Validating Procedural Knowledge in the Open Virtual Collaboration Environment

Gerhard Wickler
University of Edinburgh, UK
g.wickler@ed.ac.uk

ABSTRACT

This paper describes the OpenVCE system, which is an open-source environment that integrates Web 2.0 technology and a 3D virtual world space to support collaborative work, specifically in large-scale emergency response scenarios, where the system has been evaluated. The support is achieved through procedural knowledge that is available to the system. OpenVCE supports the distributed knowledge engineering of procedural knowledge in a semi-formal framework based on a wiki. For the formal aspect it relies on a representation used in AI planning, specifically, Hierarchical Task Networks, which corresponds naturally to the way emergency response procedures are described in existing Standard Operating Procedures. Knowledge engineering is supported by domain analysis that may highlight issues with the representation. The main contribution of this paper lies in a reasonably informal description of the analysis.

The procedural knowledge available to OpenVCE can be utilized in the environment through plans generated by a planner and given to the users as intelligent, distributed to-do lists. The system has been evaluated in experiments using emergency response experts, and it was shown that procedural uncertainty could be improved, despite the complex and new technologies involved. Furthermore, the support for knowledge engineering through domain analysis has been evaluated using several domains from the International Planning Competition, and it was possible to bring out some issues with these examples.

Keywords

Emergency planning, procedural knowledge, virtual collaboration.

INTRODUCTION

In recent decades, civil protection has become more and more organized and many governmental organizations as well as NGOs and the military are now ready to assist in various types of large-scale disasters, be they natural or man-made [Coppola, 2006]. As part of the preparation for disasters, such agencies prepare procedures and processes that should be followed if and when an event occurs. This procedural knowledge, often referred to as Standard Operating Procedures (SOPs), is usually available in textual form, ranging from a single page to procedures documented in volumes of hundreds of pages.

Crisis response and management is a complex domain in which information systems can be used to support many different aspects of the problem, as shown in the ISCRAM series of conferences. The amount of procedural knowledge that is developed in preparation for disasters indicates the importance given to this type of knowledge. While these manuals are considered valuable where they exist, there are a number of problems with such documents in practice:

- Access time: While these manuals are useful for teaching the procedures they contain, they are usually not used during an actual emergency. This is simply because there is no time to search for information in large manuals. Emergency managers may have been through the SOPs, but under stressful conditions options may be forgotten or steps may be omitted.
- Structure: The manuals are often well-structured, but there is no standard way of structuring these documents. An emergency manager who needs to be familiar with different SOPs deriving from different sources may thus find them confusing to use.
- Updating: Procedural knowledge should be updated with lessons learned after every emergency in which they have been applied. This is a cumbersome task to perform with printed manuals, and even web-based documents offer limited support for this process.

Currently, information systems that support the development, deployment and maintenance of procedural knowledge are few. Hence, procedural knowledge for crisis response and management exists but is not integral to the response process.

In this paper we describe the OpenVCE system, which is an environment that supports collaborative work and has been demonstrated and evaluated in various emergency response scenarios. The support is achieved through procedural knowledge that is available to the system in a semi-formal, distributed representation, and the system supports the whole knowledge life-cycle using various results from AI planning research.

THE OPENVCE SYSTEM

The OpenVCE project [Hansberger, 2010; Wickler et al., 2013] aimed to develop an open virtual collaboration environment that facilitates collaborative work in a virtual space. The OpenVCE space consists of two linked environments: a dynamic website and a 3D virtual world space for meetings [Tate et al., 2009] as shown in figure 1. The left-hand side shows the Drupal- and MediaWiki-based website with recently added posts, and the right-hand side shows a virtual meeting in Second Life with a screen showing automatically generated information. This environment could, for example, be used to collaborate on the development of procedural knowledge, or it could be used during an actual emergency to manage information and courses of action. In fact, this environment contains a specific piece of software that supports these two functions: the <I-N-C-A> extension for MediaWiki.

Figure 1. OpenVCE Website and 3D Virtual Meeting Space

To avoid all this technology overwhelming novice users who are attempting to collaborate in this space, the project has also developed the Virtual Collaboration Protocol, which is itself procedural knowledge that describes how this environment is meant to be used to deal with certain types of emergency. This protocol is itself supported by an extension to the website that guides users who are following the protocol.

Development of Procedural Knowledge

We have based our collaborative editing facility that can be used to write SOP manuals on MediaWiki [Barrett, 2009]. The reasons for this choice are simple: MediaWiki is open-source (a project requirement), scalable (it powers Wikipedia), and there is an active community behind it. However, wiki articles are not structured to support SOPs, which is why we have implemented an extension that allows for the structuring of an article representing procedural knowledge according to the principles underlying Hierarchical Task Network (HTN) planning [Tate, 1977; Ghallab et al., 2004], which provides a natural way of decomposing tasks into sub-tasks.

