An extended HTN knowledge representation based on a graphical notation

Francisco Palao
IActive Intelligent Solutions
Granada, SPAIN

Juan Fdez-Olivares
Dept. of Computer Science and A.I
University of Granada

Luis Castillo and Oscar García
IActive Intelligent Solutions
Granada, SPAIN

Abstract

This work presents both an extended HTN Knowledge Representation based on a graphical notation inspired in commercial standards and a suite of tools, named *IActive Knowledge Studio*, based on this representation aimed at fully supporting a knowledge engineering process that, starting with the acquisition and representation of planning knowledge, ends with the integration and deployment of planning applications. The suite is also intended to be a knowledge engineering workbench for HTN planning applications deployment. This workbench includes several interesting features like a fully integrated representation of problem and domain knowledge and a new graphical and intuitive notation for easily representing HTN domains. It also provides tools for data integration with external data sources as well as an enhanced visual environment for HTN domains and plan validation.

Motivation

AI Planning and Scheduling (AIP&S) has revealed as an enabling technology to develop "assistant applications" which support human decision making in domains where the accomplishment of tasks to carry out a given activity or achieve a goal is mandatory. Most of these applications are aimed at supporting Human-Centric processes (Dayal, Hsu, and Ladin 2001) for knowledge workers (experts or decision makers). These processes are collections of tasks which support decisions and help to the accomplishment of workflow tasks for such knowledge workers in several and diverse application domains. Indeed, the features of AIP&S are really aligned with key requirements in these application field: human-centric processes mainly embody expert knowledge, processes need to be dynamically generated through a process that must be aware of the context in which they will be executed, and the execution environments are highly dynamic, thus requiring adaptive behaviour and rapid response to the new, changing situations.

The development of such knowledge-intensive applications, where intelligent planning becomes a key component, require a great modeling and engineering effort in the main application development stages: acquisition and representation of planning knowledge, validation of such knowledge, integration with external sources of information or already existing legacy software systems, and deployment of the final planning application. Most planning applications developed at the time being (Fdez-Olivares et al. 2006), (Fdez-Olivares et al. 2011),(Boddy and Bonasso 2010), (Cesta et al. 2010) are based on a standard life-cycle based on the above stages, but each of the steps in this cycle are performed following different, ad-hoc and difficult to couple techniques or tools, not easily scalable nor reproducible to other applications (thought the same technology is being used). For example, given a set of protocols and procedures to be operationalized, it is common to manually encode such protocols in a textual planning language (PDDL (Gerevini and Long 2006), HTN-PDDL (Fdez-Olivares et al. 2006) or ANML (Boddy and Bonasso 2010) to cite some), then to generate plans in order to validate this knowledge, then to develop ad-hoc algorithms to integrate external sources of information (for example mapping data from ontologies or external data bases to PDDL data models (Castillo et al. 2010a)), then to develop extra code in order to integrate the output of the planner with existing systems, etc. This is neither an agile nor easily reproducible development process that clearly impact negatively the goal of AI Planning to be a widespread and widely used technology. Moreover, a widely recognized bottle-neck in the development of such applications is that the languages used to represent both expert knowledge and domain dynamics are textual and oriented to expert AI planning researchers, and normally, little attention is devoted to the domain objects model as well as the plan representation and integration.

This handcrafted way of working in our area is in contrast with the extremely high-technological tools developed in other, no so distant areas like Business Process Management (BPM (van der Aalst, ter Hofstede, and Weske 2003)), where a plethora of tools may be found which give support to the whole life-cycle of development in such areas: modeling, deployment, execution and monitoring of processes. BPM is, perhaps, the dominant area in IT solutions for Human-Centric processes, but BPM technology is mainly focused on the management of static and perfectly predictable tasks/processes and, at present, there is a clear trend (González-Ferrer et al. 2010) to incorporate features like dynamic composition, context awareness or adaptiveness of processes into software solutions: clearly these features may come from the incorporation of AIP&S technology into these systems. However, in order to compete on

equal terms with this extremely developed area, and to convince IT engineers and managers about the real capabilities of AI Planning technology, the current engineering process of planning applications must be radically improved.

Compared with the modeling and engineering processes of the not so different BPM area, AI P&S lacks of integrated tools (also called suites) that cover an important gap, still not fully bridged, between the conceptualization and design of a planning domain and the final deployment of a fully functional planning application. In this sense, the contribution of this work is two fold:

- On the one hand, we present an extended knowledge representation based on a graphical notation, built on both the concepts of HTN planning languages (concretely an HTN extension of PDDL, called HTN-PDDL (Castillo et al. 2006) which has been used in several applications) and inspired in the graphical notation of industrial standards devoted to the modeling of business processes (called BPMN (White 2004)).
- On the other hand, a suite of strongly coupled planning tools developed by IActive Intelligent Solutions[1], integrated into a product called *IActive Knowledge Studio*[2], conceived as an integrated development environment for planning applications. It includes several *visual working environments* in order to support the main steps of Knowledge engineering: *acquisition and representation* of planning knowledge based on the graphical notation, *validation by inspection* based on planning process debugging and plan visualization and analysis, *knowledge integration* with external sources of information and *planning application deployment*.

