O-Plan

User Guide

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

July 13, 1995

Version 2.3

Acknowledgements

The O-Plan project began in 1984. Since that time the following people have participated: Colin Bell, Ken Currie, Jeff Dalton, Roberto Desimone, Brian Drabble, Mark Drummond, Anja Haman, Ken Johnson, Richard Kirby, Glen Reece, Arthur Seaton, Judith Secker, Austin Tate and Richard Tobin.

Prior to 1984, work on Interplan (1972–4) and Nonlin (1975–6) was funded by the UK Science and Engineering Research Council and provided technical input to the design of O-Plan.

From 1984 to 1988, the O-Plan project was funded by the UK Science and Engineering Research Council on grant numbers GR/C/59178 and GR/D/58987 (UK Alvey Programme project number IKBS/151). The work was also supported by a fellowship from SD-Scicon for Austin Tate from 1984 to 1985.

From 1989 to 1992, the O-Plan project was supported by the US Air Force Rome Laboratory through the Air Force Office of Scientific Research (AFOSR) and their European Office of Aerospace Research and Development by contract number F49620-89-C-0081 (EOARD/88-0044) monitored by Northrup Fowler III at the USAF Rome Laboratory.

From 1992 to 1995, the O-Plan project was supported by the ARPA/Rome Laboratory Knowledge Based Planning and Scheduling Initiative through the US Air Force Rome Laboratory through the Air Force Office of Scientific Research (AFOSR) and their European Office of Aerospace Research and Development by contract number F49620-92-C-0042 (EOARD/92-0001) monitored by Northrup Fowler III at the USAF Rome Laboratory.

Additional resources for the O-Plan and O-Plan projects have been provided by the Artificial Intelligence Applications Institute through the EUROPA (Edinburgh University Research on Planning Architectures) institute development project.

From 1989 to 1993, research on scheduling applications of the O-Plan architecture was funded by Hitachi Europe Ltd. From 1989 to 1992, the UK Science and Engineering Research Council (grant number GR/F36545 – UK Information Engineering Directorate project number IED 4/1/1320) funded a collaborative project with ICL, Imperial College and other partners in which the O-Plan architecture was used to guide the design and development of a planner with a flexible temporal logic representation of the plan state. A number of other research and development contracts placed with AIAI have led to research progress on the O-Plan prototype.

Contact Information

The O-Plan project team can be contacted as follows:

O-Plan Team
Artificial Intelligence Applications Institute
The University of Edinburgh
80, South Bridge
Edinburgh EH1 1HN
United Kingdom

Tel: (+44) 131 650 2732
Fax: (+44) 131 650 6513
Email: oplan@ed.ac.uk

Created: July 19, 1993 by Brian Drabble
Last Modified: July 13, 1995 (10:00) by Brian Drabble
Printed: July 13, 1995

Contents

1 Introduction

1.1 Contents of this Guide

The aim of this guide is to provide the user with an overview of setting up and running the O-Plan system. This guide assumes that the O-Plan system has been successfully extracted from the tar image file and installed and compiled correctly.

You should have installed O-Plan in a directory different from any directory used for an earlier version of O-Plan to avoid any confusion between files from different versions. The installation procedure is described in the Installation Guide. In the remainder of this document it will sometimes be assumed that there is a shell command, `oplan`, that can be used to invoke O-Plan. The Installation Guide describes how to establish such a command.

This guide is divided into the following sections:

1. **Setting up to run O-Plan**: this section describes the O-Plan startup scripts and the environment variables used to run both the planner and execution agent.
2. **Running O-Plan**: this section describes how to run the O-Plan system as either a planning or execution agent.
3. **Plan and World Interfaces**: this section describes the PlanWorld interface used by O-Plan to allow the user to browse on either the plan or world state.
4. **O-Plan Processes and Internal Structure**: this section provides an overview of the way in which an O-Plan agent is implemented in terms of the the mapping of *processes* to components and the way in which these components communicate.
5. **Typing Lisp Commands**: this section deals with inputting of Lisp commands into the O-Plan system, e.g. turning on some of the debugging tools.
6. **Dealing with Lisp Errors**: this sections deals with Lisp errors which sometimes occur. These can either be caused by typing errors in entering Lisp commands or by errors in the O-Plan code. In the latter case they should be reported to the O-Plan development team.
7. **Control Panel and Trace Information within O-Plan**: this section deals with obtaining different levels of trace information within the O-Plan system.
8. **Example Demonstration**: this section provides step by step instructions on running an example demonstration. The demonstration domains available are described in detail in the Demonstration Guide.
9. **Example AutoCAD Demonstration**: this section provides step by step instructions on running AutoCAD as a PlanWorld viewer for the Pacifica and Space Station Assembly domains.

2 Running O-Plan and its Interfaces

The aim of this section is to describe the environment variables and scripts which have been provided with the O-Plan system together with details of the interface which allows for developer access to the various components.

2.1 O-Plan Scripts

Several scripts have been provided to start O-Plan and the AutoCAD bassed PlanWorld Viewers. These are as follows:

1. **oplan**
   This provides the basic O-Plan system with a textual interface for plan and world state browsing.
2. **oplan-acad-plan-view**
   This adds a graphical interface for browsing and editing plans based on the AutoCAD package. This can be used in all demonstration domains.
3. **oplan-acad-world-view**
   This adds a demonstration graphical interface for world browsing for a space platform construction task and the Pacifica NEO scenario both of which are based on the AutoCAD package.

See Section 4 of this document for further details on the Plan and World Viewers.

2.2 Environment Variables used by O-Plan

In order to run the demonstrations the user should be in the X windows environment and if this is not the case the the user should issue the necessary commands to invoke it. In most cases, it is not necessary for users to set any environment variables themselves. This means that zero work is required to switch between O-Plan versions, provided that they are invoked by different commands. However, the user may set some of the variables in order to change the defaults if that is desired. The following sections describe the environment variables which are used, their purpose and where necessary their setting or defaults.

- **setenv** OPLANDIR **<directory name>**
  The directory name should be the full name of the directory that contains the O-Plan system. This variable is set by the `oplan` script which should have been edited prior to the first invocation of O-Plan. The script's value persists only for the duration of the run. Users may also wish to set this variable in their global environment as a shorthand way of specifying the top level of the O-Plan file structure. That is how it will be used below. However, that setting will have no effect on the value used by `oplan`.

- **setenv** OPLANTFDIR **$**OPLANDIR**/demo/tf**
  Specifies a directory in which TF files containing domain descriptions are held. If the variable does not have a value, the `demo` TF directory will be used. It is also possible to change the directory at task selection time after the planner has been invoked.
- **setenv** OPLANTMPDIR **$**HOME**/oplan-tmp**
  Specifies the directory into which temporary files generated by the O-Plan system will be written. It is recommended that this environment variable be set to `oplan-tmp` in the users $HOME directory for uniformity with other O-Plan installations. Alternatively it may be:

  1. set to any user specified directory to which the user has the necessary read and write access privilege
  2. left unspecified in which case all temporary files will be written to the user's current working directory

- **setenv** OPLANACADDIR **$**OPLANDIR**/demo/autocad**
  Specifies the directory in which AutoCAD will look for drawing and other files for the PlanWorld Viewers. Note that AutoCAD requires write access to this directory since it makes locking files in the same directory in which it looks for its saved drawings.

  This environment variable is compulsory if using either the AutoCAD Plan or World Viewers.

2.3 Setting up to Run O-Plan

The O-Plan system is invoked from the startup script `oplan` which is held in the $OPLANDIR`/bin` directory. There are several ways to arrange for this script to be accessible as a shell command.

