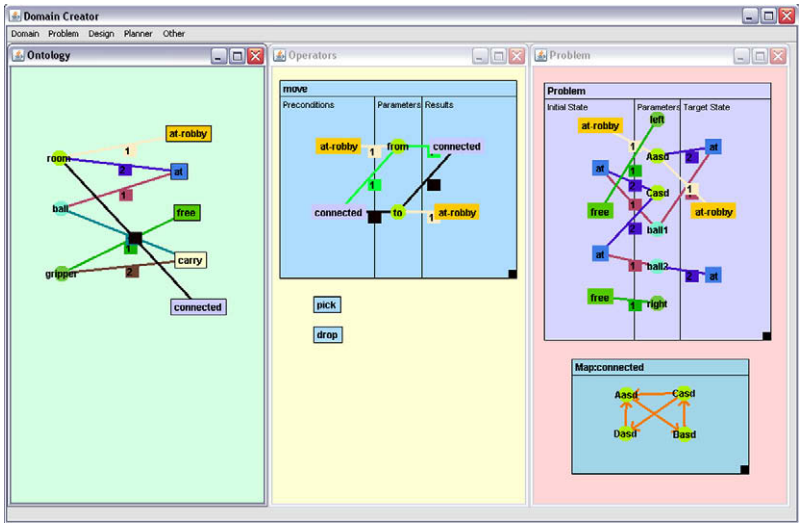

Fig. 1. A full screenshot of the system interface.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

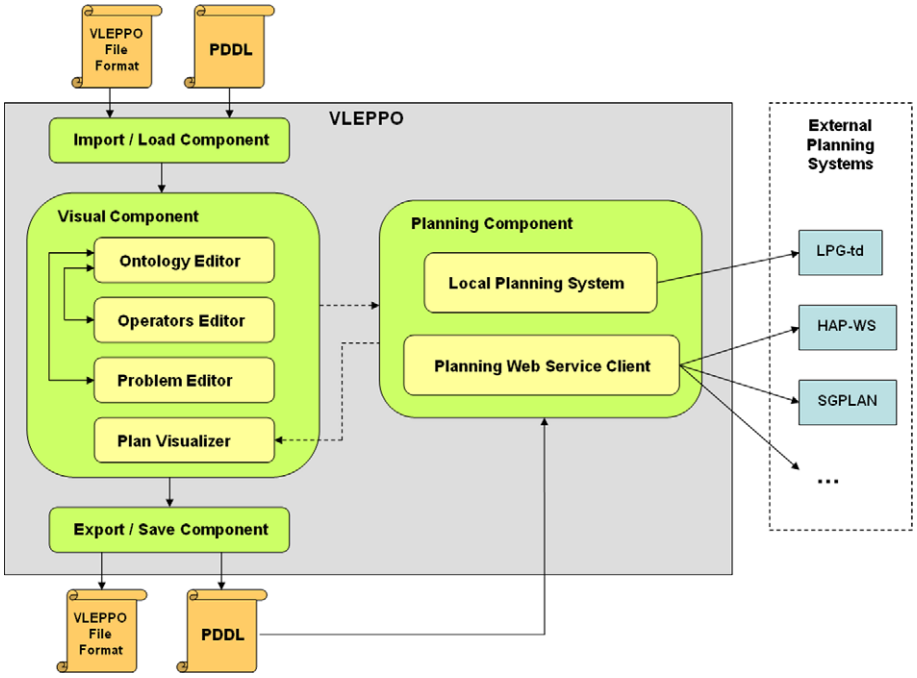

Fig. 2. The architecture of VLEPPO.

functions for both domains and problems can be alternatively used, which, unlike export and import functions, have the ability to preserve visual information such as colors and positions of the elements, even for domains that are under development.

Finally, in order to accommodate for interoperability with other planning systems that actually perform the planning procedure, the Planning Component has been developed, which accommodates cooperation with external planners. This includes a web service client component which offers the user the chance to select among planners that are implemented or wrapped and deployed as web services. Thus, the user is not restricted to a single planning algorithm, but has the ability to experiment with planners and can even attempt to solve the problem with many planners simultaneously, as the planning procedure is not executed locally. Furthermore, in order to avoid considering an internet connection mandatory for the planning operation of the system, another planning component which solves the problem locally has also been incorporated.

The full implementation code of the VLEPPO system complemented with example PDDL files can be downloaded from http://www.dit.hua.gr/~raniah/vleppo_en.html.

4. Visual design of planning problems with VLEPPO

This section provides a description of the visual elements offered by the system for modeling planning domains and problems, while their correspondence to PDDL elements explained in Section 2.2 is pointed out. The features of the system are described using well-known planning examples.

4.1. The domain entities and relationships

The VLEPPO system employs the well-known formalism of the entity-relationship model, as seen through the PDDL standard, in order to describe the structure of the planning domains; therefore, the classes in a planning domain are the entities, while the predicates are the relationships. These elements are visually represented in the system by distinctive shapes and connections between them.

A *class* in PDDL, denoted by a colored circle, represents a type of domain objects or action parameters. Based on a class, corresponding operator parameters and problem objects can be created, bearing the same color for instantaneous identification.

A *relationship*, denoted by a rectangle, corresponds to a domain predicate in PDDL and is used to describe associations between classes, or properties of classes, in the unary predicate case. As with classes, the color of the relationship can be used to track the occurrences of the predicate in the operators or problems.

Relationships are associated with classes through *connections*. Each connection represents an argument of a relationship, while the class shows the type of the argument. A relationship may have an arbitrary number of arguments of any type, which are increasingly ordered upon creation.

As an example, consider a very common domain in the planning community, the *Gripper* domain (McDermott, 1998). In this domain there is a robot with N grippers which moves in a space composed of K rooms that are all connected with each other so that the robot can reach any room from any other room with one single movement. There are also L numbered balls which the robot can carry from their initial positions to some destination, holding one ball in each gripper at any given time. The Gripper domain has four relationships: *at*, which specifies the position of a ball in a room, *holding*, which is used to denote that some gripper is holding a ball, *at-robot*, which specifies the position of the robot, and *empty*, which states that a gripper is not holding any ball. The classes and relationships of the domain, along with their connections are depicted in Fig. 3.

The aforementioned elements – classes, relationships and connections – combined together, form the entity – relationship model of the data for the planning domain the user is dealing with, which is visually represented in the Ontology Editor. Although there are other formalisms available, such as UML (UML), to describe the structure of data in a domain, this model was estimated to approach best the PDDL structure.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

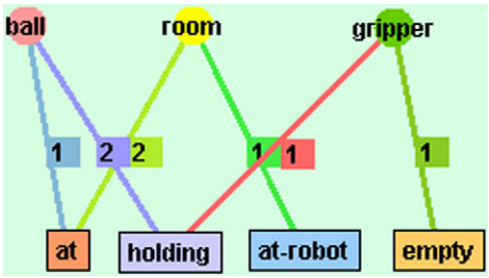

Fig. 3. The relationships in the Gripper domain.

4.2. Representing operators

Operators in VLEPPO have a direct correspondence to PDDL actions. The essential elements of their definition are the preconditions, the results (or the add/delete lists), and their parameters, each one depicted visually in a separate column of the operator shape in the Operators Editor. Preconditions and results can be created from predicates while parameters can be created from classes. Connections can also be created provided that there are corresponding connections in the Ontology Editor. Other elements that can be imported in operators will be discussed in more detail in the paragraph about advanced features.

The Gripper domain involves three operators, shown in Fig. 4: *move*, which enables the robot to move between rooms, *pick*, which allows a gripper to lift and hold a ball, and *drop*, which is the opposite of pick and is used when a ball is no longer held by a gripper.

The default view for an operator is in *preconditions/results view*, when the column on the left shows the preconditions that must hold for the action to be executed and the column on the right shows the state of the world after the execution of the action (in terms of predicates affected by this action). However, in some cases this is not convenient or desirable; therefore, the system offers another option, closer to the PDDL definition of operators, which is the *add/delete lists view*. If selected, the column on the right changes to show the facts that will be added and deleted from the current state of the world upon the application of the action. The user can switch between the two views at any time, however changes to the operators are allowed only in the preconditions/results view. For demonstration purposes, the two views for the *pick* operator are shown in Fig. 5.

