Technical Report

O-Plan Project Second Year Demonstration: Reasoning with Resources

Brian Drabble

Approved for public release; distribution is unlimited

Artificial Intelligence Applications Institute
University of Edinburgh
80 South Bridge
Edinburgh EH1 1HN
United Kingdom

January 6, 1997

ARPA-RL/O-Plan/TR/19 Version 1

Contents

1 Introduction

The aim of this report is to describe the second year demonstration of the O-Plan system. The demonstration was designed to show the ways in which a rich model of domain resources e.g. trucks, aeroplanes, fuel, runways, etc, could be encoded and used within an activity planner. The demonstration also provided a check on the development of the functionality of the emerging O-Plan system and in particular the system's ability to reason with numbers and numerical ranges. The demonstration was conducted on the 29th September 1994 at AIAI.

The second year of the O-Plan project was to investigate the ways in which resources could be flexibly handled and manipulated within an activity planner framework. As part of this investigation a study was carried out into the different types of resources present in planning domains and into previous planning approaches to resource reasoning [5]. The results of this study were twofold.

1. It became possible to identify the type of resource reasoning support which an activity planner should provide.
2. It resulted in the design of a flexible Resource Utlisation Manager (RUM) [6, 7] for use in an activity planner such as O-Plan and Sipe-2.

The support provided by the RUM would allow a range of resources types to be represented and manipulated and went beyond those types supported by KRSL. The RUM is designed as a replacement component for the current simpler Resource Utilisation Manager module in O-Plan. While the new RUM has not been implemented in the current prototype, the research has investigated the underlying mechanisms necessary to support the various resource types and to integrate resource reasoning about each type into an activity planner. The approach adopted in the research – as stated in the proposal – was to take an activity centred reasoning approach and to relate resource reasoning to this. The project does not claim this to be the best way to handle domains in which resource contentions dominate. Approaches such as in OPIS [12] or TOSCA [1] may be more appropriate. TOSCA is itself based on the O-Plan architecture but uses resource centred representations and knowledge sources.

Resource reasoning has been identified by a number of AI planning and scheduling researchers as a key provider of information concerning the management of the search. This is because one of the most powerful reasons for not doing something is the fact you have insufficient resource with which to carry it out. For example, the resource may be time, money, material, tools, etc. As such resource reasoning should provide:

1. checks that resource usage demands can be met from the resource available at any time.
2. heuristic estimates of the quality of a plan as it is generated.
3. suggestions, if possible, on the repair of a failed plan should resource usage be the problem. For example, reduce resource levels, produce more of the resource earlier, move actions back or forward in time, etc.

In any real world domain there is a need for a planner to carry out extensive reasoning about resources and to provide ways in which various resources could be manipulated, consumed and produced.

The remainder of this report is structured as follows. Section 2 describes the scenario used as the demonstration domain and the types of resources which were being modelled and reasoned with. Section 3 describes how the domain and in particular the resources were mapped to Task Formalism (TF) statements. TF is the input language used to specify domains, tasks and operators to the O-Plan system. Section 4 describes the results obtained from the demonstration and the types of plans which were generated. Finally section 5 provides a summary of the results achieved and pointers to further work to be undertaken in the area of resource reasoning.

1.1 Review of Previous Work in Resource Based Reasoning

The CAMPS (Core of a Meta-Planning System) [13, 2] was developed at MITRE in the mid 1980s and was intended to be an architecture to solve scheduling and resource allocation problems. It was a development of MITRE's previous experience with the KRS and EMPRESS expert systems. KRS [3] is an Air Force tactical mission planner and EMPRESS [8] is a cargo preparation planner for the space shuttle. Both the KRS and EMPRESS systems were delivered to their sponsors but suffered from similar criticisms. They were successful in meeting their functional requirements but were extremely difficult to extend or modify. This was the impetus for the CAMPS architecture which was to address the same application areas in two new systems: AMPS [4, 10] Airforce Mission Planning and EMPRESS-II. For the remainder of this report the planner name CAMPS will be used to denote both the AMPS and EMPRESS-II systems.

CAMPS formulates its problems as a Constraint Satisfaction Problem (CSP). In a CSP, there is a set of variables that must be instantiated according to a set of constraints where the constraints specify what the acceptable values of the variables are. In some situations the problem is unsolvable in that an assignment cannot be made without violating at least one of the specified constraints. In these cases the constraint is relaxed, i.e. the violation is noted but tolerated. In the applications to which CAMPS was applied it aim was to allocate times and resources to specified actions.

The major difference between the two systems is their approach to the "planning" problem. CAMPS views the planning problem as one of allocating times and resources to specified activities and relies on it not being necessary to perform problem solving in order to enumerate all the appropriate tasks and resources for an application. CAMPS is thus doing what should (more properly) be called *scheduling*. O-Plan is a general problem solving framework in which systems such as TOSCA which uses a similar problem solving strategy to CAMPS can be built. However, O-Plan is primarily used today as an activity planner which allows the user to generate alternative courses of action in which problem solving and search can take place but the search is minimised by the use of user provided domain knowledge and search heuristics. The main levels of compatibility with the CAMPS system is at the O-Plan constraint management level. O-Plan uses a number of *different* constraint managers to manage *separable* parts of the plan state. The following constraint managers are currently within the O-Plan framework:

- TOME and GOST Manager: deals with the assertion of effects and the satisfaction and maintenance of conditions in the plan state.
- Plan State Object Manager: deals with the object created during planning which act as place holders[1] for domain entities.
- Time Point Network Manager: deals with the metric temporal information concerning actions, links and events within the plan state
- Resource Utilisation Manager: deals with the recording of the use of resources within the plan state.

Each of these constraint managers has a support role for the decision making layer of O-Plan (the knowledge sources) and has a standardised interface [7]. This allows the constraint manager to respond in one of three ways:

1. **Yes**: the constraint can be taken under management,
2. **No**: the constraint cannot be taken under management,
3. **May be**: the constraint can be added **if** some combination of plan state element changes are made.

This has some similarity to the CAMPS constraint evaluation mode *suggestion* (see Section 2.1.2). The elements which can be changed are described in a planning ontology which has been developed for communication between O-Plan constraint managers. The main elements which can be changed are the restrictions on a plan object or the restrictions on a time point. By adhering to the definition of the planning protocol and the constraint manager interface it is possible for other researchers to replace individual constraint managers with there own without having to make any changes to the core of the planning system. This has been tested by means of an exchange of information with researchers in temporal databases at GE and Honeywell. The standard interface and protocol also allows the integration of *alien* constraint managers by other researchers. For example, an application may require to reason about free space and would therefore need a spatial reasoning manager. The constraints which apply to the spatial manager are communicated through O-Plan's TF language and then passed to the associated constraint manager. This allows O-Plan to more flexibly handle different constraints and reduces the problem of the large number of slots and simple constraints as was the case in CAMPS.

The use of dedicated constraint managers for a class of constraints e.g. time, resources, etc removes the requirement which CAMPS had for different evaluation strategies e.g. rules, hand coded and predicates. The way in which a constraint is managed and a value generated is the concern of the constraint manager. The way in which complex constraints are handled is an implementation decision as this layer of reasoning is hidden from the user by the protocol.

[1]In O-Plan they are referred to as Plan State Variables

The information returned from **all** constraint managers (eg. TGM, TPNM) is in the form of an `and-or` tree. This allows the constraints on the plan state (object and temporal restrictions) to be built up over a series of queries. In the following example, the planner is required to choose an army[2] whose `attack-firepower` is sufficient to defeat a defensive force. Using O-Plan TF this can be expressed as follows:

```
only_use_if {firepower ?army1} = ?attack-firepower,
only_use_if {firepower ?army2} = ?defence-firepower,
compute {> ?attack-firepower ?defence-firepower},
```

The `or-tree` returned from each condition query can be merged with that from a previous query and a consistent set of restrictions obtained. The merging of `or-tree`s is not restricted to the same class of constraints but across all constraint types. The current designs for the resource utilisation manager envisage it have an MTC style interface referred to as the Resource Truth Criterion (RTC) which allows queries of the form "can resource X be allocated at this point in the plan". The `or-tree` returned could then be merged with an `or-tree` created from condition queries, effect queries, temporal queries, etc. This would allow the planner to crate a single `or-tree` which would represent the consistent set of plan state changes required. Similar criteria could be envisaged for other types of constraints such as spatial, authority, etc. This merging process would greatly reduce the size of the search space and would help reduce the problem in CAMPS of requiring to evaluate slots in the correct order to efficiently restrict the search space and requiring filters and critics to be assigned to slots to rule out possible candidates from a generator.

1.2 Definition of Resource Types

Resources in the real world can be broken down into distinct categories:

1. **Consumable Resources**
   These are resources which are consumed during the life of a plan and can be subdivided into the following two sub-categories.
   (a) **Strictly**
   These are resources which maybe produced or set to a specific level at the beginning of a plan, consumed during its execution and which are *not* topped up or replaced during the plan. For example a fixed amount of money, bricks, fuel, etc.
   (b) **Producible** These are resources which can be topped up during the plan. The topping up process can be either under the control of the plan or outwith the plan, e.g. a delivery from another agent or process. These resources can be defined further into the following three categories:
      i. **By Agent**
      These are resources which maybe produced or set to a specific level by one of

[2]There is a restriction that ?army1 and ?army2 cannot be bound to the same value

more actions within a plan either as raw material or as sub-components/sub-assemblies for a future part of the plan. The production of these resources is under the direct control of the plan and as such can be generated as required within the relevant constraints.

ii. **Outwith Agent**
These are similar to By Agent resources but maybe produced or set to a specific level by an *off-line* process outwith the plan. For example a delivery of materials, etc. The main difference is these resources are *not* under the control of the planning process in terms of their numbers or quantity.

iii. **By and Outwith Agent**
These are resources which can be produced or set to a specific value either by the planner or as the result of an off line process. For example, materials can be delivered from a warehouse or the planner can chose (resources permitting) to retrieve them itself.

