McCluskey, T.L., Jarvis, Peter and Kitchin, Diane E.

OCLh: A Sound and Supportive Planning Domain Modelling Language

Original Citation

McCluskey, T.L., Jarvis, Peter and Kitchin, Diane E. (1999) OCLh: A Sound and Supportive Planning Domain Modelling Language. Technical Report. University of Huddersfield. (Unpublished)

OCL_h: a sound and supportive planning domain modelling language

T. L. McCluskey
Department of Computing Science
University of Huddersfield, UK

P.Jarvis*
AIAI
University of Edinburgh, UK

D. E. Kitchin
Department of Computing Science
University of Huddersfield, UK

November 2, 1999

Abstract

In this paper we postulate OCL_h as a prototype for future planning domain modelling languages which are foundationally sound, but offer features that are attractive and supportive to knowledge engineers. The novel contributions of this paper is that it (a) describes a truth ctiterion for OCL_h and details a proof that the criterion is sufficient for ensuring necessary truth in a partial plan structure (b) evaluates OCL_h, illustrating its pragmatic benefits by comparing it with O-Plan's *TF*. We show using a real example how OCL_h's structuring devices aid the knowledge engineer in building a model. Finally, the example and comparison with *TF* identifies further development work to advance OCL_h as potential high level research language for modelling operator based planning domains.

Keywords:
Domain Modelling, Planning Language, Truth Criterion

---

*Peter Jarvis is funded under the O-Plan project. The O-Plan project is sponsored by the Defence Advanced Research Projects Agency (DARPA) and the U.S. Air Force Research Laboratory (AFRL) under grant number F30602-99-1-0024. The U.S. Government is authorized to reproduce and distribute reprints for governmental purposes notwithstanding any copyright annotation hereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either express or implied, of DARPA, AFRL or the US Government.

1 Introduction

Knowledge acquisition for planning has received increasing attention in the last few years with the appearence of workshops at AIPS98 [1] and through the PLANET initiative[1] [13]. One problem identified is that languages designed for use with realistic systems tend to be theoretically opaque - it is not easy to give operators a clear semantics as hierarchical operators are context-dependent. Although some progress has been made in this area [21, 5, 11], 'clean' representation languages still fall short of the richness apparently required for applications [14], and are generally not designed with the knowledge engineering task in mind. In order to be able to investigate existing and novel planning techniques, and their scaling up to knowledged-based applications, one needs to encode domains in a language that is clear and well founded, aids the knowledge engineer in the knowledge acquisition and maintenance task, and is oriented towards planning applications.

In this paper we postulate OCL_h as a prototype for future planner domain modelling languages that are foundationally sound, but offer features that are attractive and supportive to knowledge engineers. OCL_h is a language, with a supporting method, which has been designed for encoding domains for both classical precondition planners and HTN planners. The rationale for an *object-centred* approach to encoding planning domains was proposed in reference [12]. A full method for the model building process was described, including the establishment of various model properties, supported by a set of tools to support the engineering process. While this defined the base language, *OCL* was later extended to OCL_h to include an extension for HTN models. Along with desirable properties of OCL_h encodings, an algorithm to check domain descriptions for the absense of these properties was introduced [11]. Further details of the language is available in reference [10], and OCL_h encodings of planning domains including an HTN transport logistic domain can be found on the web[2].

In the first part of the paper we start by briefly reviewing the constructs of OCL_h. Next we define a truth ctiterion for OCL_h, show that it is is sufficient, and can be used as the basis for a sound goal achievement algorithm. In the second part of the paper we illustrate the pragmatic benefits of OCL_h by comparing it with O-Plan's TF, and show using a real example how OCL_h's structure aids the knowledge engineer to build a model.

2 Foundations of OCL_h

A domain modeller using OCL_h aims to construct a model $\mathcal{M}$ of the domain in terms of objects, a sort hierarchy, predicate definitions, substate class definitions, invariants, and operators. Predicates and objects are classed as dynamic or static as appropriate - dynamic predicates are those which may have a changing truth value throughout the course of plan execution, and dynamic objects (grouped into dynamic sorts) are each associated with a changable state. Each object in $\mathcal{M}$ belongs to a unique *primitive sort* s, where members of s all behave the same under operator application. For example, in a transport domain the writer might start by defining objects and a

[1] online proceedings at http://www.aiai.ed.ac.uk/ paj/planning/planet/ka-tcu/99-04-workshop.htm

[2] http://www.hud.ac.uk/scom/research/Artform/resources.html

simple sort hierarchy as shown in Example 1. This shows the way 'sorts' definitions construct the hierarchy, and in particular how (static) sorts can be defined in terms of aggregation and recursion.

$sorts(item, [box, crate])$ $sorts(vehicle, [truck])$ $sorts(load, [empty, add(item, load)])$
$objects(box, [box\text{-}1, box\text{-}2, box\text{-}3])$ $objects(truck, [truck\text{-}1, truck\text{-}2])$

Example 1

Variables can appear in predicates in various components of a model. In this paper they are represented by a capital letter, and are associated with a sort s given by the predicate definition part of $\mathcal{M}$. A legal substitution of a variable is the replacement of a variable by a term which has s as a 'supersort'.