Moreover, the system supports operators that are supposed to have duration, referred to as *durative actions* in PDDL. The defini-

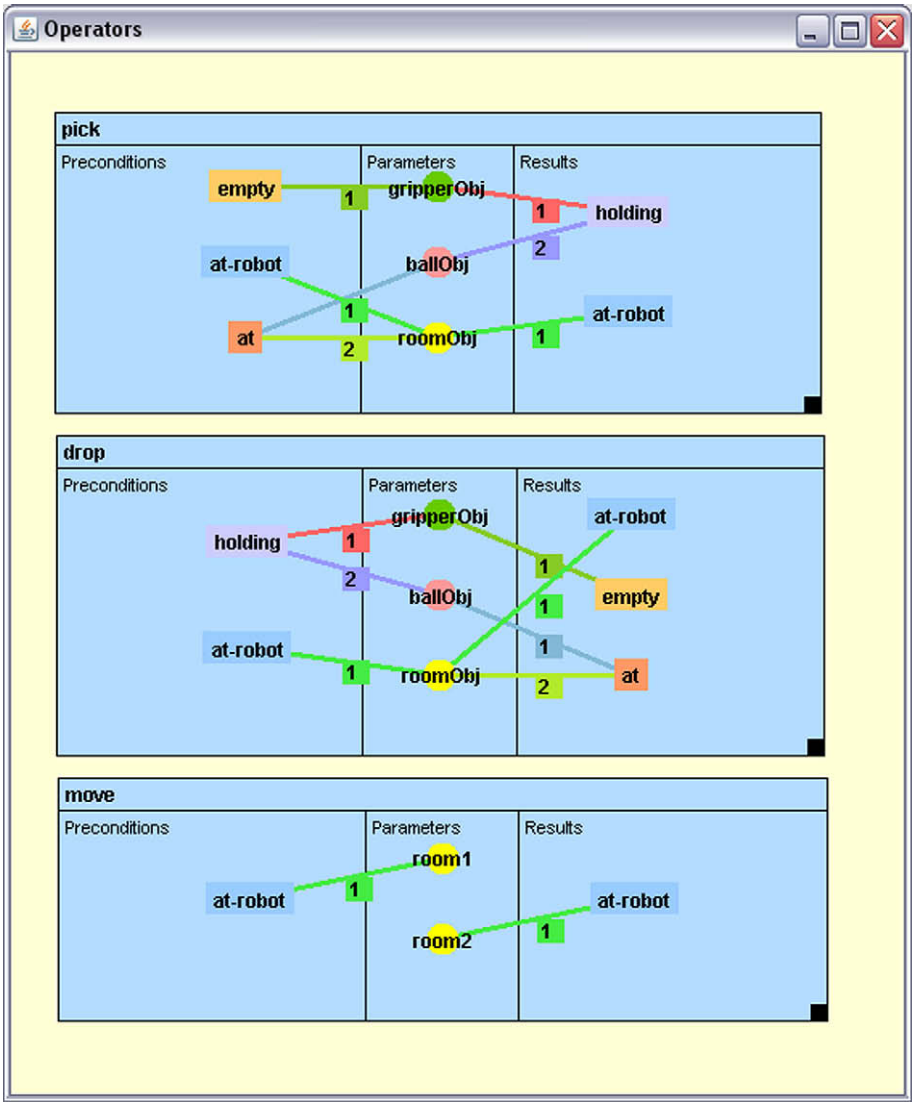

Fig. 4. The operators in the Gripper domain.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

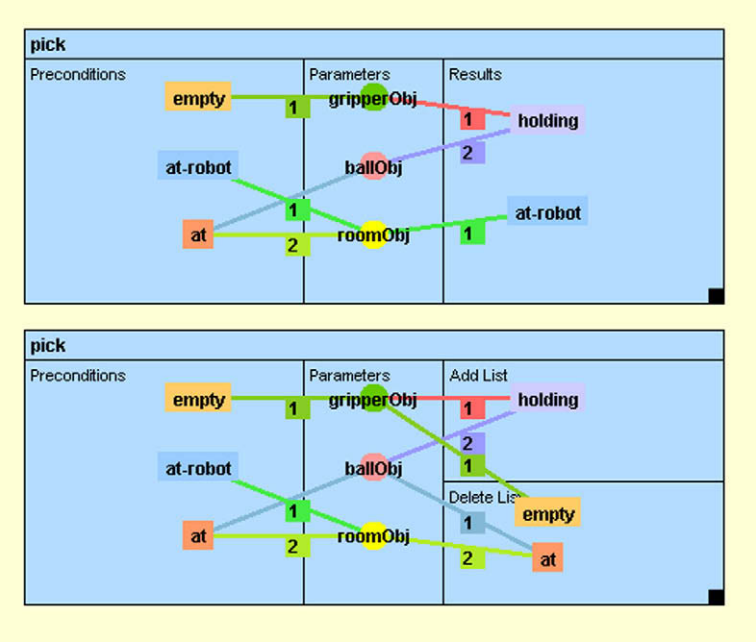

Fig. 5. The two views for the *pick* operator in the Gripper domain.

tion of such a durative action includes setting the duration of an operator, in combination with temporal annotations to the preconditions and results. The durative view of an operator, along with the dialogues for the definition of temporal annotations are depicted in Fig. 6.

4.3. Representing problems

For every domain defined in PDDL, a large number of problems that correspond to this domain can be defined by describing an initial and a goal state. The problem shape in the visual system is much like the three-column operator in form, but different semantically. The left column holds the predicates in the initial state, the right column holds the predicates in the goal state, and the middle column holds the objects that take part in the problem definition. As with operators, the aforementioned elements can be created from the corresponding elements in the Ontology Editor.

An example of a problem for the Gripper domain is presented in Fig. 7. In this instance, there are two rooms (Bedroom, Kitchen), one ball (Ball1) and the robot has two grippers (rightGripper, leftGripper). The initial state of the problem defines the location of the robot and the ball, which are the Kitchen and Bedroom, respectively, and that both grippers are free. The goal state requires that the destination of both the ball and the robot is the Kitchen.

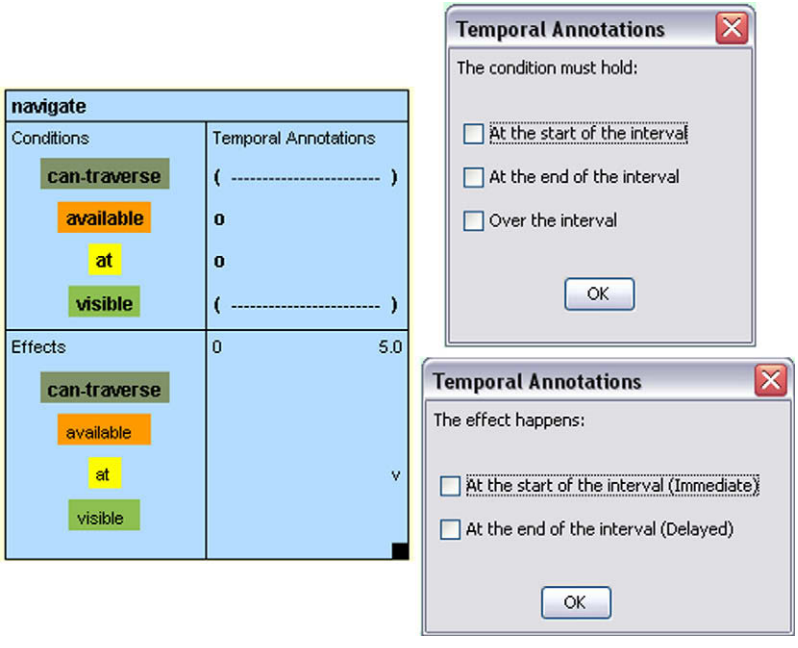

Fig. 6. A durative action and the dialogue boxed for its temporal annotations.