2. **Reusable Resources** These are resources which can be *allocated* to a plan for it to use and then possibly returned i.e. *deallocated* after use.

   (a) **Non-Sharable**
   These are unitary resources which maybe allocated and then deallocated to a central pool afterwards. For example, workman on a building site, access to a key, forklift trucks in a factory, etc.

   (b) **Sharable Resources**
   These are resources which are shared between actions which may be executing in parallel. Sharable resources can be further divided into two distinct sub-categories:

   i. **Synchronously**
   These are shared resources which are available over a defined time interval during which operations using them must be carried out. For example the cargo capacity of a ship, battery power of a spacecraft, etc, can be allocated to multiple consumers but only over the same interval of time.

   ii. **Independently**
   These are shared resources over which there is no restrictive access. For example, a warehouse may be shared by several customers each having a set amount of storage available to it.

The types defined in the O-Plan resource hierarchy can be directly mapped to those defined in the KRSL manual [9]. However, the modelling capabilities of a number of O-Plan resource types exceeds those of the KRSL definitions.

The resource types defined within the O-Plan project are described in Figure 1. The resources of a given domain can be organised into *hierarchies* as described in Figure 2. Figure 2 describes three different fuel resources and the locations in which they can be found. The resource reasoning component of O-Plan maintains checks on the utilisation of resources at the "fully qualified" level, i.e. the amount of `fuel-avgas` in tank1 and tank2 of Port 1. The level of resources at points higher up in the hierarchy i.e. the amount of `fuel-avgas` in Port1 is calculated summing the entries below it down to the fully qualified level.

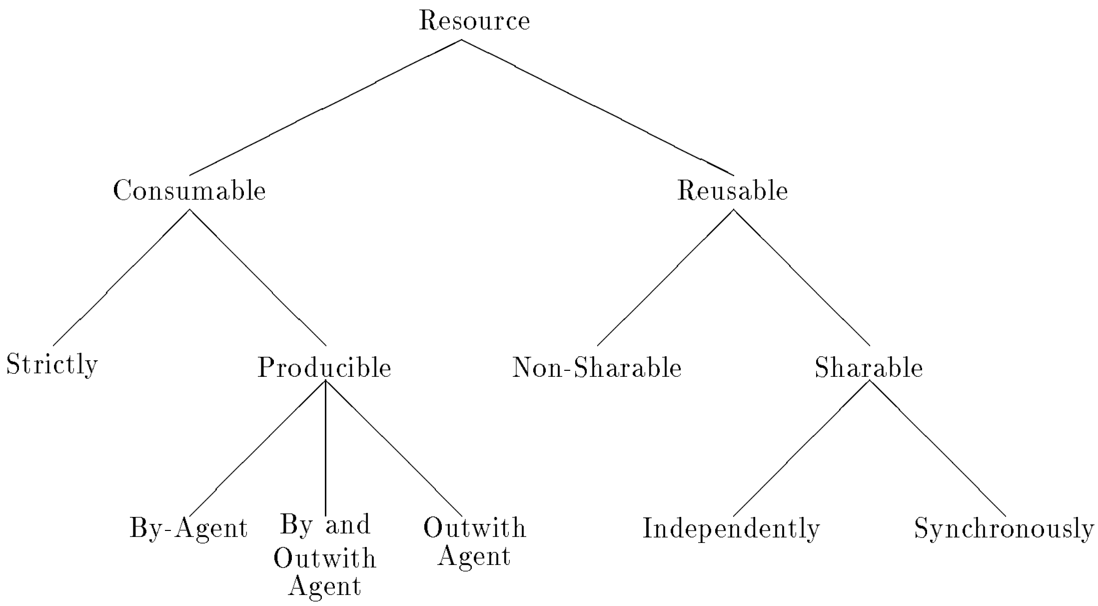

Figure 1: Example Hierarchy of Resource Types

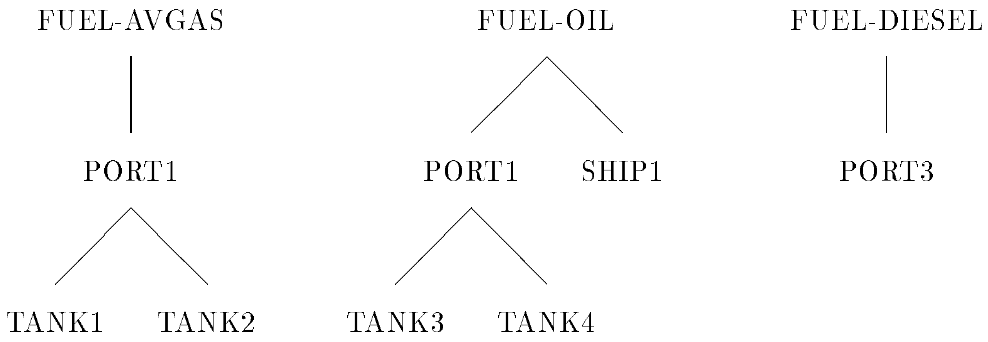

Figure 2: Example Hierarchy of Domain Resources

2 Scenario Description

The aim of this section is to describe the scenario which was used during the demonstration and in particular the resources which were modelled and manipulated.

The demonstration used a variant of the PRECiS domain [11] which is a non-confidential domain developed by a number of ARPI participants. The PRECiS domain allows AI planning and scheduling researchers to investigate a various military related logistics problems involving the evacuation of non-combatants for the fictional island of Pacifica. A map of the island of Pacifica is given in Figure 3.

The scenario explored during the demonstration was as follows:

> Civil unrest has broken out on the island of Pacifica and the US military has been requested to help evacuate a number of foreign nationals from the island. The nationals are on the island as part of WHO projects, forestry projects and as tourists. As part of its response to the request the US military has decided to generate a number of alternative courses of action (COAs) in which different numbers of nationals are moved from defined locations by ground transports and helicopters. Some transports are already on the island whereas others need to be flown in from Honolulu. The ground and helicopter transports bring the nationals to the island's main airport at Delta from which they are flown via passenger plane to Honolulu. The different COAs need to deal with:
>
> - the use of alternative transports e.g. helicopters versus trucks.
> - the number and location of nationals to be evacuated.
> - the use of in-theatre transports versus those which need to be brought with the US forces.
> - the availability and type of fuel required by the different types of transports.

A number of ground transports and helicopters need to be moved by strategic airlift from Honolulu to Pacifica and this can be achieved by the use of aircraft (C5 Galaxy and C141 Starlifters) allocated by USTRANSCOM. Due to the physical constraints of the aircraft the C5 is used to move the helicopters and the C141 is used to move the ground transports. The C141 aircraft can carry two ground transports per trip and the C5 aircraft can carry two helicopters per trip. Two other aircraft types are used in the demonstration and these are a KC-10 air tanker and a B707 passenger plane. The KC-10 aircraft remains on stand by in Honolulu and only flies to Delta should their be insufficient aviation fuel at Delta airport to refuel the B707 prior to its departure. The B707 passenger plane is used to fly the evacuees from Delta to Honolulu and is already at the airport in Delta prior to the start of the demonstration. All aircraft arriving and departing from the airport at Delta are required to use the same single runway (the other was closed by terrorist activity) i.e. all accesses need to be scheduled to avoid collisions.

The ground transports and helicopters require aviation fuel and diesel fuel respectively and this is initially located in fuel tanks at the airport in Delta. The fuel required by a transport