OCL_h is based on the assumption that the state of the world in a planning application can be decomposed into the state of each object (a 'substate') in that world. A **substate** ss describes the state of an individual dynamic object in a planning world. It is fully described as a tuple ss with components (i, s, e), where $ss.i$ is the object's identifier, $ss.s$ is the primitive sort of $ss.i$, and $ss.e$ is a set of ground dynamic predicates which all *refer* to $ss.i$[3]. All predicates in $ss.e$ are asserted to be true under a locally closed world assumption; informally, these means that any instances of predicates referring to $ss.i$ not included in $ss.e$, but which may be used in the description of another object of sort $ss.s$, are false.

$(box\text{-}1, box, [at(box\text{-}1, depot\text{-}2), waiting(box\text{-}1)])$ $(box\text{-}2, box, [in(box\text{-}2, truck\text{-}1)])$
$(truck\text{-}1, truck, [loaded(truck\text{-}1, add(box\text{-}3, add(box\text{-}2, empty))), unavaliable(truck\text{-}1), fuel(truck\text{-}1, full)])$

Example 2

A **world state** is a complete set of substates for all the dynamic, primitive objects in the planning application. Three substates that could form part of a world state are shown in Example 2. Here the local closed world assumption tells us that, for example, $waiting(box\text{-}2)$ is false. States are constrained by **invariants**. These define the truth value of static predicates and the relationships between dynamic predicates. In particular they are used to record inconsistency constraints. A world state that satisfies the invariants is called well-formed.

For each sort s, the domain modeller groups object substates together, specifying each group with a set of predicates called a **substate class expression**. When ground, each expression always decribes a unique, legal substate, and together the substate class expressions should form a complete, disjoint covering of the space of substates for objects of s. For example, the substates of a *truck* may fall into the three classes given by the first definition of Example 3.

Substate classes are normally specified at various levels in the sort hierarchy. For example, objects of sort *truck* have classes specified through their primitive sort but they also inherit the dynamic predicate $fuel(truck, fuel\_level)$ from supersort *vehicle*. A substate of an object, therefore, may have up to *n hierarchical components* representing its primitive sort (s_1) and $n - 1$ supersorts $s_2, ..., s_n$. In general therefore, the hierarchical substate class expression for an object

[3] in previous publications we identified substates and substate expressions with their predicate descriptions: here, for clarity, we decorate then with the name of the object they are attached to and the primitive sort of that object

of primitive sort s is the conjunction $h_1 \& h_2 \& ... \& h_n$ where each h_j is one component of sort s_j's substate class expressions.

$$substate\_classes(truck, [[loaded(T,L), unavailable(T), less\_than(L,5)],$$
$$[unloaded(T), unavailable(T)], [unloaded(T), available(T)]])$$
$$substate\_classes(vehicle, [[fuel(V,A)]])$$

Example 3

To ensure that *any* legal ground instantiation of a substate class expression gives a legal substate, they may contain 'static' predicates. So, for example, predicate *less_than* limits the load of the truck to be up to 4 objects. A more elaborate example of an object hierarchy and a set of class expressions that will be used in the discussion later is shown in Example 4.

1.$sorts(physical\_obj, [transport])$

2.$sorts(transport, [small\_scale\_transporter, large\_scale\_transporter])$

3.$sorts(small\_scale\_transporter, [ground\_transporter])$

4.$sorts(large\_scale\_transporter, [people\_transporter, cargo\_transporter])$

5.$objects(ground\_transporter, [gt1, gt2])$

6.$objects(cargo\_transporter, [c5])$

7.$objects(people\_transporter, [b707])$

8.$substate\_classes(physical\_obj, [[at(O,L]])$

9.$substate\_classes(transport, [[unloaded(T), available(T)], [loaded(T,C), in\_use(T)],])$

10.$substate\_classes(large\_scale\_transporter, [[is\_of\_sort(L, air\_base)],])$

11.$substate\_classes(cargo\_transporter, [[is\_of\_sort(C, equipment)]])$

12.$substate\_classes(people\_transporter, [[is\_of\_sort(C, people)]])$

13.$substate\_classes(ground\_transport, [[driving\_between(T,A,B],])$

Example 4

Here the hierarchy gives a definition of some of the sorts, objects and substate classes in the Pacifica domain[4]. The hierarchy imposes constraints on final substates using the special static predicate *is_of_sort*, so that a substate for the cargo transpoter $c5$ would be any legal grounding of:

$$[at(c5,L), loaded(c5,C), in\_use(c5)]$$

where C and L belong to the primitive sorts *equipment* and *air_base* respectively.