1. Define a shell alias, for shells that provide aliases. For instance, `tcsh` users can add the following line to their home directory's `.tcshrc` file:

   ```
   alias oplan "/applications/oplan/bin/oplan"
   ```

   Note that you do not have to use the name `oplan` for the alias.
2. Alter your `PATH` to include the directory containing the script.
3. Move the script to a directory which is on the users current `PATH`.

The O-Plan system is menu driven and requires the X Window System. Before running O-Plan:

1. If you are not already in X windows then execute the command to invoke it.
2. Ensure that the environment variables are set to the required values beforehand (normally this is done in the user's UNIX or shell startup file). These variables and their default values are described earlier in subsection 2.2.
3. Define a command that will invoke O-Plan, as described above.

3 Running O-Plan

Once the setup steps have been carried out (defined in subsections 2.2 and 2.3) then O-Plan can then be invoked by typing the relevant command. For example:

```
oplan
```

When you start O-Plan, a number of windows should appear. The first window to appear (`O-Plan Running Processes`) displays the names of the currently running parts of O-Plan. This window is described later in this document in the Pseudo-processes section. After that, one window will appear for the Task Assigner (TA) and one for each of the components of the planner (IM, AM, DM, KP-1). Finally a control panel (CP) window will appear over the top of the IM window. The window in which you typed the `oplan` command will stay connected to O-Plan. When all the other windows have appeared, the prompt `form>` should appear in this window. You can type Lisp commands in this window and consequently it is called the Lisp interaction window. For more information see the section Typing Lisp commands below.

The window configuration is specified by the `-config` argument to the `oplan` command. The default, screen-filling, configuration can also be specified by starting O-Plan as follows:

```
oplan -config oplan-basic
```

O-Plan will not allow a plan to be executed unless an execution capabilities has been specified at start up. This is specified by means of a separate configuration file and additional command line arguments. For demonstration purposes a simplified Execution Agent referred to as Micro-Exec has been developed and can be used with any of the O-Plan demonstration domains. Micro-Exec and a world simulator can be invoked by starting O-Plan as follows:

```
oplan -config oplan-plan-exec.config -allow-exec t
```

This will cause the Micro-Exec window and those of the world process simulator to appear at startup. A separate O-Plan agent can be spawned off dynamically to allow experimentation with external agents and can be invoked by starting O-Plan as follows:

```
oplan - allowexec t
```

This agent will not be spawned off until the user chooses to begin executing a plan and the Micro-Exec has not been specified in the configuration file. Note that you do not have to specify a configuration. The basic configuration is used by default. A full list of O-Plan command line arguments can be found in Section 3.1.

The planner will be brought up in its uninitialised state. This will be indicated by a message at the top of the TA window. The message will also indicate the date the image was created as well as the version number.

Each of the separate parts of the planner has its own window which allows the user to follow the plan as it is being generated. The major parts of the O-Plan systems are described in the Section 3.4.

3.1 O-Plan Command Line Arguments

The `oplan` startup script has been designed to accept a number of optional arguments which allow the user to customise certain aspects of the systems functions and display formats. A full list of the arguments are as follows:

Argument	Description
-break	Enters the Lisp break loop. This allows the user to examine the state of the world before O-Plan starts up or to call O-Plan procedures without starting O-Plan.
-config <`filename`>	Use the indicated configuration file. A series of configuration files are provided in $OPLANDIR/`lib`.
-eval <`form`>	Evaluate <`form`>, where <`form`> is a Common Lisp expression.
-load filename	Load the indicated file.
-noinit	Do not load any `oplan-init` file.
-tfc <tfc-arg, ...>	Run the TF syntax checker rather than as O-Plan. All arguments after the `-tfc` will be processed by the syntax checker See the TFC section of the Implementation Guide for further details.
-allow-exec	Informs the planner that an execution capability is allowed. The execution agent is either invoked at startup (specified via a configuration file) or dynamically as a separate agent if no configuration file is specified.

Note that all arguments are processed *before* O-Plan considers loading an `oplan-init` file. If you need something to happen earlier, use `-eval` or `-load`.

3.2 oplan-init file

When O-Plan starts, it looks for a file named `oplan-init`, first in the current directory and then in the user's home directory. If the file exists, is is loaded into Lisp.

The oplan-init file can contain (Lisp) commands that customise certain aspects of O-Plan. At present, only a few such customisations are defined. See the sections on defining a PostScript viewer and on the whats-going-on file for details.

3.3 Exiting O-Plan

The standard way to exit is to use the Quit option in the Task Assigner menu or to press the Quit button in the control panel window. However, if something has gone wrong, these methods may not work. In that case, you have several options.

The fastest way to exit is to type control and backslash in the Lisp interaction window. All the other windows should vanish, and Lisp will exit, leaving you back at the shell. This is a hard exit. Lisp may not perform cleanup tasks such as flushing buffered output. Typing control and C in the Lisp interaction window will also cause all other windows to vanish, but in most cases Lisp will not exit. To exit from Lucid Common Lisp, type `(lcl:quit)`.

3.4 Major Components of the the O-Plan System

The screen image in Figure 1 shows the screen layout for the planner after the initial setup. O-Plan provides an interface which allows for developer access to the various components. These components are as follows:

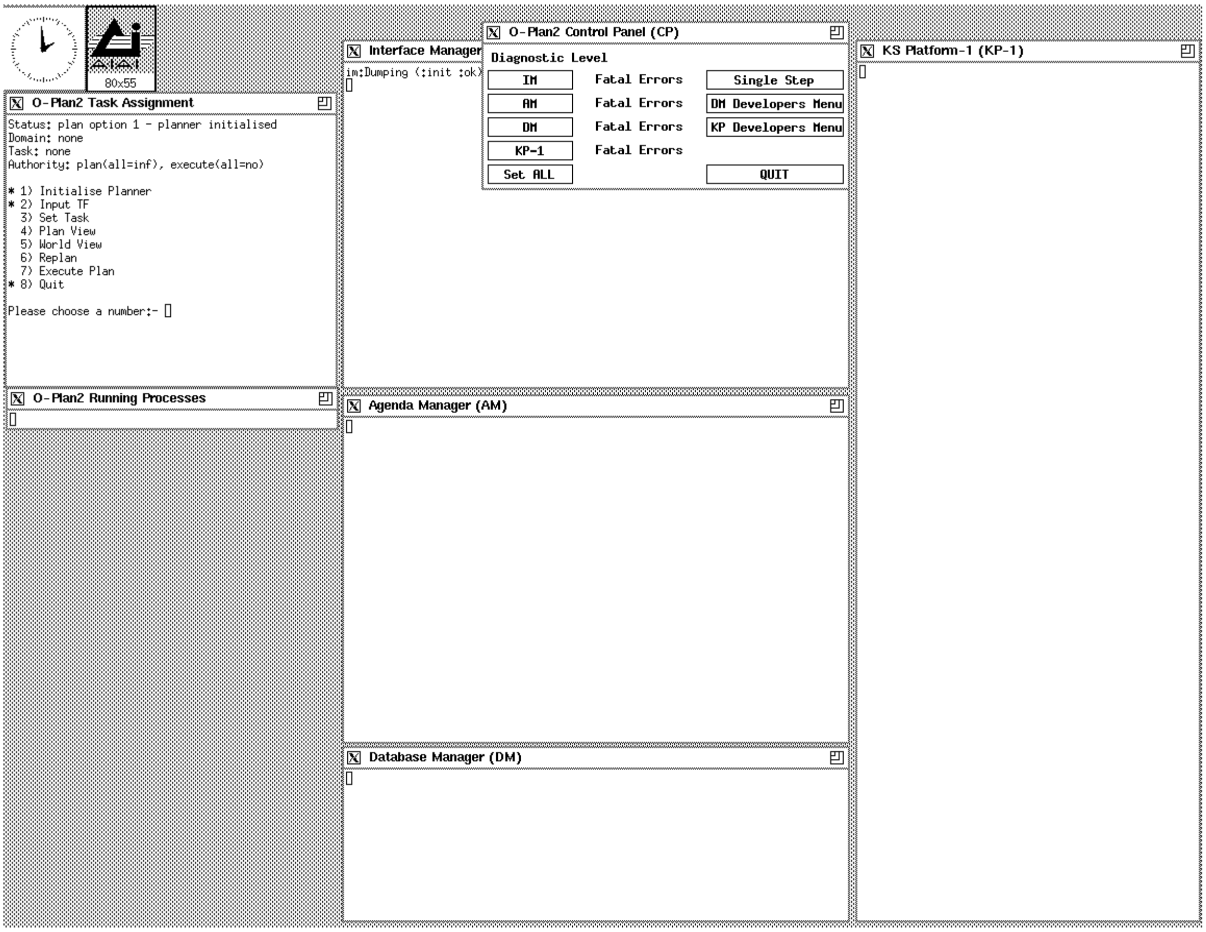

Figure 1: Example screen image from the O-Plan System

- Task Assignment Window
  This window allows the user to give a range of top level commands, such as giving the planner a new task, browsing on certain aspects of the plan once it has been generated (through a suitably defined interface, e.g. AutoCAD), etc.

- Interface Manager
  The interface manager handles interactions with agents external to the planner, e.g. the task assignment agent and the execution agent through a series of event channels. It also enables components of the O-Plan system to communicate between themselves and with a user acting in the role of a planner specialist. The control panel is part of the Interface

Manager and is used as a control panel to change the levels of diagnostics, instrumentation and monitors

- Agenda Manager
  The agenda manager keeps track of the current *flaws* within the plan and decides which flaw to tackle next. A flaw describes a particular problem in the plan which needs to be addressed. For example, expand an action to a more detailed level, make sure that a condition of an action is satisfied at the right point, ensure enough resources are available for an action, etc.

- Database Manager
  The database manager provides the support routines required by the knowledge sources and holds the emerging plan as it is being generated.

- Knowledge Source Platform
  All decision making within O-Plan is carried out through knowledge sources. There is a separate knowledge source for each possible flaw type within a plan and the invocation of the appropriate knowledge sources is one of the functions of the agenda manager. Knowledge sources run on a knowledge source platform. In the current system there is only one such platform.

- Running lights
  The running lights window shows the user which component is currently active and in the case of problems where the error may have occurred.

The user should identify each of the above components of the screen as they will be frequently referenced in the Example Demonstration Section. A more detailed description of the O-Plan architecture can be found in the Architecture Guide.

3.5 Running the O-Plan Execution Agent

With the environment variables defined appropriately the planner can be initiated via either of the methods described in Section 2.3. The O-Plan system can be configured both as a planning agent and as an external agent e.g. an execution agent. The function of an execution agent is to execute the plan in the real world and to report back to the planning agent on the plan's success or failure. At present the functionality of the execution agent is limited and the execution of the plan is simulated in a world process (see Section 3.6 for further details).

In order to simplify the O-Plan demonstration environment the external Execution Agent (as released in O-Plan Version 2.1 and 2.2) has been replaced by single pseudo-process running within the same image as the planner. This simplified execution agent has the same capabilities, both processing and communication as the external agent available in 2.1 and 2.2 and a demonstration of the simplified execution agent is provided in the file `README-pacifica.tf` in the default Task Formalism directory.

A separate O-Plan external agent can be spawned dynamically from the O-Plan planner details of which are specified at the start of this section. However, this external agent is only a shell and does not have any "capabilities" or Knowledge Sources defined. The capability to invoke external agents is provided to allow for user experimentation with external agents and does not form part of the O-Plan demonstration environment.

3.6 World Simulator

The world process allows the user access to a simulated world in which events take place at defined times. At present the world process is demonstrated as part of the execution agent and cannot be run separately. However, a separate world process has been developed and will be made avialable as a separate image with O-Plan release 2.3. The world process which comes as part of this release is automatically loaded when the execution agent is invocated from the planner's TA menu. The windows of the world process and their function are as follows:

- **World Clock** This window shows the time in the simulated world and is represented as `days hours:minutes:seconds`. The clock is automatically set to zero when the world process is loaded and operates with simulated time passing at the same rate as real time, i.e. a 1 to 1 correspondence.

- **World History** This window describes the events which have *passed* and the times at which they occurred. For example:

```
1:40 node-5-4    : End (unload gt2 delta).
1:41 node-7      : Begin (fly_passengers delta honolulu).
1:51 node-7      : End (fly_passengers delta honolulu).
1:51 node-8      : Begin (fly_transport delta honolulu).
2:00 node-8      : End (fly_transport delta honolulu).
```

- **World Agenda** This window shows the pending events which are due to occur in the near future. In the following example, two events are due to occur at time 0 00:00:30.

```
#  2  0:30 -> node-6-1    : End (drive gt1 calypso).
#  3  0:30 -> node-4-1    : End (drive gt2 abyss).
```

  If the user wishes to remove or reshedule an event they should read the following item on the World Interaction window.

- **World Interaction** This window allows the user to:
  - specify the outcome of an event in the simulated world e.g. an action starting or finishing, vehicle breaking down, etc
  - alter the time and sequence in which events execute in the simulated world.
  - view the constraints on events, e.g. the conditions on an action end, the effects of an action or world event, etc.
  - view the state of the simulated world.

The user will be prompted with a menu of these options each time an events occurs in the simulated world. The event will be descibed in the banner heading of the menu, e.g. `node-3-1 begin:{evacuate Barnacle 20}`. The menu entries are as follows:

- **Describe**:
  Describes the event in the World Interaction Window and provides details of the event, its expected execution time, due time, any conditions it requires to be satisfied, any effects it asserts.
- **Suceed**:
  Informs the simulated world that the event executed successfully.
- **Fail**:
  Informs the simulated world that the event executed unsuccessfully and that any effects it should have asserted have not taken place.
- **Partial Fail**:
  Informs the simulated world that the event was partially executed successfully and the user is required to identify which effects were successfully asserted and which were not. This is done through the World Interaction Window
- **Event First**:
  Schedule a new event before this event in the plan. The user is required to describe the event and its effects through the World Interaction Window.
- **Event After**:
  Schedule a new event after this event in the plan. The user is required to describe the event and its effects through the World Interaction Window.
- **Show World State**:
  Describes the state of the simulated world in the Worls Interaction Window.
- **Configure**:
  Changes the configuration of the simulated world
- **Set mode to: auto**:
  Sets the interaction mode to automatic so that all events are assumed to execute successfully. When choosing this mode the user must also choose the **Suceed** option in order to execute the menu.
- **Break in**:
  Allows the user to break into the code of the simulator process.

3.7 Compiling O-Plan Domains

The input language used by the O-Plan system is Task Formalism (TF) which is used to describe the actions, their decomposition and constraints, tasks, resources in the domain, etc. In order to ensure that the *syntactic* description of a domain (in TF format) is correct a command has been provided. (This assumes the user has defined an alias `oplan` to invoke the O-Plan system. If an alternative method has been defined the command should be altered accordingly)

```
oplan -tfc
```

The command will provide a trace of the compilation process and will indicate via appropriate messages where syntax errors have occurred in the encoding of a domain. A full description of the TF language can be found in the O-Plan TF Manual.

3.8 O-Plan Sanity Checker

The O-Plan sanity checker can be run from the window in which the O-Plan system was invoked. After start up the window remains attached to the O-Plan system and is capable of accepting Lisp expressions.

The sanity checker should be run after a plan has been produced and its function is to check that certain aspects of the plan are syntactically correct. For example, all conditions have a contributor, all actions which need expanding have been, etc. Any error reports will appear in the Lisp interaction window. The number of errors will be reported to the TA window, and the user will need to type return in the TA window to continue. To run the sanity checker type `(sanity-check)` in the window from which O-Plan was invoked.