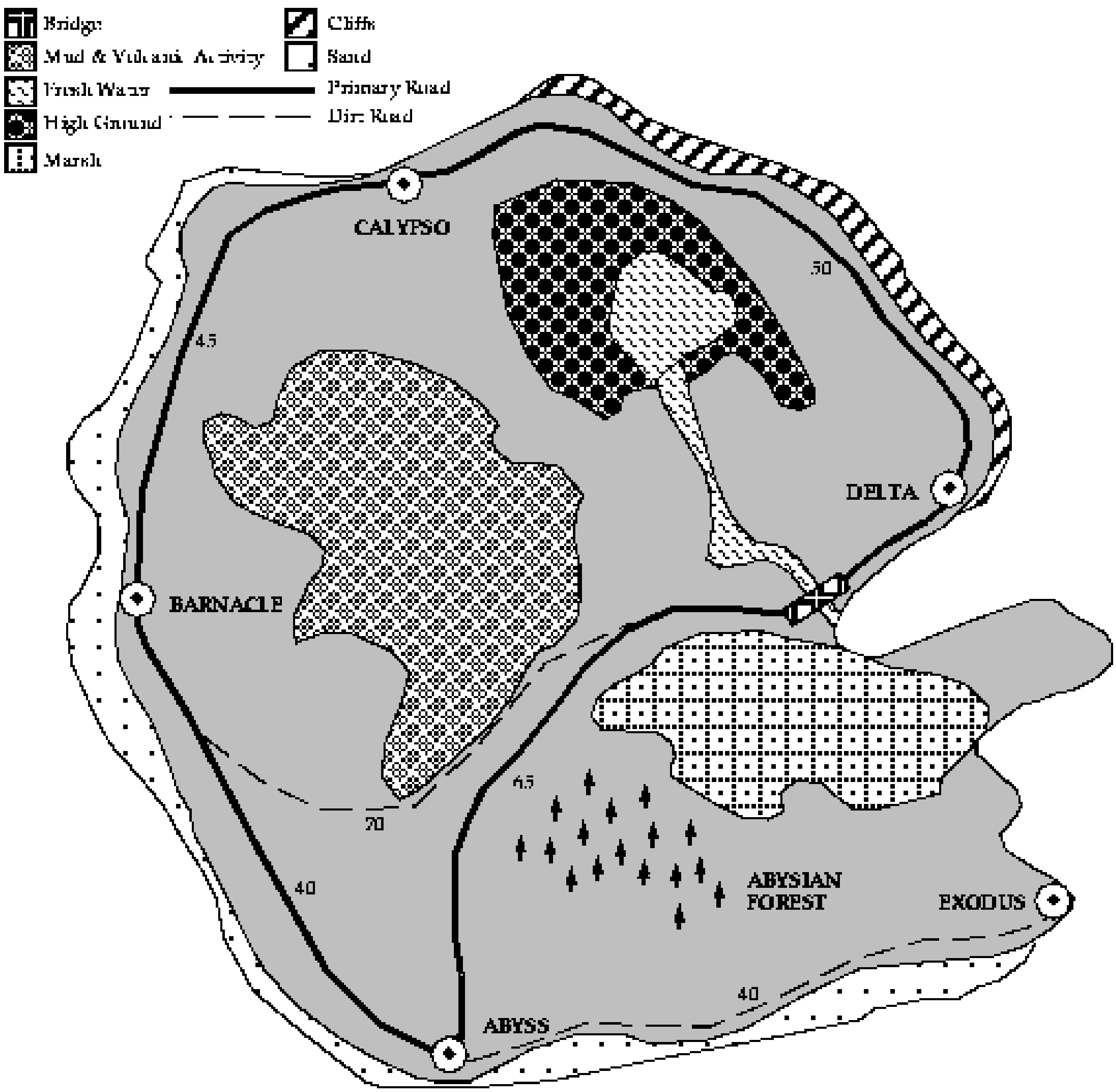

Figure 3: The Island Geography of Pacifica

is based on the transports type and the distance over which the transport must travel. The fuel load of a transport is sufficient for a single round trip to one of the out lying cities e.g. Barnacle, Calypso and Abyss. The B707 requires 20,000 gallons of aviation fuel similar to that used by the helicopters. This fuel can be obtained either from the fuel tanks at Delta or by requesting the KC-10 tanker to fly from Honolulu with extra aviation fuel. The capacities of the ground transports and helicopters is 35 and 25 respectively. The capacity of the B707 is 250 and is sufficient to move all evacuees in a single trip.

The evacuees themselves are located in three of the out lying cities Abyss, Barnacle and Calypso. There are 50 people in Abyss, 100 in Barnacle and 20 in Calypso. With the capacities of the ground transports and helicopters being 35 and 25 respectively a number of trips will be required to move all of the evacuees to Delta for evacuation by the B707.

3 Scenario and Domain Encoding

The aim of this section is to describe the methods and techniques used to encode the scenario described in Section 2 as a series of Task Formalism (TF) statements. The description of the encoding is divided into three parts. Section 3.1 describes the search method used to generate the plan and in particular, the ways in which the number and location of the evacuees was tracked. Section 3.2 describes the modelling of resources and the resource types covered in the demonstration. Section 3.3 describes the general model of the domain and provides a tutorial on the main types of statement used in the encoding. A complete listing of the TF description of the domain can be found in Appendix A.

3.1 Action Modelling

The aim of this section is to provide a description of the search method used to generate the different COAs. The overall task specification can be divided into three phases:

1. **Phase 1:** Loads the ground transports and helicopters at Honolulu onto the C5 and C141 transport planes and fly them to Delta. Once in Delta unload the planes at the airport.
2. **Phase 2:** Using a mixture of appropriate transport assets move all evacuees from the outlying cities to the airport at Delta. At the same time prepare and fuel the B707 aircraft to fly the evacuees to Honolulu. If necessary request the KC-10 tanker aircraft to fly to Delta from Honolulu with extra aviation fuel for the B707.
3. **Phase 3:** Once all evacuees have been assembled at Delta airport load them onto the B707 and the transports onto the C5 and C141 aircraft. All aircraft are then to return to Honolulu.

Phase 1 consists of creating sequences of actions which load the required transport asset onto the appropriate aircraft. The number of transports of each type required is given in the task specification together with the appropriate transport plane. This is a relatively simple problem for the planner to solve as the individual "load - fly - unload" plans only interact when taking off and landing. This is because the runways at Honolulu and Delta are modelled as `reusable-sharable-synchronously` resources which aircraft need to be scheduled to use.

Phase 2 consists of creating sequences of actions to fuel, move, load and unload a transport to pick up a batch of refugees. The fuel required is based on the transport type and the distance over which it must travel. With the transports having fixed capacities a number of trips may be required to remove all evacuees from one outlying city. In theory each of these trips could proceed in parallel except that there are a fixed number of transports of each type. The planner must therefore order the trips so that trips using the same resource e.g. helicopter-1 appear in sequence in the plan. Keeping track of the number of evacuees and the numbers at each location is coded as a type of missionary and cannibals problem. A state vector is maintained which keeps track of the number of evacuees moved and those still to be moved. For example the task specification describes the required outcome as follows:

```
action {evacuate Abyss 50},
action {evacuate Barnacle 100},
action {evacuate Calypso 20}
```

This states the number of evacuees to be evacuated from ech of the named city to the evacuation point. The evacuation point is specified by a fact asserted in the initial domain description. The information is then converted by means of task expansion into a series of `achieve` conditions which need to be satisfied. The `achieve` conditions are as follows:

```
achieve {evac_status Abyss}    = {0 50},
achieve {evac_status Barnacle} = {0 100},
achieve {evac_status Calypso}  = {0 20}
```

Which states the requirement that all evacuees are in Delta and none are left in the outlying cities. The planner now creates a plan in a backward chaining manner moving people from Delta back to the outlying cities with the number moved dependent on the capacity of the transport. This recursive process continues until the initial conditions of the problem i.e. all evacuees are in the outlying cities and no evacuee is in Delta is reached. This point is recognised by the planner by it being able to satisfying the `achieve` condition with one of the initial facts stated as follows:

```
{evac_status Abyss}    = { 50 0},
{evac_status Barnacle} = {100 0},
{evac_status Calypso}  = { 20 0}
```

At this point the planner need not add any extra trips to the plan. As an example, the planning process to evacuate the city of Barnacle using ground transports only is given. The search steps are as follows:

- The expansion of the task level results in the posting of the condition `achieve {evac_status Barnacle} = {0 100}`.
- The schema `drive` "moves" 30 people from Barnacle to Delta and then posted the condition `achieve {evac_status Barnacle} = {30 70}` to be satisfied.
- The schema `drive` is again chosen and moves a further 30 people to Delta. It then posts the condition `achieve {evac_status Barnacle} = {60 40}` to be satisfied.
- The schema `drive` is chosen twice more and on the second occasion the condition posted is `achieve {evac_status Barnacle} = {100 0}`.
- The planner can satisfy this `achieve` condition by matching it against a fact given in the initial domain description and so the search process ends.

Phase 3 is similar to Phase 1 (except that start and finish locations of the airlift are reversed) except the planner must also create a subplan for loading and flying out the passengers on board the B707.

3.2 Resource Modelling

The aim of this section is to provide a description of the methods and techniques used to describes the resources modelled in the domain. Figure 1 provided an overview of the resource types modelled in the O-Plan project. The following table provides a description of the particular instances of these resource types found in the demonstration domain.

Major Resource Type	Minor Resource Type	Instances
Consumable	Strictly	Aviation and diesel fuel at Delta Evacuees at Abyss, Barnacle and Calypso
	Producible-by-agent	Aviation fuel in the KC-10 Tanker
	Producible-outwith-agent	not handled
	Producible-by-and-outwith-agent	not handled
Reusable	Non-sharable	Ground Transports
		Helicopters C5, C141, KC-10 and B707 aircraft
	Sharable-independently	Taxi ways and airport parking
	Sharable-synchronously	Runways at Honolulu and Delta

The two resource types which were not handled during the demonstration `Producible-outwith-agent` and `Producible-by-and-outwith-agent` rely on O-Plan being able to handle external events. This functionality was not available in Version 2.2 (the one used as the basis for the demonstration) of the system and external event handling is not planned for integration into the system before the end of the current project. Even after taking these restrictions into consideration this domain proved extremely rich in resource types and instances and very suitable for the demonstration. The remainder of this section will describe the techniques used to manipulate the resource instances within the plan. The TF statements required to declare the resources types and their instances are covered in detail in the following section.

The techniques used to manipulate the resources can be divided into two types: those dealing with consumable resources and those dealing with reusable ones.

