

# I-X



## **I-X Configurer's Guide**

### **Methods for Building I-X Applications**

Jussi Stader

Contributions from Jessica Chen-Burger, Jeff Dalton, Stephen Potter, Austin Tate, Gerhard Wickler

Artificial Intelligence Applications Institute

School of Informatics, The University of Edinburgh

Appleton Tower, Crichton Street, Edinburgh EH8 9LE, UK

Web: <http://i-x.info>

E-mail: [query@i-x.info](mailto:query@i-x.info)

28 June 2007

## Table of Contents

1	Introduction .....	3
1.1	The I-Demo Suite of Demonstrators .....	3
2	Building Applications with I-X Technology .....	4
2.1	Specifying Application Domains for I-X .....	4
2.1.1	Gathering Domain Information .....	4
2.1.2	Building Domain Models .....	5
2.1.2.1	Building Organisational Models .....	5
2.1.2.2	Modelling Domain Objects .....	6
2.1.2.3	Building Process Models .....	8
2.1.2.4	Making Models Work Together .....	9
2.1.2.5	Some Constraints on Domain Models .....	10
2.2	Configuring I-X Applications .....	11
2.3	Initialising I-X Applications .....	12
2.4	Extending I-X Applications .....	12
3	A Basic Configuration Example .....	13
3.1	Files for Initialisation .....	13
3.1.1	Domain Models .....	13
3.1.2	Test Menus .....	13
3.1.3	Initial States and Plans .....	13
3.1.3.1	Building Initial States and Plans .....	13
3.2	Configuration Issues .....	14
3.2.1	Using a Configuration File .....	14
3.2.2	Using a Domain Model .....	14
3.2.3	Using an Initial Plan and State .....	14
3.2.4	Using a Test Menu .....	14
3.2.5	Running without a Name Server .....	14
4	Step-by-Step Configuration .....	14
4.1	I-Demo Cooperation .....	15
4.1.1	Conceptual Modelling .....	15
4.1.2	Building Files for Initialisation .....	15
4.1.2.1	Building Domain Models .....	15
4.1.2.2	Building Test Menus .....	15
4.1.2.3	Building Initial States and Plans .....	15
4.1.3	Configuring the Application .....	16
4.1.3.1	Adjusting Properties .....	16
4.1.3.2	Adjusting Windows Scripts .....	17
4.1.3.3	Adjusting Unix Scripts .....	17
4.2	Using the Map State Viewer Add-On .....	17
4.3	Running Agents on Different Machines .....	18
4.4	Running Across Firewalls .....	19
4.4.1	XML Across Firewalls .....	19
4.4.2	Jabber Across Firewalls .....	19
4.5	Building Your Own I-X Agent .....	19
5	Further Reading .....	19

## 1 Introduction

I-X is a rich systems integration architecture that is underpinned by a set of complementary conceptual modelling methods. It provides a suite of tools that are implemented in a flexible environment to help people work together on processes that may be distributed over different geographical locations. Among its many facilities, I-X supports the specification and implementation of best practices and standard operation procedures (SOP) through its process and workflow management facilities; communication between collaborating agents through its user and automated interfaces; keeping track of current status through its world-state management component and making plans dynamically through its automated planner while allowing the user to choose actions to perform through its option management. Through its web interfaces, I-X also supports real-time access and communication (via web services), thus making resources of I-X tools easier to share for a wider audience.

The purpose of this document is to help people create, use, configure, and extend I-X applications. Its remit is to be useful rather than comprehensive by providing a tutorial-style introduction. Throughout this document, examples and instructions are based on a set of demonstrators we use for illustration. The scenario of these demonstrators is described below in section 1.1.

In the following sections, we first give instructions for building I-X applications in general (section 2) in order to give an overview of how I-X technology is applied. Section 2 starts with instructions for populating I-X systems, i.e. what models to provide for which part of the tool and how to load them into I-X applications. We then move on to instructions for configuring I-X applications, i.e. understanding, adapting, and using process panels, and section 2 finishes on some instructions for extending existing I-X applications. Section 3 gives step-by-step instructions for creating a simple I-X application. After reading section 3, you should be able to generalize from it and build similar I-X applications. Section 4 gives step-by-step instructions for tailoring the basic application to achieve specific purposes. If at any point you feel that you need more detailed information than is provided in this document, turn to section 5 which provides pointers to in-depth I-X documentation and related papers. This complementary I-X documentation should support most real-world I-X applications development.

I-X systems can be used, configured, and developed under different platforms. To make this document more reader-friendly, we have assumed that you are working in a Windows environment, but this choice of platform is not preferred to any other. Where it is relevant we include brief instructions for running in a Unix/Linux environment.

For details on how to download and install I-X on your machine, see the I-X web pages at <http://i-x.info/> or at <http://www.aiai.ed.ac.uk/project/ix/>. There is a wealth of other I-X documentation providing background information and comprehensive descriptions of what is available (see below).

This document was written for I-X version 4.0 and revised for version 4.1. Although the instructions in this document will work for I-X version 4.2 and later versions, the new features in these later versions are not yet included. Instructions for running under Unix are basic at best and need to be revised. The step-by-step configuration instructions in section 3 and 4 do not include instructions on how to build domain models or how to build test-menu files. For this information, please consult other I-X documentation, e.g. I-X Domain Editor Guide.

### 1.1 The I-Demo Suite of Demonstrators

An I-X process panel may be seen as an independent agent who carries out tasks within an organisation while communicating and collaborating with other agents, therefore the example application is also provided in such a setting. This example application is a system that aims to support the moving of objects using different forms of transport. It is the operator agent's task to move the object; there may be a supervisor agent who can grant authority for the use of certain types of transport.

The I-Demo suite of demonstrators all uses this simple scenario. Each of the I-Demo demonstrators is documented with a demo script and explanations of what part of the I-X technology the demonstrator illustrates. There is a basic demonstrator, *idemo-basic*, which uses a stand-alone operator agent. All other I-Demo demonstrators are based on this basic demonstrator, each adding one specific aspect of I-X technology that they aim to illustrate. The next section describes how each of the demonstrators is built or configured, starting from the *idemo-basic* demonstrator. There is also a demonstrator, *I-Demo Features*, which combines the features of all other I-Demo demonstrators. This demonstrator is provided as a part of the I-X system release, but is not described in this document, as its features are already included in the

relevant individual demos. All of those I-Demo demonstrators can be found in the I-X applications area, e.g. the I-Demo Basic demonstrator is in <ix-base>/apps/ideo-basic/.

## 2 Building Applications with I-X Technology

This section describes the major steps that need to be taken when setting up an I-X application. This section is split into 4 parts:

1. Specifying application domains for I-X – describing the problem, gathering information, and building models;
2. Configuring I-X applications – building process panels and making them work in an application;
3. Initialising I-X applications – using models, states, and plans; and
4. Extending I-X applications – more extensive and challenging application development that involves programming.

Section 3 will give step-by-step instructions of how to perform different configuration tasks introduced in this section.

### 2.1 Specifying Application Domains for I-X

When building any serious application, it is important to first establish the requirements for the application with an informal problem description. Then follows a phase of domain information gathering, interleaved with knowledge modelling and capture activities.