The representation of procedural knowledge in the wiki consists of two parts: the informal wiki text that can contain any type of information in a human-readable form, and the formal representation which is closely based on <I-N-C-A> [Tate, 2003], the representation used by our HTN planning system. The authors of procedural

knowledge need to be familiar with the syntax used for the formal representation. However, the edit view is different from the normal view which transforms the internal representation into one that is easier to understand for users not familiar with <I-N-C-A>. Figure 2 contrasts the two views of the same procedural knowledge.

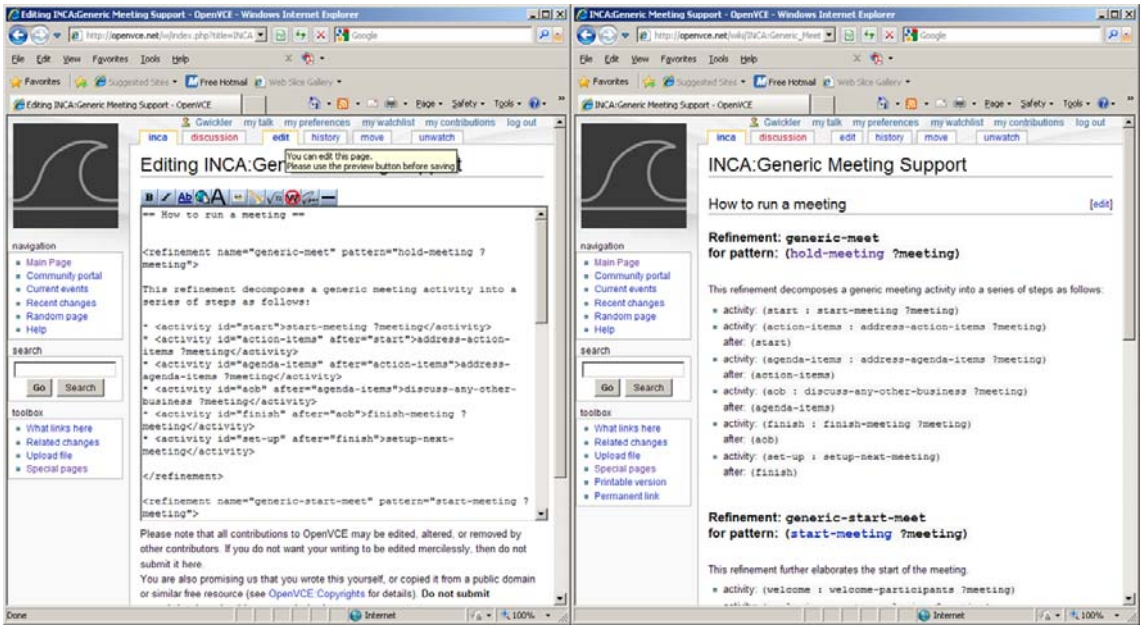


Figure 2. Procedural Knowledge in the edit view (left) and normal view (right)

The <I-N-C-A>-extension for MediaWiki also includes a number of browsing facilities that allow users to explore procedural knowledge, e.g. by searching for procedures by objective, or grouping refinements (methods) into domains. For more details on these features see [Wickler et al., 2013].