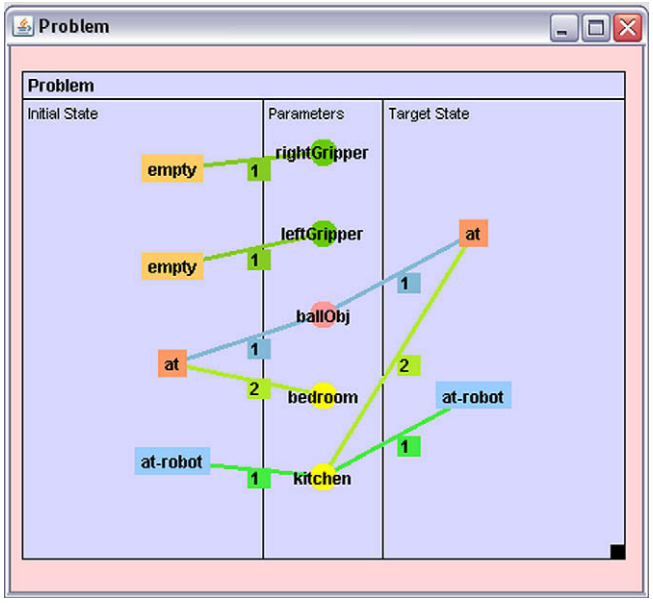

Fig. 7. A problem instance of the Gripper domain.

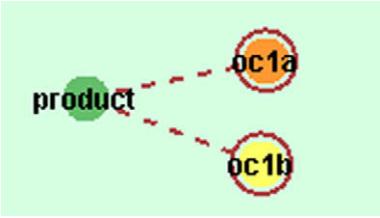

Fig. 8. An example of two constants of the type "product".

4.4. Advanced features

The basic PDDL features described above are adequate for simple planning domains and problems, while for more complex problems advanced features of the language are used, described in this section.

A PDDL constant is represented in the system similarly to a class, enhanced with a red[1] outline, as depicted in Fig. 8. Constant must be of some type, and can be used either in an operator or in a problem, where they behave similarly to parameters or objects, respectively.

A derived predicate is an advanced PDDL feature that is represented by a group of design elements in VLEPPO, which includes the predicate enhanced with an AND/OR tree (Nilsson, 1998) that indicates the way it derives from other relationships. The tree is constructed from binary AND and OR and unary NOT nodes, while each of the node arguments can be either another node of any type, or a relationship. An example of a derived predicate is presented in Fig. 9, stating that a person can go out if it is sunny, or if it is raining and an umbrella is available.

Among the advanced features is the option to indicate that a predicate is timeless, that is, the predicate is true at all times. This operation involves a lot of validity checks, which will be explained in the corresponding paragraph.

Another PDDL feature incorporated in VLEPPO are numerical expressions, which involve the definition of a number of elements

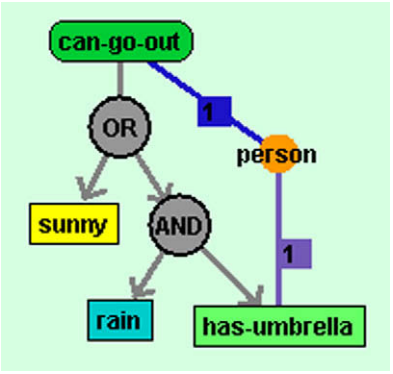

Fig. 9. An example of a derived predicate.

such as functions in the Ontology Frame and conditions or assignments incorporating these functions in the Operators Frame. Furthermore, for each function imported in the system, the initial values of the function for the problem at hand have to be defined in the Problem Frame. As an example from the *Rover* domain (IPC-5), consider a function relating a vehicle and two waypoints with the cost for the vehicle to traverse the distance between these waypoints. The definition of the function in the Ontology Frame and the corresponding values in the Problem Frame are depicted in Fig. 10.

Finally, *maps* can be created for every relationship that has exactly two arguments of the same type. Maps significantly facilitate problem readability when the relationship they represent is expected to have many instances in the initial state of a problem, as these instances can be omitted from the initial state definition. Each connection in the map represents an instance of the relationship, while the objects represent the arguments, as shown in Fig. 11. Maps do not have a direct correspondence to PDDL; therefore, in order to maintain full compatibility, their use is not mandatory. However, if used, they accommodate comprehensiveness in initial state definition.

4.5. Syntax and validity checking

A very important feature of the system is syntax and validity checking, as it detects errors and inconsistencies at the time they emerge and prevent them from propagating. Planning domains are checked for consistency within their own structures, and planning problems have to be checked for consistency and correspondence to the related domains. The remainder of this paragraph provides several examples illustrating the validity assurance processes of the system.

The user is allowed to insert a new connection in an operator or in a problem only if a corresponding connection exists in the Ontology Editor, while special care must be taken to verify that

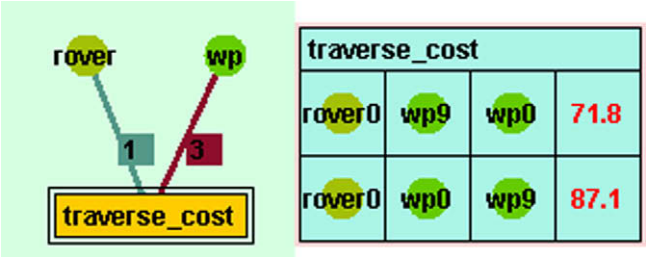

Fig. 10. An example of a function and its values.

[1] For interpretation of color in Fig. 8, the reader is referred to the web version of this article.

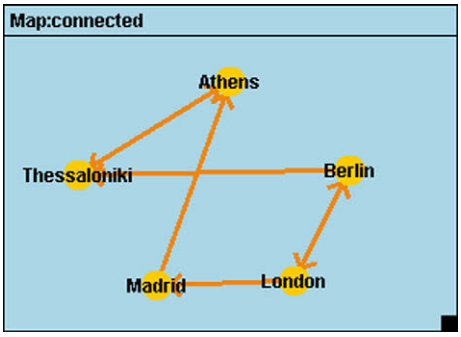

Fig. 11. A map for the relationship connected (C1, C2).

the types of parameters and objects match to the types of predicate arguments.

As far as constants are concerned, the system ensures the definition of their class before they are inserted in an operator or a problem. Furthermore, additional checks are performed about the types of arguments, similar to those performed for simple objects.

Timeless predicates are, by definition, allowed to appear in the preconditions of an operator, but not in the add or delete lists. As a consequence, adding a timeless predicate in the preconditions part of an operator will trigger its automatic appearance in the effects part as well, so the add and delete lists will not be affected. Furthermore, before setting a predicate timeless, checks will have to be performed to determine if this operation is allowed.

5. Import and export functions

5.1. Exporting to PDDL

The interoperability and increased flexibility of the system would not be possible without compliance with the PDDL standard. Visual elements taking part in domain definition, as well as comments, can be combined to formulate constructs and exported to a PDDL file, which is automatically enhanced with the appropriate requirements tag, as detected by the system. Some details clarifying export are presented in the remainder of the paragraph.

Upon export, the user is offered the option to use typing on demand, producing slightly different domains. In case the requirement *:typing* is declared, each class name is included in the *(:types)* construct of the domain definition, and for each object, parameter and constant its type must be declared wherever they appear. In case typing is not used, classes are treated as timeless unary predicates and appear along with the other predicates of the domain. In addition, for each parameter in an operator, a precondition that denotes its type must be added in the operator definition, while likewise, for each object, a new initial literal denoting its type must be included in the problem definition.

Exporting the domain operators requires a combination of several elements of the Operator Editor and the Ontology Editor, namely predicates, classes, variables and constants, functions, and obviously the operators themselves. Slight changes occur to an operator definition depending on whether the *:typing* requirement is declared, as the types of the parameters have to be included in the definition.

Exporting the problems is quite similar to exporting the operators, however, the problems are stored in a different PDDL file; therefore, numerous problems can be defined for the same domain. If maps are used, care must be taken to include the information they embody in the list of predicates included in the initial state. Furthermore, if functions are used, their initial values provided by the user in the Problem Editor will be part of the declaration of the initial state of the problem, in the corresponding construct.