#### 2.1.1 Gathering Domain Information

The kind of domain information that is required by the current implementation of I-X is mainly in the areas of processes and how they are performed, organisational structure in which agents operate, and domain objects that are manipulated by the processes. In more detail:

- Processes and how they can be broken down into sub-processes are the basis of the support that can be provided by I-X applications. Thus, most of the information gathering and modelling work will be done in this area. Details about processes are specified in “refinements”. Sub-processes within a refinement can be temporally ordered (e.g. step 1 has to be done before step 2). Refinements of processes also carry conditions and effects that can be used to query and update the world state. In addition, refinements can have issues associated with them, and variables can be used for storing information and sharing it between processes, etc. In summary, the following information is most relevant in the area of processes:
  - Sub-processes that may have ordering constraints between them;
  - Conditions that determine whether a process should be executed at certain point of time and effects that specify execution outcomes;
  - Issues that are relevant to or raised by processes; and
  - Variables that store information about objects that are manipulated by processes.
- Organisational structure with respect to participating agents covers the agents’ relationships with other agents (see below). It also indicates the interaction model when communicating with other agents. The I-Space tool is used to model other agent’s relationships with respect with self-agent’s role within an organisation. The following agent relationships are supported:
  - Peers – same level within the organisation as the agent itself; activities can be *passed* to such agents;
  - Subordinates – agents at a lower organisational level; activities can be *delegated* to them;
  - Superiors – agents at a higher organisational level; activities can be *escalated* to them;
  - Contacts – agents that may not be within the organisation - activities are *passed* to them;
  - Services – automatic services internal or external to the organisation; these services can be *invoked*;

- External Capabilities – explicit capabilities of any of the above expressed as comma-separated pairs (<agent1>:<verb1>,< agent1>:<verb2>,<agent2>:<verb1>, etc.)
- Domain Objects are manipulated as part of the application, but support that I-X provides for this is centred around the use of variables in process models and what can be managed using world state specifications, so only basic modelling is required here. World state specifications are statements about what is true in the world that the I-X application knows about. Each agent has its own world state, but it can pass (parts of) its state to other agents.

### 2.1.2 Building Domain Models

In this section we describe the models that are to be built and some strategies, techniques and steps for building them. This section does not aim to provide a guide for domain modelling in general. If you have no modelling experience you can expect to try out I-X technology, but developing real I-X applications will require at least some modelling expertise. There is a variety of domain modelling methods, some of which give an introduction to domain modelling. For readers interested in learning more about process modelling methods, the IDEF3 web site [1] provides relatively rich guidelines. For foundations in domain modelling methods, e.g. ontology building methods, see [2], [3] and [8]. To understand how Semantic Web based applications are related to processes and enactment, BPEL4WS [4] and OWL-S [5] are good references. For similar interests on Semantic Web based data representations, OWL [6] is worth a read.

What we describe in this section is an outline of how to build the models that are required for I-X applications. The different types of I-X models all relate to each other because they represent different aspects of the same domain (or application area) and they complement each other when they are used in the application. We first describe how the different models are built (organisational models, domain object models, and process models) and then give indications of how they relate to each other and how they may be used together. Finally, we give some details of modelling constraints that are imposed by the ways in which I-X and relevant add-ons use and manage the underlying models.

#### 2.1.2.1 Building Organisational Models

As a first step of building an organisational model, it is useful to sketch out what agents are in the organisation, how they relate to each other, how they communicate with each other, and what kinds of processes different agents are involved in. For each agent, take a note of:

- Organisational relationships between the agent and other agents (i.e. peers, superiors, subordinates, contacts),
- Areas of expertise (i.e. capabilities, processes it knows about or is responsible for)
- Interfaces (communication) with other agents
- Other relevant characteristics of the agents

The only information about organisational models that can directly be included in I-X applications is the organisational relationships. This information is made available through the agent's I-Space specifications. Adding relationships to other agents to the I-Space specifications of an agent will have the effect that those other agents will increase the number of options that are available for dealing with activities. For example, the agent will be able to "delegate" activities to subordinates with suitable capabilities, "pass" them to peers, etc. In addition to agent relationships, I-Space can also be used to specify agent capabilities. If capabilities are used coherently, agents with relevant capabilities will appear as options for dealing with activities. For example, consider the situation where towing is to be done (activity "tow-vehicle" is on the activity list). If there is an agent that has advertised a capability "tow-vehicle", this agent will appear on the list of possible actions for dealing with the activity. Note that if no capabilities are specified for an agent, the agent is assumed to have all capabilities (i.e. can perform any activity).

An example of an agent's organisational relationships in I-Space is:

- Superiors: Supervisor
- Peers: Operator2, Operator3
- Subordinates:
- External Capabilities: Operator2:move-vehicle,Operator2:load-vehicle,Operator3:source-objects

Currently, I-Space specifications are part of an agent's properties file (see section 2.2). It is important to decide which agent should have information about which other agents. A good rule of thumb is that if an

agent is to initiate communication with another agent (e.g. to delegate/pass and activity to that agent), the agent needs to know about that other agent. An agent does not need to know about other agents if it only needs to reply to those agents' requests.

Any information about agents that is not part of the I-Space specifications (e.g. the agent's location or contact information) can be included in the system as part of Modelling Domain Objects below.

At the moment, the onus is on the user to get the agent names right, to make capability specifications useful, and to tie in other "object information" about the agents. In future, the modelling of organisational information will be supported by I-DE and it will be tied into organisational ontologies that will also contain concepts of organisational roles.

### 2.1.2.2 Modelling Domain Objects

In the past, the modelling of domain objects and their lifecycles is not supported well in I-X. Since version 4.4, I-DE provides quite comprehensive support in this area and in future there will also be support for the use of ontologies that underpin the modelling of domain objects. However, although I-DE supports the modelling of domain object classes, the only way in which objects can currently be used in I-X process panels is through constraints on the world state. World state specifications are statements about what is true in the world that the I-X application knows about. Each agent has its own world state and it can pass (parts of) its state to other agents.

Which information about domain objects is relevant depends on the individual application, but it is often useful to start by organising objects into different types, e.g. persons, vehicles, cargo objects, warehouses, etc. Once you have identified the types of objects that are relevant in your application, try to determine what kinds of things you may want to say about them and thus include in your model. For instance, in our example scenario of transportation, vehicles will have locations, are assigned to persons who drive them, and have a weight limit for their transport capability; but their colour does not matter in our context so this information does not need to be represented in the domain model. A useful distinction between characteristics is often whether they will remain constant throughout a relevant period of time, or whether they will change over time. For example, the type of a vehicle used for transport will not change in our scenario (e.g. a car will not become a truck or vice versa), whereas the status of a vehicle, e.g. whether it is in transit or parked, will change over time. Finally, try to establish how object types may relate to each other. For example, persons and vehicles may be linked by a driver-of (or driven-by) relationship. Here are the kinds of notes you should take about your domain modelling:

- Person (Object type):
  - Represented by their names (e.g. John);
  - Status of the person (e.g. working/waiting/resting);
  - Location of the person (longitude = x, latitude = y);
  - Job(s) assigned to the person.
- Vehicle (Object type):
  - Represented by its vehicle type and an ID number (e.g. Van10);
  - Status (parked, transit, transporting);
  - Location (longitude = x, latitude = y);
  - Driver(s) assigned to the vehicle.
- Relationships (between object types):
  - Driver-of a vehicle (a relationship between a vehicle object and a person object).