Plan Generation

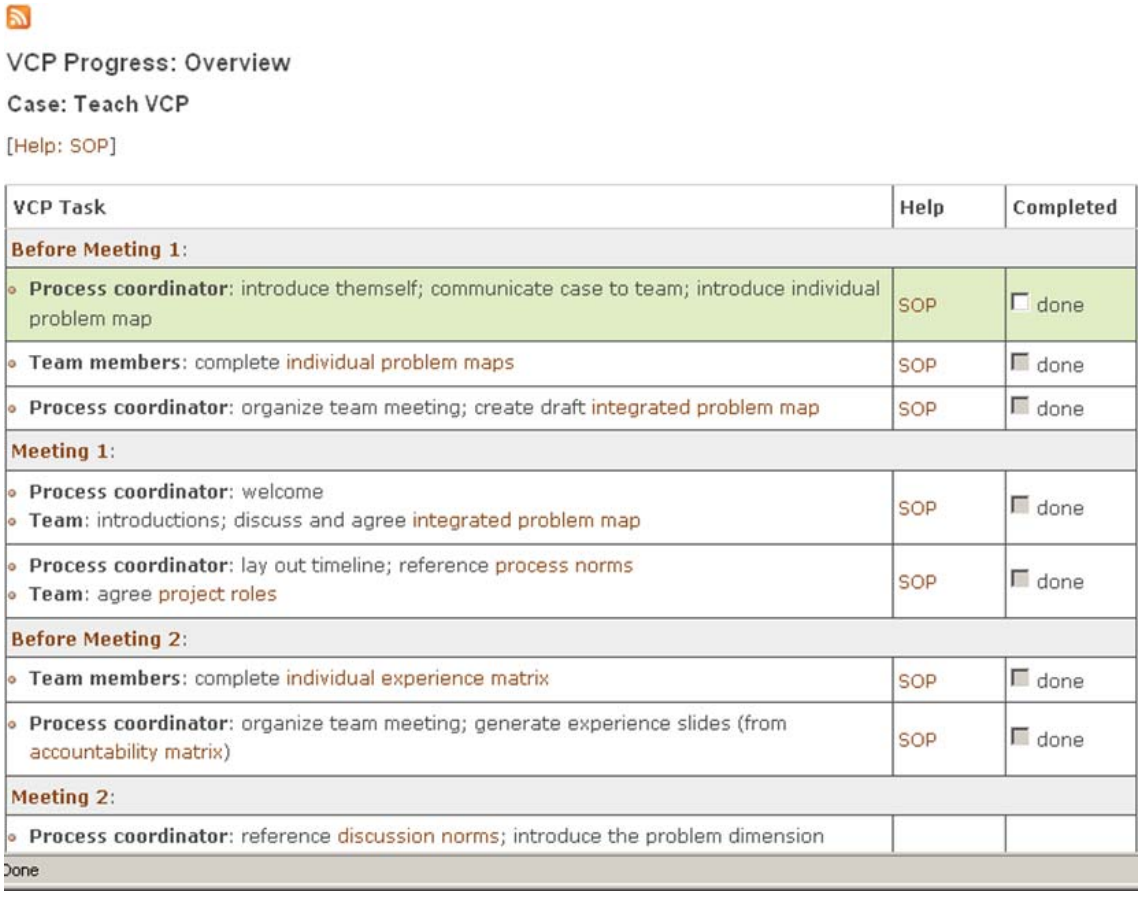
The reason for having the formal aspect of the representation is, of course, that we want to use this knowledge to generate plans. To this end, a so-called special page is part of the MediaWiki extension that allows users to look at the <I-N-C-A> elements for a specific procedure or domain. This triggers another syntax transformation, this time into exactly the syntax that is expected by the HTN planner which is part of the I-X framework.

Developers can use this view for debugging, but it is intended to be directly accessed by the planner which loads the domain directly from the wiki. The planner can then be used to generate plans that are appropriate for a given situation and which always takes into account the latest and complete procedural knowledge available.

Generic Deployment: Using the Plans

The I-X framework supports not only plan generation, but it also supports the management of plan execution though a generic interface, the I-X Process Panel (I-P2), which views a plan as an intelligent, distributed to-do list [Potter et al., 2006]. The idea of using a to-do list as a basis for a distributed task manager is not new [Kreifelts et al., 1993]. However, I-X goes well beyond this metaphor and provides a number of useful extensions that facilitate the finding and adaptation of a complete and efficient course of action.

The I-P2 was developed as an <I-N-C-A> viewer that shows the four components of an <I-N-C-A> element: issues that need to be addressed, nodes in the plan, constraints on these nodes, and annotations for additional information (such as a URL to the wiki page that defines the corresponding element). The nodes are the activities to be done. They can be ticked off in the interface just like in any to-do list. The intelligence of the panel stems from the procedural knowledge which can be applied to items on the to-do list in a mixed-initiative planning mode. It is distributed as there are multiple panels for multiple agents and activities can be passed between panels according to a capability model. The I-P2 has been used successfully in a number of emergency response applications, e.g. the Co-OPR project [Wickler et al., 2007].



VCP Progress: Overview

Case: Teach VCP

[Help: SOP]

VCP Task	Help	Completed
Before Meeting 1:		
◦ **Process coordinator**: introduce themself; communicate case to team; introduce individual problem map	SOP	☐ done
◦ **Team members**: complete individual problem maps	SOP	☐ done
◦ **Process coordinator**: organize team meeting; create draft integrated problem map	SOP	☐ done
Meeting 1:		
◦ **Process coordinator**: welcome ◦ **Team**: introductions; discuss and agree integrated problem map	SOP	☐ done
◦ **Process coordinator**: lay out timeline; reference process norms ◦ **Team**: agree project roles	SOP	☐ done
Before Meeting 2:		
◦ **Team members**: complete individual experience matrix	SOP	☐ done
◦ **Process coordinator**: organize team meeting; generate experience slides (from accountability matrix)	SOP	☐ done
Meeting 2:		
◦ **Process coordinator**: reference discussion norms; introduce the problem dimension		

Done

Figure 3. To-Do List in OpenVCE Website

The deployment support in OpenVCE uses the same idea as the I-P2, it presents the user with a hierarchical to-do list representation of the plan. The main difference is that the interface is not a stand-alone application, but is integrated into the OpenVCE dynamic website. While this version of the I-P2 is not as feature rich as the stand-alone version implemented in Java, it is fully integrated with OpenVCE allows the adding of specific code that can be invoked from the to-do list, e.g. to generate information shown on the screen in the virtual world, as shown in figure 1. Furthermore, the refinements used can be accessed directly via links to the wiki as shown in figure 3, thus providing a quick reference for the procedural knowledge.