Next sections are devoted to describe, firstly, the main issues concerned with the extended graphical representation and, secondly, the main features of IActive Knowledge Studio.

The graphical knowledge representation

The graphical knowledge representation introduced in this work can be considered as an evolution from a former, textual HTN language called HTN-PDDL (Castillo et al. 2006),(Fdez-Olivares et al. 2011) towards a more usable one, closer to the modeling practices of general purpose IT engineers and strongly focused on developing commercial planning applications. HTN-PDDL is a hierarchical extension of PDDL which incorporates all the standards concepts of the PDDL 2.2 version (Edelkamp and Hoffmann 2004). Concretely HTN-PDDL supports the modeling of planning domains in terms of a compositional hierarchy of tasks representing compound and primitive tasks at different levels of abstraction, where primitive tasks maintain the same expressiveness that PDDL 2.2 level 3 durative actions (allowing to represent temporal information like duration and start/end temporal constraints, see (Castillo et al. 2006) for details). In addition, HTN methods used to decompose compound tasks into sub-tasks include a precondition that must be satisfied by the current world state in order for the decomposition method to be applicable by the planner. The problem representation in HTN-PDDL is thus almost the same that in PDDL 2.2 problems, but with the only difference that the goal is described as a set of high-level tasks to be decomposed, instead of than a set of states to be achieved.

It is worth to note that this textual representation, though can be seen as a general-purpose hierarchical planning representation, based on the HTN paradigm, is specific for a temporally extended HTN planner, formerly known as SIADEX (Fdez-Olivares et al. 2006),(Castillo et al. 2006), which has evolved as a commercial product, developed by our start-up *IActive Intelligent Solutions*, now called *Decisor* [3]. Both, the textual language and the former planner have already been applied in several applications in diverse domains like crisis intervention (Fdez-Olivares et al. 2006),(Castillo et al. 2010a), e-learning (Castillo et al. 2010b), e-tourism (Castillo et al. 2008) or e-health (Fdez-Olivares et al. 2011), many of them being at present commercially exploited by IActive [4].

In the following, we will detail how this textual representation has evolved into a graphical knowledge representation based on three pillars: **1)** a domain objects representation based on UML that is called *Context Model* that clearly overcomes the classic representation of PDDL domain objects, **2)** a planning description language, still based on predicates but incorporating object-oriented concepts, named EDKL (Expert Knowledge Description Language) used for writing logical expressions for preconditions, effects, rules and temporal constraints in both compound tasks and actions, and **3)** a graphical notation named EKMN (Expert Knowledge Model Notation) used to represent the main concepts of an HTN domain (compound tasks, methods, primitive tasks as well as hierarchical, compositional and order relations) intended to be understandable by both, IT engineers and domain experts.

Context Model: UML-based domain objects model

Domain objects are represented in the Context Model following the standards recommendations of UML (Unified Modeling Language (Booch, Rumbaugh, and Jacobson 1999)), a standardized general-purpose modeling language in the field of object-oriented software engineering. UML includes a set of graphic notation techniques to create visual models of object-oriented software-intensive systems. Although UML supports the modeling of many aspects related with planning applications (like activities diagrams or state machine diagrams) we have opted to use UML only with those issues related with data and domain objects modeling, trying to bring the strong modeling capabilities of this standard to domain objects in AI Planning.

Indeed, all the HTN-PDDL concepts (mostly inherited from the classical PDDL types and objects representation) have their associated representation in UML. The Context