These notes are important because they will help you to be consistent about how you refer to objects and their characteristics and relationships. There will always be several choices of how relevant information can be represented. Some of these choices will make life harder or more difficult later, but often there are several alternatives that are all suitable. However, it is important that a choice is made and that the choice is used coherently. If you find later that a modelling choice is making your life difficult, do try to go back to the modelling exercise and make the changes required (expecting to throw away much of the work that was based on the old models). If it is not possible to make these changes, make sure you document the choice, its problem, why it could not be changed, and how to work around it (i.e. live with it).

Using the I-X Domain Editor, I-DE, an object class hierarchy can be specified. Each object class has a name (e.g. Vehicle) and it can have a set of properties and a set of sub-classes. Each class can have one or more parent classes, but cycles in the class hierarchy are not allowed. Object classes can inherit properties from their parent classes, but inheritance is currently primitive and overwriting or merging of properties is not advised. Figure 1 shows a simple object class model created using I-DE.

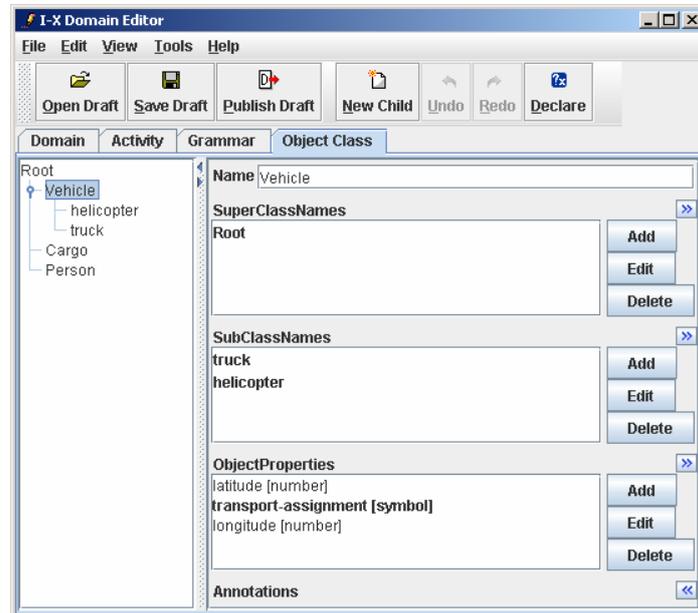


Figure 1 Simple Object Class Model

Once you have decided what types of object are relevant and what their relevant characteristics are, you can decide on how to represent these in *world state constraints* and start to put in some instances to make your ideas more concrete. World state constraints always have the following structure: *pattern = value*. In our models, including the example below, we usually choose representations in the form of: *attribute object = value*.

All world state constraints of a process panel are shown in the "state" sub-panel of the process panel. Figure 2 shows a world state based on the modelling information above. The world state includes a representation of a person, Operator1, at a location.

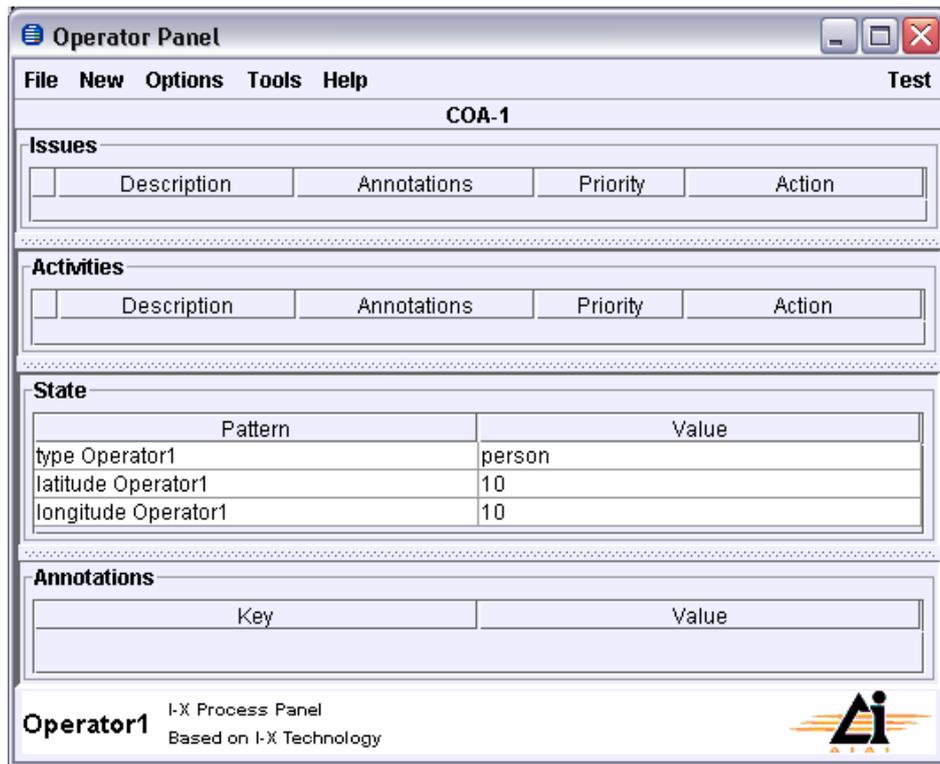


Figure 2: World State in iDemo-basic

If you have modelling experience, some of the representations in this model may seem odd to you. Our representation choices are restricted by I-X and by some of the tools we are using in this tutorial. See section 2.1.2.5 for more detail.

For details on how to set up an initial world state see section 2.3.

### 2.1.2.3 Building Process Models

Having gathered information about how things are (or should be) done (see section 2.1.1), you now need to organise this information into process models that are part of a domain model. These process models can then be used within process panels. It is not necessary to keep all process information in one domain model – different agents have different areas of expertise, so different process panels can have different domain models. Note, however, that if agents (process panels) need to communicate about objects and processes, their models have to be consistent.

I-X process models are organised as a set of *refinements*. A refinement has a *name* and a *pattern*: the *name* is used for display purposes; the *pattern* is used to determine which activities the refinement can be used for execution (i.e. the refinement is one way of dealing with any activity that matches its pattern). A refinement may also have a set of *issues* which have to be resolved, and a set of *nodes* which are the steps (sub-activities) to be taken when dealing with an activity in this way. A refinement can also have constraints. There are different types of constraints: a set of specific constraint types (precedence constraints, conditions, and effects), and there may be other constraints that the user can specify. Precedence constraints are used to specify node *orderings* within a refinement (e.g. “node1 must end before node2 can begin”). *Conditions* and *effects* are both constraints on the world state. Conditions specify aspects of the world state that have to be met in order for the refinement to work correctly, effects are specifications of how performing the steps in the refinement change the world state. Finally, refinements can have annotations, which can be used for comments.

Patterns are a central part of I-X process models. As mentioned above, each refinement has a pattern that is used to determine the activities for which the refinement can be used. Patterns also appear as specifications of nodes and issues within refinements, and as part of constraints. A pattern is made up of any number of terms (words and variables). For example, “move ?cargo-object ?to-x ?to-y” is a pattern. Variables are used to stand for (and match against) parts of patterns. In I-X, variables are terms that start with “?”, e.g. “?cargo-object”. They are needed to generalise specific situations and make them more generally applicable (“move ‘medical-box1’ ...” vs. the above). During process execution, matching variables in specific situations assigns values to the variables and thus makes generic descriptions specific. In process models, variables

can be used to query and update the world state, and to carry information into refinements and between process steps.