Related Work

A wiki that supports procedural knowledge is available at wikihow.com, but the knowledge remains essentially informal. A system that uses a similar approach, namely, representing procedural knowledge in a wiki is CoScripter [Leshed et al., 2008]. However, their representation is not based on AI planning and thus does not support the automated composition of procedures. The Incidone system [Lijnse et al., 2012] uses Task-Oriented Programming to represent and use procedural knowledge in emergency response, but the representation is closer to the specific programming language used.

KNOWLEDGE ENGINEERING FOR PLANNING

The MediaWiki extension for procedural knowledge can support the knowledge engineering process by performing a domain analysis. Apart from the syntax, this performs a semantic analysis in terms of domain features described below. The basic idea here is that authors of procedural knowledge can specify these features. The values these features take for a given domain can also be computed independent of their explicit specification. A comparison of the computed features to the ones specified in the formal domain definition can then be used to validate the formalization, thus supporting the domain author in producing a consistent domain.

Specifying a planning domain and a planning problem in a formal description language defines a search space that can be traversed by a state-space planner to find a solution plan. It is well known that this specification process, also known as problem formulation [Russell and Norvig, 2003], is essential for enabling efficient problem-solving though search [Amarel, 1968].

Domain Features

We will now formally define some of the domain features that can be used to assist knowledge engineers during the problem formulation process, i.e. the authoring of a planning domain which defines the state space. The features used in the domain analysis are: domain types, relation fluency, inconsistent effects and reversible actions. These features are not new, at least at an informal level. Their specification is either already part of PDDL [Fox and Long, 2003] or could easily be added to the language. Applying this approach to various planning domains shows that the features defined here can be used to identify certain representational problems.

Related Work

Amongst the features mentioned above, domain types have been discussed most in the planning literature. A rigorous method for problem formulation in the case of planning domains was presented in [McCluskey and Porteous, 1997]. In the second step of their methodology types are extracted from an informal description of a planning domain. Types have been used as a basic domain feature in TIM [Fox and Long, 1998]. Their approach exploits functional equivalence of objects to derive a hierarchical type structure. This work has later been extended to infer generic types such as mobiles and resources that can be exploited to optimize plan search [Coles and Smith, 2006].

The distinction between rigid and fluent relations [Ghallab et al., 2004] is common in AI planning and will be discussed only briefly. Inconsistent effects of different actions are exploited in the Graphplan algorithm [Blum and Furst, 1995] to define the mutex relation. However, this is applied to pairs of actions (i.e. fully ground instances of operators) rather than operators. Reversible actions, as a domain feature, are not related to regression of goals, meaning this feature is unrelated to the direction of search (forward from the initial state or regressing backwards from the goal). The reversibility of actions (or operators) does not appear to feature much in the AI planning literature. However, in generic search problems they are a common technique used to prune search trees [Russell and Norvig, 2003].

Preprocessing of planning domains is a technique that has been used to speed up the planning process [Dawson and Siklossy, 1977]. Perhaps the most common preprocessing step is the translation of the STRIPS (function-free, first-order) representation into a propositional representation. An informal algorithm for this is described in [Ghallab et al., 2004, section 2.6].

TYPE INFORMATION

Many planning domains include explicit type information. In PDDL the :typing requirement allows the specification of typed variables in predicate and operator declarations. In problem specifications, it allows the assignment of constants or objects to types. If nothing else, typing tends to greatly increase the readability of a planning domain. However, it is not necessary for most planning algorithms to work.

In this section we will show how type information can be inferred from the operator descriptions in the planning domain definition. If the planning domain includes explicit type information the inferred types can be used to perform a consistency check, thus functioning as a knowledge engineering tool. In any case, type information can be used to simplify parts of the planning process. For example, if the planner needs to propositionalize the planning domain, type information can be used to limit the number of possible values for variables, or a ground backward searcher may use this information to similar effect.

The formalism that follows is necessary to show that the derived type system is maximally specific given the knowledge provided by the operators, that is, any type system that further subdivides a derived type must necessarily lead to a search space that contains type inconsistent states.

The simplest kind of type system often used in planning is a *type partition* $\langle C,T,\tau \rangle$ in which the set of all constants C used in the planning domain and problem is divided into disjoint types T and the function τ defines the type of each constant. That is, each type corresponds to a subset of all constants and each constant belongs to exactly one type. This is the kind of type system we will look at here.

A type partition divides the set of all constants that may occur in a planning problem into a set of equivalence classes. The availability of a type partition can be used to limit the space of world states that may be searched by a planner. In general, a world state in a planning domain can be any subset of the powerset of the set of ground atoms over predicates *P* with arguments from *C*. We shall now define a type system that is derived from the operator descriptions in the planning domain.