5.2. Importing from PDDL

VLEPPO also offers the feature of importing planning domains and problems expressed in PDDL, visualizing them, and thus enabling the user to manipulate and maintain them. Both typed and non-typed PDDL files are supported; however, importing non-typed PDDL is subject to some restrictions. If no typing is used, syntax alone might not be enough, and semantic information might be necessary in order to discriminate types of objects from unary predicates. A module for translating non-typed to typed PDDL has been developed, which scans the non-typed PDDL file for unary predicates, which are candidates for being considered as "types" and consequently examines which of them could be timeless, by ensuring that they do not appear in the add or delete lists of any operators, a distinctive property of timeless predicates. After determining the types, the translation process can proceed. Although the non-typed to typed PDDL translation module has the best possible performance provided the information contained in PDDL files, when the domain includes ordinary unary predicates which do not happen to appear in any add or delete lists, they might be mistaken for types. In such cases, the intervention of a user, who can identify semantics, is required, but even so, the effort is minimal compared to the effort of designing the domain from scratch.

5.3. Save and load functions

Importing from and exporting to PDDL is essential as it provides formalization of the planning domains and problems created by the visual system and interoperability with planning systems. However, PDDL cannot capture visual information, although in some cases the user might have spent valuable effort to customize the way the planning domain and problems appear. To compensate this, an internal file format has been developed with the extension ".vff" (VLEPPO File Format), which is able to save not only the essential information of a planning domain and problem, i.e. the design elements such as objects, relationships, operators, but also the visual information such as colors and positions of these elements. Consequently, the load function is able to read this file format and restore the planning domain and problem to the exact state that it was in when saving was performed. Save and load functions are particularly useful during the development of the domains, when they are considered incomplete and exporting them to formal PDDL is not suggested.

6. Obtaining solutions to planning problems

6.1. Interface with planning systems implemented as web services

As VLEPPO is intended to be an integrated system not only for designing but for solving planning problems as well, an interface with planning systems is necessary. This is achieved by providing the ability to discover and communicate with web services which offer implementations of various planning algorithms. Moreover, existing planning systems can expose their functionality through web services and be utilized by VLEPPO (HAP-WS).

To this end, a dynamic web service client has been developed as a subsystem. The requirement for flexibility in selecting and invoking a web service justifies the decision to implement a dynamic client instead of a static one. In this way, the system can exploit alternative planning web services according to the problem at

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

hand, as well as cope with changes in the definitions of these web services.

The communication with the web services is performed by means of exchanging SOAP (Simple Object Access Protocol) messages, as the web service paradigm dictates. However, in a higher level, the communication is facilitated by the use of the PDDL language, which constitutes the common ground between the visual system and the planners. An additional advantage of using PDDL is that the visual system is released by the obligation to determine the PDDL features that a planner can handle, thus leaving each planning system to decide for itself.

The planners implemented as web services accept as inputs the PDDL descriptions of the planning domain and problem, perform the planning procedure and respond with the plan for the problem at hand. An example WSDL (Web Services Description Language) file describing typical planning web service API is depicted in Fig. 12.

The employment of web services technology results in the independency of VLEPPO from the planning or problem solving module and increased flexibility. Such a decoupling is essential since new planning systems which outperform the current ones are being developed. Each of them can be exposed as a web service and then invoked for solving a planning problem without any further changes to system or the domains and problems already designed and exported as PDDL files.

6.2. Solving planning problems locally

Although VLEPPO aims at exploiting the capabilities of the web service technology in order to be independent from the planning module, and thus take advantage of different planners according to the problem at hand, an option to solve the problems locally is also offered. This option can be used at any time without any special machine set up; therefore, the lack of internet connection or the lack of available planners implemented as web services do not prevent the user from obtaining valid plans, although the planning process might not be optimal. Currently, the planner used for solving the problems locally is LPG-TD (Gerevini, Saetti, & Serina, 2004; Gerevini et al., 2005), which proved to perform very well based on the results of the 4th International Planning Competition (IPC-4) (IPC, 2004).

7. Case study

In order to illustrate the use of the system, even on alternative domains, a case study concerning a real world problem has been selected, rather than the typical planning domains used in competitions and during the description of the system in the previous sections. The case study concerns web service composition; therefore, a brief introduction to web services along with a discussion about how a web service composition problem can be viewed as a planning problem is provided (Vrakas, Hatzi, Bassiliades, Anagnostopoulos, & Vlahavas, 2008). A system with visual capabilities such as VLEPPO is very useful when one attempts to approach web service composition problems through planning, as visual modeling significantly facilitates understanding of the dynamics of the available web services and their possible interactions during composition.

```
<definitions targetNamespace="PPP-WS">
<types>
        <s:schema elementFormDefault="qualified" targetNamespace="PPP-WS">
        <s:element name="PPP"> <s:complexType> <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="domain" type="s:string"/>
        <s:element minOccurs="0" maxOccurs="1" name="problem" type="s:string"/>
        </s:sequence> </s:complexType>
        <s:element name="PPPResponse"> <s:complexType> <s:sequence>
        <s:element minOccurs="0" maxOccurs="1" name="PPPResult" type="s:string"/>
        </s:sequence> </s:complexType>
        </s:element>
        </s:schema>
</types>
<message name="SoapIn">
        <part name="parameters" element="s0:PPP"/>
</message>
<message name="SoapOut">
        <part name="parameters" element="s0:PPPResponse"/>
</message>
<portType name="PPPWSClassSoap">
        <operation name="PPP">
        <documentation> Example WSDL document for planner </documentation>
        <input message="s0: SoapIn"/>
        <output message="s0:SoapOut"/>
        </operation>
</portType>
<binding name="PPPWSClassSoap" type="s0:PPPWSClassSoap">
        <soap:binding style="document" transport="http://schemas.xmlsoap.org/soap/http"/>
        <operation name="PPP">
        <soap:operation soapAction="PPP-WS/PPP" style="document"/>
        <input> <soap:body use="literal"/> </input>
        <output> <soap:body use="literal"/> </output>
        </operation>
</binding>
<service name="PPPWSClass">
        <documentation> Example Web Service interface of a planner </documentation>
        <port name="PPPWSClassSoap" binding="s0:PPPWSClassSoap">
        <soap:address location="http://exampleLocation.com"/>
        </port>
</service>
</definitions>
```

Fig. 12. An example WSDL document for a planner web service.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

7.1. Semantic web service composition

A web service is a software system identified by a URI (Universal Resource Identifier), whose public interfaces and bindings are syntactically defined and described the Web Services Description Language (WSDL) (WSDL, 2001). Its definition can be discovered by other software systems, which may then interact with the web service in a manner prescribed by this definition, using XML (XML) based messages conveyed by internet protocols (Booth et al., 2003). Web services aim to simplify the process of distributed computing by defining a standardized mechanism to describe, locate, and communicate with online software systems.

When individual web services are limited in their capabilities, they can be composed to create new functionality in the form of complex web services, a process called web service composition (Piccinelli, 1999), which is emerging as a new model for interactions among distributed and heterogeneous applications. To truly integrate application components on the Web across organization and platform boundaries merely supporting simple interaction using standard messages and protocols is insufficient (van der Aalst, 2003) and web services composition languages, such as BPEL4WS (Thatte, 2003), are needed to specify the order in which messages are exchanged and operations are executed.

Automatic application interoperability is hard to achieve at low cost without the existence of the Semantic Web. Semantic Web is the next big step of evolution in the Web (Berners-Lee, Hendler, & Lassila, 2001) that includes explicit representation of the meaning of information and document content (namely *semantics*). The combination of the Semantic Web with web services, namely *semantic web services*, achieves the automatic discovery and composition of web services (McIlraith, Son, & Zeng, 2001) through the use of semantic web service descriptions, on top of WSDL descriptions, such as OWL-S (OWL Services, 2003), that allow the web service semantics to become comprehensible by other agents/programs through ontologies.

The capability of automating web service composition is very important for the survival of web services in the industrial world as manual composition becomes significantly hard as the number of available web services increases (Bassiliades, Anagnostopoulos, & Vlahavas, 2005). A very promising direction to automatically composing semantic web services is through the use of AI planning techniques (Alfredo, 2003).

