I-X Process Panels – User Guide

Austin Tate, Jeff Dalton, Jussi Stader and Stephen Potter
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`

Version 4.3 – 27 Mach 2006

CONTENTS

0 How to Use This Guide

This document is intended to provide an introduction to the I-X system, its foundations and its tools, to give a user's manual allow you to apply I-X to your task, and to provide some guidelines for tailoring I-X to your particular task, domain and environment.

Section 1 gives an introduction to the motivation behind the I-X project and the facilities it provides its users.

Section 2 provides a **quick-start guide**: those eager to try the I-X tools should skip to this section.

The subsequent five sections describe the use of individual I-X tools in greater detail:

- **Section 3** describes the use of *I-P*^{2}, Process Panels.
- **Section 4** covers the use of *I-DE*, the domain and process editor.
- **Section 5** explains the use of the *I-Space* agent organisation and relationship tool.
- **Section 6** describes the use of *Messenger* instant messaging tool.
- **Section 7** discusses the use of *I-Plan* an AI activity planning tool.

Sections 8 and **9** outline the steps necessary to move beyond a basic use of the I-X tools by tailoring them for particular applications.

Finally, **Appendices A, B** and **C** provide handy reference information.

1 Introduction to I-X and I-X Process Panels (I-P^2)

1.1 I-X Research Programme

I-X is a multi-faceted research programme whose goal is to produce a well-founded system that allows humans and computers to cooperate in the creation or modification of some artefact. This artefact may be a plan, a design or a physical entity – in other words, I-X supports **synthesis tasks**. Beyond this, it can also be used to provide a framework for any collaborative activity.

The I-X research draws on earlier work on O-Plan (Tate et al. 1998; 2000; 2002), <I-N-OVA> (Tate, 1996), the Enterprise Project (Fraser and Tate, 1995; Uschold, et al., 1998) and the Task-Based Process Management Project (Stader, 2000) but seeks to make the framework generic and to clarify terminology, simplify the approach taken, and increase the re-usability and applicability of the core ideas.

The I-X research programme includes the following threads or work areas:

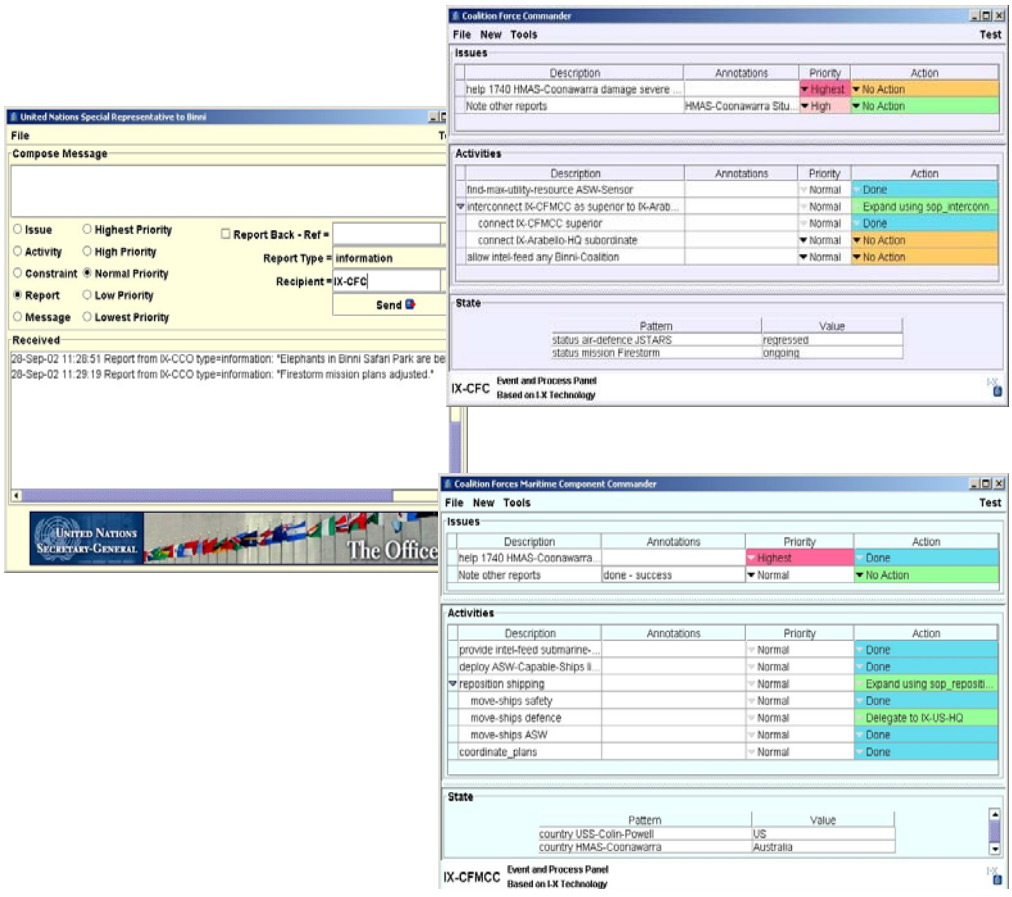
1. **I-Core**, comprising the core architecture, an underlying ontology for activity and processes (termed <I-N-C-A> - **I**ssues, **N**odes, **C**onstraints and **A**nnotations), and the terminology used to describe systems or applications built in the I-X framework.
2. **I-P^2**, which are I-X Process Panels used to support user tasks and cooperation.
3. **I-DE**, which is the I-X Domain Editor, which is itself an I-X application but is also used to create and maintain the domain description, process models and activity specifications used elsewhere.
4. **I-Plan**, which is the I-X Planning System. This is also used within I-P^2 and other applications as it provides generic facilities for supporting planning, process refinement, process composition, dynamic response to changing needs, etc.
5. **I-Views**, which are viewers for processes and products, and which are employed in other applications of I-X. I-Views can be for a wide range of modalities and types of user.
6. **I-Faces**, which are underlying support utilities to allow for the creation of user interfaces (*User I-Faces*), inter-agent communications (*Communications I-Faces*) and repository access (*Repository I-Faces*).
7. **I-X Applications** of the above work areas in a variety of domains. These currently include:
    a. Coalition Operations (*CoAX*)
    b. Emergency and Unusual Procedure Assistance (*I-Aid*, *I-Help*, *I-Rescue*)
    c. Support Desks (*I-Support*)
    d. Multi-Perspective Knowledge Modelling and Management (*I-AKT*)
    e. Medical Best Practice Procedures or Protocols (*I-Medic*)
    f. Natural Language Presentations of Procedures and Plans (*I-Tell*)
    g. Collaborative Meeting and Task Support (*I-Space*, *I-Room* and *I-World*).
    h. Intelligent Messaging (*I-Me*).
8. **I-X Student Projects**, which are deepening and refining a number of aspects of the I-X research programme.
9. **I-X Technology Transfer**, including work on standards committees, especially for process, plan, activity and capability models.

1.2 I-X Process Panels (I-P^2)

An I-X Process Panel (I-P^2) is designed to act as a workflow, reporting and messaging 'catch all' for its user. It can act in conjunction with other panels for other users if desired.

- Can take ANY requirement to:
  - Handle an issue
  - Perform an activity
  - Maintain a constraint
  - Note an annotation
- Deals with these via:
  - Manual (user) activity
  - Internal capabilities
  - External capabilities (*invoke* or *query*)
  - Reroute or delegate to other panels or agents (*escalate*, *pass* or *delegate*)
  - Plan and execute a composite of these capabilities (*expand*)
- Receives reports and messages and, where possible, interprets them to:
  - Understand current status of issues, activities, constraints and annotations
  - Understand current world state, especially status of process products
  - Help control the situation
- Copes with partial knowledge

Three example process panels are shown in the figure below. These panels are from a demonstration of agent systems within a military Coalition context – part of the Coalition Agents eXperiment – CoAX (Allsopp et al. 2001; 2002).



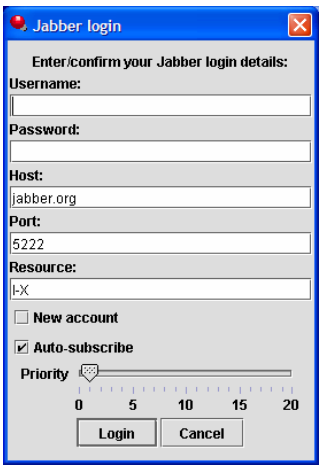
An I-X Process Panel supports a user or collaborative users in selecting and carrying out *processes* and creating or modifying *process products*. Both processes and process products are abstractly considered to be made up of *Nodes* (activities in a process, or parts of a process product), which may contain sub-nodes, thereby constituting a hierarchical description of the process or product. The nodes are related by a set of detailed *Constraints* of various kinds. A set of *Issues* is associated with the processes or process products to represent unsatisfied requirements, problems raised by analysis or critique, and so on.