Let *P* be the set of all the predicate symbols used in all the operators. A *type name* is a pair ‹*N,k*› where *N* is a predicate (name) and *k* is an integer. A type name can be used to refer to a type in a derived type system. There usually are multiple names to refer to the same type. The basic idea behind the derived types is to partition the set of all type names into equivalence classes, and then assign constants used in a planning problem to different equivalence classes, thus treating each equivalence class as a type.

Such a derived type shall be called an *O-type*, and it is simply a set of type names. Two type names ‹N_1, k_1› and ‹N_2,k_2› are equivalent (they belong to the same O-type) if:

- N_1 is an operator name and v is its k_1th parameter, and N_2 is a precondition or effect predicate of the operator and v is its k_2th parameter; or
- they are in the transitive closure of the above.

Then we can define the (derived) O-type partition as consisting of the constants that occur in a planning problem, the O-types just defined, and the function τ that assigns to each constant c the O-type that contains all those pairs ‹*N,k*› that describe an occurrence of c in a relation-instance defined in the planning problem. Note that $\tau(c)$ is not necessarily well-defined for every constant mentioned in the initial state, e.g. if a constant is used in two relations that would indicate different derived types (which rely only on the operator descriptions). In this case the O-type partition cannot be used as defined above. However, if appropriate unions of O-types are taken then this results in a new type partition for which $\tau(c)$ is defined. In the worst case this will lead to a type partition consisting of a single type. Given that this approach is always possible, we shall now assume that $\tau(c)$ is always defined.

Then we can show that the following proposition must hold: *Let ‹C,T,τ› be the O-type partition derived from a planning problem. Then every state that is reachable from the initial state is type consistent.* The proof is fairly straight-forward and can be examined in [Wickler, 2011]. This shows that the type system derived from the operator definitions is indeed useful as it creates a state space of type consistent states. However, the question that remains is whether it is the best or even only type system. Clearly, there may be other type systems that give us type consistent state space. The system that consists just of a single type is a trivial example. A better type system would divide the set of constants into more types though, as this reduces the size of a type consistent state space.

We will now show that the above type system is maximally specific given the knowledge provided by the operators: *Let ‹C,T,τ› be the O-type partition derived from a problem. If two constants c_1 and c_2 have the same type $\tau(c_1) = \tau(c_2)$ then they must have the same type in every type partition that creates a type consistent search space*. Again, the full proof is omitted here, but can be found in [Wickler, 2011]. What this shows is that this and only this derived type system is the most specific set of types that result in a consistent search space.

INCONSISTENT EFFECTS

In a STRIPS-style operator definition the effects are specified as and add- and delete-lists consisting of a set of (function-free) first-order atoms, or a set of first-order literals where positive elements correspond to the add-list and negative elements correspond to the delete-list. Normally, the definition of an operator permits potentially inconsistent effects, i.e. a positive and a negative effect may be complementary.

It is fairly common for planning domains to define operators with potentially inconsistent effects. For example, the move operator in the DWR domain is defined as follows:

```
(:action move :parameters (?r ?fr ?to)
        :precondition (and (adjacent ?fr ?to) (at ?r ?fr) (not (occupied ?to)))
        :effect (and (at ?r ?to) (occupied ?to) (not (occupied ?fr)) (not (at ?r ?fr))))
```

This operator has a positive effect (at ?r ?to) and a negative effect (at ?r ?fr). These two effects are unifiable and represent a potential inconsistency. Since this is a common feature in planning domains there is no need to raise this to the domain author. Effects that are necessarily inconsistent may be more critical. We shall say that an operator has *necessarily inconsistent* effects iff it has a positive effect and a negative effect which are equal.

Given the definition of the state-transition function for STRIPS operators [Ghallab et al., 2004] it should be

clear that the negative effect can be omitted from the operator description without changing the set of reachable states. Thus, the presence of the negative effect does not change the range of the state-transition function.

From a knowledge engineering perspective this means that an operator with necessarily inconsistent effects indicates a problem and should be raised to the domain author. However, this is only true for simple STRIPS operators where actions are instantaneous and thus, all effects happen simultaneously. If effects are permitted at different time points then only those that are necessarily inconsistent at the same time point must be considered a problem.

Since actions are ground instances of operators, there is no need to distinguish between necessarily and potentially inconsistent effects. All effects must be ground for actions and therefore inconsistent effects are always necessarily inconsistent. Even if necessarily inconsistent operators are not permitted in a domain, actions with inconsistent effects may still occur as instances of operators with potentially inconsistent effects.