[1] http://www.iactive.es

[2] Download at http://www.iactive.es/productos/iactive-knowledge-studio/

[3] http://www.iactivecompany.com/products/iactive-intelligent-decisor/

[4] http://www.iactive.es/casos-de-exito/casos-de-exito/

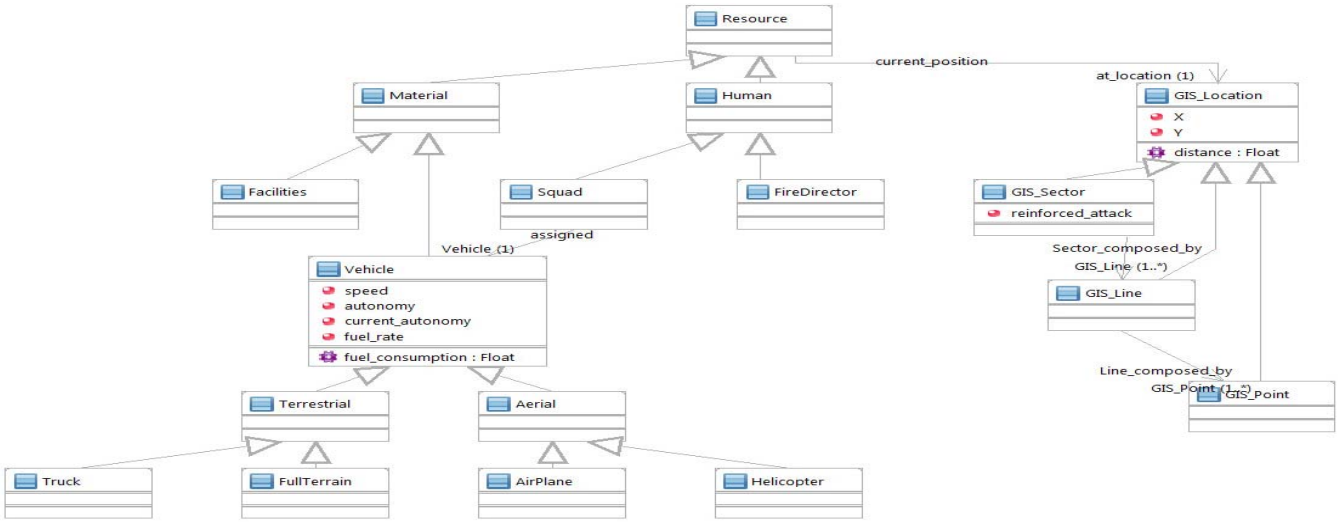

Figure 1: A (simple) UML model of a fire-fighting ontology: a Resource is located at a GISLocation (represented by a UML association). Every location is represented by two coordinates and an associated operation to compute the distance between two points (special Google libraries are provided by the language to implement this operation). There are material (Vehicles and Facilities) or Human (Squads and Fire Director) resources. A Vehicle has an speed, autonomy, current autonomy and a fuel rate. The fuel consumption of every vehicle (Terrestrial or Aerial) is a dynamic value computed by a procedure. Attributes, operations and relations are inherited throughout the is-a hierarchy.

Model is based on five key UML concepts: *Class, Attribute, Operation, Association and Generalization*. Object types are represented as UML Classes (see Figure 1), and the hierarchy of types is represented as an is-A hierarchy, using the UML standard Generalization relationship between classes; object properties are represented as UML Class attributes and relations as UML Association. Domain objects are represented as instances of the UML classes defined. In addition, UML operations are used as special attributes of objects that need to be computed by a procedure, thus allowing to manage and represent resources. Finally, it is also possible to represent temporal information as attributes of a class the value of which is of a special type called *DateTime* (supporting the representation of time and dates associated to objects).

This modeling approach based on UML eliminates any barrier between the modeler and the AIP&S technology, and it is intended to offer a standard way to model planning domain objects close to IT engineers. Moreover, as seen in the next section, the Context Model defined as a UML Class Diagram is directly incorporated and managed by the planning language. The incorporation of this object-oriented approach into de planning language provides new features that increase both the expressiveness of the planning language and the user-friendliness with respect to previous languages like PDDL or HTN-PDDL. Among others enhancements, these features allow to introduce a solid interpretation of inheritance of properties and operations in the language as well as a better modeling of objects relations, now much easier to manage. Perhaps the most important one is the integration with external data models. That is, most external sources of information are based on relational data bases or ontologies. Both models can be directly mapped into UML models, thus increasing the integration capabilities of planning domains and problems with external sources information, a key issue to be addressed in the development of any planning application.

EKDL: Expert Knowledge Description Language

As opposed to other well known approaches, like for example itSIMPLE3.0 (Vaquero et al. 2009), the aim of EKDL[5] is not to translate the UML Context Model into PDDL, but maintaining this object-oriented representation as modeled in the Context Model and introducing it into the planning language that will be described in this and following sections. Predicates are still the basic construction of EKDL, but the standard syntax used in most planning languages has been modified in order to cope with objects and classes.

The first consequence is that predicates now respond to the syntactical pattern $(C.p\ o\ v)$, where C stands for the name of a class, p stands for the name of a property or a relation, o stands for an instance or a variable of type C, and v stands for an instance or variable of the range of p (that is, the Class which the allowed values of the property p belong to). For example, a typical PDDL predicate $(at\ ?x$ - $Person$ $?y$ - $Place)$ shall require firstly to define a *Class Person*, with a property at of type *Place* (*Place* must also by defined as a class). This definition will result in an EKDL predicate $(Person.at\ ?x\ ?y)$ that can be used as desired in any logical expression. This notation (indeed a prefix form of the standard $< object\ attribute\ value >$) forces to represent all the

[5]A manual (in spanish) describing EKDL and EKMN can be found in http://help.iactive.es/

expressions as logical combinations of binary predicates, but has the key advantage that is aligned with standards knowledge interchange formats, like for example RDF triplets, a common way to serialize knowledge present in ontologies or relational data bases, thus opening the way to easily integrate the planning knowledge modeled under EKDL with other standard knowledge representation formalisms.

In summary, EKDL is an adaptation and extension of PDDL and HTN-PDDL that tries to maintain the expressiveness of logical expressions based on predicates, but also maintaining the object-oriented approach in logical expressions.

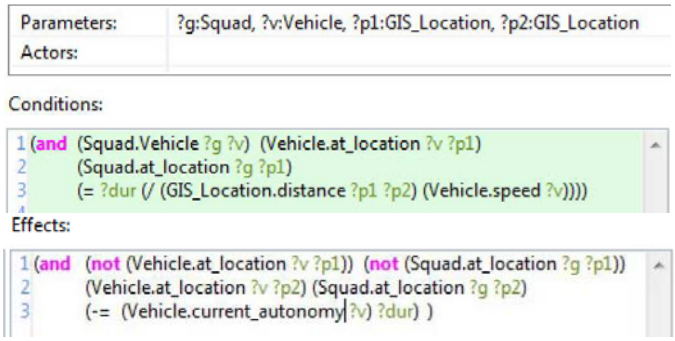

Figure 2: A primitive task represented in EKDL. It is like a PDDL durative action representing the movement of a human group transported by a vehicle. Temporal (duration of the transport) and resource constraints (the autonomy of the vehicle decreases) are also represented.