7.2. Web service composition as a planning problem

Using planning, the semantic web service composition problem can be described as a desired state to be achieved by the complex service and the planner will be responsible to find an appropriate sequence of simple web service invocations, to achieve this state. In this way, non-predetermined web services can be formulated on demand. The abilities of simple web services can be described in semantic languages such as OWL-S and can be considered as planning operators.

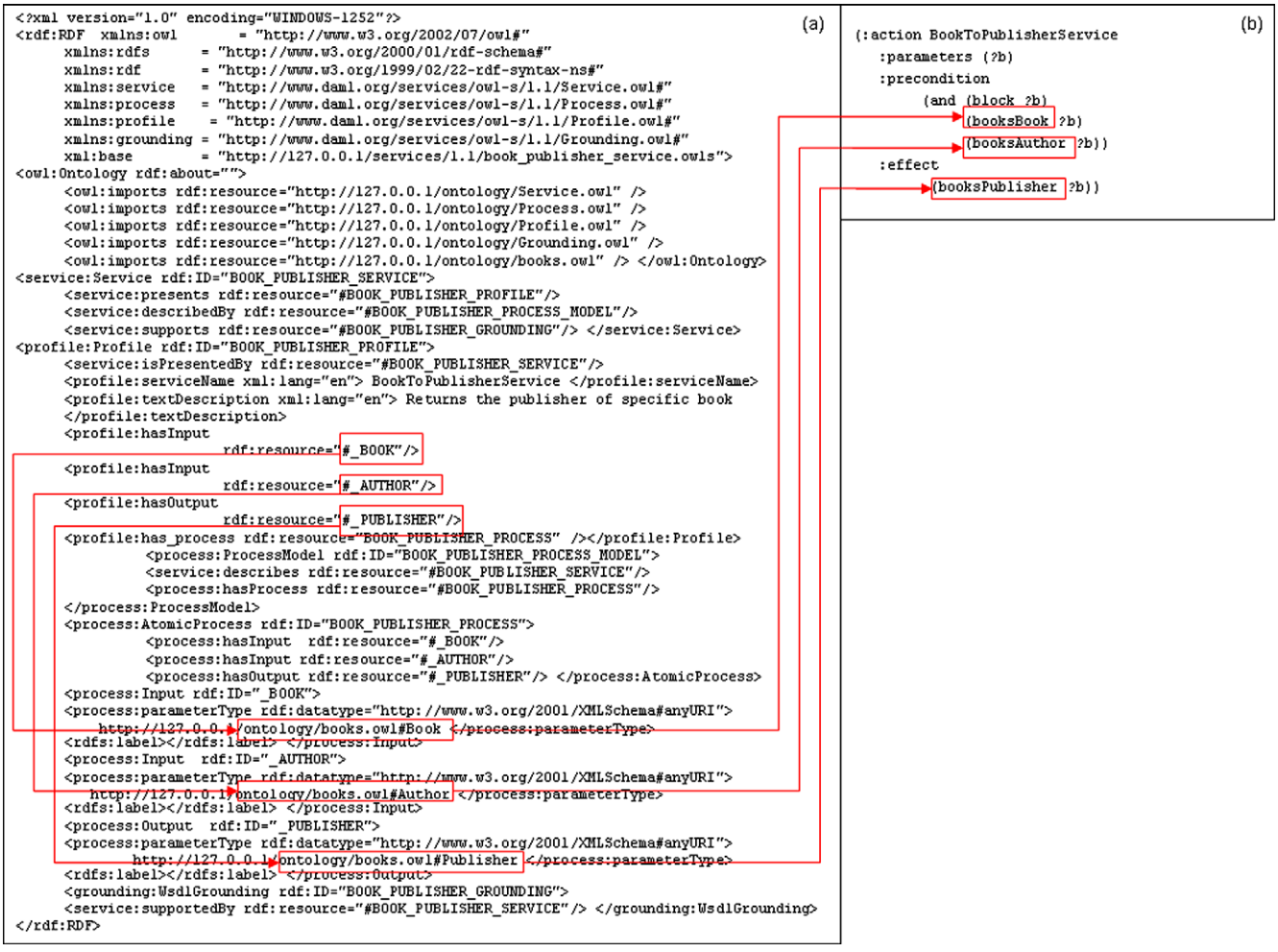

Fig. 13. An example of OWL-S to PDDL transformation. (a) The OWL-S profile of the BookToPublisherService. (b) The PDDL action produced.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

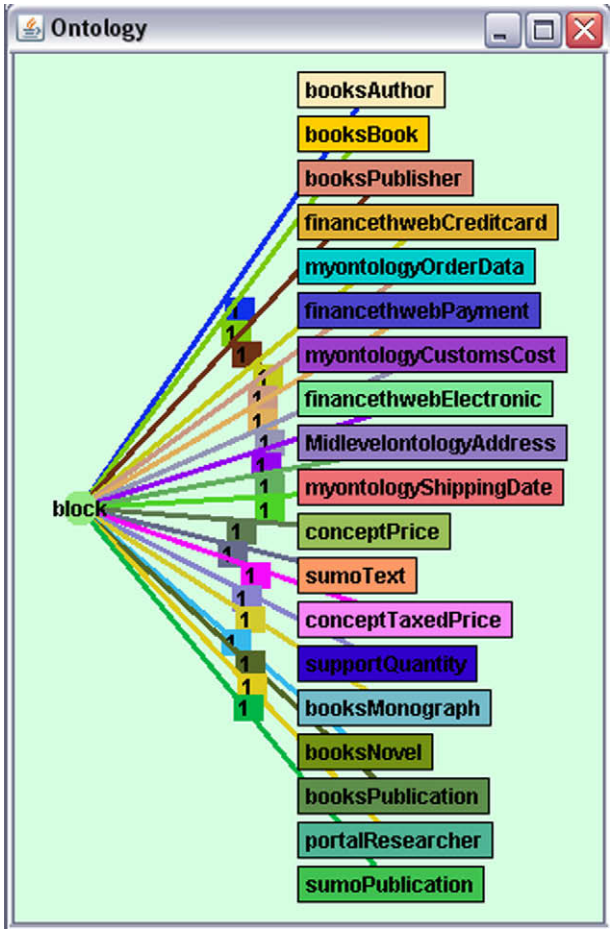

Fig. 14. The predicates for the electronic bookstore domain.

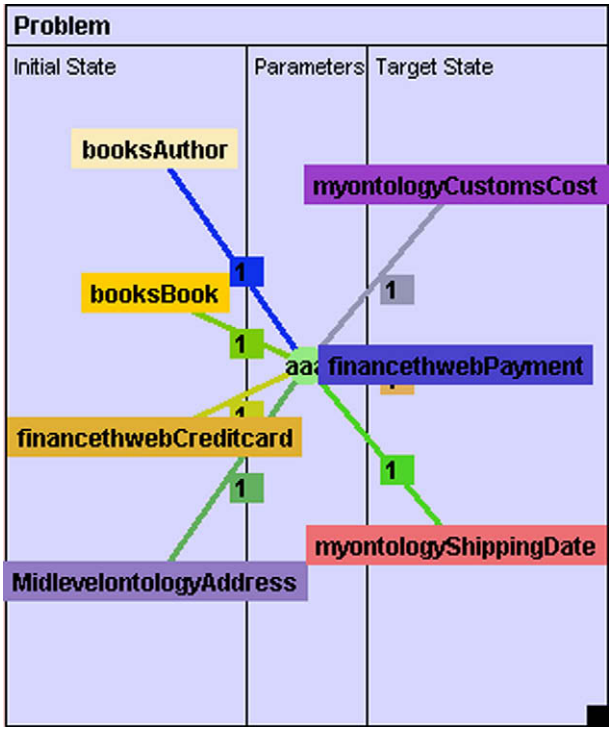

Fig. 16. A problem for the electronic bookstore domain.