Consumable :

This type of resource is used to model the fuel resources which are "consumed" each time a ground transport or helicopter is set out to pick up a batch of evacuees. Consumable resources make explicit use of the current resource utilisation manager which is one of the main constraint managers with the O-Plan planning system. The amount of fuel required is dependent on the type of transport and the distance over which the journey takes place. The fuel required is stated as an `always` fact relating the amount of fuel to the two cities involved in the journey. For example, the following statements define the amount if aviation fuel (avgas) and diesel fuel required to for a return journey from Delta to Abyss.

```
{avgas_fuel_required Abyss Delta 140},
{diesel_fuel_required Abyss Delta 40},
```

Similar pairs of facts are needs for the other cities in which the evacuees may be found. This information can be "looked" up by a schema using a `only_use_if` condition. For example:

```
only_use_if {diesel_fuel_required ?from Delta ?fuel_required}
```

Assuming the schema variable `?from` is bound to Abyss before this condition is satisfied then the value 40 will be bound to the schema variable `?fuel_required`. If the schema variable `?from` is not bound then the schema variable `?fuel_required` is converted to a numeric plan state variable which will be bound later. Given that the schema variable `fuel_required` has a specific value then this amount of fuel can be "consumed" from the fuel tanks at Delta airport. For example:

```
consumes {resource diesel_fuel_delta_tank2} = ?fuel_required gallons
```

This statement will reduce the amount of diesel fuel in the tank by the required amount.

Reusable :

This type of resource is used to model the in-theatre and air transports which can be "allocated" from a central pool, used and then returned to the pool afterwards. This type of resource manipulation can be modelled by a series of conditions and effects and does not make explicit use of the current resource utilisation manager [3]. This is a generic technique and can be used to model *all* types of reusable resources, e.g. manpower, trucks, keys, runways, etc.

The main resources in PRECiS domain which fall into this category are the ground transports and helicopters which move the evacuees from the outlying cities to the airport at Delta. Figure 4 describes the assertion of the required effects and condition ranges to support reusable resources.

Figure 4 shows two actions `A1` and `A2` which move a group of evacuees from a city to Delta. In this example the city is Abyss however the use of schema variables would allow the same schema to be used for movement between any of pair of cities and with any transport. Actions in the O-Plan system have two ends a `begin_end` and an `end_end` representing the start and finish of an action. Conditions and effects can be required or asserted at either end of an action. In Figure 4 a condition is represented as an arrow entering an action end and an effect is represented as an arrow leaving an action end. For example, `A1` has two conditions on its `begin_end`.

1. that GT2 be available at that point

[3] As explained in Section 1 a new more flexible RUM design has been outlined during research on the O-Plan project [6]

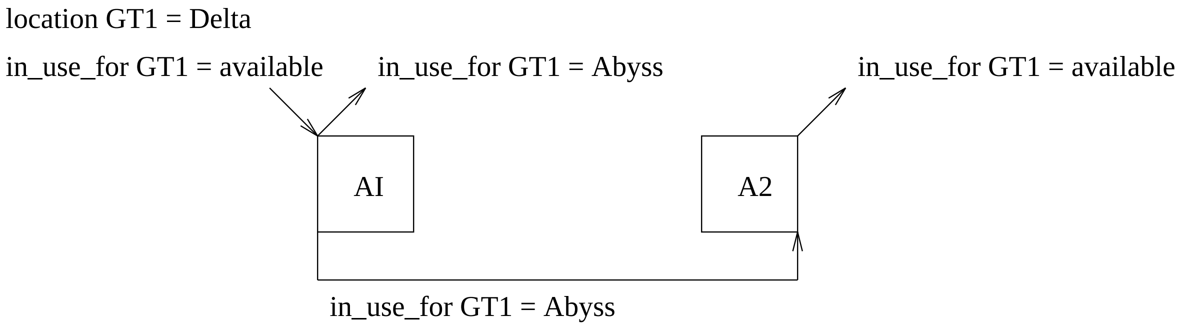

Figure 4: Manipulation of Reusable Resources via Conditions and Effects

2. that GT2 remain assigned to the Abyss trip for the duration of the actions `A1` and `A2`.

The first condition can be modelled as an `unsupervised` condition i.e. it is a scheduling constraint on the availability of the ground transport. The second condition can be modelled as a `supervised` which informs the system of the required contributor for the condition, i.e. the effect `in_use_for GT2` from the `begin_end` of `A1`. Action `A1` asserts the effect that it changes the status of GT2 from `available` to `Abyss` thus reserving the ground transport. The second action `A2` asserts the effect that GT2 is `available` once the transfer from Abyss has been completed. By attaching the assertion of an effect and the satisfaction of a condition on the same `begin_end` it avoids the problem of a "hole" occurring in the protected range of a condition.

The TF condition and effects statements required to model the plan schema in Figure 4 is as follows[4]:

```
conditions unsupervised {location_at ?at} = ?to at begin_of 1,
           unsupervised {in_use_for ?at} = available at begin_of 1,
             supervised {in_use_for ?at} = ?from
                        at end_of 2 from begin_of 1;
effects {in_use_for ?at} = ?from at begin_of 1,
        {in_use_for ?at} = available at end_of 2;
```

[4] ?at and ?from are schema variables.

3.3 Domain Encoding

The aim of this section is to provide a description of the domain encoding in the form of a tutorial covering the main statement types and their function. The statements types described in this section are as follows:[5]:

Type Definitions :
Type information is used to declare that certain instances are members of specific classes. For example, in the demonstration there are two ground transports, GT1 and GT2 which are of type `ground_transport`. Types can also be used to describe the instances of a functional relationship. For example, a transport (ground or helicopter) can either be `available`, `in_transit` (being flown to or from Honolulu), or allocated to go to a particular city e.g. `Abyss`, `Barnacle` or `Delta`. Given this definition a transport can only have one of these "usages" at a time. A complete definition of the types used in the demonstration is as follows:

```
types
 ;;;
 ;;; Transport Assets
 ;;;
      ground_transport         = (GT1 GT2),
      helicopter               = (AT1),
      air_transporter_cargo    = (C5 C141),
      air_transporter_evacuees = (B707),
      air_fuel_transporter     = (KC10),
      runway_status            = (clear in_use),
      cargo_types              = (passengers ground_transports air_transports),
 ;;;
 ;;; Geographic Information
 ;;;
      country          = (Pacifica Hawaii_USA),
      location         = (Abyss Barnacle Calypso Delta Honolulu),
      city             = (Abyss Barnacle Calypso Delta),
      air_base         = (Delta Honolulu),
      transport_use    = (in_transit available Abyss Barnacle Calypso Delta);
```

Resource Definitions :
Resource information is used to declare the type of the resource, the location of the resources and the units in which the resource should be measured. The resources are declared as being of type `consumable_strictly` which allows the fuel to be consumed but not added to. By using two such aviation fuel sources it became possible to model `producible_by_agent` resources as the second fuel source (on board a KC-10 tanker aircraft) is only brought to the island when the level of aviation fuel on the island falls below

[5] A complete listing of TF statements and their usage can be found in the TF Manual which forms part of the documentation for the O-Plan system.

that required by the B707. For the demonstration three sources of fuel are modelled, one diesel and two aviation. In all three cases the fuel is measured in gallons. A definition of the resources used in the domain are as follows:

```
 ;;;
 ;;; resource information
 ;;;
 ;;;   resource_units gallons = count;

resource_types
     consumable_strictly {resource aviation_fuel_delta_tank1} = gallons,
     consumable_strictly {resource diesel_fuel_delta_tank2} = gallons,
     consumable_strictly {resource aviation_fuel_KC10_tank1} = gallons;
```

Always and Initially Facts :
Always facts cannot be refuted by any effects asserted by actions within the plan, i.e. they always have a certain value. For example, the cities always remain on the island of Pacifica. Initial facts are asserted as having a certain value at the start of the plan but unlike always facts can be refuted by plan effects. A definition of the always and initial effects used in the domain are as follows:

```
always {avgas_fuel_required Abyss Delta 140},
       {avgas_fuel_required Barnacle Delta 160},
       {avgas_fuel_required Calypso Delta 180},
       {diesel_fuel_required Abyss Delta 40},
       {diesel_fuel_required Barnacle Delta 60},
       {diesel_fuel_required Calypso Delta 80},

       {country Abyss} = Pacifica,
       {country Barnacle} = Pacifica,
       {country Calypso} = Pacifica,
       {country Delta} = Pacifica,
       {country Honolulu} = Hawaii_USA;

initially
;;;    {evac_status <city>}   = {<# left at city> <# safe at evac pt>}
       {evac_status Abyss}    = { 50 0},
       {evac_status Barnacle} = {100 0},
       {evac_status Calypso}  = { 20 0},
       {evacuate_to Delta};   ;;; the evacuation point in Pacifica
```

Task Definitions :
A task definition inform the planner of the specific requirements the plan must achieve, e.g. effects which must be achieved, actions to be expanded, etc. A domain description

may contain more than one task definitions representing alternative requirements and initial conditions. The task schema may also specify:

- the top level ordering of activities
- the initial effects state of the domain specific to this task unlike initially facts which are common to all task definitions.
- the temporal constraints which the generated plan must meet
- the initial levels of resources

The task definition for demonstration is as follows:

```
task Operation_Columbus;
  nodes sequential
          1 start,
          parallel
            3 action {transport_ground_transports Honolulu Delta},
            4 action {transport_helicopters Honolulu Delta}
          end_parallel,
          parallel
            5 action {evacuate Abyss 50},
            6 action {evacuate Barnacle 100},
            7 action {evacuate Calypso 20}
          end_parallel,
          parallel
            8 action {fly_passengers Delta Honolulu},
            9 action {transport_ground_transports Delta Honolulu},
           10 action {transport_helicopters Delta Honolulu}
          end_parallel,
          2 finish
        end_sequential;

  effects {location_gt GT1} = Honolulu at 1,
          {location_gt GT2} = Honolulu at 1,
          {in_use_for GT1} = in_transit at 1,
          {in_use_for GT2} = in_transit at 1,

          {location_at AT1} = Honolulu at 1,
          {in_use_for AT1} = in_transit at 1,

          {apportioned_forces GT} at 1,
          {apportioned_forces AT} at 1,

          {at C141} = Honolulu at 1,
          {at C5} = Honolulu at 1,
          {at KC10} = Honolulu at 1,
```

```
        {at B707} = Delta at 1,
        {runway_status_at Delta} = clear at 1,
        {runway_status_at Honolulu} = clear at 1,
        {gt_capacity 25} at 1,
        {at_capacity 35} at 1;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 0 .. 15000 gallons overall,
    consumes {resource aviation_fuel_KC10_tank1} = 0 .. 30000 gallons overall,
    consumes {resource diesel_fuel_delta_tank2} = 0 .. 8000 gallons overall;

end_task;
```