4 Plan and World Viewers

4.1 Invoking and Running the O-Plan System with the AutoCAD Interface

With the environment variables set appropriately (see subsection 2.2 for details) the planner can be initiated via either of the methods described in subsection 2.3. The O-Plan system uses its own PlanWorld (PW) Adaptor program to interface to external software in order to display plan and world information. It is envisaged that the PW adaptor will interface to software such as AutoCAD, automatic mapping tools, overlays, etc. At present only AutoCAD output is supported and the user is required to invoke AutoCAD on behalf of the planner. In order to make this easier two scripts have been provided in $OPLANDIR`/bin` each of which invokes a copy of the AutoCAD system. The scripts and their requirements are as follows:

1. `oplan-acad-plan-view`
   The `oplan-acad-plan-view` will generate a graphics window in which the state of the current plan can be displayed graphically. This can then be edited by the user via the menu options.
2. `oplan-acad-world-view <drawing name>`
   The `oplan-acad-world-view` will generate a graphics window in which the state of the world can be displayed graphically. The script requires an argument <drawing name> which specifies the name of file containing the AutoCAD drawing information.

In the case of both scripts they require that the user specify the environment variable OPLANACADDIR which is the directory containing the AutoCAD specific drawing files. The AutoCAD package will read and write from the directory specified in the environment variable OPLANTMPDIR. It is recommended that the user invoke these scripts before invoking the planner, making sure that temporary directory specified in $OPLANTMPDIR is the same in **all** scripts.

At present `oplan-acad-world-view` support is limited to two domains: space station construction and Pacifica. However, the `oplan-acad-plan-view` can be used in all demonstration domains.

Each of the scripts will initiate *two* AutoCAD windows. When either script is executed a window will appear entitled `Notice` which will prompt the user with the following message

```
Cannot write to that file and changes cannot be saved to it,
                Do you want to continue?
```

The user should click on the `yes` button as this message is merely a warning[1]. In the case of the World View the window will be divided into three panes and a number of menu options will

[1] Write permission on certain AutoCAD files has been removed to stop accidental overwritting by the user.

appear along the top and left side of the window. In the case of the Plan View a single pane will be displayed and a number of menu options will appear along the top

Each of the separate components of the planner has its own window which allows the user to follow the plan as it is being generated. The major units of the O-Plan systems are described earlier in this section.

The Space Station construction application and the Pacifica NEO domain have been included in the set of demonstration of the O-Plan system in order to demonstrate this interface.

The user is now advised to read Section 10 of this document `Example Demonstration` which describes the steps required to run a particular demonstration (with specific emphasis on steps **7** and **9** which deal with plan state and world state browsing respectively) and to then refer to the relevant sections of the the Demonstration Guide which provides specific information concerning the Space Station and Pacifica applications.

4.2 Running O-Plan with a PostScript Viewer

In addition to the AutoCAD Viewers O-Plan also provides the ability to view plans via a PostScript viewer. One of the plan-view options (option 5) in the Task Assignment menu is `Print graph`. This causes a PostScript picture of the current plan nodes and their order relationships to be printed on a PostScript printer. It is assumed that the Unix command `lpr` refers to such a printer by default. (That is, `lpr` is called without a `-P` argument to select a non-default printer.)

However, a number of PostScript viewers are available, and they allow a user to display PostScript output on their screen rather than waiting for a hardcopy. It is possible to inform O-Plan of such a viewer by defining the `:ps-viewer` parameter in an `oplan-init` file. For instance, to use GhostView, a user would place the following line in their `oplan-init` file:

```
(set-parameter :ps-viewer "ghostview -landscape")
```

When this parameter has a value (the default is as above), a “Viewer one page” option will appear in the “Printer format” menu (the menu that appears when a user selects `Print graph` as the plan display mode). Selecting this option will cause the viewer to be run. The viewer is run in the background, and O-Plan can continue independently. However, O-Plan may not exit completely until the viewer has exited as well.

If you have difficulty getting a viewer to work, it may help to know that the viewer and `lpr` are invoked using the Unix command

```
(viewer tmpfile ; /bin/rm tmpfile)&
```

where viewer is, e.g., `lpr` or `ghostview -landscape` and `tmpfile` is the name of a temporary file generated by O-Plan.

5 User Interaction with the Planner

The O-Plan planner allows the user to intervene at any time during the planning process to browse on the plan and world state and to set certain modes which allow the user to take choices on behalf of the planner. To intervene in the planning process the user needs to click in the `Intervene as User` button on the `O-Plan Control Panel`. As a result a menu will appear with the following options:

```
Plan View
World View
Set Modes
Break in
Poison
Quit
```

Each of the options will now be described in further detail.

Plan View : This requests a view of the Plan which can be generated in several different formats which are as follows:

- Text to screen: the plan will appear in the PlanWorld Viewer window.
- Text to file: the plan will be sent to a user named file.
- Print Graph: the plan can be sent to an appropriate print device.
- AutoCAD: the plan will be represented as a series of AutoCAD commands and placed in the file `tfin.lsp`.
- KRSL: the plan will be represented as a series of KRSL statements and placed in a user named file.

World View :This requests a view of the World at a particular point in the plan and can be generated in several different formats which are as follows:

- Text to screen: the description will appear in the PlanWorld Viewer window.
- Text to file: the description will be sent to a user named file.
- AutoCAD: the world will be described as a series of AutoCAD commands and placed in the file `tfin.lsp`.

Set Modes : This allows the user to change the mode of *three* type of choice in the planner. The modes can either be set to `ask` or `auto` and can be changed by clicking on the option with the mouse. The modes are as follows:

Set bind mode to: deals with the binding of specific values to variables in the plan state.

Set schema selection mode to: deals with the choice of schema for an expansion of a node.

Set poison handler mode to: deals with the choice of an alternative plan state should the current one prove fruitless.

When the planner encounters a choice in a mode set to `ask` it will prompt the user with an appropriate menu from which the user must choose. The user is free to change mode as often as desired during the planning process.

Break in : This requests the planner to allow the user access to the O-Plan system. When this option is chosen a window will appear (`Planner User (KS-USER)` as follows:

```
>>Break: User Intervention

KS-USER-BREAK:
:C  0: Return from Break
    1: Return to round-robin scheduler

->
```

The user is allowed to type any Lisp expression which can be evaluated. The user must remember to return from the break by chosing the appropriate option from the menu[2].

Poison : This requests the planner to poison the current plan state and to provide the user with a list of alternative plan states from which the user must choose one. The planner will then continue the planning process from this new plan state.

[2]The option and number will vary according to the outcome of expressions typed by the user

6 O-Plan Processes and Internal Structure

Most of the time, users do not have to know anything about how O-Plan works internally. However, a reasonably accurate model of how the system works may make it easier to understand its behaviour, especially when something goes wrong. Since the processing model used in version 2.x may not be familiar to all users, this section provides a brief description. It also serves to explain the contents of the "run-lights" window.

An O-Plan agent, such as the planning or execution agents, contains a number of components (Database Manager, Agenda Manager, etc) that run in a semi-independent fashion and accomplish certain tasks by sending each other messages.

O-Plan version 2.x runs as a single invocation of Lisp as a single Unix process. Version 1 ran as several separate invocations of Lisp which communicated via sockets. Consequently, version 2.x is sometimes referred to as the "single-process" version. This has lower communication overheads and is easier to manage, but the appearance of independent operation is sometimes less complete.

In the single-process version of O-Plan, each component is implemented as a pseudo process or p-process. They are pseudo because, unlike real processes, they don't represent an independent thread of control. They are more like event handlers. An agent's main loop calls each p-process in turn so long as there is a message for it to handle. The p-process usually handles one message and then returns to the main loop.