To elaborate, consider the case were a user wishes to use a complex web service which, when provided with some input data, will return some required information. There may be a number of alternatives when formalizing the problem of web service composition as a planning problem (Carman, Serafini, & Traverso, 2003), however the most straightforward solution is the following: the inputs provided by the user form the initial state of the problem, while the desired outputs form the goals of the problem. The available semantic web service descriptions are used to obtain the actions or operators available in the planning domain, as they are described in the STRIPS formalism. Each action has the same name as the name of the corresponding web service, while its preconditions list is formed by the inputs of the service. The add list of the action includes the outputs of the service and the delete list is left empty.

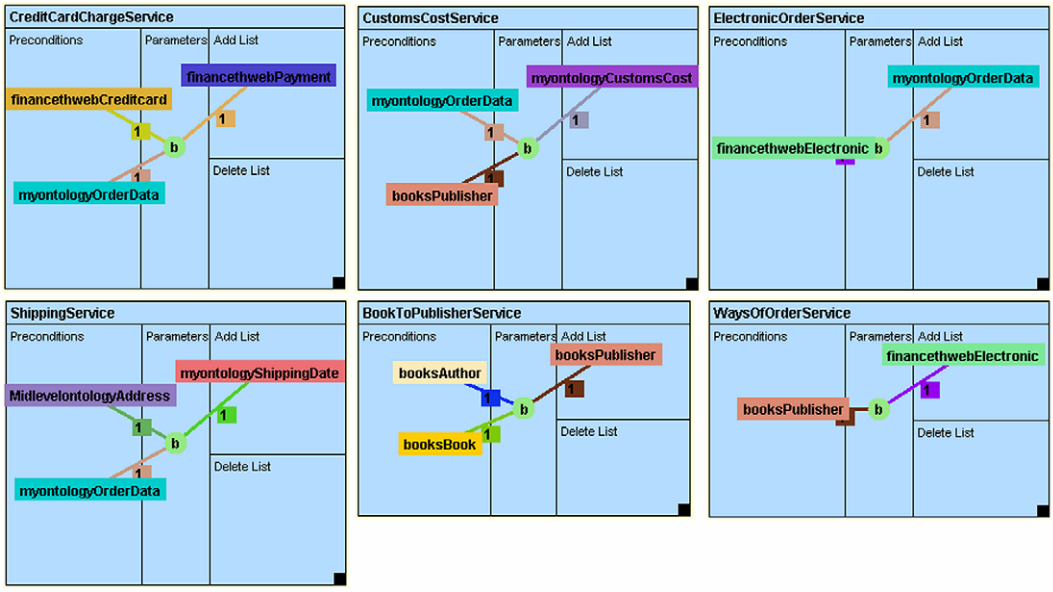

Fig. 15. The actions for the travel planning domain.

It should be noted that the formalization presented above in some cases might require the planning system to be aware of possible semantic similarities among syntactically different concepts. If this situation occurs, it can be dealt with utilizing ontologies, which describe relationships and similarities among concepts.

7.3. Example

Initial stages of experimentation on web service composition using VLEPPO included modeling the web services in the system from scratch. However, in order to obtain realistic results, case studies involving real test sets were pursued. The test sets containing semantic web service descriptions in OWL-S and their corresponding ontologies were obtained from OWLS-TC (OWLS-TC). Consequently, they were modified, parsed by the PORSCE II system (Hatzi et al, 2008), translated into PDDL and imported in VLEPPO. An example of the transformation of an OWL-S web service description into a PDDL action is presented in Fig. 13, however further discussion about the transformation process is beyond the scope of this work.

The domain presented here refers to electronic bookstores and the predicates it includes are shown in Fig. 14. It should be noted that in this web service scenario the predicates do not have any arguments, therefore an auxiliary argument not affecting the results had to be added, as some of the planners tested had trouble handing predicates without arguments.

The objective of the web service composition scenario is the electronic purchase of a book. Fig. 15 presents some of the domain web services turned into actions, in add/delete lists view, that will be used in the complex service satisfying the user requirements. As already mentioned, the inputs of a web service are the preconditions of the action, while the outputs of the service are the add list. The delete list is left empty, since no web service at this point is considered to be able to retract information or data from the world state (Hatzi, Vrakas, Bassiliades, Anagnostopoulos, & Vlahavas, 2007b).

A number of different problems can be defined corresponding to this domain definition. An example is presented in Fig. 16 where the supposed user provides a book title and author, credit card info and the address that the book will be shipped to, and requires a charge to credit card for the purchase, as well as information about the shipping dates and the customs cost for the specific item.

A full screenshot of the system interface with the domain and problem described above is depicted in Fig. 17.

The problem was solved locally using the LPG-td planning system, and the produced plan is presented in Fig. 18, while the visualization of the plan in levels is also provided in Fig. 19.

8. Related work

This section presents the most prominent experimental efforts to construct general-purpose planning tools that have appeared so far.

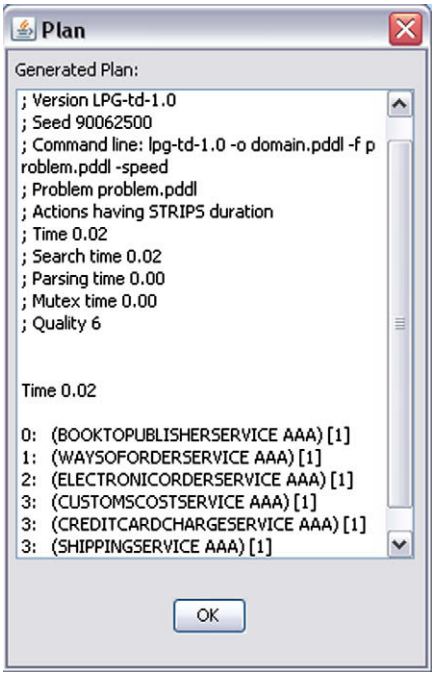

Fig. 18. The plan produced by LPG-td.

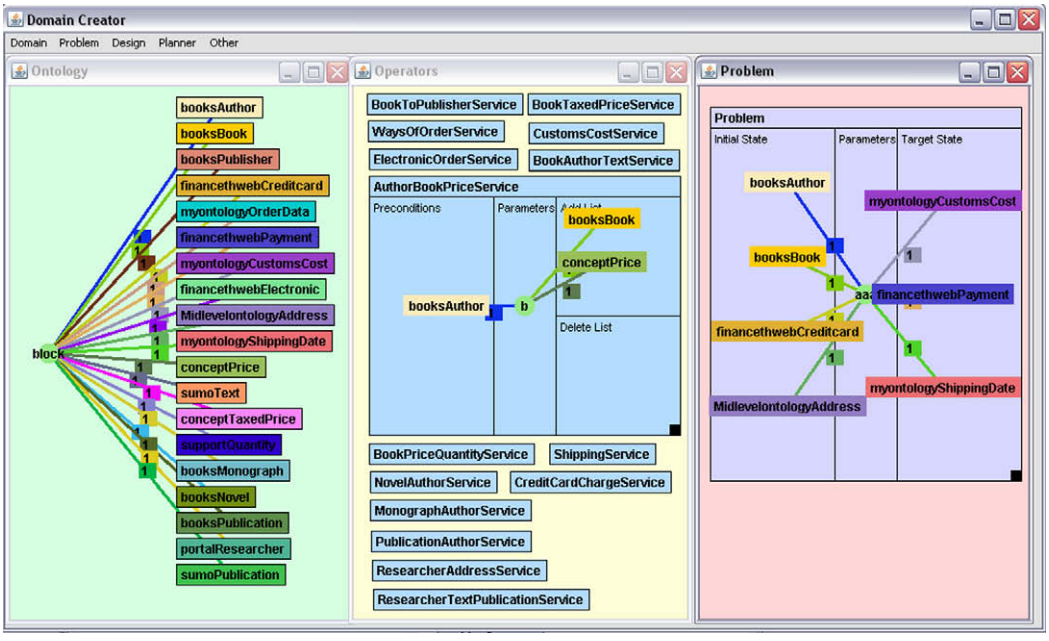

Fig. 17. A full screenshot of the interface with the case study presented above.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