Primitive Action Representation

Let P be the set of all possible predicate structures in the model (where an argument of a predicate can contain any appropriate object identifier, variable or legally-formed term). If $z, z' \in P$, for z and z' to be 'equal' we assume they must be identical. For example, if x, y, z are sort variables,

[4] available from http://www.aiai.ed.ac.uk/ oplan/web-demo/show-tf.cgi/pacifica.tf

p a predicate name defined in $\mathcal{M}$, then $p(x, y)$ is distinct from $p(x, z)$. If $z, z' \in P$ then $z' \subseteq z$ means that z' is a subset of z with this definition of equality.

If i is a variable or an object identifier, s is a sort-name, and e is a set of predicates taken from P, then se with components (i, s, e) is called a **substate expression** if $se.s = ss.s$ and there is a legal substitution t such that $se.o_t = ss.o$ and $se.e_t \subseteq ss.e$, for at least one substate ss. A consequence of this definition is that any subset of the predicates in a substate class expression form a substate expression.

A **substate transition** is an expression of the form $(o, s, se \Rightarrow ssc)$ where o is a dynamic object identifier or a variable of sort s, and se and ssc are a substate expression and a substate class expression respectively. In a state containing the substates in example 2, a transition might be:

$$(T, truck, [loaded(T, X] \Rightarrow [loaded(T, add(B, X)), unavaliable(T)])$$

brought about by the loading of another box onto a truck. For each component of se from the *nth* level in the hierarchy, ssc must contain a complete substate class expression component from the nth level. For the levels in the sort hierarchy that are not mentioned in se, it is assumed that predicate desctiptions of the object persists. Since the hierarchical component inherited from the supersort *vehicle* is not referred to in the example, the truck's fuel level remains unaffected by this transition.

Operator Definition: An action in a domain is represented by an either a primitive or hierarchical operator. A primitive operator schema O has components $(id, prev, nec, cond, cons)$, such that $O.id$ is the operator's identifier, $O.prev$ is the prevail condition consisting of a set of substate expressions, $O.nec$ is a set of necessary substate transitions, $O.cond$ is a set of (conditional) substate transitions, and $O.cons$ is a set of static predicates acting as constraints. Each expression in $O.prev$ must be true before execution of O, and, at least in the case of primitive operators, will remain true throughout operator execution.

Operator Execution: A primitive operator O can be **executed** in world state S if there is a substitution sequence t such that
(a) for all $(X, s, L) \in O.prev$, there is a substate $(o, s, E) \in S$ such that $X_t = o$ and $L_t \subseteq E$
(b) for all $(X, s, L \Rightarrow R)$ in $O.nec$, there is some substate $(o, s, E) \in S$ such that $X_t = o$ and $L_t \subseteq E$
(c) $O.cons_t$ is consistent i.e. there is a legal binding u such that $O.cons_{tu}$'s static predicates all evaluate to true in $\mathcal{M}$.
The new world state is S with the changes made as specified in the necessary object transitions, and any other objects changed by the conditional transitions if the *LHS* of the transitions were satisfied in S.

3 A Truth Criterion for Use in Primitive Partial Plans

The rigorous formulation of OCL_h, briefly reviewed above, leads to properties of domain models such as consistency and transparency that have been used as the basis for tool support [11]. Here we show how a truth criterion can be formulated and used as the basis for investigating goal achievement in object-centred goal-directed planners. The truth criterion is *sufficient* for ensuring the truth of a substate expression at a step in a plan, and can be used to ensure a planner accepting OCL_h is sound and complete. More details of this truth criterion and a set of planners that are based on it is given in reference [9].

Assume a plan structure containing only primitive operators $\mathcal{P}$ is any set of steps, temporal constraints, and variable constraints having the form $(steps, tc, vc)$. A step is the occurence of an operator within a plan. A completion of $\mathcal{P}$ is a ground, linear sequence of all the steps in $\mathcal{P}$ which obeys $\mathcal{P}.pc$ and $\mathcal{P}.vc$. In this context $possibly(X = Y)$ means that the term X can be unified to the term Y without making vc inconsistent, and likewise $possibly(A < B)$ means that that temporal link can be added between steps A and B without making tc inconsistent. A sound plan is one in which all the goal conditions (which are posed as substate expressions) and preconditions of operators are necessarily established in the plan. In OCL_h these preconditions are the prevail conditions and the left hand sides of necessary transitions of steps.

In the classical formulation, establishing a literal p at a point t in a plan is often cast as proving the necessary truth of p. There are various planners which embody conditions *sufficient* for established a literal as pointed out in [8]. That is, if the condition evaluates to true in a partial plan structure then the literal will be necessarily true in all completions of the plan. Often, this is carried out in planning by finding a step A before t with p in its effects, and ensuring that that effect is not undone between the temporal position of A and t. In OCL, operators (steps) describe the transitions of objects, rather than the adding and deleting of literals, and substate expressions rather than literals have to be established before steps can be executed.