When O-Plan is started, a small one-line window will appear before all of the other windows. This window is called the `Running Processes` window. It displays the names of the parts of O-Plan that are currently running or waiting for a running component to return a value. The window has the title "O-Plan Running Processes" and is usually placed just below the Task Assignment window.

Ordinarily, only one p-process is active at once and they run in turns in a round-robin fashion. However, in some cases (such as DM requests), one p-process calls another as a subroutine. In this case the calling p-process will be shown to the left of the one it calls. For instance, if the DM is running in the ordinary way, a user will see:

```
+----------------------+
|:DM                   |
+----------------------+
```

But if KP-1 calls the AM and the AM calls the DM before returning, a user will see:

```
+----------------------+
|:KP-1 :AM :DM         |
+----------------------+
```

In this case, only the scdm is actually running. The AM is waiting for the DM to return some result, and KP-1 is waiting for the AM.

Certain things cannot happen while the system is waiting for a p-process to return. In particular, no other p-processes can run. So if a button in pressed in the control panel, or an expression typed in the Lisp interaction window, or a "Plan View" or "World View" selected in the Task Assigner window, the requested effects may not happen immediately.

Whenever O-Plan does not seem to be responding as expected when something has been typed or selected from a menu entry, user should look at the "running processes" to see what part of O-Plan is currently running. It is possible that the user may have forgotten to answer a question or to select a menu entry, so that O-Plan is waiting for the user to do that before going on.

7 Typing Lisp Commands

When starting O-Plan, the window in which the `oplan` command was typed does not return to the shell prompt. Instead it functions as a read-eval-print loop in which a user can type Lisp commands. It is also used for some low-level messages from O-Plan. This window is called the "Lisp interaction window". Most of the time, a user can ignore it, but may have to use it from time to time, e.g. to turn on some of the debugging tools. Any Lisp expression can be typed as a command.

It's possible to type commands because there's a (pseudo-)process that waits for input from this window and evaluates it when it comes. When a command is typed, there may therefore be a delay, until the next time this pseudo-process gets to run, before the command is evaluated.

Note that if the input typed into the interaction window doesn't amount to a complete Lisp expression, the whole system will eventually stop and wait for the expression to be completed. In particular, this will happen if you `<return>` is typed in the interaction window. A user can see this has happened by looking at the one-line "Running processes" window. If nothing seems to be happening, but the :LISP-LISTENER process is running, it may be because the user has typed some incomplete input.

The listening p-process prints `form>` as a prompt, but the output that goes to the interaction window sometimes causes the prompt to go off the top of the screen.

If a Lisp error occurs during the evaluation of a command that was typed into this window, it is possible to get out of the error *break loop*, and hence allow O-Plan to continue running, by using a *restart* that has been set up for this purpose. Look for the option that says `Return to O-Plan`. A user can usually select this option by typing `:a` or `1`. An example will make this a bit clearer:

```
form> (not-a-function)
>>Error: The function NOT-A-FUNCTION is undefined

SYMBOL-FUNCTION:
   Required arg 0 (S): NOT-A-FUNCTION
:C  0: Try evaluating #'NOT-A-FUNCTION again
:A  1: Return to O-Plan

-> :a
Return to O-Plan
NIL
form>
```

8 Dealing with Lisp Errors

By "a Lisp error" we mean that O-Plan or the Lisp system invokes the Lisp error signalling mechanism rather than, say, just printing a message. Unless the error is caused by an erroneous command typed into the Lisp interaction window, it is usually due to a bug in O-Plan and should be reported.

Your options for dealing with the Lisp error depend on what O-Plan was doing when it occurred.

When a Lisp error occurs in the TF compiler, a menu will appear to make sure you notice the error. There will be only one option, "ok". Select "ok" to continue. The TF compiler will exit and report that the file contained an error. That is, a Lisp error is treated as if it were a single syntax error that applied to the whole TF file.

When the error occurs in some other part of the planner, there is typically no neat way to recover from it, and a more complex menu appears. It will look something like this:

```
Lisp error in process <process-name>
------------------------------------
Allow the error
Return to scheduler
Force global reset
```

`<Process-name>` will be replaced by the name of the process that was running when the error occurred: :AM, :DM, :KP-1, etc. You should look in the window associated with that process to see the error message. The meanings of the options are as follows:

Description Allow Lisp to handle the error. This will usually result in a "break loop" as described below. Select this option if you plan to report the bug, because the bug report should include a "backtrace" (also described below).

Return to scheduler Exit the process in which the error occurred so that the system can continue running. In most cases, this will not work very well, because the process in which the error occurred needed to finish some task that was interrupted by the error. This sort of recovery can also be attempted from a break loop, as described below (again).

Force global reset Attempt to reinitialise O-Plan. This will abandon the attempt to find a plan (or whatever O-Plan was doing) and start over as if you had selected the "Initialise planner" option in the TA. It is equivalent to evaluating the Lisp expression (force-reset).

When the Lisp system handles an error, Lisp usually enters a "break loop" in one of the O-Plan windows. Ordinarily, you will see a menu first, as described above. In any case, you can tell that Lisp has entered a break by the following signs:

1. O-Plan will stop producing any output.

2. The one-line ”running processes” window will stop changing.
3. There will be an error message in a window associated with the rightmost name in the ”running processes” window.
4. The window will be displaying the ”¿” prompt.

The first two can also happen when O-Plan is merely taking a long time to do something; the others cannot.

In a break loop, the normal ways of exiting O-Plan by using the control panel or the TA menu will not work. However, you can type Lisp debugger commands and Lisp expressions in the the window, after the ”>” prompt. Typing

```
(exit-oplan)
```

should cause O-Plan to exit.

```
(force-reset)
```

attempts to reset the system by wiping out all pending messages, sending itself an :INIT message, and returning to the process scheduler.

If you're planing to report a bug, you should get a ”backtrace” before exiting. It shows the nested Lisp procedure calls that led up to the error. Get a backtrace by typing ”:b” after the ”>” prompt. Grab the resulting output and the error message with the mouse and put them both in the bug report.

In some cases you can (and may want to) get O-Plan to continue after an error. This is most likely to be so when the error is in a relatively peripheral part of the planner, such as the PW viewer, but it can happen in other cases too.

In some cases, the Lisp system will offer you a reasonable way to continue from the error, so always look at the options that appear after the error message.

Another form of recovery that sometimes makes sense is to get O-Plan to abandon its current activity and allow other parts of the system to run. You can accomplish this by selecting ”Return to round-robin scheduler” from the restart options presented after the Lisp error message. (This is equivalent to choosing ”Return to scheduler” from the menu appears when the error is first noticed.)

This approach should be tried only when the error occurs during a relatively self-contained activity. An example might be a Lisp error when the PlanWorld Viewer tried to display a plan. By returning control the the scheduler, you could get the viewer to abandon this attempt This would work fairly well because nothing else in the system depends very strongly on the viewer succeeding. On the other hand, it would not work very well if, for instance, the DM ran into an error somewhere deep in the TOME/GOST manager. In such cases, you might try typing the command (force-reset) instead.

9 Control Panel and Trace Information within O-Plan

The control panel allows the developer to set the levels of diagnostic information and monitors within the system. Diagnostic levels can be set in a range of values from `none` through to `all` where all messages are printed. In most cases setting the level to `emergency only` is usually sufficient. The control panel also allows the developer to place the system into single step mode in which the developer is prompted on each agenda cycle to choose the next agenda to be processed. A description of the single step mechanism is given in step 6, (Agenda Manager Processing Options) of Section 10 which describes how to run an O-Plan demonstration.