Operator Definitions :

Operator schemas define the actions which can be executed in the domain. A schema define the schema variables, decomposition, conditions, effects, resource and temporal constraints. Details of the structure for the schema components can be found in the O-Plan TF manual. The schema to transport evacuees from an outlying city to Delta is as follows:

```
schema Road_Transport;
  vars ?from     = ?{type city},
       ?to       = ?{type air_base},
       ?gt       = ?{type ground_transport},
       ?e_left   = ?{satisfies numberp},
       ?e_safe   = ?{satisfies numberp},
       ?c_left   = ?{satisfies numberp},
       ?c_safe   = ?{satisfies numberp},
       ?capacity = ?{satisfies numberp},
       ?take     = ?{satisfies numberp};
  only_use_for_effects {evac_status ?from} = {?e_left ?e_safe};
  nodes 1 action {drive ?take in ?gt from ?from},
        2 dummy;
  conditions only_use_if {apportioned_forces GT},
             only_use_if {evacuate_to ?to},
             only_use_if {gt_capacity ?capacity},
             compute {transport_step ?capacity ?e_left ?e_safe}
                          = {?c_left ?c_safe},
             compute {- ?e_safe ?c_safe} = ?take,
             achieve {evac_status ?from} = {?c_left ?c_safe} at 2,
             unsupervised {location_gt ?gt} = ?to at begin_of 1,
             unsupervised {in_use_for ?gt} = available at begin_of 1,
               supervised {in_use_for ?gt} = ?from
                          at end_of 1 from begin_of 1;
  effects {in_use_for ?gt} = ?from at begin_of 1,
```

```
          {in_use_for ?gt} = available at end_of 1;
end_schema;
```

4 Results and Plans Obtained

The aim of this section is to describe the plans which were obtained and their relevance to military logistics planning.

The tasks specified in the domain description required the planner to create plans in which a specified number of evacuees were to be moved from a number of known locations. The evacuees were to be moved by an appropriate mixture of transport assets, e.g. ground transports (trucks and buses) and helicopters. In all tasks the ground transports, helicopters and transports planes (C141 and C5) are initially positioned in Honolulu and the B707 passenger plane is at Delta airport. The tasks and their descriptions are as follows:

Operation_Columbus :
: The evacuees can be moved from the outlying cities to Delta by either ground transports or helicopters. Aviation fuel on the island is limited and this should result in the plan containing steps to bring the KC-10 tanker from Honolulu with extra aviation fuel for the B707. Diesel fuel on the island is plentiful and should allow all evacuations to be achieved via ground transports. However, the time constraints imposed by the task make this option impossible forcing the planner to create plans involving a mixture of transport assets.

Operation_Columbus_Mixed_Transports :
: This task in similar to `Operation_Columbus` however, in this task there is sufficient aviation fuel and diesel fuel on the island. The time constraints imposed in `Operation_Columbus` have also been removed which would allow all evacuations to be carried out with the same type of transport.

Operation_Columbus_Ground_Transports_Only :
: This task is similar to `Operation_Columbus` except the only aviation fuel available on the island is allocated to the B707. All evacuations must be made using ground transports. There is sufficient diesel fuel on the island for this to be achieved.

Operation_Columbus_Helicopters_Only :
: In this task the time constraints and road conditions are such that the evacuations can only be achieved using helicopter transports. There is sufficient aviation fuel on the island for this to be achieved.

Plans were successfully generated for each of the tasks specified above and where possible a number of alternative replans were also obtained. The replans show how it would be possible to achieve the *same* task requirements but with a different tactical plan, e.g. different resource allocations `send GT2 instead of GT1`, transport method, etc.

The tasks described in the demonstration represent an interesting class of problems faced by the US military. They demonstrate how a series of transport assets (planes, helicopters, ground transports) and cargo (fuel) can be moved and coordinated between one location and another. The plans produced took into consideration many real world constraints on time and resources.

However, in order to simplify the demonstration a number of restrictions were introduced, e.g. fuel for the C5 and C141, crew schedules and maintenance periods were ignored. It would have been possible however, to introduce these resource and time constraints without drastically altering the performance of the system. The overall performance of the system showed that plan domains can be encoded with resource information and that plans can be generated which use this knowledge to restrict the options and choices to be considered.

5 Summary and Further Work

The aim of the section is to summarise the results of the demonstration and to provide pointers to further work in this domain.

The demonstration successfully showed that plans could be generated for a number of different resource related tasks specified in the PRECiS domain. A number of techniques were explored and validated which showed how resources could be defined and manipulated using a range of methods. These methods made explicit use of O-Plan's Resource Utilisation Manager to track consumable resources and O-Plan's TOME and GOST Manager to track reusable/sharable resources. Whilst these method allow the same breadth of coverage as was expected with the new RUM [6, 7] they do not have the same level of flexibility and support.

In tasks where the resources were limited (e.g. small amounts of diesel fuel) the system was able to use knowledge of resources to rule out certain options as being impossible. In tasks where the choices were more extensive i.e. use either transport types with no temporal restrictions the system was still able to find a solution in an acceptable period of time.

Since the demonstration was concluded further changes have been made to the domain description. The main changes have occurred in two areas and concern the movement of the C141 and C5 cargo transports.

Number of Transports :
The number of transports to be moved and their designations (e.g. ground transports GT1, GT2, GT3, etc) was encoded explicitly and each had the same attributes, capacity, size, weight. etc were the same in the original encoding. Subsequent changes now allow the system to "calculate" the number of transports it requires and their type e.g. 4 large ground transports, 2 small ground transports, etc. The system then selects from an inventory of transport type and creates a plan to move the required number of each type.

Transporter Capacities :
A specific type of transport e.g. ground or helicopter was assigned to be move by a specified air transporter in the original encoding. Subsequent changes now allow the transports to be mixed and loaded on to the C5 and C141 in the most appropriate order. For example, a C5 can carry 50 tons of cargo which could be made up of five small ground transports (10 tons each) or three large ground transports (15 tons each) and one helicopter (4 tons). The system uses the list of transports to be moved and creates "loads" for the C5 and C141 planes to carry.

As described in Section 1 the second aim of the demonstration was to validate the system's numerical representation and reasoning capabilities. The need to reason about resource levels and the tracking of evacuees provided a good test of these capabilities. It was possible to use `compute` conditions and O-Plan's built in mathematical functions e.g. `add`, `subtract`, etc to calculate the numerical changes occurring in the plan. In carrying out this validation a number of functionality issues concerning plan state variables (PSVs) were identified. These require the PSV Manager to allow numeric PSVs and to allow the domain encoder to specify range constraints on the numeric PSVs. These changes were made during the preparation work for the demonstration and will appear on the next planned release of O-Plan in July 1995.

References

[1] Beck, H., *TOSCA: A Novel Approach to the Management of Job-Shop Scheduling Constraints*, in Realising CIM's Industrial Potential, proceedings of the Ninth CIM-Europe Annual Conference, (eds) C. Kooij and P.A. MacConaill and J. Bastos, Amsterdam, 12-14 May, 1993.

[2] Brown, R., *Knowledge-based Scheduling and Resource Allocation in the CAMPS Architecture*, in proceedings from the International Conference on expert Systems and the Leading Edge in Production Planning and Control (Ed. M. Oliff.), Benjamin/Cummings, Menlo Park, CA 1987.

[3] Brown, R.H., Millen, J.K. and Ethan, A., *KNOBS: Final Report*, Technical Report RADC-TR-86-95, Rome Air Development Centre, Griffis Air Force Base, NY, 13441-5700, 1982.

[4] Dawson, B., Day, D.S. and Mulvehill, A., *The AMPS Final Report*, Final Report, USAF Rome Laboratory Technical Report RADC-TR-90-131. July 1990.

[5] Drabble, B., *Comparison of the CAMPS system with O-Plan2: Architecture and Reasoning Capabilities*, Technical Report DARPA-RL/O-Plan2/TR/12.

[6] Drabble, B. and Tate, A. *Resource Representation and Reasoning in O-Plan2*, Technical Report DARPA-RL/O-Plan2/TR/6.

[7] Drabble, B. and Tate, A., *The Use of Optimistic and Pessimistic Resource Profiles to Inform Search in an Activity Based Planner*, in proceedings of the Second International Conference on AI Planning Systems" University of Chicago, Chicago Illinois, June 1994. Publishers the American Association for Artificial Intelligence, Menlo Park, California.

[8] Hankins, G.B., Jordan, J.W., Katz, J.L., Mulvehill, A.M., Dumoulin, J.N. and Ragusa, J. *EMPRESS: Expert Mission Planning and REplanning Scheduling System*, in proccedings of the Expert Systems in Government Symposium, 1985.

[9] Lehrer, N., (ed.), ARPI KRSL Reference Manual 2.0.2, February, 1993. ISX Corporation.