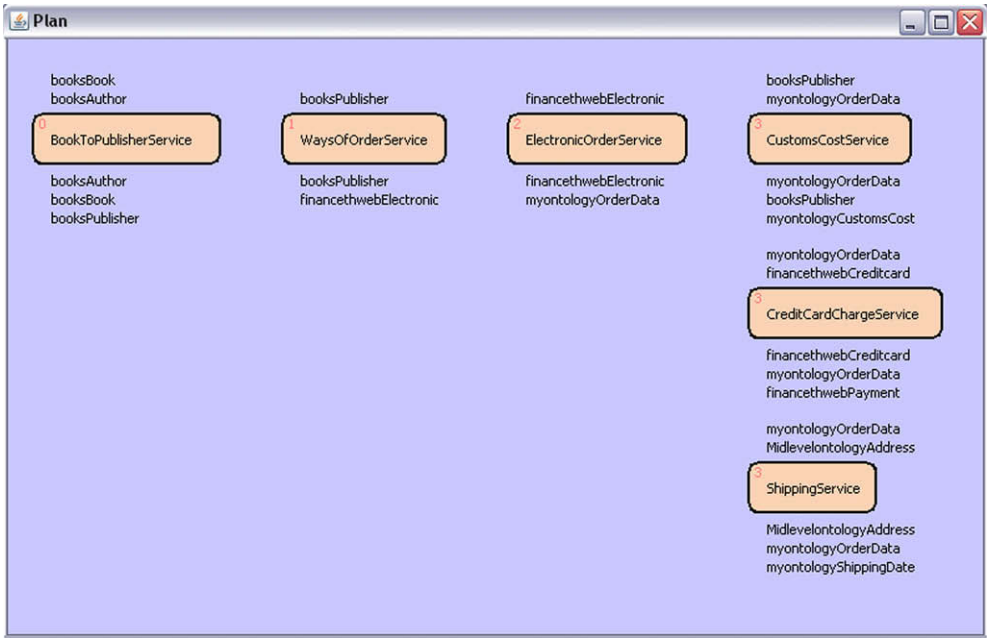

Fig. 19. Visualization of the plan.

The GIPO system (McCluskey, Liu, & Simpson, 2003) is based on an object-centric view of the world. The main idea behind it is the notion of change in the state of objects throughout plan execution. Therefore, the domains are modeled by describing the possible changes to the objects included in them. The GIPO system is designed to work with both classical and Hierarchical Task Network (HTN) domains. In both cases, it offers graphical editors for domain creation, planners, animators for the derived plans and validation tools. The domain models are represented mainly in an internal representation language called Object Constraint Language (OCL), which is object oriented, in accordance with the GIPO system. Only limited translation from the internal representation format to PDDL is offered. Local planners can be integrated into the GIPO system, however they must adhere to a number of restrictions, while the integration process is burdensome.

As far as the graphical aspects of GIPO are concerned, the most fundamental of them are the definition of classes of objects, predicates, and object class states. The above elements have to be defined in the order mentioned. The operators are thought of simply as transitions between states of an object. In addition, GIPO provides the *opmaker*, a tool which can derive operator definitions from partially defined domains and sample plans. Furthermore, planning problems, referred to as *tasks*, can be defined in the form of initial and goal states. Finally, some additional tools are included, such as the *Life History Editor,* which accommodate more advanced features and better understanding and manipulation of the domain.

The major difference between GIPO and the proposed VLEPPO system is the way they deal with planning domains. The object oriented GIPO has its own internal representation based on the OCL language. Thus, the design process is driven to follow this representation. On the contrary, the VLEPPO system provides design elements which correspond as accurately as possible to the well established elements of STRIPS/PDDL. Therefore, designing a planning domain and problems is much more straightforward with respect to the prevailing standards, interoperability with planning systems that conform to those standards comes naturally. Furthermore, the proposed system offers design elements that cover a wider part of PDDL, such as derived predicates and constants. In addition, an effort has been made to preserve the semantics and the names of the PDDL elements in order to decrease the learning curve of those users already familiar with the language. Finally, another significant difference lies in the interaction with external planning systems, which in the case of GIPO requires manual configuration and the planning systems available are limited to those that fulfil certain requirements, while in the case of VLEPPO, the only requirement about external planning systems is their compatibility with the PDDL standard.

SIPE-2 is another System for Interactive Planning and Execution of the derived plans (as the name implies) (Wilkins, 2000). As it is designed to be performance-oriented, it embodies many heuristics for increased efficiency, as well as the ability for providing advice to the built-in planner. Among its useful features is the plan execution monitoring, which enables the user to feed new information to the system in case changes occur in the world and the graphical interfaces for knowledge acquisition and representation, as well as plan visualization. The operators, plans and problems can be represented using the *ACT* formalism (Wilkins & Myers, 1994), and they can be visually manipulated through the ACT-Editor. The operators have the same semantics as in PDDL, meaning that they describe the changes in the state of the world, but there are also significant differences. They can have various levels of abstraction, and they can be applied to any situation, producing different effects each time, which are determined using deductive rules. Moreover, the operators state additional information, such as (but not restricted to) constraints, conditions and purpose, which denotes which goals the operator can solve. On the other hand, the sort hierarchy (classes) structured in the form of frames and can be manipulated through the GKB-Editor, a tool for editing knowledge bases in Frame Representation Systems. The user interface allows browsing any part of this hierarchy, as well as creating and editing classes and instances.

SIPE-2 is an elaborate system with a wide range of capabilities. However, it uses the ACT formalism, which is quite complicated and does not correspond directly to PDDL. Therefore, there is no way to easily use an existing PDDL file to construct a domain in SIPE-2, or export the domain or problem to PDDL, especially if the PDDL domain uses extended features, a restriction which sig-

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

nificantly decreases the interoperability of the SIPE-2 system with external planning systems. Another difficulty of using the system is the two totally separate graphical environments for elements of the domain, instead of a unified one. Finally, the system does not offer the capability to employ different planning algorithms other than the one implemented in it.

ASPEN is an environment for automated planning and scheduling (Fukunaga, Rabideau, Chien, & Yan, 1997). It is an object-oriented system initially targeted to space mission operation domains. Its features include an expressive constraint modeling language which is used for defining the application domain, systems for defining activity requirements and resource constraints, as well as temporal constraints. In addition, a graphical user interface is included, but its use is confined to visualizing plans and schedules in systems where the problem solving process is interactive.

ASPEN is not intended to be a general purpose system for domain and plan generation; on the contrary, it was developed for the specific purposes of space mission operations. Therefore, it uses its own object-oriented language, AML (Aspen Modeling Language). AML has only a few distant correspondences to PDDL, and the system does not provide the capability of exporting the domains and problems in PDDL. Furthermore, it does not offer a graphical interface for modeling the planning domains and problems.

Other systems that provide user interfaces such as Cox and Veloso (1997) and Pegram, St. Amant, and Riedl (1999) are beyond the scope of this discussion as they are developed for specific purposes and therefore cannot cover the needs of a general purpose planner.

In conclusion, although the above systems are useful, none of them offers direct visual representation of PDDL elements. This is essential for facilitating interoperability with external planning systems and design for users already familiar with the language and. Moreover, even the systems which offer translation to PDDL do not cover important features of the language. In addition, most of the aforementioned systems are confined to a single built-in planning algorithm. The proposed system attempts to tackle these issues by directly representing and using PDDL elements to define planning domains and problems and by cooperating with a variety of planning systems implemented or wrapped and deployed as web services.

9. Conclusions and future work

This paper presented VLEPPO, a visual system for facilitating the design and maintenance of planning domains and problems through a convenient graphical user interface, as well as obtaining solutions to planning problems utilizing different planners. A high degree of flexibility is provided in cooperating with planners, as they may be either local, or implemented as web services, as the system offers a web service client in order to exploit such functionality. One of the main concerns while designing and developing the system was to maintain the interface as simple as possible, while at the same time provide high correspondence to the PDDL language, in order to accommodate interoperability and compatibility with other planning systems through import and export functions.

The use of the system was demonstrated with a case study involving web service composition, viewed as a planning problem. This is an example not typical in the area of planning, but it depicts the convenience of the interface and the ability of the system to handle not only classical planning problems but also alternative problems that can be approached through planning.