Processes and process products in I-X are represented in the <I-N-C-A> (Issues – Nodes – Constraints – Annotations) model of synthesised artefacts (Tate, 2000; 2003).

2 Quick Start Guide

This section describes the quickest way to run an I-X Process Panel and begin exploring the capabilities of I-X. This approach uses the *Jabber* instant messaging protocol for communicating with other users' panels; for this you will need to have an account on a Jabber server (step 4 below). However, this is not necessary: a panel can be invoked in a stand-alone mode (but of course, in order to take full advantage of the potential of I-X as a collaborative environment, you will eventually want to communicate with other panels, though not necessarily using Jabber).

1. Since it is written in the Java programming language, to use I-X you will need to have a Java interpreter (the Java Virtual Machine (JVM)) installed on your computer, and its location specified in the value of your path environment variable. (To test whether you have a JVM installed, type "java" at command-line prompt – if it is correctly installed this should return a message about how to use Java.) If you need to install Java, you should download the Java Software Development Kit (SDK) from `http://java.sun.com` - in addition to the JVM this also contains a Java compiler and debugger, useful if you then go on to develop code to extend the I-X system for your own applications. Alternatively, ask your local computing support staff – they should be able to help you install and run a JVM.
2. Download the I-X distribution from `http://i-x.info`. This is packaged as a zip file. Unpack this in a convenient location on your local system, where it will create a single directory with all necessary files.
3. Go to this directory, and then double-click the `ix.exe` icon (MS Windows users) or the `IX-Mac` icon (Apple Mac users) or from a command-line prompt call the `ix` application script (Unix, Linux and Mac OS X users). This should start a basic I-X Process Panel.
4. Along with the panel, a "Jabber login" window should now appear (pictured below):



- If you wish to simply run a stand-alone panel, you should click "Cancel".
- If, on the other hand, you want to communicate with other I-X panels *and you already have a valid Jabber username and password*, you should enter these details, along with the name of the Jabber server on which this name is registered, and then click on "Login". If successful, the name of your panel should be updated to show your username.
- Finally, if you want to communicate *but do not have a username*, then you can attempt to create a new account through this window: you need to type in the name of a Jabber server that allows new accounts to be created (`jabber.org`, for example) and enter a new username and password (taking care to remember your choices!) and tick the "New account" box. Now click on "Login": if the new account has been successfully created, the name of your panel should now be

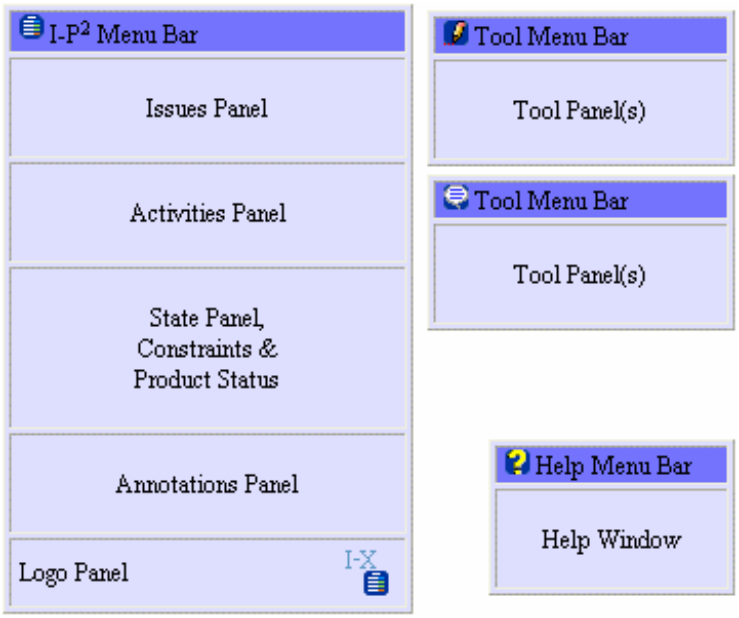
updated with your new username. However, if the attempt fails (perhaps because the username you chose is already in use), you will see an error message, and will be prompted to try different values. (To learn more about Jabber Instant Messaging see `http://www.jabber.org/`.)