We give a *sufficient* condition for the necessary truth of a substate expression (X, S, L) before a step O in a plan structure $\mathcal{P}$ in terms of the **established** condition (i.e. if this condition is true then the substate expression will be established in any completion of the plan). Here (X, S, L) could be a member of $O.p$, or L could be the left hand side of some state transition[5] (see figure 1). (X, S, L) is established by step $A \in \mathcal{P}.steps$ if

1. A is necessarily before O and has a necessary transition $(X, S, N \Rightarrow R)$ such that $L \subseteq R$.
2. there is no such step $C \in \mathcal{P}.steps$ such that
   (a) C is possibly in between A and O, and
   (b) C contains a necessary or conditional transition $(Y, S, M \Rightarrow U)$ such that
      i. possibly $X = Y$, and
      ii. if $X = Y$, then either $\neg(L \subseteq U)$ or U and R contain class expression components from the same level of the class hierarchy

Essentially this states that, to check or make a substate expression (X, S, L) true in a developing plan, it is sufficient to make sure it has an establisher (A), and to check that no step possibly inbetween can possibly change the state of the object concerned. The exception is where a step can change the state of X: whenever this happens the change in state will establish the substate expression, *or* the change in state affects a distinct part of X's hierarchy.

[5] we take the liberty of using tuples rather than names with selectors as the resulting discussion is simpler

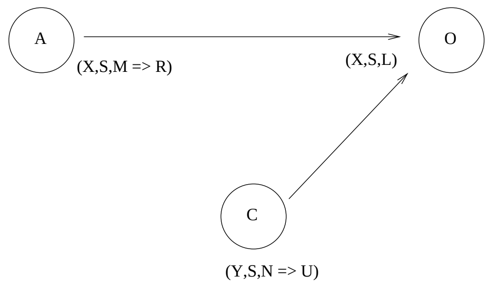

Figure 1: Establishment of a Substate Expression

To show the establish condition is sufficient, assume we have to a partial plan structure $\mathcal{P}$, and a substate expression (X, S, L) within it that satisfies the truth criterion before step O. Further assume that $\mathcal{Q}$ is a completion of $\mathcal{P}$, and (X', S, L') is the ground version of (X, S, L). Then $(X', S, L') = (X, S, L)_t$ for some grounding substitution t. Then:

– step A (used as the establisher in $\mathcal{P}$) is before O in the completion $\mathcal{Q}$ by the definition of 'necessarily before'

– A's transtion $(X, S, N \Rightarrow R)$ will be grounded to $(X', S, N' \Rightarrow R')$ in $\mathcal{Q}$. Since $L \subseteq R$, by definition this holds true for any consistent binding of variables in L and R. Hence the condition $L' \subseteq R'$. is met in $\mathcal{Q}$.

– Assume there is in the completion a C in between A and O which acts as a clobberer. Then C must be in $\mathcal{P}$, and furthermore it must have been possibe to order it in between A and O. For C to be a clobberer in the completion, it must contain $(X', S, M' \Rightarrow U')$ in its transitions such that X' gets translated into a substate that does not satisfy L', that is $\neg(L' \subseteq U')$, and U' must affect at least some of the hierarchical components as R' does. In $\mathcal{P}$, therefore, C must have contained a transition such that $L \subseteq R$ given C translates object X. Hence we get a contradiction, and so in the completion there can be no such clobberer.

Since we have proved that, if any substate expression *se* satisfies the condition in a plan $\mathcal{P}$, it follows that a ground version of *se* is established, we have sufficiency. Any plan $\mathcal{P}$ which has all its substate expressions (prevails, overall goals and lhs of transitions) satisfying the truth criteria, means that all the completions of that plan are sound solutions[6].

3.1 The Application of the Truth Criterion to HTN Planning

Hierarchical Representation of Actions: By allowing operators to contain 'bodies' (networks of tasks), the primitive operator easily generalises to the hierarchical case. Hierarchical operators are related to the primitive operators that result in expansions of the hierarchy, and are similar in this respect to the formulations of Yang [21] and Erol [4]. A hierarchical operator will change the substates of objects in ways conditional on its expansion into more detailed task networks. It

[6] space does not permit us to discuss the case where an expression is established by a conditional transition, however this is discussed in [9]

will *necessarily* change the state of zero, one or more objects into a well defined new state (i.e. well defined according to the substate class expressions). If it does not necessarily change the substate of any object, then it is called a *filter* operator, but if it is 'indexed' with one or more necessary transitions, it is called a *method* operator [11].