As the area of planning for web service composition seems very promising, future extensions of the system will include components oriented to this direction. Added functionality will include integration with the PORSCE II (Hatzi et al., 2008) system in order to obtain planning domains from OWL-S web service descriptions automatically. Such an integration will also allow planning with semantic relaxation provided by the Ontology Management module of PORSCE II. Moreover, communication with Universal Description, Discovery and Integration (UDDI) registries (UDDI, 2004) is possible, in order to obtain new web services descriptions and formulate the available actions.

In addition, extensions to the visual interface in order to accommodate the visual representation of web services as planning actions are necessary. As the number of available actions in such domains is expected to increase over time, different representation schemes have to be developed in order for the interface to remain efficient. Such extensions should include modular view of operators in related groups, and search capabilities. Furthermore, plan manipulation would be a useful feature, as the produced plans might not always satisfy the user. Finally, the addition of plan metrics, apart from covering the corresponding PDDL feature, will enable the user to compare plans produced by different planners and select the best one based on several criteria.

Acknowledgments

This work was partially supported by a PENED program (EPAN M.8.3.1, No. 03ΕΔ73), jointly funded by the European Union and the Greek Government (General Secretariat of Research and Technology).

References

AIPS Planning Competition (2000). <http://www.cs.toronto.edu/aips2000/>.

Alfredo, M. (2003). Planning and scheduling for the web roadmap. Technical Coordination Unit for Planning and Scheduling for the Web, PLANET Network of Excellence <http://www.dipmat.unipg.it/~milani/webtcu/>.

Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2005). Web service composition using a deductive XML rule language. *Distributed and Parallel Databases, 17*(2), 135–178.

Berners-Lee, T., Hendler, J., & Lassila, O. (2001). The semantic web. *Scientific American, 284*(5), 34–43.

Booth, D. et al. (2003). Web services architecture, W3C working draft, August. <http://www.w3.org/TR/ws-arch/>.

Carman, M., Serafini, L., & Traverso, P. (2003). Web service composition as planning. In *ICAPS 2003 workshop on planning for web services*.

Cox, M. T., & Veloso, M. M. (1997). Supporting combined human and machine planning: An interface for planning by analogical reasoning. In *ICCBR 1997* (pp. 531–540).

Edelkamp, S., & Hoffmann, J. (2004). PDDL 2.2: The language for the classical part of IPC-4. In *Proceedings of the international planning competition international conference on automated planning and scheduling (Whistler 2004)* (pp. 1–7).

Fikes, R., & Nilsson, N. J. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence, 2*, 189–208.

Fox, M., & Long, D. (2002). PDDL+: Modeling continuous time dependent effects. In *Proceedings of the 3rd international NASA workshop on planning and scheduling for space*.

Fox, M., & Long, D. (2003). PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research, 20*, 61–124.

Fukunaga, A., Rabideau, G., Chien, S., & Yan, D. (1997). Towards an application framework for automated planning and scheduling. In *Proceedings IEEE aerospace conference, Snowmass, Colorado* (pp. 375–386).

Ghallab, M., Howe, A., Knoblock, C., McDermott, D., Ram, A., Veloso, M., et al. (1998). *PDDL – The planning domain definition language*. Technical Report, Yale University, New Haven, CT.

Gerevini, A., & Long, D. (2005). *Plan constraints and preferences in PDDL3*. Technical Report R.T., Department of Electronics for Automation, University of Brescia, Italy.

Gerevini, A., Saetti, A., & Serina, I. (2004). LPG-TD: a fully automated planner for PDDL2.2 domains (short paper). In *International planning competition, 14th international conference on automated planning and scheduling (ICAPS-04)*.

Gerevini, A., Saetti, A., & Serina, I. (2005). LPG-td planning system. <http://zeus.ing.unibs.it/lpg/>.

HAP-WS planner. <http://lpis.csd.auth.gr/systems/HAP-WS.wsdl>.

Hatzi, O., Meditskos, G., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2008). A synergy of planning and ontology concept ranking for semantic web service composition. In *Eleventh Ibero-American conference on artificial intelligence (IBERAMIA 2008)*.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047

Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2007a). VLEPpO: A visual language for problem representation. In R. Bartak (Ed.), *Twenty-sixth workshop of the UK planning and scheduling special interest group (PlanSIG 2007)* (pp. 60–66).

Hatzi, O., Vrakas, D., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2007b). Visual Representation of web service composition problems through VLEPpO. In J. D. Velasquez (Ed.), *International workshop on intelligent web based tools (IWBT07), 19th IEEE international conference on tools with artificial intelligence (ICTAI07)* (pp. 42–49).

Hendler, J. A., Tate, A., & Drummond, M. (1990). AI planning: Systems and techniques. *AI Magazine, 11*(2), 61–77.

Hsu, C. W., Wah, B. W., Huang, R., & Chen, Y. X. (2006). Handling soft constraints and preferences in SGPlan. In *Proceedings of the ICAPS workshop on preferences and soft constraints in planning*.

IPC (2004). International planning competition. <http://ls5-www.cs.uni-dortmund.de/~edelkamp/ipc-4/>.

IPC-5. Benchmark domains and problems. <http://zeus.ing.unibs.it/ipc-5/>.

McCluskey, T. L., Liu, D., & Simpson, R. M. (2003). GIPO II: HTN planning in a tool-supported knowledge engineering environment. In *International conference on automated planning and scheduling (ICAPS)/conference on artificial intelligence planning systems (AIPS)*.

McDermott, D. (1998). The 1998 AI planning systems competition. *AI Magazine, 2*(2), 35–55.

McIlraith, S., Son, T. C., & Zeng, H. (2001). Semantic web services. *IEEE Intelligent Systems, 16*(2), 46–53 [Special issue on the semantic web].

Nilsson, N. J. (1998). *Artificial intelligence: A new synthesis*. San Francisco, CA: Morgan Kaufmann Publishers Inc..

OWL Services Coalition (2003). OWL-S: Semantic markup for web services, OWL-S 1.0 Release. <http://www.daml.org/services/owl-s/1.0/>.

OWLS-TC version 2.2 revision 1. <http://projects.semwebcentral.org/projects/owls-tc/>.

PDDL4J project (2008). <http://sourceforge.net/projects/pdd4j>.

Pegram, D. A., St. Amant, R., & Riedl, M. O. (1999). An approach to visual interaction in mixed-initiative planning. In *Mixed-Initiative Intelligence workshop, AAAI-99*.

Piccinelli, G. (1999). *Service provision and composition in virtual business communities*. Tech. Report HPL-1999-84, Hewlett-Packard, Palo Alto, CA. <http://www.hplhp.com/techreports/1999/HPL-1999-84.html>.

SATPLAN (2004). <http://www.cs.rochester.edu/u/kautz/satplan/>.

Thatte, S. (Ed.). (2003). Business process execution language for web services (Version 1.1). <http://www-106.ibm.com/developerworks/webservices/library/ws-bpel>.

UDDI Specification (2004). <http://www.uddi.org/pubs/uddi_v3.htm>.

UML, Unified Modeling Language. <http://www.uml.org/>.

van der Aalst, W. (2003). Don't go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems, 18*(1), 72–76.

Vrakas, D., Hatzi, O., Bassiliades, N., Anagnostopoulos, D., & Vlahavas, I. (2008). A visual programming tool for designing planning problems for semantic web service composition. In F. Ferri (Ed.), *Visual languages for interactive computing: Definitions and formalizations* (pp. 302–326). Idea Group Publishing.

Weld, D. (1999). Recent advances in AI planning. *AI Magazine, 20*, 2.

Wilkins, D. E. (2000). SIPE-2: System for interactive planning and execution. <http://www.ai.sri.com/~sipe/>.

Wilkins, D. E., & Myers, K. L. (1994). *A common knowledge representation for plan generation and reactive execution*. SRI International. Artificial Intelligence Center.

WSDL (Web Services Description Language) (2001). <http://www.w3.org/TR/wsdl>.

XML, Extensible Markup Language. <http://www.w3.org/XML/>.

Please cite this article in press as: Hatzi, O., et al. A visual programming system for automated problem solving. *Expert Systems with Applications* (2010), doi:10.1016/j.eswa.2009.12.047