5. You are now free to explore the different I-X facilities:

   - From the "New" menu you can create new issues, activities, constraints and annotations, and see these displayed on the panel (and once you've created a plan, save this through the "File" menu).
   - From the "Tools" menu you can start a **Domain Editor**, and try creating your own process models. You can start a simple **HTML Viewer** for accessing directly local I-X HTML resources. You can use the **I-Space** tool to enter the names of other agents and your relationships with them (and see how this affects your options for handling the issues and activities on the main panel). If you are logged on to a Jabber server and you know the Jabber Identification (JID) of another online user (who may be running a conventional Jabber client, rather than an I-X panel) you can try communicating using the **Messenger** tool by typing their JID into the "Recipient" field, adding some text to the "Compose Message" field, and clicking "Send" (if you do not know of any other online users, you can still test your connection by sending a message to `ix-test@jabber.org` - this is an agent that simply responds with an acknowledgment of any message it receives).
   - The options in the "Help" menu provide more information about using I-X.

Soon you'll be ready to learn more about I-X and how to tailor it to your own applications.

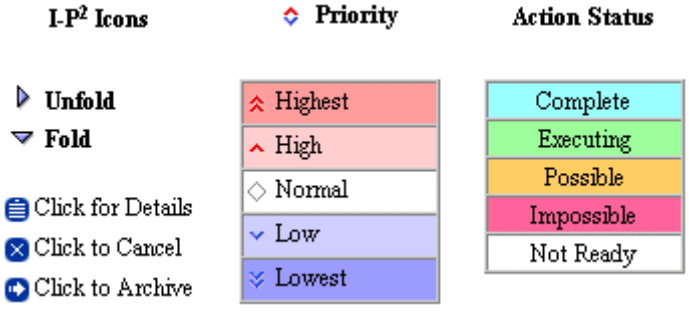
3 Using an I-X Process Panel



An I-X Process Panel (I-P^2) (shown schematically above) contains a number of sub-panels that describe:

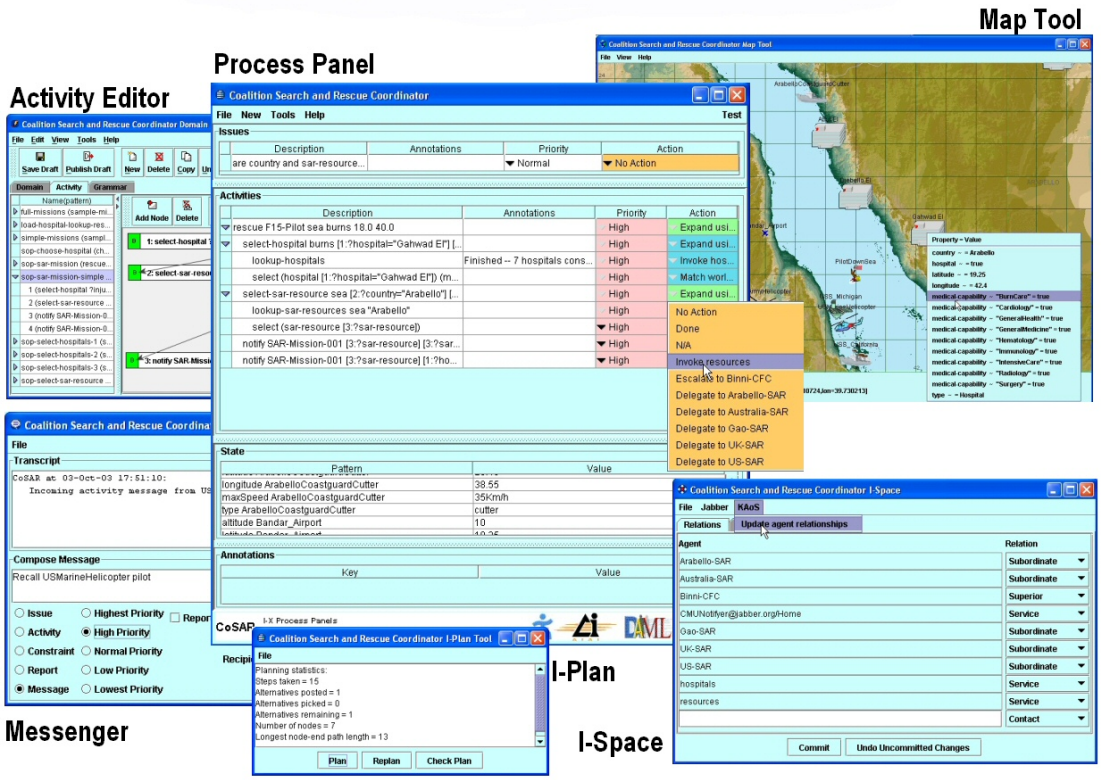
- A set of issues to be handled.
- A set of activities to be performed.
- Current state information reflecting the current set of constraints to be respected. This includes the status of a range of process products being created or manipulated by the processes.
- Annotations in the form of `keyword=value` pairs.