An hierchical operator O has components $(id, pre, index, cons, nodes)$, such that $O.id$ is the operator's parameterised identifier, $O.pre$ is a set of substate expressions, $O.index$ is a set of necessary state transitions (possibly null), $O.cons$ is a set of static predicates acting as constraints (which include temporal constraints on *nodes*), and $O.nodes$ is a set of *nodes*. A node is either the name of a primitive operator, the name of a hierarchical operator, or an expression of the form 'achieve(G)', where G is a substate expression. Each expression in $O.pre$ must be true before execution of O, but may be affected by an operator's execution. **Hierarchical Partial Plan Structures** A task network m in OCL_h is defined as a structure $(id, pre, index, cons, nodes)$. *pre* are the preconditions of the network (for a top level network this be the initial state), and *index* is the set of transitions that the network must achieve. *id*, *cons* and *nodes* are as defined above. The refinement step is carried out by reducing m to network m', by replacing a node with operator *op* of the same identifier, or a node of type $achieve(G)$ is replaced by the name of a primitive or the nodes in a method operator which necessarily achieves a substate satisfying G. The refinement step is legal if $m.cons$, supplemented with other constraints brought about by the refinement, is consistent in $\mathcal{M}$.

The *transparency* property developed in reference [11] was stated in terms of transition sequences. We can apply the truth criterion developed above to restate this property as follows. Every expansion of a method operator should have the following property: for each object X whose substate transition is declared in its index, every substate expression in the prevail, precondition or necessary transitions must be *established* according to the definition above.

4 A Practical Evaluation of OCL_h using the O-Plan System

O-Plan [3, 17] and SIPE [19, 20] are HTN centered planning systems that have been developed to support applied research. This application focus has lead to the formation of constructs and representational devices that are designed to meet the modelling requirements of real-world planning problems. In this section, we compare OCL_h with O-Plan's Task Formalism TF to identify their similarities and differences. As well as highlighting their relative strengths, the results provide an insight into the practical utility of OCL_h, and indicate where further research must be focused to unify the relative benefits of these representation laguages.

Our comparison is in two stages. First, we examine the benefits of using only the substate / substate class ideas from OCL_h. We motivate this with the scenario of supporting a domain writer in understanding and modifying an existing domain description encoded in TF. This scenario is designed to demonstrate the modelling assumptions that are explicitly captured in OCL_h but not in TF and the utility of using substate OCL_h elements as a pencil and paper activity alongside the general development of the model. Second, we take each of the major constructs in TF in turn and consider how they can be mapped to OCL_h. Throughout we use the Pacifica domain [15]. Pacifica is an unclassified version of a non-combatant military evacuation planning application.

It entails the movement of transportation equipment to an island, the evacuation of the population of outlying districts to a central point, and finally the evacuation of the assembled population and transportation equipment from the island. The Pacifica domain is one a number of standard O-Plan demonstrations that can be run over the World Wide Web[7].

4.1 Analyis of a TF Encoding Using the OCL_h Method

The Pacifica type definitions, loosely equivalent to OCL_h sorts, are shown in Example 5. The substates that instances of these types can occupy are not explicitly stated in the domain model but are instead implicitly recorded in the model's operator definitions.

ground_transport = (*GT*1 *GT*2),
air_transport = (*C*5 *B*707),
country = (*Pacifica Hawaii_USA*),
location = (*Abyss Barnacle Calypso Delta Honolulu*);

Example 5

Part of the specification of the *fly_transport* operator is given in Example 6. We can deduce from the *vars* statements that transports only operate between locations that are of the type *air_base* and from the *effects* statements that instances of the type *ground_transport* can be at locations and have an *in_use* status set to at least *in_transit*. Careful examination of this action reveals some subtle substate constraints in the domain that are not explicitly documented. For example, instances of the type *ground_transport* can be carried only by the *C5* instance of the *air_transport* type. A *C5* is a large military transport aircraft while a *B(oeing) 707* is a standard civilian passenger aircraft. If a domain writer charged with modifying the model was unaware of this constraint, he or she might change the *C5* token in lines 5 and 6 of Example 6 to *B707*. The result would be an action that enables the invalid state of *loaded(B707, ground_transport)* to be formed. As the current model does not include a specification of the constitution of a valid state, there is no specification for the domain writer to manually check his or her new model against and therefore identify the error.

1.*schema fly_transport*
2.*expands fly_transport* ?*FROM* ?*TO*;
3.*vars* : *FROM* ?*type air_base*, ?*TO* ?*type air_base*,
4.*only_use_for_effects at GT*1 ?*TO*, *at GT*2 ?*TO*;
5.*conditions achieve at C*5 ?*FROM*, *unsupervised at GT*1 ?*FROM*,
6.*effects at C*5 ?*TO*, *in_use_for GT*1 *in_transit at begin_of self*,
*in_use_for GT*2 *in_transit at begin_of self*,
7.*end_schema*;

Example 6

[7] http://www.aiai.ed.ac.uk/ oplan/web-demo/

Example 4 shows the objects, sorts and substate classes for for Pacifica expressed in OCL_h. The process of producing this encoding forces the developer to think deeply about the objects in the domain and the states that they can occupy and, hence, explicitly document the assumptions underlying the original TF encoding of the domain. For example, the distinction between the *B707* and the *C5* identified above is made explicit through the division of sort *large_scale_transporter*[8] into the sorts *cargo_transporter* and *people_transporter* at line 4. The invariant on line 14 states that it is inconsistent to load an object *C* onto a *people_transporter* when object *C* is of the sort *Cargo*.