The running lights process shows the developer which component is currently running and in the case of a problem allows the developer to easily identify in which component a problem has occurred. The running lights window is controlled from a variable which allows the output to be turned on and off. This can be achieved as follows:

To turn the running lights on (the default), have Lisp evaluate:

```
(run-lights-on)
```

To turn the running lights off, have Lisp evaluate:

```
(run-lights-off)
```

O-Plan can provide a trace of what it is doing for use in debugging. The trace is written to a `whats-going-on` file in the users `$OPLANTMPDIR` directory. The file name will be constructed be concatenating `whats-going-on` with the users login name, e.g. `whats-going-on.bd`. This facility is primarily intended for developers and is normally turned off.

To turn on the whats-going-on file, have Lisp evaluate:

```
(setq *print-whats-going-on-p* t)
```

To turn it off, have Lisp evaluate:

```
(setq *print-whats-going-on-p* nil)
```

The `whats-going-on` file is off by default.

One way to get Lisp to evaluate these expressions is to type them in the Lisp interaction window; another is to put them in the user's "oplan-init" file. Note, however, that the oplan-init file is examined only when O-Plan first starts up.

10 Example Demonstration

The demonstrations allow the user to gain a gentle introduction to the O-Plan systems through a series of simple worked examples. This section describes the basic steps involved in running any of the demonstrations to be found in the Demonstration Guide.

The O-Plan system is menu driven (via the Task Assignment Window) which allows the user to issue high level commands to the system. The Task Assignment Window contains the following menu items:

```
Status: Version <no> <date> <status of the planner> either:
        uninitialised, planner initialised, planning,
        replanning, executing, planner finished
Domain: one of none, <file> or <file> + <file>
Task  : one of none or <task name>
Authority: plan(all=inf), execute(all=no)

* 1) Initialise Planner
  2) Input Task
  3) Set Task
  4) Add to Task
  5) Plan View
  6) World View
  7) Replan
  8) Execute Plan
* 9) Quit

Please choose a number:-
```

In the Task Assignment Window items marked with a * are the only ones available from the planner at any point. A demonstration can be broken down into ten distinct stages:

1. **Initialising the System**
   Initialises the system ready for a new TF domain description to be accepted.

   In order to carry out this step the following should be used:

   > Move the mouse to the Task Assignment Window and enable the window for keyboard input. Enter 1, (followed by carriage return) and this will initialise the planner ready for the demonstration. Wait until the planner initialised message appears in the `Status` banner of the Task Assignment Window.

2. **Specifying the Domain**
   This part of the demonstration informs the planner of which domain is to be used, e.g. block stacking, house building, etc.

   In order to specify this information the following step should be used:

Enter 2 from the Task Assignment Window. This will cause a further menu to appear which describes the problem domains available in the $OPLANDIR/TF directory. Use the mouse pointer to either:

(a) choose the file which contains the domain description you wish to use.

(b) choose the `Change directory` option. The user will be presented with a menu listing the previous TF directories visited (where appropriate) and an option to type in a new directory name. The user must either chose a directory from the menu or type the name of a new directory after the prompt in the Task Assignment window. The TF files in the specified directory will now be displayed.

(c) choose the `Enter filename` option and type the name of the file after the prompt in the Task Assignment window.

Wait until the Task Assignment Window `Status` banner displays the message that is has successfully set up for the chosen domain.

3. **Specifying the Task**

   This part of the demonstration informs the planner of the particular task to be carried out within the domain. For example, the order of the completed stack of blocks, the type of house, etc.

   In order to specify this information the following step should be used:

   Enter 3 from the Task Assignment Window. This will cause a new menu to appear which describes the particular tasks in the domain you may choose from (the list of task schemas in the TF file). Use the mouse pointer to choose the task you wish the planner to solve.

4. **Adding a New Action to the Plan**

   This part of the demonstration informs the planner that the user wishes to add further requirements to the plan. These can be added once a plan has been generated and during the subsequent execution of the plan.

   In order to specify this information the following step should be used:

   Enter 4 from the Task Assignment Window. This will cause a new menu to appear which has a single option to "Add an action". Use the mouse pointer to choose the option. The Task Assignment Window will now be cleared to allow the input of the new action e.g. (evacuate Calypso 20), (install kitchen_equipment), etc.

5. **Running the Planner on the Task**

   This part of the demonstration usually involves little interaction from the user apart from watching the messages from various windows. However, the user is free to intervene as described in Section 5 should they wish to though it is not expected as part of a demonstration. As the planning processing moves forward text will appear in each of the interface manager, agenda manager, database manager and ks-platform windows respectively. The text gives a trace of the planner as it solves the problem. It is not necessary as part of the demonstration to understand the information as it is being displayed.

6. **Messages and Diagnostics**
   The level of messages will be the default (normally short one line messages). Greater or lesser levels of messages can be selected for O-Plan using the Interface Manager's Control Panel.

7. **Agenda Manager Processing Options**
   The user may also control the order in which outstanding tasks (agenda entries) are processed by the the O-Plan system. This is achieved by placing the O-Plan system into single step mode. This is achieved by chosing the `Single Step` button from the Control Panel. This will result in the following menu being displayed in the Agenda Manager window.

   Agenda Manager Interrogator:
   Stopped in cycle 61

   (a) `Display the Agenda Table`: Display current list of outstanding tasks
   (b) `Display the Untriggered Agenda Table`: Display the current list of untriggered agenda entries.
   (c) `Display the Alternatives Agenda Table`: Display the list of alternative plan state to explore
   (d) `Process the top agenda entry`: Send the top entry for processing
   (e) `Process any agenda entry`: Send a specified entry for processing
   (f) `Schedule KS-USER`: This will schedule the KS-USER knowledge source to run on the next cycle of the planner. This will allow the user to choose options from the menu described in Section 5.
   (g) `Break in`: This allows the user to interrogate the contents of particular datastructures within the LISP code. You may resume O-Plan by typing `0`
   (h) `Quit single step mode`

8. **Browsing on the Plan**
   Once the plan has been generated (or while it is being generated) by the O-Plan system it can be viewed in a number of ways. In order to specify the display mode Enter 5 from the Task Assignment Window. This will invoke[3] the PlanWorld window which will display information from the planner and allow the user to specify file names where necessary.

   The user will now be presented with a menu showing the different viewing formats available. These are as follows:

   (a) `Text to screen`: Send each node of the plan to the PlanWorld window in the following format:

```
Node Number: node-3-10-2
   Begin_end Predecessors : (end_of node-3-9-1 begin_of node-3-10)
   Begin_end Successors   : (end_of node-3-10-2)
   End_end Predecessors   : (begin_of node-3-10-2)
```

[3] Assuming the window is not already present

```
End_end Successors     : (begin_of node-3-9-8 begin_of node-3-10-3)
Earliest_start_time    : 7~00:00:00
Latest_start_time      : infinity
Earliest_finish_time   : 9~00:00:00
Latest_finish_time     : infinity
Minimum_duration       : infinity
Maximum_duration       : infinity
Node_type              : action
Node_pattern           : "(pour basement floor)"
```

In most cases the size of the plan will cause information to be lost off the top of the window and the user is advised to place the window in `scroll mode` buy pressing the middle mouse button and the control key simultaneously and chosing the `Enable Scrollbar` option from the menu which appears.

(b) `Text to File`: Send the plan to the file name specified by the user and place it in the directory specified in the environment variable $OPLANTMPDIR. The user will be prompted for the file name via the PlanWorld window. The format of each node in the file is as follows:

```
node
  <node number>
  (<begin end predecessors>)
  (<begin end successors>)
  (<end end predecessors>)
  (<end end successors>)
  (<earliest-start> <latest-start> <earliest-finish> <latest-finish>
        <min-duration> <max-duration>)
  <node type>
  <node pattern>
end_node
```

An example of a node is as follows:

```
node
  node-3-4-1-2
  (end_of node-3-4-1-1)
  (end_of node-3-4-1-2 begin_of node-3-4-1-2-1)
  (begin_of node-3-4-1-2 end_of node-3-4-1-2-1)
  (begin_of node-3-4-1-3)
  (0 1728000 172800 1900800 172800 1900800)
  action
  "(ground_move 107th_acr_%byair ft_meade_port andrews_afb_naf_afb)"
end_node
```

(c) `PostScript Graph`: Send the plan to a Postscript printer for printing. The user will be presented with a second menu from which they must chose:

   i. `Single page`: Send a copy of the plan (scaled to fit on a single sheet of A4) directly to a specified hardcopy device.
   ii. `Multiple Pages`: Send a copy of the plan (split across multiple sheets of A4) directly to a specified hardcopy device.