Whether it is desirable for the planner to consider such actions depends on the other effects of the action. For example, in the DWR domain no action with inconsistent effects needs to be considered. However, if an action has side effects then it may make sense to permit such actions. For example, circling an aircraft in a holding pattern does not change the location of the aircraft, but it does reduce the fuel level. If such side effects are important actions with inconsistent effects may need to be permitted. And, of course, every action has the side effect of taking up a step in a plan.

REVERSIBLE ACTIONS

A common feature in many planning domains (and in many classic search problems) is that they contain actions that can be reversed by applying another action. There is usually no need to consider such actions during the search process. The idea here is to apply the concept of reversibility to operators: an operator may be reversed by another operator (or the same operator), possibly after a suitable substitution of variables occurring as parameters in the operator definition. Note that this definition is somewhat narrow as it demands this pattern to be consistent across all instances of the two operators, i.e. it excludes the possibility of an operator sometimes being reversed by one operator, and sometimes by another, depending on the values of the parameters.

An action a that is applicable in a state s is reversed by an action a' if the state that results from applying the sequence aa' in s results in s, i.e. the state remains unchanged. An operator O is reversed by an operator O' under substitution σ' iff for every action $a = \sigma(O)$ that is an instance of O, if a is applicable in a state s then $a' = \sigma(\sigma'(O'))$ is applicable the resulting state and the result is again s.

For example, consider the (move ?r ?l1 ?l2) operator from the DWR domain. This can be reversed by another move operation with different parameters, i.e. (move ?r ?l1 ?l2) is reversed by (move ?r ?l2 ?l1). While this definition captures the idea of a reversing operator, it is not very useful from a computational point of view. Another way to avoid exploring states that are the result of the application of an action followed by its reverse action is to store all states in a hash table and test whether the new state has been encountered before, an approach that is far more general than just testing for reversing actions. Computationally, it is roughly as expensive as the test suggested by the above definition. The key here is that both are state specific. A definition of reversibility that does not depend on the state in which an action is applied would be better.

From a domain author's perspective, it is often possible to specify which operators can be used to reverse another operator, as we have shown in the DWR move example above. If this information is available during search then there is no need to apply the reverse action, generate the state, and compare it to the previous state. Instead a relatively simple substitution test would suffice. Let O_1 be an operator that is reversed by O_2 under some substitution. Then the two sets of positive/negative effects must cancel each other under that substitution.

This means we can let the domain author specify reversing operators and then use the above necessary criterion for validation. Or we could treat the above criterion as sufficient and thus exclude a portion of the search space. This may lead to incompleteness in the search, but the domains we have used for our evaluation do not show this problem.

EVALUATION

The problem with evaluating the support given to knowledge engineering is that one should not use domains that were produced for the evaluation. Any flaw in the knowledge engineering that could be discovered by the method described here would have to be built into the domains and hence the evaluation would be contrived. To evaluate the domain features we have applied them to a small number of planning domains. To avoid any bias

we used only planning domains that were available from third parties, mostly from the international planning competition. Since our algorithms work on domains and the results have to be interpreted manually only a limited number of experiments was possible. Random domains are not suitable as they cannot be expected to encode an implicit type system.

A planning domain on which the algorithm has been used is the DWR domain [Ghallab et al., 2004]. In this domain types are defined explicitly, so it was possible to verify consistency with the given types. The algorithm produced the following, listing the argument positions in predicates where they are used:

type: [loaded-0, unloaded-0, at-0]
type: [attached-0, top-1, in-1]
type: [occupied-0, attached-1, belong-1,adjacent-1, adjacent-0, at-1]
type: [belong-0, holding-0, empty-0]
type: [loaded-1, holding-1, on-1, on-0, in-0, top-0]

The first type states that it is used as the first argument in the loaded, unloaded and at predicate. This corresponds exactly to the robot type in the PDDL specification of the domain. Similarly, the other types correspond to pile, location, crane and container, in this order. The main difference is that the derived types do not have intelligible names.

The other domains that were used for testing did not come with type information specified in the same way as the DWR domain. However, they all use unary predicates to add type information to the preconditions (but not every unary predicate is a type). The domains used are the following STRIPS domains from the international planning competition: movie, gripper, logistics, mystery, mprime and grid. The algorithm derives between 3 and 5 types for each of these domains which appear consistent with what the domain authors had in mind. The only domain that stands out is the first, in which each predicate has its own type. However this appears to be appropriate for this very simple domain.

If actions with inconsistent effects are considered by the planner, this may lead to further complications. This is because the definition of the state-transition function first subtracts negative effects from a state and then adds positive effects. For actions that have no inconsistent effects this order is irrelevant. However, if actions with inconsistent effects are permitted the result may be surprising. For example, returning to the move operator in the DWR domain, this has been defined with a positive effect (occupied ?to) and a negative effect (occupied ?fr). Thus, the action (move r loc loc) will result in a state in which (occupied loc) holds. Now suppose the domain had been defined using the predicate free instead of occupied. In this case the result of (move r loc loc) would result in a state in which (free loc) holds. This problem occurs only with inconsistent effects.