The process of writing the OCL_h description of Pacifica in Figure 3 forced us to think carefully about the states that objects can occupy. As a result, we have made explicit distinctions such as that between cargo and people transporters and documented them. Even in the absence of tool support, for argument say the description in Figure 3 was added as a comment within the Pacifica TF file, the assumptions underlying the model would be documented. However, OCL_h goes further by offering tool support for checking the consistency of actions against the substate class and invariant specifications. In the following section we consider the mapping between O-Plan TF and OCL_h constructs to determine if the tool support provided for OCL_h can be extended to a rich formalism such as TF.

4.2 Comparing O-Plan TF and OCL_h Constructs

In the previous section we demonstrated that the OCL_h method of object-centered structuring can be used to identify slips in operator definitions that place an object of a sort into an invalid state. Although TF does not currently support such constructs, it would be straightforward to integrate them with O-Plan. In this section we consider more the complex question of reconciling the operator representations of both languages, and hence the likelihood of providing tool support for checking O-Plan TF models for transparency. First, we consider how the indexing of methods in OCL_h with a state transition index is achieved in O-Plan TF. Second, we consider the mapping of each of the condition types supported by O-Plan to OCL_h.

Method Indexing

In OCL_h, each method must be indexed by a set of necessary state transitions, $LHS \Rightarrow RHS$, and a set of dynamic filter predicates, P. To determine how the equivalent index can be expressed in TF, consider the example TF schema in Example 7. The *RHS* component of the state transition index of this schema is stated in the *only_use_for_effects*. In this case, the operator is designed to bring about the state of the ground transport GT1 being at the *?to* location. The side effects of the operator are typed as just *effects*. In this case, the effect that the *C5* is also at the *?to* location is a side effect. One would not use this operator for the purpose of achieving this effect. Thinking in terms of the domain, it would not make sense to load a transporter with cargo and then fly the

[8] The clarity of modelling afforded to us by OCL_h has caused us to replace the original *air_transport* and *ground_transport* types in the original TF encoding with *large_scale_transporter* and *small_scale_transporter*. The true distinction between these sorts is not that they travel by land or air, but that a *large_scale_transport* can carry a *small_scale_transport* but not the converse.

transporter to a location if one only wanted the transporter to be at that location. In this case the loading of the cargo would be superfluous.

$$\begin{aligned}
&\textit{schema fly_transport vars } ?\textit{FROM } ?\textit{type air_base}, \ ?\textit{TO } ?\textit{type air_base},\\
&\textit{expands fly_transport } ?\textit{FROM } ?\textit{TO};\\
&\textit{only_use_if fuel_at } ?\textit{FROM assigned_to_unit}; \ \textit{only_use_for_effects at } GT1 \ ?\textit{TO},\\
&\textit{effects at } C5 \ = ?\textit{TO},\\
&\textit{conditions achieve at } C5 \ ?\textit{FROM},\\
&\textit{unsupervised at } GT1 \ ?\textit{FROM}, ..
\end{aligned}$$

Example 7

Reconstructing a *LHS* of an OCL_h transition, to form an index for a method operator, is more involved. Initially, it appears that the *only_use_if* condition type is equivalent to the *LHS*. In TF *only_use_if* conditions are used to choose between different methods for refining a given high level action. The planner will not make any attempt to satisfy the condition. If an *only_use_if* condition is not satisfied at the point in the planning process when the planner considers it, then the method of which it is a part is not applicable. This is the equivalent to the intended behavior of an OCL_h planning engine when considering the *LHS* of a state transition index. However, in OCL_h dynamic objects in a transition also have a target substate embodied in the *RHS*. In domain modelling terms, this means we only specify objects of dynamic sorts in a state transition index if we are concerned about both the state that they are in when we decide to use a method and after it has been executed. For an OCL_h hierarchical operator *O* the $O.pre$ component provides a set of substate expressions for specifically defining dynamic conditions that we are concerned about only when selecting operators. Mapping this to O-Plan TF, we can distinguish between two sub-types of *only_use_if* conditions. The first, which correspond to $O.pre$, are typed as *only_use_if* but there is no associated expression in the operators *only_use_for_effects*. The second, which are equivalent the *LHS* of an OCL_h state transition index are typed as *only_use_if* and for which there is also an associated expression in the operators *only_use_for_effects*. Considering what we can learn form this mapping, it is first clear that we can automatically compile from an O-Plan TF operator the $O.pre$ and the index transitions in OCL_h. There is no difference in expressiveness between the two formalisms in this aspect. The advantage of OCL_h in this is that it forces the domain writer to explicitly distinguish between these sets. This additional structure gives the domain writer some additional guidance when writing operators.