Like many process modelling techniques, I-X supports the concept of “hierarchical” models, i.e. breaking down (expanding) activities into sub-activities, which can themselves be expanded into further detail. Unlike most techniques, I-X process models use flexible links between activities and their expansions. In the process models, each refinement specifies via its pattern that it can be used to expand “this kind of activity”, i.e. any activity that matches the pattern. It is not until process enactment that a connection between an activity and an expansion is made, leaving options of how to perform the activity (alternative expansions) open until a choice is required. During process modelling, this means that you will produce a “flat” set of refinements rather than an expansion tree. Figure 3 shows a simple process model in I-DE.

To produce I-X process models, the best support currently available is I-DE, the I-X domain editor. This editor can be started from any I-X process panel (under the Tools menu) or it can be used stand-alone, although this stand-alone mode is not yet documented. When using I-DE, remember to save your work frequently! Its user guide describes details on how to use the editor to enter and view process information.

### **2.1.2.4 Making Models Work Together**

The most used connection between models is that between process models and world state. However, there may be connections between the agents and the world state, for example if you want to represent, monitor, and visualise an agent's geographical location and there may be connections between agents and process models where an agent's identifier can be passed to a process via parameters or world state conditions.

The connection between world state constraints and process models is at the heart of I-X techniques. Having represented objects and their characteristics as world state constraints, you can then use them in activity specifications of your process models. Because the process models are designed to be generally useful, they will contain variables. The variables can be used to link different parts of the specifications. For example there may be an activity that assigns a vehicle to a person who drives it. In order to do this, we need to first find a vehicle that is free. After the assignment activity, the vehicle is no longer free. This could be modelled as follows: the activity's name is “assign-vehicle-to-driver”. Its pattern is “assign ?driver ?vehicle”. The activity has the conditions “type ?driver = person”, “type ?vehicle = vehicle”, “assigned ?vehicle = false”. The activity has the effects “assigned ?vehicle = true”, “driver-of ?vehicle ?driver”, and “assigned ?driver = true”. (Note: in this activity we do not care whether the driver has already been assigned a vehicle.) If we take a world state containing a person “John” and a vehicle “Truck1”, we can perform the activity: its conditions are met if we assign the variables ?driver=John and ?vehicle=Truck1. After we performed the activity, the state changes to include: “assigned Truck1 = true”, “assigned John = true”, “driver-of Truck1 = John”.

By now you will appreciate that the notes on object types and characteristics will be useful when working on other models (e.g. process models).

Note: the conditions, effects, and world state descriptions may need to be implemented in a somewhat verbose way because of the restrictions in the kinds of constraints I-X can manage (see next sub-section).

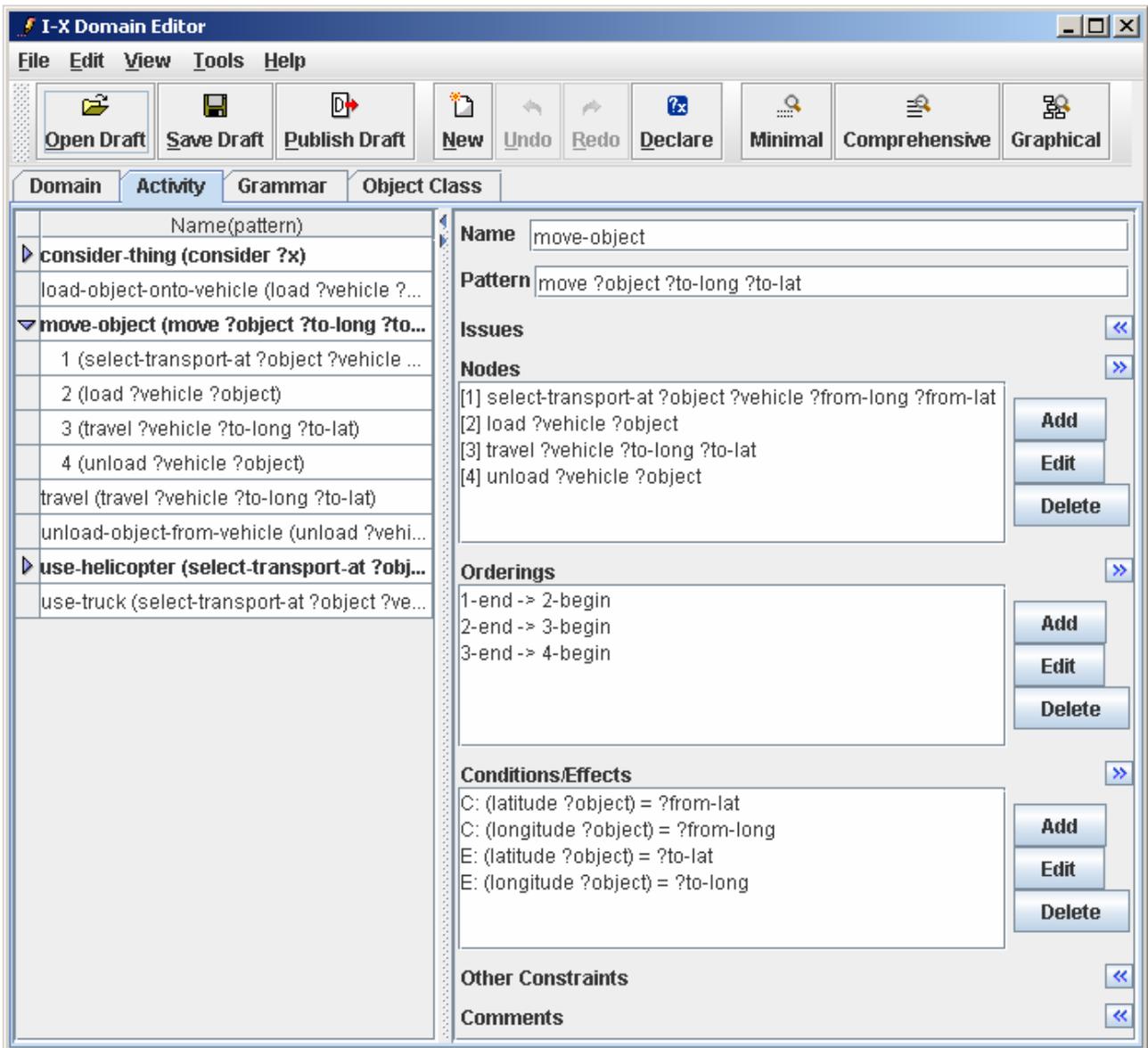


Figure 3: Simple Process Model in I-DE

### 2.1.2.5 Some Constraints on Domain Models

In addition to the constraints imposed by the specific approach that I-X adopts with respect to domain models, there are some constraints that are imposed either by limitations of the current version of I-X or by the approach taken by add-ons we are using in this document.

The main limitation of I-X with respect to process models is that I-X cannot handle logic within the conditions of refinements (process models) in a straightforward way. This means that some models are more long-winded than they should be. The only logical operator between conditions is “and”, which means that alternatives and negations cannot be expressed directly. For example, there is no direct way to say that a vehicle cannot be assigned to a resting driver (cannot say “status ?driver != resting”, and cannot say “status ?driver = working OR status ?driver = waiting”). If this kind of specification is required, we may have to introduce a separate “assignable” characteristic to driver objects, e.g. “assignable ?driver = true”. We can then use this to check whether we can assign a vehicle to the driver.