Looking at reversible actions, we have made an even stronger assumption to carry out some experiments with the domains mentioned above: we have assumed that there is at most one operator that reverses a given operator. We have then, for each domain, done a pairwise test on all the operators defined in the domain to see whether the necessary criterion holds. This resulted in discovering that the move operator can be reversed by itself with a substitution automatically derived from the operator definition, and similarly it discovered the reversibility between the take and put operators and the load and unload operators in the DWR domain.

Perhaps surprisingly, the unique reversibility was not given for all domains. The logistics domain contains load and unload operators for trucks and airplanes. These are specified as four distinct operators. However, in terms of their effects the two load operators and the two unload operators cannot be distinguished. The only difference lies in the preconditions where the ?truck parameter is required to be a truck and the ?airplane parameter is required to be an airplane.

This result can be interpreted in two ways: one could argue that the necessary condition may not be used as sufficient in this domain. Or one could argue that this domain contains redundancy that can be removed by merging the two load and unload operators, which would not change the set of reachable states in this example but means the planner has fewer actions to consider. Either way, testing for the necessary reversibility condition has highlighted this domain feature.

CONCLUSIONS

This paper has described the OpenVCE environment for virtual collaboration. This environment benefits from a number of results achieved in AI planning. Firstly, the MediaWiki extension uses a mixture of formal and informal aspects to represent procedural knowledge. The formal representation is based directly on HTN representations used in AI for a long time, and more specifically, it is based in the <I-N-C-A> ontology. Behind the scenes the environment also makes use of a planner to generate to-do lists. The user does not interact with

the planner directly, although the planner has been used in mixed initiative mode, indicating a possible future extension of the system. The contribution of OpenVCE to AI planning lies in the support for knowledge engineering and the features used for the domain analysis, and we will look at these in more detail below. Finally, the environment overlaps with research in AI planning in its execution support, which is provided through the I-X-like to-do list integrated into the website.

The first feature, the type system, is a rather simple, flat division into equivalence classes. This may not be suitable for very complex planning domains, but the domains we have analyzed do not exhibit much hierarchical structure. The advantage of such a type system is that it can be easily added to the operator descriptions in the form of unary preconditions. Furthermore, we showed that the type system derived by our algorithm is the most specific type system of its kind based solely on the operator descriptions. An open question is whether this is identical to the least general generalization [Plotkin, 1969] used in machine learning. The algorithm could be refined to derive a hierarchical type system if one takes into account the directionality of the operators, but for a type system consisting of equivalence classes this is irrelevant. Also, the algorithm described in this paper should also be applicable to hierarchical task network domains, but this has not yet been implemented.

Actions with inconsistent effects are another feature we have defined. For most domains, such actions are probably not desirable. In fact, the admission of such actions leads to a different planning problem as the state spaces with or without such actions may be different for the same planning domain and problem. Also, planners that translate a STRIPS planning problem (with negative preconditions) into a propositional problem (without negative preconditions) need to be more careful if actions with inconsistent effects are permitted. The translation method described in [Ghallab et al., 2004, section 2.6] does not work in this case as it introduces independent predicates for a predicate and its negations, which can become true in the same state if an action with inconsistent effects is applied. This would render the planner potentially unsound.

The final feature which defines reversible actions is somewhat different as it can only be usefully used as a necessary criterion to test whether one operator is the reverse of another. The more strict, sufficient definition does not provide any computational advantage. The difference is simply that the necessary criterion can be computed on the basis of the operator descriptions, whereas the sufficient test requires knowledge of the state in which an action is applied. The difference is quite subtle though, and may not matter in practice. The necessary criterion requires the positive and negative effects to cancel each other. However, if a state contains an atom that is also added by the first action, but then deleted by the second action, then the state will be changed. If an operator listed all the relevant atoms also as preconditions, this exception would not hold.

The OpenVCE system was evaluated in a viral outbreak scenario involving around 20 experts divided into two groups, only one of which was using OpenVCE with its procedural knowledge. The complexity of the technology was addressed in a short, one-hour overview session itself delivered in Second Life. Furthermore, technical support was available during the experiment to address specific issues. Despite the technological challenges, the experiment showed that OpenVCE with its built-in procedural knowledge can improve procedural uncertainty. As for the technological challenges, these were usually related to firewall issues and audio setup problems, which only needed to be solved once.

ACKNOWLEDGEMENTS

Effort sponsored by USJFCOM-Army Research Labs parent contract DAAD19-01-C-0065, subcontract no. SFP1196749DP (via Alion Science and Technology), task order no. 118, and the Air Force Office of Scientific Research, Air Force Material Command, USAF, under grant number FA8655-09-1-3090. The University of Edinburgh and research sponsors are authorized to reproduce and distribute reprints and on-line copies for their purposes notwithstanding any copyright annotation hereon.