Condition Types

O-Plan's TF contains a number of condition types [18] which, in terms of the planning process, determine the mechanisms O-Plan will use to satisfy a given condition. It is well argued (for example, in the contractor metaphor in the house building domain [16]) that these types correspond to domain features and can be written without knowledge of the underlying search strategies deployed in O-Plan. In this section, we consider each condition type and determine if it can be mapped to OCL_h or if the foundaations of OCL_h (and the corresponding truth criteria and domain property defintitions) must be modified to accommodate them.

Achieve (x = v): O-Plan will seek to make *x* have value = *v* at the point in the plan that this condition is placed. It will exploit any available mechanism to do this, including the addition of new plan structure (which has the worst implications for expansion of the search space). There is a one to one mapping between this condition type and the OCL_h achieve condition. In OCL_h conditions of this type are expressed as $achieve(g)$, where *g* is the substate expression representing an object *x* with attribute *v*.

Only_use_if: As outlined in the previous section, this condition type is used by O-Plan to select between different methods for refining the same non-primitive action. A method *O* will only be considered applicable for refining iff at the time in the planning process *O.pre*, *O.cons* and the *RHS*'s of the transitions in *O.nec* hold in the current plan state. OCL_h divides *only_use_if* conditions as follows:

only_use_if on Static predicates: are expressed within *O.cons*

only_use_if on Dynamic predicates, also mentioned in the only_use_for_effects of an operator: are expressed on the *LHS* of a state transition index in *O.nec*.

only_use_if on Dynamic predicates, not mentioned in the only_use_for_effects of an operator: these are expressed in *O.pre*.

Unsupervised Conditions: O-Plan restricts the mechanisms it can deploy in satisfying an *unsupervised* condition. It will only use effects that are already in the plan and will not consider the inclusion of new plan structure. This causes problems with the existing definition of transparency as it is no longer necessary for a method to achieve all the stages of a transition itself. In essence, methods are no longer self-contained.

Supervised Conditions: For example, ($x = v$ at node 1 from node 2) means that *x* must equal *v* at a point in a plan and that it will be satisfied by an effect at a specified node or by an expansion of that node. In OCL_h terms, the use of supervised conditions tightens the definition of linear soundness. A supervised condition would stipulate that the *LHS* of a given transition must be established by the *RHS* of a specified transition or the set of transitions inserted in the sequence by some higher level one. Currently, the initial conditions or any *RHS* occurring before a transition can satisfy the *LHS*.

4.3 Discussion

This comparison has taken the O-Plan system as an example of an applied planning system and compared it with OCL_h to give insights into the practical utility of OCL_h. We have identified that even when applied as just a paper and pencil activity alongside model development, OCL_h can benefit a domain writer. The discipline of constructing sort hierarchies, their substate class components, and invariants to document design decisions forces the domain writer to think deeply about the sorts and the states that their objects can occupy. Including these aspects in a domain model documents modelling assumptions that would otherwise only be implied by operator definitions. In the second stage we compared TF's schemas with OCL_h's hierarchical operators, with the intention of . Many of the constructs in O-Plan's TF have an immediate mapping in OCL_h. Specifically, only_use_if, only_use_for_effect, effects, and achieve condition and effect types either map directly or can be automatically compiled. However, in the case of *supervised* and *unsupervised* condition types, the definitions of linear soundness and transparency are over-restrictive. In the case of the *supervised* condition type this should not be a problem. *Supervised* conditions have the effect of tightening the definition of linearly sound by specifying a subset of the possible contributors to establishing a condition. In the case of *unsupervised* the issues are more complex. Unsupervised removes the obligation on a method to be self contained in ensuring that it establishes the LHS of the overall transition that it is designed to achieve. It is not immediately obvious what obligation this places on other methods in the model. Pragmatically, *unsupervised* has proved a useful construct in modelling real-world problems and therefore cannot be discarded without careful consideration. The integration of *supervised* and *unsupervised* condition types into OCL_h is an important issue for further research. The *unsupervised* condition type is likely to require the most effort.

In terms of practical application, the O-Plan team is currently working on a planning application for the supporting Small Unit Operations in the US Army. This work is demanding much effort in domain modelling and requirements determination. The O-Plan team already uses IBM=92s Business Systems Design Method (IBM 1992a; 1992b) to identify the fundamental entities in a domain and the transition that those entities can make. The concepts within OCL_h support this emphasis through the provision of a planning oriented formalism for tightly specifying these models. In the absence of tool support, OCL_h concepts are being applied as pencil and paper activities alongside the model development. The additional structure provided by OCL_h is helping to clarify thinking. While not providing a complete definition of transparency with respect to O-Plan TF, the notion provides a useful review check for models.

5 Related Work

A related development in knowledge acquisition for planning is the development of tools for manipulating, analysing and compiling domain models. Fox and Long [6] show how efficient tool support can a type structure and model invariants from an operator set. Effectively, their tools can deduce parts of the OCL_h language (i.e. sort hierarchies and invariants) from literal-based precondition and effects operators. Gerevini and Schubert have shown the potential of type analysis in planning [7], and McCluskey and Porteous showed the potential of combined domain independent heuristic extracion [12]. Tools based on this work help the knowledge engineer build a model by (a) cross checking stated assumptions and properties of the model (b) making explicit implicit knowledge that is particularly helpful to a planner. Biundo and Stephan have also worked on systematic modelling of planning domains, but in the area of deductive planning [2]. They use a rich language which is inspired by formal methods in software engineering.