Another limitation is that I-X does not yet include the concept of object class hierarchies. This means that we cannot refer to hierarchical aspects of the object class model. For example, our model of the move-object activity should include a condition that the ?vehicle must always be an object of class Vehicle. During an I-X session, we should be allowed to use all suitable vehicles to bind the variable. However, with the Vehicle class condition, the current version of I-X would not have the knowledge that a Helicopter (or a Truck) is a Vehicle and would not allow any of these objects to be used in the activity. The only object class constraints

that I-X can currently deal with are ones that relate to an object's immediate class, without having to traverse the object class hierarchy. For example, a condition to say that the ?vehicle must be a Helicopter should be fine, as long as all helicopters are defined as having the object class Helicopter. I.e., this usually means using the leaf-nodes of the object class hierarchy.

The add-on we use in this document is the I-X Map Tool which can be used to view geographical information of agents and objects. The main limitation imposed by the I-X Map Tool add-on is on attribute names. I-X Map Tool inspects the world state of a process panel and derives its own conclusions based on a set of pre-determined keywords. When I-X Map Tool finds a constraint pair of the form "longitude ?who ?x" , "latitude ?who ?y" it translates it into a point on the map. If there is also a "type ?who ?type-name" constraint, I-X Map Tool checks whether there is a specific icon to display the ?type-name objects or an even more specific icon for displaying the individual ?who. This means that if I-X Map Tool is to be used in conjunction with an I-X process panel, locations have to be represented using longitude and latitude information for object types, using the exact keywords and the exact statements as described in the models above although it may seem preferable to use different attribute names (e.g. "object-type" instead of "type") or different representational styles (e.g. "location ?place-name ?x ?y").

## 2.2 Configuring I-X Applications

Applications are kept in the applications sub-directory called "apps". A good starting point for developing I-X applications is to use the sample application, idemo-basic, and to adapt it to your own application's requirements. The first step is to copy the whole application (copy the idemo-basic directory) and rename the copy to something you like (we call it my-app here). Within the application's area, configuration work will take place in two sub-directories:

1. In the config area (my-app/config/) the properties of agents are set. Nearly all your configuration work will take place in this directory
2. In the scripts area (my-app/scripts/win) you will place scripts to run your agents. You may also specify add-ons here (see below), if they are required by your application

The building blocks of applications are process panels. The behaviour and appearance of panels can be configured as appropriate for the application. The main way of configuring applications is through property files (which conventionally have the file extension "props"). Property files are kept in the config directory. They are used to supply values of various parameters of process panels (see below) and they identify any test menu files used to adapt the test menus of their respective panels. Property files can also include initial plans, domain libraries, background colour, etc. If you are using extensions and add-ons, they too have to be linked into the application here. There is also a file called bg-colour.txt that contains the code of a few useful colours that work well as backgrounds of panels. This file is just for documentation/guidance.

In summary, property files set parameters that get passed to a panel at start-up. All parameter settings are optional. The parameters that you are likely to want to set for a process panel agent are:

- the panel's symbol name (e.g. Supervisor) which is used when communicating with other agents,
- the panel's label or display name (e.g. Operations Supervisor Panel), the panel's logo, and its background colour, all used in the panel's display to suit the panel's purpose and make it easily recognised (especially when more than one panel is used on one display),
- the domain model to load at start-up,
- the plan (saved process panel state) to load at start-up,
- the test-menu-adaptation file to use (if any), and
- the I-Space configuration (relationships to other agents).

For a full set of process panel parameters and their descriptions see the I-X Parameters List at <ix-base>/doc/parameters.txt.

Each process panel usually has its own script (in my-app/scripts/win) that can be used to start up the panel. For simple applications, the only parts of the script that may have to be adapted are the name of the panel's properties file and whether the panel is to run a name server. There are more advanced parameters that can be used e.g. if non-standard locations are used for the application's files, if Java add-ons are to be used, or if different communication protocols are to be used. It is also possible to pass command-line parameters to the panel. This can provide an alternative way of specifying initial plans, domain models etc. If add-ons like the

map state viewer are to be used, their file locations have to be specified here. If a non-standard (custom programmed) IX-Agent other than the standard one (ix.ip2.lp2) is to be used, the command line for calling the java class will be edited here.

Relating agents has the advantage that additional options will appear in the panel to handle activities and issues. For example, if the panel's agent has designated subordinates, the agent may delegate activities to them using the commands that will now be available through the Action menu.

Setting up test menus has the advantage that potentially complex effects can be achieved with a single user action. For example, if another staff member joins the team, we can use a single test-menu selection as a short-cut to start up an agent, give it a physical location, and re-distribute jobs to share the work evenly. Unfortunately, setting up test menus involves writing XML.

Add-ons are modules that are linked into an I-X application and used to perform tasks on (parts of) the application. If you are using add-ons, these are specified in the scripts area, and their resources are kept in the resources area (my-app/resources). We provide an example of an add-on, I-X Map Tool, which is used to display geographical information about world state objects.

### 2.3 Initialising I-X Applications

Initialising (or populating) I-X applications is done in three areas:

1. **Process models:** These are models of best-practice or instructions of how things can be done. It will often be necessary to have different domains of expertise for different panels (e.g. operator vs. supervisor). Process models can be built using the integrated domain editor (I-DE) that comes with every instance of an I-X panel. Alternatively, domain models can be produced as XML documents in any editor. Although process models can be created and modified while the application is used, process models will often be built before the application is used, i.e. when running the application they are available to support activities.
2. **World state:** By world state we mean a description of the current state of the world, in which an I-X process panel operates. In I-X, such a world state is described using "constraints" such as "colour = red". The world state of a panel can be manipulated directly from within the panel by explicit user actions or through effects of activities that have been carried out. To explicitly change the world state, use "New | New Constraints" menu option to add a new constraint (which will appear in the panel's state area), or modify existing world state values by typing into the value fields in the world state area. The world state of a panel can also be changed by other process panels. A process panel can pass (parts of) its own world state to another panel in bulk (using the "File | Send state" option, followed by choosing a panel to send it to) or in part by sending constraints as messages to another panel. A world state of a panel can be saved as a plan. Previously saved world states can later be loaded and may be used as an initial state. Initial world states will usually be available with an application, and users may save their own world states if required.
3. **Plans:** A plan is a collection of activities in which an agent is involved in some way. Usually these activities are related to each other and to the world state (which can be saved along with a plan). A plan changes over time as activities are performed, expanded, delegated, etc. Initial plans may be provided with an I-X application, but the user can save their own plans if required. This is often done to save a session that is to be continued later.

See also later sections for step-by-step instructions.

### 2.4 Extending I-X Applications

I-X is implemented in Java and its architecture and components are accessible to Java programmers. The I-X source is well-documented using javadoc.

Apart from general programming, there are two main areas in which I-X applications can be extended and adapted: by providing a custom world state viewer, and by providing a communications strategy. The I-X Process Panels User Guide gives details on both of these. In addition, the map viewer described in the next section (4.2) is an example of a custom world state viewer. See also sections 4.3, 4.4, and 4.5 below.

### 3 A Basic Configuration Example

This section describes I-Demo Basic, the demo application that illustrates the basic concepts of I-X technology. The next section describes further configuration issues, each illustrated with their own I-Demo application. The configuration tasks that are covered in I-Demo Basic are:

1. Using a domain,
2. Building an initial plan and state,
3. Using an initial plan and state,
4. Using a test menu, and
5. Running without a name server.

Below, we first describe the initialisation files that are required and give an indication of how they are built. We then describe in detail how the configuration tasks above are achieved in I-Demo Basic.