Primitive tasks are represented textually in EKDL (see Figure 2), but supported by visual forms. They are represented as a name, typed parameters (now referring to Context Model classes), and logical expressions to describe preconditions and effects. Preconditions and effects are represented as logical expressions of predicates taking into account the object-centered syntax above described. Numerical function are also allowed and, therefore, it is also possible to represent discrete numerical resources. Indeed numerical resources are represented as numerical properties of objects, for example the remaining autonomy of a Vehicle can be represented as the attribute *current-autonomy* of a class *Vehicle*. Then EKDL provides arithmetic and increment/decrement operators that can be applied to numerical attributes. For example, the expression $(= (Vehicle.currentautonomy ?v) ?dur)$ stands for a decreasing $?dur$ time units the autonomy of a Vehicle $?v$.

In addition, primitive tasks representation inherits the concepts of PDDL 2.2 level 3 durative actions (allowing to represent temporal information like duration and start/end temporal constraints). EKDL also allows the representation of PDDL 2.2 axioms. As shown in next sections, the introduction of this knowledge is supported by graphical interfaces, with powerful capabilities to make the introduction of this knowledge an easier task.

It is worth to note that neither UML activity diagrams nor state machine diagrams are intended to capture and represent all the categories of knowledge found in temporal HTN paradigms. Although they may be useful to represent some aspects of domain dynamics related to changes of state, and thus able to model basic primitive actions, they lack of necessary mechanisms to adequately represent temporal constraints as well as compound tasks and alternative decomposition methods. UML can represent relationships between activities at different levels of abstraction, but it is not designed to represent alternative decompositions to be dynamically managed at reasoning time by a planner (one of the key aspects of HTN). Moreover, thought some time constraints on activities can be represented in UML, they are intended to be used on sets of predefined, already fixed activities. The temporal constraints management in HTN is much more expressive, since it supports to represent which temporal constraints must be satisfied on a dynamically generated, and initially unknown set of tasks. Therefore, the modeling of HTN planning domains requires additional characteristics to the ones presented in UML, since UML is not designed to be a knowledge representation language. Because of this, we have defined a new graphical approach for this purpose, detailed in the next section.

EKMN: Expert Knowledge Model Notation

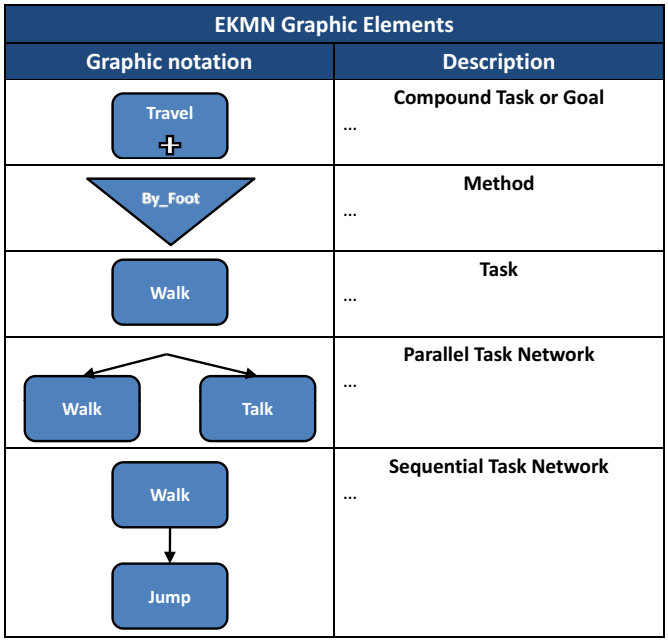

Figure 3: Main elements of the graphic notation.

Compound tasks, decomposition methods and primitive actions represented in an HTN planning domain mainly encode the procedures, decisions and actions that experts must follow, according to a given protocol, when they deal with a given decision problem.

Therefore, the main advantage of HTN planning approaches is their capability of capturing planning expert knowledge in form of either protocols or operating procedures. Hence, an HTN planning model should be seen as

a knowledge representation mechanism to represent human expertise and operating procedures as well as using them as a guide to the planning process.

The main goal of EKMN is to face these aspects and provide a notation that is understandable by IT modelers (responsible of encoding the knowledge finally managed by the planner) as well as domain experts (containers of the knowledge). It is inspired in BPMN (White 2004), the current standard notation for process modeling, and its aim is to display an intuitive visual metaphor of the main categories of knowledge found in an HTN domain. On the one hand, *compound tasks (or goals)*, *methods*, *task networks* and *primitive tasks*. On the other hand, the relationships between these categories: *hierarchical relationships* between tasks and methods, *is-part-of* relationships between methods and subtasks, and *order relations* between tasks inside a method). Figure 3 shows the visual representation used for compound tasks (goals), methods, primitive tasks and sequential and parallel tasks. The main idea behind this graphic notation is to support a modeling process that starts with the development of a visual skeleton of the task hierarchy, carrying out (if possible) a collaborative process between knowledge engineer and expert, and then to detail this skeleton by filling out more detailed knowledge, using EKDL, in successive refinement steps. Figure 4 shows an example of the skeleton of an HTN domain which incorporates the above described elements.