   Once the user has chosen their required output format a further menu will appear indicating the number of levels. The user can either chose to display the plan to a required level or display all nodes of the plan by chosing the `all` option.

(d) `KRSL`: Convert the plan to a series of KRSL statements and send them to the file name specified by the user. The files will be placed in the directory specified in the environment variable $OPLANTMPDIR. The user will be prompted for the file name via the PlanWorld window. The plan will be converted to a series of KRSL statements which describe:

   i. The nodes at each level of the plan, together with their ordering links
   ii. A detailed description of each node in the plan. The description includes information on condition satisfaction, effects, time, parameters, etc.

(e) `AutoCAD`: Convert the plan to a series of AutoCAD commands and send the plan to the file `tfin.lisp`. The file will be placed in the directory specified in the environment variable $OPLANTMPDIR. (In order to view the contents of the file the user will need to invoke the AutoCAD Viewer via the script `oplan-acad-plan-view` which is stored in the directory $OPLANDIR/bin.)

   A menu will appear indicating the number of levels in the plan which can be displayed. The user can either chose to display the plan to a required level or display all nodes of the plan by chosing the `all` option. When the `tfin.lsp` file is complete the viewer will indicate this in two ways.

   i. the message `The Autocad file is ready for viewing` will appear in the PlanWorld window
   ii. the console will bleep twice.

   After both of these steps the file may be loaded into the AutoCAD system in the usual way.

9. **Browsing on Plan State Information**
   Further information concerning the effects asserted in the plan and the goals satisfied during the generation of the plan, etc can be obtained through the Interface Manager's Control Panel. Two separate browsing menus are provided:

   Interrogate the Contents of the DM To view this information chose the **DM Developers Menu** option from the Control Panel window. This will result in a new menu appearing in the Database Manager window. The options in the menu are as follows:

   (a) `Display nodes`: Display the nodes of the plan together with their predecessors and successors.
   (b) `Display TOME`: Display the effects asserted in the plan.

(c) `Display GOST`: Display the protected ranges over which preconditions must be preserved.

(d) `Display PSV`: Display the plan state variables which are in the plan.

(e) `Break in`: You may resume O-Plan by typing `0`.

(f) Quit interrogation. Quit this menu

10. **Browsing on the World State**

Information concerning the state of the world at a particular node in the plan graph can be obtained and viewed on the screen. In order to view this information Enter 6 from the Task Assignment Window. This will invoke[4] the PlanWorld window which will display information from the planner and allow the user to specify file names where necessary.

The user will be prompted for a node number by the phrase `View at end_of node-` appearing in the Task Assignment Window. The user may then select the plan node of interest. Following the choice of `node_end` the user will be presented with a menu showing the different viewing formats available. These are as follows:

(a) **Text to Screen**: The information concerning the specified node of the plan will appear in the `PlanWorld` window as a set of triples of the form <`pattern`> `=` <`value`>. For example:

```
{on a b} = true
{clear a} = true
{clear b} = false
```

(b) **Text to File**: Send the world view to the file name specified by the user and place it in the directory specified in the environment variable $OPLANTMPDIR. The user will be prompted for the file name via the PlanWorld window.

(c) **AutoCAD World Viewer**: The default AutoCAD viewer filters the `patterns` by comparing them with those found in the variable `ok-preds` and only processes those which are members of this list. For example, the value of `ok-preds` for the space state assembly domain is as follows:

```
(setq ok-preds '(joint_at truss_at tube_at module_at radiator_at
                 solar_panel_at antenna_at))
```

The AutoCAD commands are written to the file `tfin.lsp` and is placed in the directory specified in the environment variable $OPLANTMPDIR. (In order to view the contents of the file the user will need to invoke the AutoCAD Viewer via the script `oplan-acad-world-view` which is stored in the directory $OPLANDIR/bin.) When the `tfin.lsp` file is complete the viewer will indicate this in two ways.

i. the message `Autocad file is ready for viewing` will appear in the PlanWorld window.

ii. the console will bleep twice.

[4] Assuming the window is not already present

While the information is being written to the file `tfin.lsp` the Viewer checks that at least **one** pattern passes the filter test. If no pattern passes the test the following message will be displayed:

```
AutoCAD file is empty
```

The usual cause of this message is an incorrect set of patterns in the `ok-preds` structure. The user may define their own processing function by replacing the function `output-world` with their own and by defining a valid set of filter patterns if necessary.

11. **Executing the Plan** To execute the plan choose the Execute Plan option from the Task Assignment Window and the plan will be sent to the execution agent. For demonstration purposes a simple execution agent has been provided but can only be run with a suitable configuration file (See Section 3). The simplified execution agent referred to as Micro-Exec is invoked at the same time as the Task Assignment Agent and Planner Agent and can be used on any of the O-Plan Demonstration domains.

11 Example AutoCAD PlanWorld Viewer Demonstration

The directory $OPLANACADDIR contains the demonstration AutoCAD files for use with the `oplan-acad-plan-view` and `oplan-acad-world-view` scripts. The files are as follows:

1. `acad-plan.config`
   A file used with xrdb to set up suitable windows sizes for AutoCAD - suitable for the Plan Viewer.
2. `acad-plan.config`
   A file used with xrdb to set up suitable windows sizes for AutoCAD - suitable for the World Viewer.
3. `plan.*`
   Files needed for the Plan Viewer e.g. `.dwg`, `.lsp` and `.mnu`. Others will be created in this directory when AutoCAD runs e.g. `.mnx` and temporarily `.dwk` and `.mnk`.
4. `plan-parts.dwg`
   Used to build the root drawing for plan.dwg (visible components used to make drawing blocks in AutoCAD).
5. `space-platform.*`
   Files needed for the Plan Viewer in the space station domain, e.g. `.dwg`, `.lsp`, `.mnu`, `.scr` and `.sld`. Others will be created temporarily in this directory when AutoCAD runs e.g. `.dwk`.
6. `map-view.*`
   Files needed for the Plan Viewer in the Pacifica domain, e.g. `.dwg`, `.lsp`, `.mnu`, `.scr` and `.sld`. Others will be created temporarily in this directory when AutoCAD runs e.g. `.dwk`.

The demonstrations, files, etc are further defined in [1], [2]. The user is intended to make use of the PlanWorld Viewer using AutoCAD via the scripts `oplan-acad-plan-view` and `oplan-acad-world-view` respectively.

Before running the demonstration it is necessary to ensure that you have access to AutoCAD. If you can successfully execute the command "acad" then all is well. Otherwise, before running this demonstration you should ensure that you are set up to use AutoCAD. See your system administrator.

A screen image of the O-Plan system together with both Plan and World Viewers in given in Figure 2.

The following text explains how the AutoCAD PlanWorld Viewers can be used with O-Plan.

1. Prior to running the O-Plan system invoke the AutoCAD Plan and World Viewers by using the scripts `oplan-acad-plan-view` and `oplan-acad-world-view` respectively (See Section 4 of the User Guide).