#### 3.1 Files for Initialisation

The files that are used to initialise I-Demo Basic are the domain model, the initial plan, the initial state, and the test menu.

##### 3.1.1 Domain Models

The operator's domain model can be found in I-Demo Basic's domain-library sub-directory: "apps/ideo-basic/domain-library/domain-operator.xml". Future versions of this document may have a section detailing the process of building this model. For the moment, use I-DE to look that the domain model file provided.

##### 3.1.2 Test Menus

The operator has a test menu file to provide short-cuts for placing tasks onto the operator's panel. The file is in the configuration sub-directory: "apps/ideo-basic/config/operator-test.xml". Unfortunately, setting up test menus involves writing XML. Future versions of I-X may have support for building such files. For the moment, use an XML viewer to look that the test menu files provided and edit files with any means you can find. Test menus can be specified within one test menu file or in several files whose entries will be appended to the menu. The test menu setup text (<ix-base>/doc/pages/notes/test-menus.txt) describes how to put together test menus for process panels.

##### 3.1.3 Initial States and Plans

Initial states and plans are located in the domain-library sub-directory. The operator has both an initial state ("apps/ideo-basic/domain-library/state-operator-init.xml") and an initial plan ("apps/ideo-basic/domain-library/plan-operator-init.xml"). They are kept separate to support maintenance of the initialisation files. Both are best viewed by starting up the operator panel.

###### 3.1.3.1 Building Initial States and Plans

The easiest way to create an initial state is to run the operator panel. First make sure that the panel's script allows you to save the plan information you need. Make sure the -plan-state-to-save part of the application parameters is set correctly. For saving the whole plan, the parameter should be set to "\*". For Windows, make sure the file scripts/win/operator.bat contains a line like this: "set app\_params=-ipc=xml -load config/operator.props -plan-state-to-save=\*". Once this is done, start the operator panel, reset it to ensure there is no plan or state information already there, and enter the state information in the form of new constraints which will be shown in the panel's state area. When you are happy with the constraints, save the panel's state (using the save plan option). Note that building initial states and plans should be done after the configuration of the application.

```
double-click on scripts/win/operator.bat (ignore messages about not finding files)
In the "File" menu, select "Reset" ► "All"
In the "New" menu, select "New Constraint" and type "type Helil = helicopter"
then click "Ok"
...add more constraints as required...
In the "File" menu, select "Save Plan As...", then specify file "state-operator-init.xml"
```

The easiest way to create the operator's initial plan is to again reset the operator panel and use the test menu to place an initial task onto the agent's agenda by using the test menu option. Then the plan can be saved in the same way that the state was saved.

```
In the "File" menu, select "Reset" ► "All"  
In the "Test" menu, select "Move...locagation-a"  
...add more activities or partially run the demo as required ...  
In the "File" menu, select "Save Plan As...", then specify file "plan-operator-  
init.xml"
```

### 3.2 Configuration Issues

#### 3.2.1 Using a Configuration File

The Operator in I-Demo Basic uses the configuration file "config/operator.props". It is linked into the application as an application parameter in the agent's script. For Windows the file "scripts/win/operator.bat" contains the line "set app\_params=-ipc=xml -load config/operator.props"

#### 3.2.2 Using a Domain Model

The domain is specified as part of the panel's configuration. For I-Demo Basic, the file "config/operator.props" contains the line "domain=domain-library/domain-operator.xml" to link in its domain model in that location.

#### 3.2.3 Using an Initial Plan and State

The initial plans and states are specified as part of the panel's configuration. For I-Demo Basic, the file "config/operator.props" contains the line "plan=domain-library/state-operator-init.xml,domain-library/plan-operator-init." to link in both plan and state. Note that there are no spaces in the line at all.

#### 3.2.4 Using a Test Menu

Test menu files are linked into the application in the configuration files. For I-Demo Basic, the file "config/operator.props" contains the line "test-menu=config/operator-test.xml" to link in its test menu file in that location.

#### 3.2.5 Running without a Name Server

I-Demo Basic only has one agent and runs without a name server. This is achieved using the application parameters in the script files. For Windows, the file "scripts/win/operator.bat" contains the line "set app\_params=-ipc=xml -load config/operator.props -no name-server".

## 4 Step-by-Step Configuration

This section describes how to build each of the I-Demo demonstrators from the basic demonstrator, I-Demo Basic. There are three distinct approaches for developing your own application with the help of the I-Demo suite of demonstrators:

1. start with a copy of I-Demo Basic and add I-X technologies one at a time until you have included all the features required for your application;
2. start with a copy of the I-Demo demonstrator that is closest to what you need, then add I-X technologies one at a time until you have included all the features required for your application;
3. start with a copy of I-Demo Features and remove all the features you do not need for your application.

Which of these approaches you take will depend on your own preference for building applications, and on the complexity of your application, i.e. how many of the features you need to include.

The descriptions below all assume the first approach. Each starts with taking a copy of idemo-basic and then describes how to add the specific part of I-X technology that the demonstrator aims to illustrate. The separate tasks to perform for building an I-Demo application are:

- Conceptual modelling,
- Building files for initialisations (domain models, test menus, and initial states and plans), and

- Configuring the application.

The descriptions of the I-Demo applications below is organised along these tasks, also listing the specific aspects of I-X application building that is illustrated. Detail on how to run each I-Demo application can be found in their respective demo scripts.

### 4.1 I-Demo Cooperation

I-Demo-Cooperation illustrates how to use I-X process panels to support two agents who are working together on a common task. This section describes how to build the I-Demo Cooperation application, starting from I-Demo Basic, the base application. The configuration tasks that are covered in building this demo are:

1. Running a name server,
2. Relating agents, and
3. Combining test menus.

#### 4.1.1 Conceptual Modelling

Add a supervisor to the scenario whose job it is to take on tasks (for which the supervisor is then responsible), and to authorize the use of helicopters. The supervisor agent will be called "Supervisor".

#### 4.1.2 Building Files for Initialisation

##### 4.1.2.1 Building Domain Models

Build a domain model for the supervisor that reflects this. Call it domain-supervisor.xml and put it into the domain-library sub-directory.

```
Create "apps/ideo-coop/domain-library/domain-supervisor.xml"
```

Future versions of this document may have a section detailing this process. For the moment, use I-DE to look that the domain model file provided.

##### 4.1.2.2 Building Test Menus

Both operator and supervisor have additional test menu entries to delegate/escalate tasks directly to the other agent. These menu entries are short-cuts to use in a demo situation. If you do not want to use such short-cuts, the user of the application can achieve the same by typing the activities into the Messenger tool and sending them to the other agent.

To add to the test menus, build XML files of the test menu entries and place them in the config area:

```
Create "apps/ideo-coop/config/operator-test.xml"  
Create "apps/ideo-coop/config/supervisor-test.xml"
```

Future versions of I-X may have support for building such files. For the moment, use an XML viewer to look that the test menu files provided.

##### 4.1.2.3 Building Initial States and Plans

Both agents have initial states, and the supervisor agent also has an initial plan. The operator's initial state is the same as for ideo-basic and therefore needs no change. The supervisor's initial state needs to be created.