Problems remain with the formulation of expressive HTN languages because of the complexities in analysing complex conditional behaviour in an abstract operator. Calculating ‘implicit preconditions’, for example, of such operators is thus not as straigtforward as that of a linear sequence of primitive operators. This does not mean, however that progress towards that goal should not be made. Tsuneto et al’s ‘external conditions’ idea is a step in this direction - conditions (excluding initial conditions) that are needed to be satisfied before any completed plan can be sound - is an important idea here. They have an algorithm which finds some external conditions.

6 Conclusions

In this paper we have briefly reviewed the foundations of OCL_h, and defined (a) a sufficient truth criterion for plans containing primitive operators (b) a truth citerion, based on transparency and sort abstraction, for hierarchical task networks. In the second half of the paper we have compared OCL_h to TF, a powerful representation language which has been used to encode many complex domains. This comparison has (a) given rise to a mapping between many of the constructs (b) highlighted the features of OCL_h that may need further development (c) illustrated some of the advantages in using such an object-centred approach. The two halves of the paper, therefore, provides evidence that OCL_h is both a realistic yet transparent language, capable of both supporting theoretical analysis and providing the constructs required for domain modelling.

References

[1] Benjamins, Nunes de Barros, Shahar, Tate and Valente (eds). *Workshop on Knowledge Engineering and Acquisition for Planning: Bridging Theory and Practice.* AIPS98, 1998.

[2] S. Biundo and W. Stephan. Modeling Planning Domains Systematically. In *Proceedings of the 12th European Conference on Artificial Intelligence*, 1996.

[3] K. Currie and A. Tate. O-Plan: the open planning architecture. *Artificial Intelligence*, 52:49 – 86, 1991.

[4] K. Erol. *Hierarchical Task Network Planning: Formalization, Analysis, and Implementation.* PhD thesis, Department of Computer Science, University of Maryland, 1995.

[5] K. Erol, J. Hendler, and D. S. Nau. UMCP: A Sound and Complete Procedure for Hierarchical Task Network Planning. In *Proc. AIPS*. Morgan Kaufman, 1994.

[6] M. Fox and D. Long. The Automatic Inference of State Invariants in TIM. *JAIR vol. 9*, pages 367–421, 1997.

[7] A. Gerevini and L. Schubert. Computing Parameter Domains as an Aid to Planning. In *Third International Conference on Artificial Intelligence Planning Systems*, 1996.

[8] S. Kambhampati and D. S. Nau. On the Nature of Modal Truth Criteria in Planning. In *Twelfth National Conference on Artificial Intelligence*, 1994.

[9] D. E. Kitchin. *Object-Centred Generative Planning.* PhD thesis, School of Computing and Mathematics, University of Huddersfield, forthcoming,1999.

[10] D. Liu. The OCL Language Manual. Technical report, Department of Computing Science, University of Huddersfield , 1999.

[11] T. L. McCluskey and D. E. Kitchin. A Tool-Supported Approach to Engineering HTN Planning Models. In *Proceedings of 10th IEEE International Conference on Tools with Artificial Intelligence*, 1998.

[12] T. L. McCluskey and J. M. Porteous. Engineering and Compiling Planning Domain Models to Promote Validity and Efficiency. *Artificial Intelligence*, 95:1–65, 1997.

[13] PLANET. *First Workshop of the PLANET Knowledge Acquistion Technical Coordination Unit.* Salford, UK, 1999.

[14] R. Tsuneto, J. Hendler, D. Nau. Analyzing External Conditions to Improve the Efficiency of HTN Planning. In *Sixteenth National Conference on Artificial Intelligence*, 1998.

[15] G. Reece, A. Tate, D. Brown, M. Hoffman, and R. Burnard. The precis environment. In *AAAI-93: Proceedings of ARPA-RL planning initiative workshp*, 1993.

[16] A. Tate. Generating Project Networks. In *Fifth International Joint Conference on Artificial Intelligence*, 1977.

[17] A. Tate, J. Dalton, and J. Levine. Generation of multiple qualitatively different plans. In *Proc. AIPS*, 1998.

[18] A. Tate, B. Drabble, and J. Levine. The Use of Condition Types to Restrict Search in an AI Planner. In *Twelfth National Conference on Artificial Intelligence*, 1994.

[19] D. Wilkins. *Practical Planning: Extending the Classical AI Paradigm.* Addison-Wesley, 1988.

[20] D. Wilkins and K. Myers. A Multiagent Planning Architecture. In *Proc. AIPS*, pages 154–162, 1998.

[21] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6, 1990.