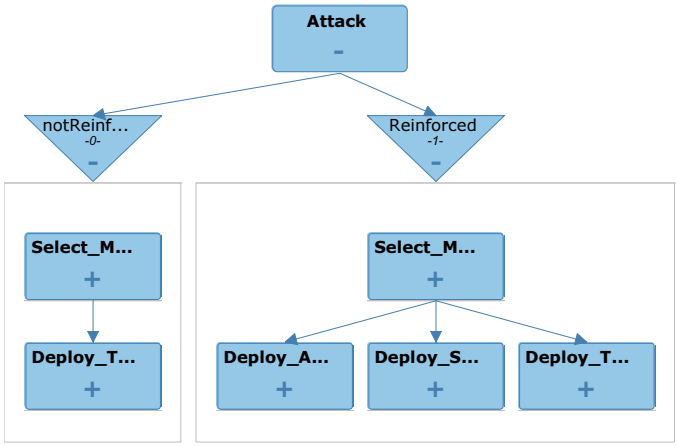

Figure 4: An example domain using EKMN implementing a standard operating procedure to attack a sector in a forest fire. The doctrine establishes that, in case of not being a reinforced attack, first select appropriated human means, then deploy and mobilize them. In case of reinforced attack, first select means, then deploy every human, aerial and terrestrial mean selected.

More concretely, the knowledge representation language as well as the planner are also capable of representing and managing different workflow patterns (van Der Aalst et al. 2003) which can be found in most business process models. A knowledge engineer might then represent control structures that define both, the execution order (sequence, parallel, split or join), and the control logic of processes with conditional (represented by alternative methods) and iterative ones (represented by recursive decomposition schemas).

In addition, this knowledge representation supports to explicitly represent and manage time and concurrency at every level of the task hierarchy in both compound and primitive tasks, by allowing to express temporal constraints on the start or the end of an activity. Any sub-activity (either task or action) has two special variables associated to it, $?start$ and $?end$, that represent its start and end time points, and some constraints (basically $<=$, $=$, $>=$) may be posted on them (it is also possible to post constraints on the duration with the special variable $?duration$). In order to do that, any activity may be preceded by a logical expression that defines a temporal constraint. For example, it is possible to encode constraints of the form $((and\ (>=\ ?start\ date1)(<=\ ?start\ date2))\ (t))$ what provides flexibility for the start time of t's execution, indicating that t should start neither earlier than date1 nor later than date2. These constraints are represented using EKDL and can be posted using a user-friendly interface, embedded into the integrated development environment that is described in the next section.

IActive Knowledge Studio

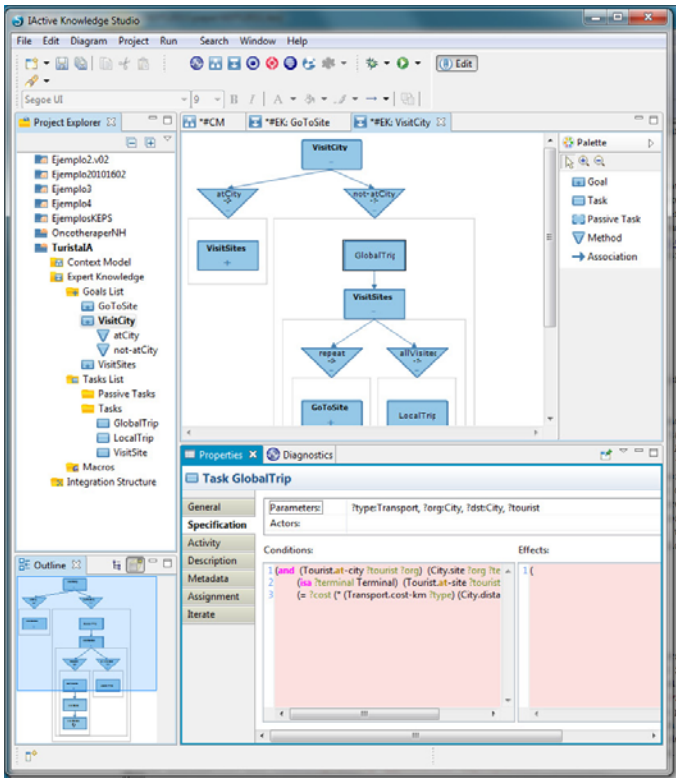

Figure 5: The knowledge edition environment of IActive Knowledge Studio. From left to right: the *project tree* showing the main parts of a planning project, the task hierarchy, an outline of this hierarchy, and a properties view showing the EKDL representation of a primitive task.

The above described knowledge representation and

graphical notation is the cornerstone of a suite of planning tools developed by IActive Intelligent Solutions, integrated into a product called *IActive Knowledge Studio*. This suite is conceived as an integrated development environment for planning applications. It is intended to support the main steps in the process of Knolwedge Engineering for Planning and Scheduling: *Knowledge Acquisition and Representation* based on the above described extended graphical representation, *Knowledge Validation* based on a validation-by-inspection process supported by an enhanced debugging process and plan analysis tools, *Knowledge Integration* with external sources of information, based on the Context Model above described and *Planning Application Deployment*. IActive Knowledge Studio has been developed using Eclipse Java technology, and it includes several *visual working environments* in order to support each one of these steps, which will be detailed in the four following sections.