The panel supports its user in handling issues, deciding on a course of action and performing activities, maintaining awareness of the current state, constraints, process products, etc., and making annotations of various kinds.



Entries on panels can be expanded using information provided in the process library used by a panel, or the entries can be passed between panels.

Right click on a line to get a context-sensitive menu that describes operations you can perform on the entry. Where relevant, this includes the ability to pop-up a window with more details of the entry, or to expand or contract the display of some levels of hierarchically specified activities, to send information about the entry to the Messenger tool for sending on to others (perhaps in a modified form), etc.



A *Tools* menu is available to make accessible the following:

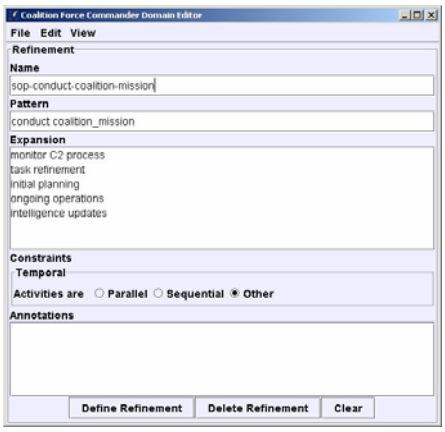
- A domain or process library editor to view, edit or add to the list of process descriptions which may be used to 'expand' entries on the process panel (*I-DE* – see section 4).
- A tool to view and change the relationships of the current panel to others (*I-Space* – see section 5).
- An instant messaging or 'chat' tool to communicate in free format or the encouraged <I-N-C-A> structured forms with other I-X Process Panels and other systems ("intelligent messaging" or "semantically augmented messaging": *Messenger* – see section 6).
- Other tools can be added to I-P^2, for example, a Map Tool, as the application demands.

In addition, an AI planner is available to try to construct alternative ways of performing current activities and achieving objectives (*I-Plan* – see section 7).

4 I-X Domains and the I-X Domain Editor (I-DE)

4.1 Editing Domains: Overview

The process model descriptions used by I-X Process Panels are kept in *domain libraries*. These can be loaded when a panel is started, and can be augmented dynamically by the user of the panel. There are two editing tools available to I-X users to help them create and manage their own domains: the *Simple Editor* and the *I-X Domain Editor* (*I-DE*). These are introduced below. The current editor may be invoked from a Process Panel by selecting *Tools > Domain Editor.*



Simple Editor interface

Basic Editing: The *Simple* Editor

The process panels contain a simple, form-based domain and process editor (shown above). This allows simple task breakdowns to be specified along with a temporal constraint that the sub-steps should be either sequentially ordered or performed in parallel.

A More Sophisticated Editor: *I-DE*

To create richer descriptions of process models and allow more sophisticated management of domain libraries, the user of I-X will want to use I-DE. I-DE is a more powerful domain and process editor, and offers multiple perspectives and views onto models. The use of I-DE will now be described. (I-DE is currently the default domain editor in I-X; appendix A describes how it can be replaced by the Simple Editor for applications where this is adequate. I-DE is also available as a stand-alone application for the maintenance of domain and process libraries.)

4.2 Using the I-DE Domain Editor Tool

The main I-DE Window provides access to most functions via its menu bar and access to the most commonly used ones via its tool bar.