The easiest way to create the supervisor's initial state is to run the supervisor panel, to reset it to ensure there is not plan or state information already there, and to enter the state information in the form of new constraints which will be shown in the panel's state area. When you are happy with the constraint, save the panel's state (using the save plan option). Do all this after the configuration of the application.

```
double-click on scripts/win/supervisor.bat (ignore messages about not finding files)  
In the "File" menu, select "Reset" ► "All"  
In the "New" menu, select "New Constraint" and type "type Helil = helicopter"  
then click "Ok"  
In the "File" menu, select "Save Plan As...", then specify file "state-supervisor-init.xml"  
In the "File" menu, select "Exit"
```

## I-X Configurer's Guide - Methods for Building I-X Applications

The supervisor's initial plan was previously the operator's initial plan, so all you have to do is to rename the file:

```
Go to the domain library directory, "apps/ideo-coop/domain-library"  
Rename "plan-operator-init.xml" to "plan-supervisor-init.xml"
```

### 4.1.3 Configuring the Application

Take a copy of idemo-basic. Rename it idemo-coop. You will have to work in all sub-directories. Start up a to-do list to note things that you should remember to do later. Copy the operator's property files and script files to make the supervisor's files: copy config/operator.props and rename it config/supervisor.props, copy scripts/win/operator.bat and rename it scripts/win/supervisor.bat.

```
Take a copy of idemo-basic. Rename it idemo-coop  
Take a copy of config/operator.props and rename it config/supervisor.props  
Take a copy of scripts/win/operator.props and rename it  
scripts/win/supervisor.props
```

#### 4.1.3.1 Adjusting Properties

Adjusting properties is done in the config directory. Edit the operator's properties to fit the new environment:

```
Edit config/operator.props
```

- The operator does not have an initial plan in this application, so remove "domain-library/plan-operator-init.xml" from the plan parameter:

```
Change "plan=domain-library/state-operator-init.xml, domain-library/plan-  
operator-init.xml"  
to "plan=domain-library/state-operator-init.xml"
```

- Add test menu entries for communicating with the supervisor to the operator's test menu and take a note to build an XML file with the new test menu entries with the right name in the right location, unless you have already done this.

```
Change "test-menu=config/move-thing-test.xml"  
to "test-menu=config/move-thing-test.xml, config/operator-test.xml"
```

**Note:** build config/operator-test.xml

- Add the supervisor to the operator's I-Space, making sure there are no spaces (watch for trailing ones)

```
Change "superiors=" to "superiors=Supervisor"  
Save and close
```

Edit the supervisor's properties to reflect that it has different characteristics from the operator:

```
Edit supervisor.props
```

- Replace all occurrences of "Operator" with "Supervisor" and "operator" with "supervisor" and take a note on your todo list to make the supervisor's initializations and the domain model with the right names and in the right locations, unless you have already done this.

```
Change "symbol-name=Operator" to "symbol-name=Supervisor"  
Change "display-name=Operator Panel" to "display-name=Supervisor Panel"  
Change "domain=domain-library/domain-operator.xml"  
to "domain=domain-library/domain-supervisor.xml"  
Change "plan=domain-library/state-operator-init.xml, domain-library/plan-  
operator-init.xml"  
to "plan=domain-library/state-supervisor-init.xml, domain-library/plan-  
supervisor-init.xml"
```

**Note:** build domain-library/domain-supervisor.xml  
build domain-library/state-supervisor-init.xml  
build domain-library/plan-supervisor-init.xml

- Add test menu entries for communicating with the operator to the supervisor's test menu and take a note to build an XML file with the new test menu entries with the right name in the right location, unless you have already done this.

```
Change "test-menu=config/move-thing-test.xml"
to "test-menu=config/move-thing-test.xml,config/supervisor-test.xml"
```

**Note:** build config/supervisor-test.xml

- add the operator to the supervisor's I-Space, again making sure there are no spaces (watch for trailing ones)

```
change "subordinates=" to "subordinates=Operator"
```

- Change the background colour to help users distinguish between different agents' windows, especially panels. Consult config/bg-colour.txt for good colours to use, e.g. ffeeff.

```
Change "metal-theme-secondary3=0xeeeeff" to "metal-theme-secondary3=0xffeeff"
Save and close
```

### 4.1.3.2 Adjusting Windows Scripts

Adjusting scripts is done in the scripts directory, "apps/demo-coop/scripts/win". Edit the operator's script file to suit the new environment:

```
Edit scripts/win/operator.bat
```

- Remove the note to not use a name server

```
Change "set app_params=-ipc=xml -load config\operator.props -plan-state-to-
save=* -no name-server" to "set app_params=-ipc=xml -load
config\operator.props -plan-state-to-save=*"
Save and close
```

Take a copy of operator.bat and call it supervisor.bat to make the supervisor's script. Edit the supervisor's script:

```
Edit supervisor.bat
```

- Replace all occurrences of Operator with Supervisor and "operator" with "supervisor"; also replace the note not to use a name server with a note to run a name server.

```
Change "title Operator Command Console" to "title Supervisor Command Console"
Change "set app_params=-ipc=xml -load config\operator.props -plan-state-to-
save=* -no name-server" to "set app_params=-ipc=xml -load
config\osupervisor.props -plan-state-to-save=* -run-name-server"
Save and close
```

### 4.1.3.3 Adjusting Unix Scripts

Adjusting Unix scripts is done in the directory, "apps/demo-coop/scripts/unix". Edit the operator's script file to suit the new environment:

```
Edit scripts/unix/operator
```

- Remove the note to not use a name server

Take a copy of the file "operator" and call it "supervisor" to make the supervisor's script. Edit this file:

```
Edit supervisor
```

- Replace all occurrences of Operator with Supervisor and "operator" with "supervisor"; also replace the note not to use a name server with a note to run a name server.

```
Change "title Operator Command Console" to "title Supervisor Command Console"
Change "set app_params=-ipc=xml -load config\operator.props -plan-state-to-
save=* -no name-server" to "set app_params=-ipc=xml -load
config\osupervisor.props -plan-state-to-save=* -run-name-server"
Save and close
```

## 4.2 Using the Map State Viewer Add-On

I-Demo Map will illustrate how to use the map state viewer add-on. For the moment, use the following instructions to guide you.

There is a stand-alone tool that can be used to provide a view of the world state that is based on maps and geographical locations. In order to use this tool as a world state viewer, you should get map image of your area. This could be, for example, a street plan, an aerial photograph, or a plan-view of your office space. You can also provide different images that are used to show the locations of agents or objects that have location information. Any world state objects that have longitude and latitude information will then be shown in the map viewer. By default, object locations are shown with the help of a default icon. You can replace the default icon and you can also use your own icons to show object locations. To do this, use object names and object type names to link objects to specific icons via the icon file names. The tool will show the most specific icon it can find.

To set up the map tool, decide what images you want to use and set about getting the images ready. Getting the images to look good requires specific skills and tends to take a long time so plan ahead for this. If there are images you want to use that are not available yet, put a placeholder image in the right location. Then set up the tool:

- edit resources/map/world-map/map.props

```
"jpgmaplayer.jpgPath=resources/map/world-map/myWorld.jpg"
```

To give the rescue centre an overview of the locations of drivers and vehicles:

- edit config/rescueCentre.props file

```
"state-viewer-class=ix.ip2.StateViewMap
map-properties=resources/map/world-map/map.props
map-type-icons=resources/map/icons/type-icons/
map-object-icons=resources/map/icons/object-icons/"
```

For more detail, see the I-X Map Tool documentation in `<ix-base>/apps/addon/map/java/resources/html/ix-map-help.html`.