Edition environment

The edition environment is intended to facilitate the acquisition and representation of planning domain knowledge. It allows to model planning objects based on the UML Context Model, to describe the task hierarchy based on EKMN and to fill out the properties associated to every category of knowledge through EKDL. Figure 5 shows a snapshot of the main perspective. It provides a *Project Tree* where the main categories of a planning project are shown: the *context model* in the terms above described, the *expert knowledge* showing an expandable list of tasks and the *integration structure* devoted to represent planning problems (see section below). A region to graphically model both the Context Model objects and the EKMN domain is also provided. Finally, the environment also includes a *properties window* in order to represent in EKDL the knowledge required for action preconditions and effects, temporal constraints, axioms, etc.

Integration environment

The integration environment has a triple role: (1) it is intended to define the necessary data definitions to allow the integration of both the input (initial state) and the output (the plans) of the planner with external systems, (2) to describe the initial state, and (3) to define the goal. Through this environment (see Figure 6), a domain modeler not only is able to represent the initial values of object properties or relations, but to define the way in which external sources of information can be accessed from the planner. Aimed at preserving its integration capabilities at the maximum, the data definition schemas managed by the planner (input data of initial state, input goal and plans) are stored as XML files (XSD templates), and these definitions are intended to both, be used to easily define the mapping from external data sources into the internal structure defined in the Context Model, and to integrate the plans obtained with external systems that need as input the output of the planner (for example, a BPM running engine).

The integration environment turns around the concept of *Integration Structure*, what can be seen as a metaphor of the "old" concept of planning problem, but redefining it into a more ambitious way, mainly centered on exploiting and enhancing the integration capabilities of the planner. Through this metaphor a Knowledge Engineer is able to define a planning problem by using the following concepts: *Local Data, Incoming Data and Incoming Goal*. **Local Data** are intended to be used in the first validation tests of the planning domain. The working environment allows to describe de initial conditions of every object as set of data tables (locally managed by the tool) that are automatically generated from the Context Model. These tables are automatically generated in such a way that the attributes of every table T corresponds to the properties and relations defined for a corresponding class C_T. In addition, each row in the table corresponds with an object instance of type C_T. Therefore, a domain modeler can describe the initial state of the world in a friendly and commonly accepted interface based on relational data base tables. The section **Incoming data** is aimed at managing the Data Template Definition (DTD), stored as XML files, that an external source should fit in order for the planner to access these external data. This is the cornerstone of the integration features of this approach: on the one hand, the XML schemas so defined can be used to easily develop data mappings to acquire external information, on the other hand these files can be used to provide the XSD specifications to third-party developers responsible of integrating the planning application with other systems. Finally, through the **Incoming Goal** Section the modeler defines the goal, that is, the task at the highest level to be decomposed as well as its parameters and start/end temporal constraints. These definitions are also stored as XML files that are given as inputs to the planner in order to carry out a planning process. Finally, the resulting plan is also stored as an XML file which can be used to integrate this output with other external systems.

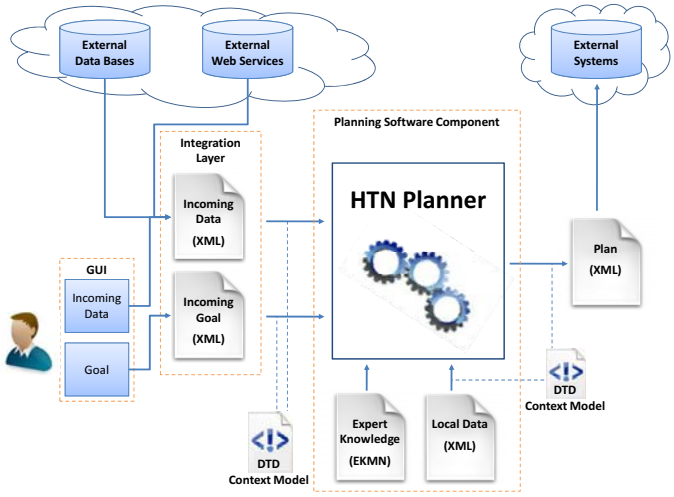

Figure 6: A diagram of the integration environment.