The Menu Bar

The menu bar has 5 standard menus:

1. *File* for closing the Domain Editor and for file access (open/save). All functions here manipulate the domain as a whole, not individual constructs;
2. *Edit* for manipulating the current construct, i.e. the construct that is currently shown in the Domain Editor's panel;
3. *View* for changing which panel is shown in the Domain Editor and - if applicable - for changing which view is shown in that panel – see **Views** below;
4. *Tools* for additional support like consistency checks etc.;

5. *Help* for access to this manual, other help, and information about the application.

The Tool Bar

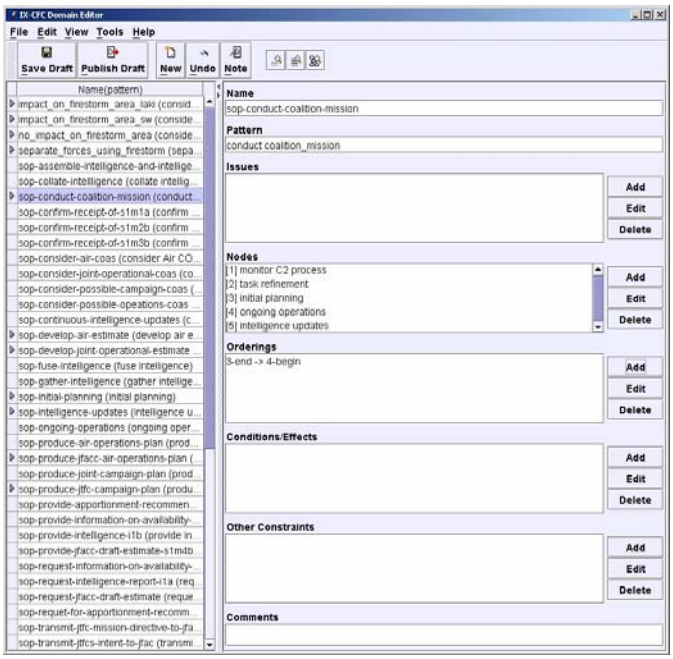
The tool bar provides access to the most commonly used functions via buttons. All these functions are also available via the menu bar (with the image alongside the menu item being that used on the corresponding toolbar button). The toolbar can be switched on and off via the *Toolbar* check box in the *View* menu. Moving the mouse over a toolbar button will, after a while, display a 'tool tip text' that gives a brief explanation of the button's function.

Views

The window can display in one of three styles: *single*, *tabbed*, and *card* style. The style can be changed via the options through the *View* > *Panel Style* menu. In addition there are several other *View* options for changing the appearance of I-DE.

I-DE can be run in two modes (*View* > *Editor Mode*):

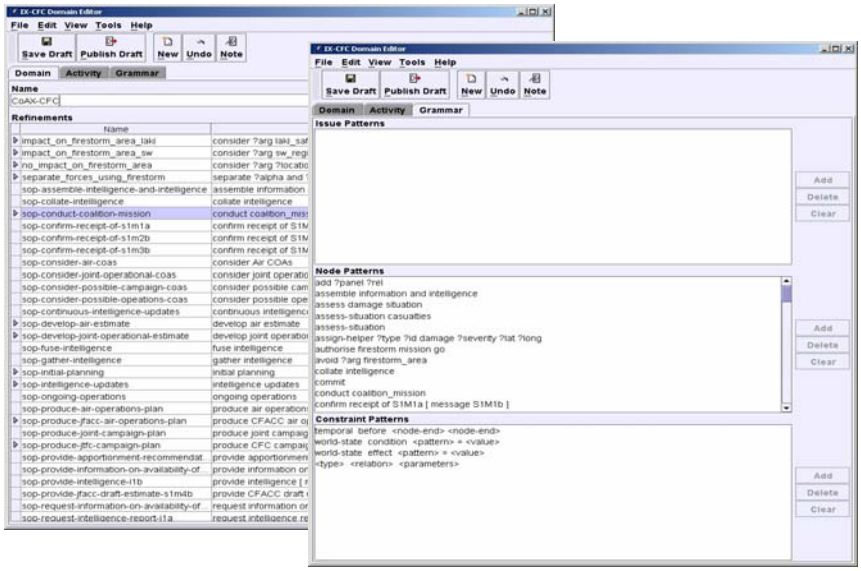
- *Simple mode*. This mode is for simple editing only, as it lacks the more sophisticated editing functions. The simple mode restricts the user to the *single* display style and *minimal* view (described below).
- *Advanced mode*. In Advanced mode, the user can, amongst other things, place more complex ordering, precondition and effect constraints on the steps of a process model.