[10] Mulvehill, A., *CAMPS/AMPS FY88 Year end Report*, The MITRE Corporation, Technical Report M89-28, May 1989.

[11] Reece, G., Tate, A., Brown, D., and Hoffman, M. *The PRECiS Environment*, papers of the ARPA/Rome Laboratory Planning Initiative Workshop at the National Conference on Artificial Intelligence (AAAI-93), Washington D.C., USA. ARPI Report ARPA-RL/CPE/Version 1, August 1993. Also available as Artificial Intelligence and Applications Institute Technical Report AIAI-TR-140, University of Edinburgh.

[12] Smith, S.F., *A Constraint-Based Framework for Reactive Management of Factory Schedules*, in proceedings International Conference of Expert Systems and the Leading Edge in Production Planning and Control, Charleston, South Carolina, May, 1987.

[13] Zweben, M., *CAMPS: A Dynamic Replanning System*, The MITRE Corporation, Technical Report M87-50, December 1986.

6 Appendix A

This appendix provides a description of the TF domain used in the second year demonstration of the O-Plan system.

```
;;; Concept: Glen A. Reece (DAI, Edinburgh) and Austin Tate (AIAI)
;;;          12th December 1992.
;;;
;;; File: pacifica3.tf
;;;
;;; Purpose: PRECiS NEO (Non-combatant Evacuation Operations) Scenario.
;;;          Domain description for transportation logistics problems.
;;;
;;; Created:  Brian Drabble and Jeff Dalton:  20th September 1994
;;;
;;; This file contains the TF which forms the second year demonstration of
;;; the O-Plan project. The aim of the project is to show the benefits of
;;; having a rich model of resources in an activity planner, The type
;;; of resources in the O-Plan resource hierarchy which are handled in the
;;; demonstration are as follows
;;;
;;;     1. consumable-strictly:
;;;        Aviation and diesel fuel in the tanks at Delta and the evacuees
;;;        at Abyss, Barnacle and Calypso.
;;;
;;;     2. consumable-producible-by-agent:
;;;        Aviation fuel brought by the KC-10 Tanker aircraft from Honolulu
;;;        when it is there is insufficient fuel at Delta to refuel the
;;;        B707 passenger aircraft prior to departure
;;;
;;;     3. consumable-producible-by-outwith-agent:
;;;        This resource is not handled as expected due to the version 2.2
;;;        not having the required event handling ability
;;;
;;;     4. reusable-nonsharable:
;;;        The ground transports, helicopters, C5, C141, KC-10 and B707.
;;;
;;;     5. resusable-sharable-independently:
;;;        The taxi ways at both Honolulu and Delta airbases.
;;;
;;;     6. reusable-sharable-synchronously:
;;;        The runways at both Honolulu and Delta airbases
;;;
;;; A City contains a particular numbers of people to be evacuated.
;;; GTs and ATs have a capacity, so more than one trip to a city may be
;;; needed. The number still to be moved is tracked using techniques
;;; developed for the Missionaries and Cannibals problem.  The M&C approach
;;; avoids uncertainty about the numbers (i.e. avoids PSVs).
```

```
;;;
;;;-------------------------------------------------------------------
;;;                 Domain Fact and Type Definitions

types
 ;;;
 ;;; Transport Assets
 ;;;
      ground_transport         = (GT1 GT2),
      helicopter               = (AT1),
      air_transporter_cargo    = (C5 C141),
      air_transporter_evacuees = (B707),
      air_fuel_transporter     = (KC10),
      runway_status            = (clear in_use),
      cargo_types              = (passengers ground_transports air_transports),
 ;;;
 ;;; Geographic Information
 ;;;
      country          = (Pacifica Hawaii_USA),
      location         = (Abyss Barnacle Calypso Delta Honolulu),
      city             = (Abyss Barnacle Calypso Delta),
      air_base         = (Delta Honolulu),
      transport_use    = (in_transit available Abyss Barnacle Calypso Delta);
 ;;;
 ;;; resource information
 ;;;
 ;;;   resource_units gallons = count;

resource_types
     consumable_strictly {resource aviation_fuel_delta_tank1} = gallons,
     consumable_strictly {resource diesel_fuel_delta_tank2} = gallons,
     consumable_strictly {resource aviation_fuel_KC10_tank1} = gallons;

always {avgas_fuel_required Abyss Delta 140},
       {avgas_fuel_required Barnacle Delta 160},
       {avgas_fuel_required Calypso Delta 180},
       {diesel_fuel_required Abyss Delta 40},
       {diesel_fuel_required Barnacle Delta 60},
       {diesel_fuel_required Calypso Delta 80},

       {country Abyss} = Pacifica,
       {country Barnacle} = Pacifica,
       {country Calypso} = Pacifica,
       {country Delta} = Pacifica,
       {country Honolulu} = Hawaii_USA;

initially
;;;    {evac_status <city>}   = {<# left at city> <# safe at evac pt>}
       {evac_status Abyss}    = { 50 0},
       {evac_status Barnacle} = {100 0},
```

```
       {evac_status Calypso}  = { 20 0},
       {evacuate_to Delta};   ;;; the evacuation point in Pacifica

;;;
;;;----------------------------------------------------------------------
;;;                      Task Definitions

;;; Task Evacuate_to_Delta evacuates people from Abyss, Barnacle and
;;; Calypso to Delta.  It used GTs already present in Delta so that
;;; no air transport is needed.

task Operation_Evacuate_to_Delta;
  nodes sequential
          1 start,
          2 finish
        end_sequential;
  conditions achieve {evac_status Abyss}    = {0  50} at 2,
             achieve {evac_status Barnacle} = {0 100} at 2,
             achieve {evac_status Calypso}  = {0  20} at 2;
  effects {location_gt GT1} = Delta at 1,
          {location_gt GT2} = Delta at 1,
          {in_use_for GT1} = available at 1,
          {in_use_for GT2} = available at 1,
          {gt_capacity 50} at 1;
end_task;

;;; Task Evacuate_to_Hawaii shows what happens when air transport is
;;; added.

task Operation_Columbus;
  nodes sequential
          1 start,
          parallel
            3 action {transport_ground_transports Honolulu Delta},
            4 action {transport_helicopters Honolulu Delta}
  end_parallel,
          parallel
            5 action {evacuate Abyss 50},
            6 action {evacuate Barnacle 100},
            7 action {evacuate Calypso 20}
          end_parallel,
          parallel
            8 action {fly_passengers Delta Honolulu},
            9 action {transport_ground_transports Delta Honolulu},
           10 action {transport_helicopters Delta Honolulu}
          end_parallel,
          2 finish
        end_sequential;

  effects {location_gt GT1} = Honolulu at 1,
          {location_gt GT2} = Honolulu at 1,
```

```
          {in_use_for GT1} = in_transit at 1,
          {in_use_for GT2} = in_transit at 1,

          {location_at AT1} = Honolulu at 1,
          {in_use_for AT1} = in_transit at 1,

  {apportioned_forces GT} at 1,
  {apportioned_forces AT} at 1,

  {at C141} = Honolulu at 1,
  {at C5} = Honolulu at 1,
  {at KC10} = Honolulu at 1,
  {at B707} = Delta at 1,
  {runway_status_at Delta} = clear at 1,
  {runway_status_at Honolulu} = clear at 1,
          {gt_capacity 25} at 1,
          {at_capacity 35} at 1;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 0 .. 15000 gallons overall,
    consumes {resource aviation_fuel_KC10_tank1} = 0 .. 30000 gallons overall,
    consumes {resource diesel_fuel_delta_tank2} = 0 .. 8000 gallons overall;

end_task;

task Operation_Columbus_Mixed_Transports;
  nodes sequential
          1 start,
          parallel
            3 action {transport_ground_transports Honolulu Delta},
            4 action {transport_helicopters Honolulu Delta}
  end_parallel,
          parallel
            5 action {evacuate Abyss 50},
            6 action {evacuate Barnacle 100},
            7 action {evacuate Calypso 20}
          end_parallel,
          parallel
            8 action {fly_passengers Delta Honolulu},
            9 action {transport_ground_transports Delta Honolulu},
           10 action {transport_helicopters Delta Honolulu}
          end_parallel,
          2 finish
        end_sequential;

  effects {location_gt GT1} = Honolulu at 1,
          {location_gt GT2} = Honolulu at 1,
          {in_use_for GT1} = in_transit at 1,
          {in_use_for GT2} = in_transit at 1,

          {location_at AT1} = Honolulu at 1,
```

```
        {in_use_for AT1} = in_transit at 1,

  {apportioned_forces GT} at 1,
  {apportioned_forces AT} at 1,

  {at C141} = Honolulu at 1,
  {at C5} = Honolulu at 1,
  {at KC10} = Honolulu at 1,
  {at B707} = Delta at 1,
  {runway_status_at Delta} = clear at 1,
  {runway_status_at Honolulu} = clear at 1,
        {gt_capacity 25} at 1,
        {at_capacity 35} at 1;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 0 .. 40000 gallons overall,
    consumes {resource aviation_fuel_KC10_tank1} = 0 .. 30000 gallons overall,
    consumes {resource diesel_fuel_delta_tank2} = 0 .. 8000 gallons overall;

end_task;

task Operation_Columbus_Ground_Transports_Only;
  nodes sequential
        1 start,
        parallel
          3 action {transport_ground_transports Honolulu Delta},
          4 action {transport_helicopters Honolulu Delta}
  end_parallel,