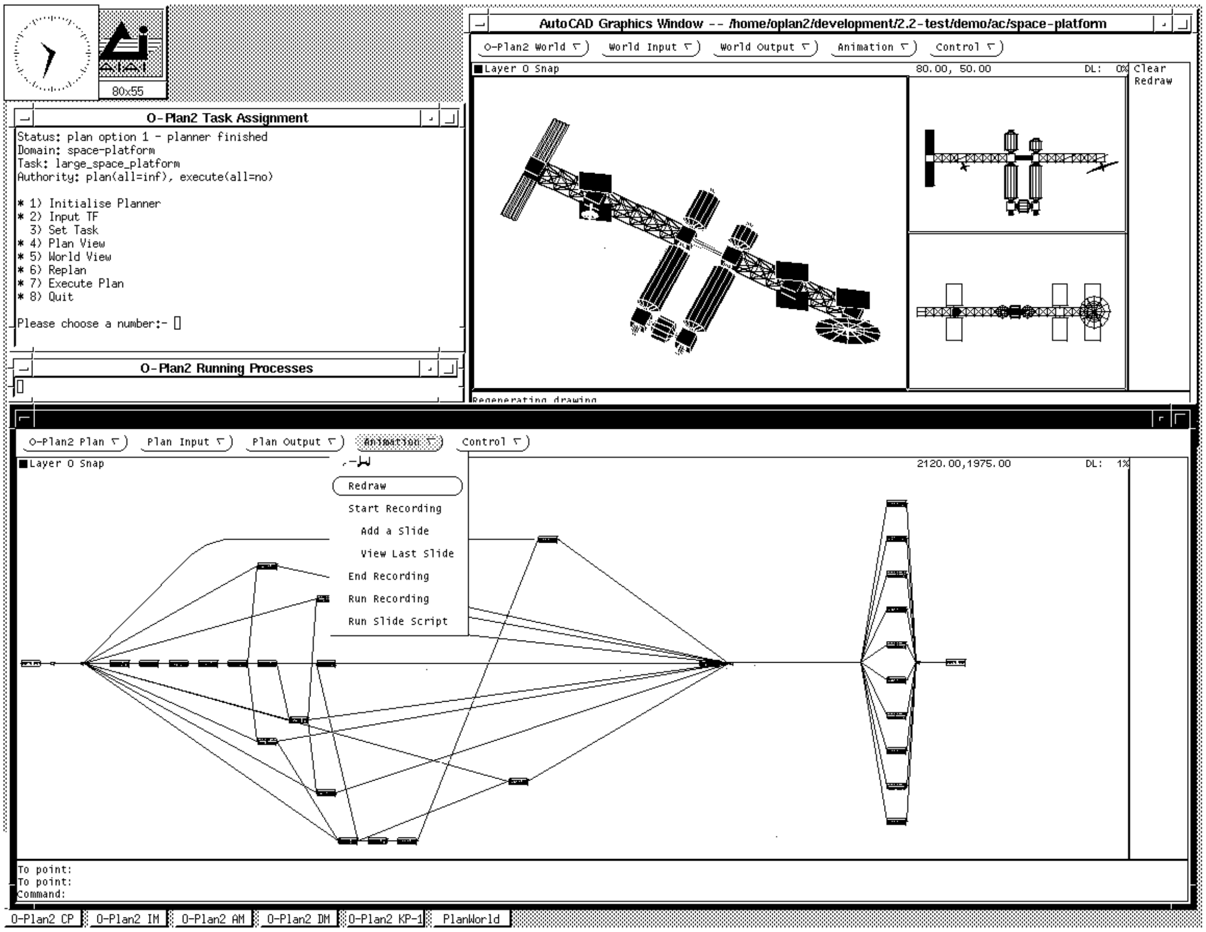

Figure 2: Example screen image from the O-Plan System with its AutoCAD Viewers

2. Enter O-Plan with the script `oplan` and select the required TF domain, either `space-platform` or `pacifica.*`.

3. The demonstration shows how a graphics interface can be used for input of information to the planner and for output from the planner for plan networks and domain specific simulation output. Using the TF Input pull down menu a schema can be drawn and an interface file to describe it passed to O-Plan. However, at present this "link" is not yet implemented. It is also envisaged that a domain specific "drawing package" could be used to construct some object and then a file created to describe this to the planner in TF schema form too. However, in the current demonstration, the TF input is not connected to O-Plan.

4. Choose one of the tasks available and O-Plan should start to create a plan for the platform chosen and terminate after some messages. The Plan and World Viewers i.e. the two AutoCAD graphics windows can be used during plan construction, but to ease explanation below, we will assume the plan has been completed.

5. Choose the Plan View Option in the Task Assignment window and then select the AutoCAD option from the menu that appears. A second AutoCAD information menu will appear indicating the number of levels[5] in the plan which can be displayed. The user can either chose to display the plan to a required level or display all nodes of the plan by chosing the `all` option. This will create a file with name `tfin.lsp` which will be placed in the directory specified by the environment variable $OPLANTMPDIR. This file is in a form suitable for use in the Load Plan option under the Plan Output pull down menu in the AutoCAD Graphics Window – plan. Use the right mouse button to pull down the Plan Output menu and hold it down to select Load Plan or release the right button to leave the pull down menu and use the left mouse button top select.

   Note that Initialisation is not needed before a Load Plan can be done. It is possible to zoom in and out, to clear the screen etc using the Control pull down menu.

   It is possible to pick an item in the plan and see its details printed. This can demonstrate the ability to pick an item to pass to the planner so that the "context" simulation at a node can be seen. It is not currently possible to pass this information directly to the planner, but this is anticipated in future. Always clear the plan window before loading another plan or you will overlay one on top of another.

6. Choose the World View Option in the Task Assignment window. A new display menu will appear and the user must choose the AutoCAD option. A prompt will now appear in the Task Assignment window and the user should now select a node of interest (node 2 is the last node in the plan and will show the World View at the end of the plan). This will again create a file with name `tfin.lsp`.

   This file is in a form suitable for use in the Load Plan option under the Plan Output pull down menu in the AutoCAD Graphics Window – space-platform. If you do this in the wrong window (eg the plan output window you will get an AutoCAD error, but it not be serious, just tidy up if necessary and redo the Load Plan in the right window.

7. The idea behind the demonstration, is that a chosen item would be done on a plan network in the plan output window. The plan item chosen would be passed to the planner where a context simulation would be carried out and drawn via the WorldView window. Any AutoCAD commands could then be used to tidy up, hide hidden lines, shade the drawing, change viewing position, etc. Such images can then be captured in a file - the AutoCAD terminology for these are PICS. The idea is that a sequence of these can then be replayed (maybe to show an execution sequence) in what is called the FLICKS. A demonstration of this is available in the Run Slide Show option on the Plan Output pull down menu.

   To stop the slide show type `ctrl/c` in the AutoCAD window (after selecting the window for input by clicking with the left mouse button if necessary).

8. It may be helpful to have a text window you can pull down to view the TF for the chosen domain. This can be done with for example

```
emacs $OPLANTFDIR/space-platform.tf
```

[5] These levels indicate the longest path to a node through the network.

9. Close down AutoCAD by typing `quit` (after selecting the window for input by clicking with the left mouse button if necessary) or use the Quit menu option under the `Control` Pull Down menu. Confirm that you do not want to change the saved drawing by typing "y".

10. The following files are created temporarily by the demonstration in the current working directory. They can be deleted after use

    - `tfin.lsp`
    - `tfout.txt`

    In addition the file `acad.lsp` is made as a copy of a lisp startup routine. This can be deleted unless it is needed for other AutoCAD applications in the user directory.

References

[1] Drabble, B. and Tate, A. *Using a CAD system as an interface to an AI Planner*, In the Proceedings of the European Space Agency Conference of Space Telerobotics, European Space Agency, Noordwijk, Holland, April, 1992.

[2] Tate, A. *Interfacing a CAD System to an AI Planner*, Available as AIAI Technical report AIAI-TR-76, January 1990.