I-DE: Advanced mode Activity panel

The main window of the Domain Editor contains several editor sub-panels for editing different aspects (or constructs) of the domain (*View* > *Windows*, or choose the appropriate tab or card, depending on selected *Panel Style*):

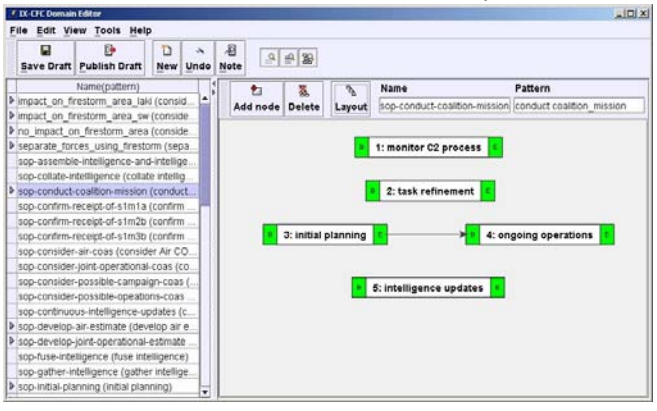
- *Domain* panel, used to edit global information about the domain itself (e.g. the domain name).
- *Activity* panel, used to edit information about activities and how they break down into sub-activities (refinement).
- *Grammar* panel, which currently only shows the patterns that are in use in the domain.



I-DE: Domain (left) and Grammar (right) panels

An editor panel may itself have different 'views' that are used to display and edit the panel's constructs. The activity panel has three such views:

- *Minimal view*: a simplified version of the activity and its refinement. The main simplification is that no constraints are shown. (This option provides an interface that is intended to be visually similar to that of the Simple Editor.)
- *Comprehensive view*: a view that can display and edit all of an activity's specification, allowing the user to specify more complex temporal and world-state (condition/effect) constraints. Other constraints, such as spatial constraints or constraints on resources, can also be specified using this view.
- *Graphical view*: a graphical view that uses nodes and arcs to show an activity's sub-activities and the temporal relationships between them. (Other constraints and specifications cannot be viewed or edited using this view.) The user can add, manipulate and delete the nodes and arcs to develop the task breakdown structures.



I-DE: Graphical view

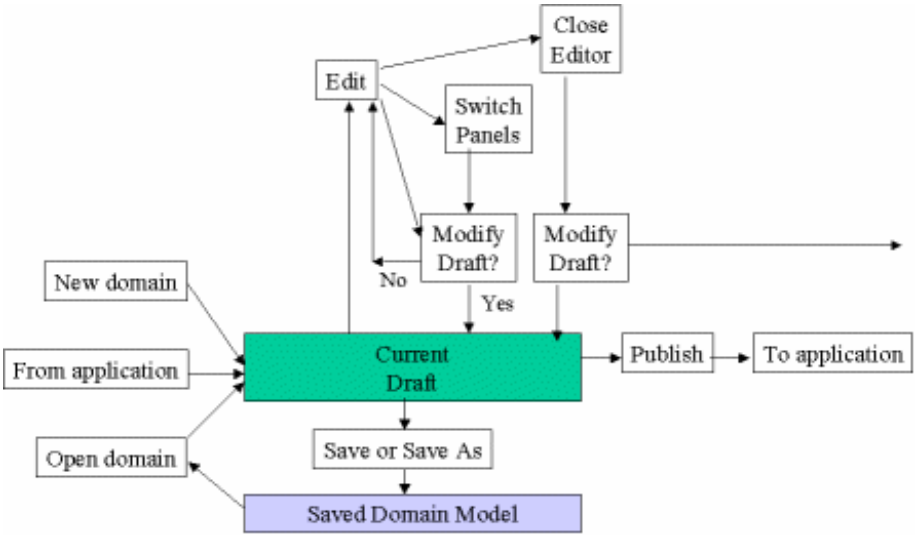
4.3 Building and Maintaining Libraries with I-DE

For a given application, the user of an I-X Process Panel will have access to the process model definitions in a particular domain library to support him or her in performing the task in hand. The richness of the definitions in this domain will depend in part on the nature of the

task and the extent to which it can be codified – and the extent to which it *has* been codified. The codification and subsequent maintenance of process models is, in effect, a knowledge engineering task, and the role of I-DE is to support this exercise.