        parallel
          5 action {evacuate Abyss 50},
          6 action {evacuate Barnacle 100},
          7 action {evacuate Calypso 20}
        end_parallel,
        parallel
          8 action {fly_passengers Delta Honolulu},
          9 action {transport_ground_transports Delta Honolulu},
         10 action {transport_helicopters Delta Honolulu}
        end_parallel,
        2 finish
      end_sequential;

  effects {location_gt GT1} = Honolulu at 1,
        {location_gt GT2} = Honolulu at 1,
        {in_use_for GT1} = in_transit at 1,
        {in_use_for GT2} = in_transit at 1,

        {location_at AT1} = Honolulu at 1,
        {in_use_for AT1} = in_transit at 1,

  {apportioned_forces GT} at 1,
```

```
  {at C141} = Honolulu at 1,
  {at C5} = Honolulu at 1,
  {at KC10} = Honolulu at 1,
  {at B707} = Delta at 1,
  {runway_status_at Delta} = clear at 1,
  {runway_status_at Honolulu} = clear at 1,
          {gt_capacity 25} at 1,
          {at_capacity 35} at 1;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 0 .. 40000 gallons overall,
    consumes {resource aviation_fuel_KC10_tank1} = 0 .. 30000 gallons overall,
    consumes {resource diesel_fuel_delta_tank2} = 0 .. 8000 gallons overall;

end_task;

task Operation_Columbus_Helicopters_Only;
  nodes sequential
          1 start,
          parallel
            3 action {transport_ground_transports Honolulu Delta},
            4 action {transport_helicopters Honolulu Delta}
  end_parallel,
          parallel
            5 action {evacuate Abyss 50},
            6 action {evacuate Barnacle 100},
            7 action {evacuate Calypso 20}
          end_parallel,
          parallel
            8 action {fly_passengers Delta Honolulu},
            9 action {transport_ground_transports Delta Honolulu},
           10 action {transport_helicopters Delta Honolulu}
          end_parallel,
          2 finish
        end_sequential;

  effects {location_gt GT1} = Honolulu at 1,
          {location_gt GT2} = Honolulu at 1,
          {in_use_for GT1} = in_transit at 1,
          {in_use_for GT2} = in_transit at 1,

          {location_at AT1} = Honolulu at 1,
          {in_use_for AT1} = in_transit at 1,

  {apportioned_forces AT} at 1,

  {at C141} = Honolulu at 1,
  {at C5} = Honolulu at 1,
  {at KC10} = Honolulu at 1,
  {at B707} = Delta at 1,
  {runway_status_at Delta} = clear at 1,
```

```
  {runway_status_at Honolulu} = clear at 1,
          {gt_capacity 25} at 1,
          {at_capacity 35} at 1;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 0 .. 40000 gallons overall,
    consumes {resource aviation_fuel_KC10_tank1} = 0 .. 30000 gallons overall,
    consumes {resource diesel_fuel_delta_tank2} = 0 .. 8000 gallons overall;

end_task;

;;; Task Large_one_city_evacuation uses lower-capacity GTs so that more
;;; trips are required -- the "large" is relative to GT capacity.  GT
;;; allocation is simplified by evacuating only one city.

task Operation_Large_one_city_evacuation;
  nodes sequential
          1 start,
          2 finish
        end_sequential;
  conditions achieve {evac_status Barnacle} = {0 100} at 2;
  effects {location_gt GT1} = Delta at 1,
          {location_gt GT2} = Delta at 1,
          {in_use_for GT1} = available at 1,
          {in_use_for GT2} = available at 1,
          {gt_capacity 25} at 1;
  time_windows 0..12 hours at end_of 2;
end_task;

;;; The "Fast" version can be done only if some road transport steps
;;; occur in parallel.  There are enough GTs to do two at once, but
;;; our methods for counting people don't allow it.

task Operation_Fast_large_one_city_evacuation;
  nodes sequential
          1 start,
          2 finish
        end_sequential;
  conditions achieve {evac_status Barnacle} = {0 100} at 2;
  effects {location_gt GT1} = Delta at 1,
          {location_gt GT2} = Delta at 1,
          {in_use_for GT1} = available at 1,
          {in_use_for GT2} = available at 1,
          {gt_capacity 25} at 1;
  time_windows 0..6 hours at end_of 2;
end_task;

;;;
;;;------------------------------------------------------------------------
;;;                 Transportation and Counting Operators
```

```
schema evacuate_City;
  vars ?city   = ?{type city},
       ?number = ?{satisfies numberp};
  expands {evacuate ?city ?number};
  conditions achieve {evac_status ?city} = {0 ?number};
end_schema;

;;; Road_transport Air Transport and people-counting

;;; Variables that refer to evac_status values are marked e_ for effect,
;;; c_ for condition.  The schema must be invoked "for effects".
;;; All e_ variables must be bound by the time the schema is selected.
;;; The c_ values are then computed from the e_ values without
;;; introducing any PSVs.  OTOH, ?gt does become a PSV.

schema Road_Transport;
  vars ?from     = ?{type city},
       ?to       = ?{type air_base},
       ?gt       = ?{type ground_transport},
       ?e_left   = ?{satisfies numberp},
       ?e_safe   = ?{satisfies numberp},
       ?c_left   = ?{satisfies numberp},
       ?c_safe   = ?{satisfies numberp},
       ?capacity = ?{satisfies numberp},
       ?take     = ?{satisfies numberp};
  only_use_for_effects {evac_status ?from} = {?e_left ?e_safe};
  nodes 1 action {drive ?take in ?gt from ?from},
        2 dummy;
  conditions only_use_if {apportioned_forces GT},
     only_use_if {evacuate_to ?to},
             only_use_if {gt_capacity ?capacity},
             compute {transport_step ?capacity ?e_left ?e_safe}
                         = {?c_left ?c_safe},
             compute {- ?e_safe ?c_safe} = ?take,
             achieve {evac_status ?from} = {?c_left ?c_safe} at 2,
             unsupervised {location_gt ?gt} = ?to at begin_of 1,
             unsupervised {in_use_for ?gt} = available at begin_of 1,
               supervised {in_use_for ?gt} = ?from
                          at end_of 1 from begin_of 1;
  effects {in_use_for ?gt} = ?from at begin_of 1,
          {in_use_for ?gt} = available at end_of 1;
end_schema;

;;; Variables that refer to evac_status values are marked e_ for effect,
;;; c_ for condition.  The schema must be invoked "for effects".
;;; All e_ variables must be bound by the time the schema is selected.
;;; The c_ values are then computed from the e_ values without
;;; introducing any PSVs.  OTOH, ?gt does become a PSV.

schema Air_Transport;
  vars ?from     = ?{type city},
```

```
       ?to       = ?{type air_base},
       ?at       = ?{type helicopter},
       ?e_left   = ?{satisfies numberp},
       ?e_safe   = ?{satisfies numberp},
       ?c_left   = ?{satisfies numberp},
       ?c_safe   = ?{satisfies numberp},
       ?capacity = ?{satisfies numberp},
       ?take     = ?{satisfies numberp};
  only_use_for_effects {evac_status ?from} = {?e_left ?e_safe};
  nodes 1 action {fly ?take in ?at from ?from},
        2 dummy;
  conditions only_use_if {apportioned_forces AT},
             only_use_if {evacuate_to ?to},
             only_use_if {at_capacity ?capacity},
             compute {transport_step ?capacity ?e_left ?e_safe}
                        = {?c_left ?c_safe},
             compute {- ?e_safe ?c_safe} = ?take,
             achieve {evac_status ?from} = {?c_left ?c_safe} at 2,
             unsupervised {location_at ?at} = ?to at begin_of 1,
             unsupervised {in_use_for ?at} = available at begin_of 1,
               supervised {in_use_for ?at} = ?from
                          at end_of 1 from begin_of 1;
  effects {in_use_for ?at} = ?from at begin_of 1,
          {in_use_for ?at} = available at end_of 1;
end_schema;

;;; Suppose we have oufe {evac_status A} = {0 100}, with capacity = 10.
;;; Well, we can do that if we can get {evac_status A} = {10 90}.  So
;;; we'll post {10 90} as an achieve.  The next road_transport activation
;;; will post {evac_status A} = {20 80} as an achieve, and so on until
;;; we reach {evac_status A} = {100 0} which is true initially.  Each
;;; step from an oufe to the required achieve is computed by the
;;; transport_step function below.

;;;
;;;-------------------------------------------------------------------------
;;;            Transport and Passenger Movement Operators

schema transport_ground_transports;
;;;
;;; This schema is used to provide ground transportation from
;;; one air base location to another air base location.  This must take
;;; place by using a cargo plane to fly the nominated transportation
;;; vehicles from the source base to the destination base.
;;;
;;; fly all transport explicitly from one base to another. No "forall" yet.
;;;
  vars  ?FROM = ?{type air_base},
        ?TO   = ?{type air_base};

  expands {transport_ground_transports ?FROM ?TO};
```

```
  nodes 1 action {load ground_transports},
2 action {take_off_from ?FROM},
3 action {fly_to ?TO},
4 action {land_at ?TO},
5 action {unload ground_transports};

  orderings 1 ---> 2, 2 ---> 3, 3 ---> 4, 4 ---> 5;

  conditions achieve {at C5} = ?FROM at 1,
             unsupervised {location_gt GT1} = ?FROM at 1,
             unsupervised {location_gt GT2} = ?FROM at 1,