### 4.3 Running Agents on Different Machines

I-Demo Distributed will illustrate how to run agents on different machines. For the moment, use the following instructions to guide you.

The easiest way to run agents on different machines is by using a Jabber server. You can use a third party one such as jabber.org (and you can create a new account there using I-X and the Jabber communication strategy - just tick the "New Account" box). Different jabber users usually have to be mutually "subscribed" to send messages - but again, this can be done from within I-X. However, you then become reliant both on having an Internet connection and on your chosen jabber server being 'live'.

Once the Jabber accounts are set up, all you have to do is to alter `-ipc=xml` and `ipc=xml` entries in the scripts to *jabber* (`ipc=jabber`).

The alternative is to stick with the nameserver/xml strategy as used in the rest of this document. To do this you must know the names or IP addresses of the end points. This is not so good across the Internet where DHCP (Dynamic Host Configuration Protocol) is in use. However, to use this approach, start up one panel with a name server as described above. In the other panels on the other machines give a parameter (`app_param`) of `-name-server=localhost:5555` if you run on the same machine (5555 is the port you are using). Replace "localhost" with the name or IP address of the machine that is running the name server, e.g. `-name-server=somemachine.aiai.ed.ac.uk:5555`.

In summary, to use the nameserver/xml strategy:

- edit scripts/win/rescueCentre.bat

```
"set app_params=-ipc=xml -load config/rescueCentre.props -run-name-server"
```

- edit scripts/win/rescueDriver1.bat

```
"set app_params=-ipc=xml -load config/rescueCentre.props
-name-server=eoiRescue.aiai.ed.ac.uk:5555", assuming that rescueCentre panel
is running on eoiRescue.aiai.ed.ac.uk.
```

### 4.4 Running Across Firewalls

If you are running an I-X panel behind a local firewall and wish to connect to other panels via the Internet, it may be necessary to open ports in the firewall to allow messages to be sent and received. The steps that need to be taken depend on the particular communication strategy that you are using; below we describe what needs to be done for the XML and Jabber strategies. You will probably also need to consult the user guide (<ix-base>/doc/IX-IP2-User-Guide.pdf) for details of how to set particular parameters etc.

#### 4.4.1 XML Across Firewalls

If you are running a panel behind a firewall, and you wish to communicate with a name server and/or other panels that are outside that firewall, then you will need to ensure that the ports that you use for communication are opened to allow Internet traffic. You should probably stipulate the particular port that you would like to use and ensure that it is opened (note that unless a particular port is stipulated, the behaviour of the name server is to allocate a port dynamically). If you are administrator of the firewall, then you should open this port using the TCP protocol; otherwise you should ask your local network administrator to do this for you.

Moreover, if you are also running a name server behind a firewall, and you wish to allow it to be accessed by panels outside the firewall, then it is expected that the designated port used by the name server (default: 5555) will be opened to allow communications through. Again, if you have administrative privileges you should open this port; otherwise you should ask your local network administrator for help.

#### 4.4.2 Jabber Across Firewalls

If you are using Jabber, and the Jabber server that you are using is outside your firewall, then it will be necessary to open a port through the firewall to enable this communication. In most cases the port to open will be 5222 (this is the default port used for Jabber client connections to a server). If you are administrator of the firewall, then you should open this port using the TCP protocol; otherwise you should ask your local network administrator to do this for you. (If you wish to use only certain designated servers, for added security port 5222 can be constrained to allow connections only to those servers' IP addresses.)

### 4.5 Building Your Own I-X Agent

For certain applications, it may be useful to interact with agents that have perhaps been developed for different purposes but which share some underlying concepts with I-X panels. For instance, the ability to invoke seamlessly an agent that is able to perform a particular type of activity that arises in your application, and then to confirm that this activity has been successfully completed, would be of great use. This can be done by 'wrapping' this agent with an I-X wrapper which deals with the interactions with the other agents – sending and receiving messages – and, when appropriate, invoking and monitoring the agent's intrinsic abilities, and collecting any results that are generated. In other words, since its internal mechanisms are hidden, to other I-X users the wrapped agent now appears to be an I-X agent like any other. For those with a little programming experience, the I-X architecture provides some support to allow this sort of I-X agent to be developed.

For java programmers, the easiest way to do this is to extend the class `ix.icore.IXAgent`: this provides basic initialisation methods (allowing, for instance, standard I-X parameters to be read, either from the command line or from a props file, an agent to be started, and its communications to be initialised, all done in the same manner as for an I-X panel) and methods for handling the different types of message that might be received (some or all of which might be overridden to invoke the specific capabilities of this agent). For further details, consult the developer's notes (see <ix-base>/doc/notes/index.html) and the I-X javadoc. In future, there will be an I-X Developer's Guide to cover this area.

While using java to extend the existing `IXAgent` class provides the most convenient approach to this sort of development, it is not the only approach possible. Any programming language that offers the basic functionality of generating, sending, receiving and parsing appropriate messages could, in theory, be used to develop an I-X agent of this sort. The I-X Syntax document provides more details about the message formats that I-X uses.

## 5 Further Reading

In the first instance, see the I-X Web pages at <http://i-x.info/> or at <http://www.aiai.ed.ac.uk/project/ix/>, in which you can find links to the documents listed below. The web pages also come as part of the core release and can be found in <ix-base>/doc/. The main sources of information currently available are:

## I-X Configurer's Guide - Methods for Building I-X Applications

- The I-X Process Panels User's Guide at <ix-base>/doc/IX-IP2-User-Guide.pdf;
- The I-DE User's Guide at <ix-base>/doc/IX-Domain-Editor-Guide.pdf;
- The I-X Map Tool – Addon Help at <ix-base>/apps/addon/map/java/resources/html/ix-map-help.html;
- The two I-Demo scripts <ix-base>/doc/IX-IDemo-Basic.pdf and <ix-base>/doc/IX-IDemo-Coop.pdf;
- A set of notes intended for developers and technical users in <ix-base>/doc/notes/. These notes include syntax specifications and details for configuration (e.g. for test menus);
- The I-X Parameters List at <ix-base>/doc/parameters.txt.

See also <ix-base>/doc/index.html for an overview of what other I-X papers and documents are available.

In future, the documentation will include

- The I-X Developer's Guide – a comprehensive guide to tailoring I-X further to suit the application at hand.
- The I-X Method for Domain Modelling – a comprehensive guide to how a domain modeller should go about capturing the information required for I-X applications.

Advanced Readings:...

[1] IDEF3: <http://www.idef.com/>.

[2] John Sowa's home page on Ontology: <http://www.jfsowa.com/ontology/index.htm>.

[3] Ontological Engineering: With Examples from the Areas of Knowledge Management, E-Commerce and the Semantic Web Asunción Gómez-Pérez, Mariano Fernandez-Lopez, Oscar Corcho.

[4] Business Process Execution Language for Web Services: <http://www-106.ibm.com/developerworks/webservices/library/ws-bpelcol1/#1>, and <http://www-106.ibm.com/developerworks/library/ws-bpel/>.

[5] OWL-S: <http://www.daml.org/services/owl-s/>.

[6] OWL: <http://www.w3.org/TR/owl-features/>.

[7] Java 2 SDK 1.4.2 Installation Notes for Microsoft Windows: <http://java.sun.com/j2se/1.4.2/install-windows.html>.

[8] Asuncion Gomez-Perez, Mariano Fernandez-Lopez, Oscar Corcho, Ontological Engineering, Springer,