Execution and debugging environment

Once the domain knowledge as well as the information required for the planner has been defined, the next step con-

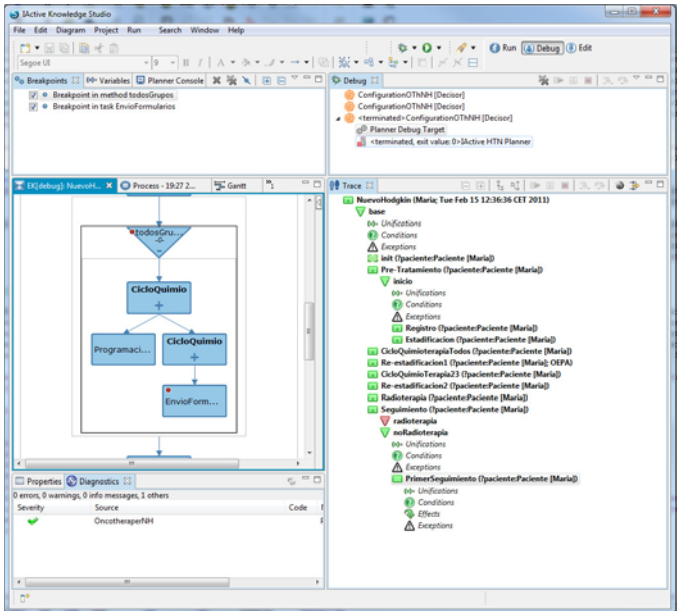

Figure 7: A snapshot of the debugging environment.

sists on using the execution and debugging environment in order to validate the knowledge. The execution and debugging environment provides the necessary functionalities to perform a validation-by-inspection process. On the one hand, it provides a trace facility that allows to execute step by step the planning process guided by the knowledge described in the EKMN notation (see Figure 7. The trace window allows to visualize and analyze through a expandable decision tree the decisions performed by the planner during the planning process. In addition, the tool also supports to define *breakpoints* associated either to compound tasks (goals), methods or primitive tasks, thus allowing to interrupt the knowledge-based reasoning process at any point defined by the modeler. The trace tree informs about the selected HTN methods as well as the discarded options during a given problem solving episode. In addition, this environment also allows to show the intermediates states produced by the planner. Moreover, it provides powerful tools for plan analysis and validation. Firstly, the plan obtained can be visualized either as a sheet or as a gantt diagram. The gantt diagram visualization also allows to intuitively analyze order dependencies between actions. In addition, there is a section to show several statistics about the resulting plan (resource usage, actions duration, etc.).

Deployment environment

The deployment environment allows to obtain a software component with the functionality defined in the previous steps and which will be able to be executed as a standalone application. The deployment process starts when the planning knowledge has been acquired and represented, when the data models necessary to integrate external information have been defined and when the knowledge has been validated. The deployment environment allows to deploy a planning project either as a local application (a *jar* file that can be accessed through an API) or as a web service (allowing to access to the planning application through remote procedure calls). In both cases, an API (application programming interface) is automatically provided. This API allows, among other operations, to perform calls to the planner and to obtain plans. Concretely, the planning component developed will return a plan from a given goal, specified as an XML file and a given set of data specified as an XML file accomplishing the data schema defined in the integration environment. The plan representation is based on an internal model that can also be known through the API. Therefore, this opens the way to integrate (by third party developers) the plans obtained with external systems that require as input the solutions offered by the planner.

Related work

Regarding other Knowledge Engineering Tools, GIPOII (McCluskey, Liu, and Simpson 2003) and itSIMPLE3.0 (Vaquero et al. 2009) are pursuing similar purposes as the work here presented. On the one hand, GIPOII is mainly aimed at supporting an object-centered Knowledge Acquisition process, focused on how domain objects change their properties or relations by specifying object transitions. Then these transitions are grouped to finally form action specifications. As opposite, the approach here presented supports a task-centered knowledge engineering process, based on the HTN paradigm. With respect to isSIMPLE3.0, one of the main common points is the domain objects model, also based on UML diagrams. itSIMPLE3.0 is devoted to capture user data requirements by using class diagrams and state based diagrams in order to represents the dynamics of actions in non-hierarchical domains, and then to translate this model into PDDL. Our approach, on the contrary, maintains the object-oriented approach to its last consequences and, so, the planning language (EKDL) has been designed in order to adopt the object-oriented paradigm. Perhaps, one of the stronger points of itSIMPLE3.0 is its capability to perform domain analysis by translating the UML and PDDL specification into Petri-Nets, what suposses a great help when focusing on validating the dynamic aspects of the domain. However these techniques are not fully applicable to hierarchical domains like the one addressed in our approach. On the other hand, itSIMPLE3.0 does not deal with temporal constraints, nor faces the representation of hierarchical planning knowledge.

Another work somehow related is (Boddy and Bonasso 2010) where authors describe a knowledge engineering process that includes the representation of operating procedures in NASA's PRL (Procedure Representation Language)(Kortenkamp et al. 2008), and then the (manual) translation into ANML (Action Notation Modeling Language) in order to be manageable for a planner. Indeed the representation of operating procedures is supported by a different tool (Kortenkamp et al. 2008), than the one used to represent planning domains with ANML (Boddy and Bonasso 2010). Our knowledge representation, based on the graphical HTN notation here presented is indeed intended to represent operating procedures (as HTN domains) and, moreover, to make

these protocols directly interpretable by a planner, thus offering a more integrated approach than the one described by Boddy et al.