     unsupervised {runway_status_at ?FROM} = clear at begin_of 2,
     supervised {runway_status_at ?FROM} = in_use at end_of 2 from
begin_of 2,
     unsupervised {runway_status_at ?TO} = clear at begin_of 4,
     supervised {runway_status_at ?TO} = in_use at end_of 4 from
begin_of 4;
  effects {at C5} = ?TO at 5,
          {location_gt GT1} = ?TO at 5,
          {location_gt GT2} = ?TO at 5,
          {in_use_for GT1} = available at 5,
          {in_use_for GT2} = available at 5,
  {runway_status_at ?FROM} = in_use at begin_of 2,
  {runway_status_at ?FROM} = clear at end_of 2,
  {runway_status_at ?TO} = in_use at begin_of 4,
  {runway_status_at ?TO} = clear at end_of 4;
end_schema;

;;;
;;;----------------------------------------------------------------------------
;;;

schema transport_extra_fuel;
;;;
;;; This schema is used to provide air transportation of fuel from Honolulu
;;; to Delta. Planes transport explictly from one base to another.
;;; No "forall" yet.
;;;
  vars  ?FROM = ?{type air_base},
        ?TO   = ?{type air_base};

  expands {transport_fuel_reserves ?FROM ?TO};

  nodes 1 action {take_off_from ?FROM},
2 action {fly_to ?TO},
3 action {land_at ?TO};

  orderings 1 ---> 2, 2 ---> 3;

  conditions achieve {at KC10} = ?FROM at 1,
```

```
     unsupervised {runway_status_at ?FROM} = clear at begin_of 1,
     supervised {runway_status_at ?FROM} = in_use at end_of 1 from
begin_of 1,
     unsupervised {runway_status_at ?TO} = clear at begin_of 3,
     supervised {runway_status_at ?TO} = in_use at end_of 3 from
begin_of 3;
  effects {at KC10} = ?TO,
  {runway_status_at ?FROM} = in_use at begin_of 1,
  {runway_status_at ?FROM} = clear at end_of 1,
  {runway_status_at ?TO} = in_use at begin_of 3,
  {runway_status_at ?TO} = clear at end_of 3;
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;

schema transport_helicopters;
;;;
;;; This schema is used to provide air transportation from
;;; one air base location to another air base location.  This must take
;;; place by using a cargo plane to fly the nominated transportation
;;; helicopters from the source base to the destination base.
;;;
;;; fly all transport explictly from one base to another. No "forall" yet.
;;;
;;;
  vars  ?FROM = ?{type air_base},
        ?TO   = ?{type air_base};

  expands {transport_helicopters ?FROM ?TO};

  nodes 1 action {load air_transports},
2 action {take_off_from ?FROM},
3 action {fly_to ?TO},
4 action {land_at ?TO},
5 action {unload air_transports};

  orderings 1 ---> 2, 2 ---> 3, 3 ---> 4, 4 ---> 5;

  conditions achieve {at C141} = ?FROM at 1,
             unsupervised {location_at AT1} = ?FROM at 1,
     unsupervised {runway_status_at ?FROM} = clear at begin_of 2,
     supervised {runway_status_at ?FROM} = in_use at end_of 2 from
begin_of 2,
     unsupervised {runway_status_at ?TO} = clear at begin_of 4,
     supervised {runway_status_at ?TO} = in_use at end_of 4 from
begin_of 4;
  effects {at C141} = ?TO,
          {location_at AT1} = ?TO at 5,
          {in_use_for AT1} = available at 5,
```

```
  {runway_status_at ?FROM} = in_use at begin_of 2,
  {runway_status_at ?FROM} = clear at end_of 2,
  {runway_status_at ?TO} = in_use at begin_of 4,
  {runway_status_at ?TO} = clear at end_of 4;
end_schema;

;;;
;;;----------------------------------------------------------------------
;;;

schema fly_passengers;
;;;
;;; This schema transports the evacuees from the airport at Delta to
;;; Honolulu. The passenger transport needs 20,000 gallons of fuel for the
;;; journey which it obtains from the tanks at Delta. .
;;; The evacuees must be moved by passenger aircraft.
;;;
  vars ?TO   = ?{type air_base},
       ?FROM = ?{type air_base};

  expands {fly_passengers ?FROM ?TO};

  nodes 1 action {load passengers},
2 action {take_off_from ?FROM},
3 action {fly_to ?TO},
4 action {land_at ?TO},
5 action {unload passengers};

  orderings 1 ---> 2, 2 ---> 3, 3 ---> 4, 4 ---> 5;

  conditions unsupervised {at B707} = ?FROM at 1,
     unsupervised {runway_status_at ?FROM} = clear at begin_of 2,
     supervised {runway_status_at ?FROM} = in_use at end_of 2
from begin_of 2,
     unsupervised {runway_status_at ?TO} = clear at begin_of 4,
     supervised {runway_status_at ?TO} = in_use at end_of 4
from begin_of 4;

  effects {at B707} = ?TO at 5,
  {runway_status_at ?FROM} = in_use at begin_of 2,
  {runway_status_at ?FROM} = clear at end_of 2,
  {runway_status_at ?TO} = in_use at begin_of 4,
  {runway_status_at ?TO} = clear at end_of 4,
          {nationals out} = true at 5;

  resources
    consumes {resource aviation_fuel_delta_tank1} = 20000 gallons;
end_schema;

;;;
;;;----------------------------------------------------------------------
```

```
;;;

schema fly_passengers_and_refuel_transporter;
;;;
;;; This schema transports the evacuees from the airport at Delta to
;;; Honolulu. The passenger transport needs 20,000 gallons of fuel for the
;;; journey which has to be brought from Honolulu in a KC-10 tanker transport.
;;; The evacuees must be moved by passenger aircraft.
;;;
  vars ?TO   = ?{type air_base},
       ?FROM = ?{type air_base};

  expands {fly_passengers ?FROM ?TO};

  nodes 1 action {load passengers},
2 action {take_off_from ?FROM},
3 action {fly_to ?TO},
4 action {land_at ?TO},
5 action {unload passengers};

  orderings 1 ---> 2, 2 ---> 3, 3 ---> 4, 4 ---> 5;

  conditions achieve {passenger_transporter_refuelled} at 2,
             unsupervised {at B707} = ?FROM at 1,
     unsupervised {runway_status_at ?FROM} = clear at begin_of 2,
     supervised {runway_status_at ?FROM} = in_use at end_of 2
from begin_of 2,
     unsupervised {runway_status_at ?TO} = clear at begin_of 4,
     supervised {runway_status_at ?TO} = in_use at end_of 4
from begin_of 4;

  effects {at B707} = ?TO at 5,
  {runway_status_at ?FROM} = in_use at begin_of 2,
  {runway_status_at ?FROM} = clear at end_of 2,
  {runway_status_at ?TO} = in_use at begin_of 4,
  {runway_status_at ?TO} = clear at end_of 4,
          {nationals out} = true at 5;

  resources
    consumes {resource aviation_fuel_KC10_tank1} = 20000 gallons;
end_schema;

;;;
;;;---------------------------------------------------------------------
;;;

schema refuel_transporter;

  only_use_for_effects {passenger_transporter_refuelled};

  nodes 1 action {transport_fuel_reserves Honolulu Delta},
```

```
        2 action {refuel_transporter},
        3 action {transport_fuel_reserves Delta Honolulu};

  orderings 1 ---> 2, 2 ---> 3;
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;          Primitive Level Schemas

schema drive;
  vars ?from          = ?{type city},
       ?take          = ?{satisfies numberp},
       ?fuel_required = ?{satisfies numberp},
       ?gt            = ?{type ground_transport};
  expands {drive ?take in ?gt from ?from};

  conditions
    only_use_if {diesel_fuel_required ?from Delta ?fuel_required};

  resources
    consumes {resource diesel_fuel_delta_tank2} = ?fuel_required gallons;

  time_windows duration self = 3 hours .. 4 hours;
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;

schema fly;
  vars ?from          = ?{type city},
       ?take          = ?{satisfies numberp},
       ?fuel_required = ?{satisfies numberp},
       ?at            = ?{type helicopter};

  expands {fly ?take in ?at from ?from};

  conditions
    only_use_if {avgas_fuel_required ?from Delta ?fuel_required};

  resources
    consumes {resource aviation_fuel_delta_tank1} = ?fuel_required gallons;

  time_windows duration self = 3 hours .. 4 hours;
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;
schema load_cargo;
```

```
  vars ?cargo = ?{type cargo_types};

  expands {load ?cargo};

  effects {loaded ?cargo};
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;

schema Add_fuel_to_transporter;

  expands {refuel_transporter};

  effects {transporter_refuelled};
end_schema;
;;;
;;;------------------------------------------------------------------------
;;;

schema take_off_from_airbase;
  vars ?airbase = ?{type air_base};

  expands {take_off_from ?airbase};

  effects {taken_off_from ?airbase};
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;

schema Fly_to_Airbase;
  vars ?airbase = ?{type air_base};

  expands {fly_to ?airbase};

  effects {in_transit_to ?airbase};
end_schema;

;;;
;;;------------------------------------------------------------------------
;;;

schema Land_at_airbase;
  vars ?airbase = ?{type air_base};

  expands {land_at ?airbase};

  effects {cargo_landed_at_airbase ?airbase};
```

```
end_schema;

;;;
;;;----------------------------------------------------------------------
;;;

schema Unload_cargo;
  vars ?cargo = ?{type cargo_types};

  expands {unload ?cargo};

  effects {unloaded ?cargo};
end_schema;

;;;
;;;----------------------------------------------------------------------
;;;                       Language Lisp Definitions

language lisp;

  (defun transport_step (capacity e_left e_safe) ; -> (c_left c_safe)
    (let ((take (min e_safe capacity)))
      (list (+ e_left take)
            (- e_safe take))))

end_language;

;;; End
```