REFERENCES

1. James Allen, James Hendler, and Austin Tate, editors. Readings in Planning. Morgan Kaufman, 1990.
2. Saul Amarel. On representations of problems of reasoning about actions. In Donald Michie, editor, Machine Intelligence 3, pages 131–171. Elsevier/North-Holland, 1968.
3. Daniel J. Barrett. MediaWiki. O'Reilley, 2009.
4. Avrim L. Blum and Merrick L. Furst. Fast planning through planning graph analysis. In Proc. 14th International Joint Conference on Artificial Intelligence (IJCAI), pages 1636–1642. Morgan Kaufmann, 1995.

5. Andrew Coles and Amanda Smith. Generic types and their use in improving the quality of search heuristics. In Proc. 25th Workshop of the UK Planning and Scheduling Special Interest Group (PlanSIG 2006), 2006.
6. Damon P. Coppola. Introduction to International Disaster Management. Butterworth-Heinemann, 2006.
7. Robert Cross and Robert Thomas. Driving Results through Social Networks. Jossey-Bass, 2009.
8. Clive Dawson and Laurent Siklossy. The role of preprocessing in problem-solving systems. In Proc. 5th international Joint Conference on Artificial Intelligence (IJCAI), pages 465–471. Morgan Kaufmann, 1977.
9. Maria Fox and Derek Long. The automatic inference of state invariants in TIM. Journal of Artificial Intelligence Research, 9:367–421, 1998.
10. Maria Fox and Derek Long. PDDL2.1 : An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research, 20:61–124, 2003.
11. Malik Ghallab, Dana Nau, and Paolo Traverso. Automated Planning. Morgan Kaufmann, 2004.
12. Jeffrey Hansberger. Understanding distributed collaboration within virtual worlds. In Proc. Sunbelt XXX Social Network Analysis Conference, 2010.
13. Th. Kreifelts, E. Hinrichs, and G. Woetzel. Sharing To-Do lists with a distributed task manager. In G. Michelis and C. Simone, editors, Proc. 3rd European Conference on Computer Supported Cooperative Work, pages 31–46, 1993.
14. Gilly Leshed, Eben Haber, Tara Matthews, and Tessa Lau. CoScripter: Automating and sharing how-to knowledge in the enterprise. In Proc. ACM Conf. Human Factors in Computing Systems (CHI), 2008.
15. Bas Lijnse, Jan Martin Jansen, and Rinus Plasmeijer. Incidone: A task-oriented incident coordination tool. In Leon Rothkrantz, Jozef Ristvej, and Zeno Franco, editors, Proc. 9th Int. Conf. on Information Systems for Crisis Response and Management (ISCRAM), 2012.
16. T.L. McCluskey and J.M. Porteous. Engineering and compiling planning domain models to promote validity and efficiency. Artificial Intelligence, 95:1–65, 1997.
17. Gordon Plotkin. A note on inductive generalization. In Bernard Meltzer and Donald Michie, editors, Machine Intelligence 5, pages 153–164. Edinburgh University Press, 1969.
18. Stephen Potter, Austin Tate, and Gerhard Wickler. Using I-X process panels as intelligent to-do lists for agent coordination in emergency response. In Proc. 3rd Int. Conf. on Information Systems for Crisis Response and Management (ISCRAM), 2006.
19. Stuart J. Russell and Peter Norvig. Artificial Intelligence: A Modern Approach. Prentice Hall, 2nd edition, 2003.
20. Austin Tate, Stephen Potter, and Jeffrey Dalton. I-Room: a virtual space for emergency response for the multinational planning augmentation team. In James Lawton, Jitu Patel, and Austin Tate, editors, Proc. 5th Int. Conf. on Knowledge Systems for Coalition Operations (KSCO), 2009.
21. Austin Tate. Generating project networks. In Proc. 5th International Joint Conference on Artificial Intelligence (IJCAI), pages 888–893. Morgan Kaufmann, 1977. Reprinted in [Allen et al., 1990, pages 291–296].
22. Austin Tate. <I-N-C-A>: A shared model for mixedinitiative synthesis tasks. In Gheorghe Tecuci, editor, Proc. IJCAI Workshop on Mixed-Initiative Intelligent Systems, pages 125–130, 2003.
23. Gerhard Wickler, Austin Tate, and Jeffrey Hansberger. Supporting collaborative operations within a coalition personnel recovery center. In Proc. 4th Knowledge Systems for Coalition Operations (KSCO), pages 14–19, 2007.
24. Gerhard Wickler, Austin Tate, and Jeffrey Hansberger. Using shared procedural knowledge for virtual collaboration support in emergency management. IEEE Intelligent Systems, 2013.
25. Gerhard Wickler. Using planning domain features to facilitate knowledge engineering. In Proc. Knowledge Engineering for Planning and Scheduling (KEPS), 2011.