Conclusions

In this work we have presented an extended Graphical Knowledge Representation for HTN domains with three main features: (1) it allows to model domain objects following an object-centered approach based on UML visual diagrams, overcoming several weaknesses of PDDL, mainly related with expressiveness and user-friendliness issues, specially the most relevant one is that this planning domain objects representation is closer to the modeling practices of IT engineers; (2) this Context Model is also directly embedded into the planning language and, in order to manage this object-centered model, EKDL (an object-oriented redefinition of basic PDDL constructs) has been described; (3) the most important aspect of this knowledge representation is EKMN, a graphical notation based on standard BPM modeling notations, which is aimed at visually and intuitively representing HTN domains as hierarchical and expandable diagrams based on compound tasks (goals)/methods/primitive tasks and the relationships between them. This graphical knowledge representation is intended to be understandable by both, IT engineers and domain experts. On the other hand, the knowledge representation is the basis on which IActive Knowledge Studio has been built. It is a development suite intended to support a knowledge engineering process that bridges the gap between the conceptualization of a planning domain and the final deployment of a planning application. The tool may also be seen as a workbench that can be used for academic and research purposes, including interesting features like a fully integrated representation of problem and domain knowledge and a new graphical and intuitive notation for easily representing HTN domains. IActive Knowledge Studio, also provides tools for data integration with external data sources, plan statistics and visualization methods for plan validation. But its most distinguishing features is that it has been designed to be used by IT engineers when developing commercial planning applications. Indeed, we have achieved to significatively increase the number of users of AIP&S technology through this development suite, since it is a commercial product that is being used in several industrial projects developed in collaboration with the partners of IActive Intelligent Solutions.

References

Boddy, M., and Bonasso, R. 2010. Planning for human execution of procedures using ANML. In *Scheduling and Planning Applications Workshop (SPARK), ICAPS*.

Booch, G.; Rumbaugh, J.; and Jacobson, I. 1999. *The unified modeling language user guide*. Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA.

Castillo, L.; Fdez-Olivares, J.; García-Pérez, O.; and Palao, F. 2006. Efficiently handling temporal knowledge in an HTN planner. In *Proceeding of ICAPS06*, 63–72.

Castillo, L.; Armengol, E.; Onaindía, E.; Sebastiá, L.; González-Boticario, J.; Rodríguez, A.; Fernández, S.; Arias, J.; and Borrajo, D. 2008. samap: An user-oriented adaptive system for planning tourist visits. *Expert Systems with Applications* 34(2):1318–1332.

Castillo, L.; Fdez-Olivares, J.; González, Milla, G.; Prior, D.; Morales, L.; Figueroa, J.; and Pérez-Villar, V. 2010a. A knowledge engineering methodology for rapid prototyping of planning applications. In *Proceedings of FLAIRS 2010*.

Castillo, L.; Morales, L.; Gonzlez-Ferrer, A.; Fdez-Olivares, J.; Borrajo, D.; and Onainda, E. 2010b. Automatic generation of temporal planning domains forelearning problems. *Journal of Scheduling* 13:347–362.

Cesta, A.; Finzi, A.; Fratini, S.; Orlandini, A.; and Tronci, E. 2010. Validation and verification issues in a timeline-based planning system. *The Knowledge Engineering Review* 25(03):299318.

Dayal, U.; Hsu, M.; and Ladin, R. 2001. Business process coordination: State of the art, trends, and open issues. In *Proceedings of the 27th VLDB Conference*.

Edelkamp, S., and Hoffmann, J. 2004. The language for the 2004 international planning competition. http://ls5-www.cs.uni-dortmund.de/ edelkamp/ipc-4/pddl.html.

Fdez-Olivares, J.; Castillo, L.; García-Pérez, O.; and Palao, F. 2006. Bringing users and planning technology together. Experiences in SIADEX. In *Proceedings ICAPS06*, 11–20.

Fdez-Olivares, J.; Castillo, L.; Czar, J. A.; and Prez, O. G. 2011. Supporting clinical processes and decisions by hierarchical planning and scheduling. *Computational Intelligence* 27(1):103122.

Gerevini, A., and Long, D. 2006. Plan constraints and preferences in PDDL3. *ICAPS 2006* 7.

González-Ferrer, A.; Fdez-Olivares, J.; Sánchez-Garzón, I.; and Castillo, L. 2010. Smart Process Management: automated generation of adaptive cases based on Intelligent Planning technologies. In *Proceedings of the Business Process Management 2010 Demonstration Track*.

Kortenkamp, D.; Bonasso, R.; Schreckenghost, D.; Dalal, K.; Verma, V.; and Wang, L. 2008. A procedure representation language for human spaceflight operations. In *Proceedings of i-SAIRAS-08*.

McCluskey, T. L.; Liu, D.; and Simpson, R. M. 2003. GIPO II: HTN planning in a tool-supported knowledge engineering environment. In *13th ICAPS*.

van Der Aalst, W.; Ter Hofstede, A.; Kiepuszewski, B.; and Barros, A. 2003. Workflow patterns. *Distributed and parallel databases* 14(1):5–51.

van der Aalst, W.; ter Hofstede, A.; and Weske, M. 2003. Business process management: A survey. *Business Process Management* 1019–1019.

Vaquero, T.; Silva, J.; Ferreira, M.; Tonidandel, F.; and Beck, J. 2009. From requirements and analysis to PDDL in itSIMPLE3. 0. In *ICKEPS'09: Proceedings of the 3rd. International Competition on Knowledge Engineering for Planning and Scheduling*, 54–61,.

White, S. 2004. Introduction to BPMN. *IBM Cooperation* 2008–029.