Realising that complex applications may demand complex process models, and the task of modelling these might itself be complex and involve many revisions and alterations, I-DE (unlike the Simple Editor) imposes a particular regime on the saving and updating of domains. The twin aims of this regime are, on the one hand, to try to ensure that the integrity of the base domain (whose models may concurrently be in use in a Process Panel, for instance) is maintained by not admitting partial or provisional modifications, while, on the other hand, allowing the modeller to regularly save changes to provide 'roll-back' points in the event of system crashes or second thoughts.

Accordingly, when the user invokes I-DE, a *draft* copy of the current *base* domain is 'checked out' and presented for modification. Any edits saved using I-DE are saved only to this draft domain, with the base domain remaining unaltered, *until the edits are explicitly 'published' by the I-DE user.* In this manner users can develop a domain by creating and modifying processes, and, when they are satisfied that their alterations constitute a coherent body worthy of addition to the base domain, can then confirm these changes by publishing them.



Saving and Reverting

To implement this regime, there are 3 levels of saving:

1. When a construct has been edited, initially these changes may be made only to the interface itself, and not to the underlying draft construct in memory. In order to transfer changes from the interface into the construct in the draft domain, the user has to modify the draft, i.e. *Edit > Note Domain/Activity/Grammar into Draft*. In addition, at relevant times the system will automatically update the draft for the user, so as to keep the draft domain in step with what is shown in the interface. If a user has edited a construct and without noting changes to the draft then decides to switch constructs, views, or panels, or else decides to save or publish the draft domain, the system will first note the changes.
2. Modifying the draft (noting changes) does not save to file, so the next level of saving is to save the draft domain to file (via toolbar button or *File > Save draft to file*). As with all editing applications, it is recommended that this be done frequently to ensure that work is not lost. Saving the draft domain to file will write the whole domain with all its constructs into a file in XML format. This can later be loaded back into I-DE for further editing, or it can be accessed by other applications.
3. The underlying base library is not changed by either of the above levels (noting to draft or saving the draft to file). *The only way to update the base library is to publish*

the draft domain via the toolbar button or the File > Publish Draft *menu command*. When this happens, the changes are made to the base domain and these changes will be seen by any applications that have registered as listeners to this domain. Note that publishing is always done for a whole domain, and not for individual constructs. Note also that publishing a domain will *not* save it to file, but the same effect can be achieved by saving the draft domain to file immediately before or after publishing. At that point the draft domain and the public domain are represented by the same XML structures. (It is a good idea to publish from time to time even if I-DE is running in stand-alone mode because it will make the editor more efficient.)

It is worth noting that the *Domain* panel, i.e. the panel that is responsible for editing global details about the domain like its name, only considers domain details as part of its editing remit, not the constructs within the domain.

While there is no function that undoes individual editing steps, the following functions are available, corresponding to the 3 levels of saving:

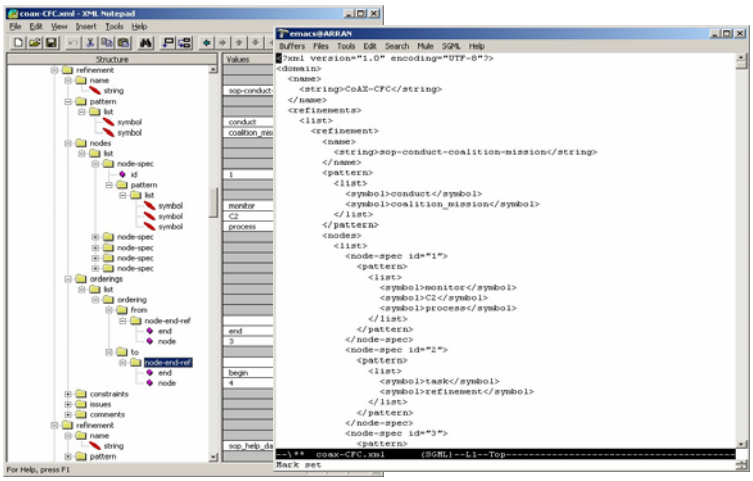
Corresponding to the different levels of editing and saving, there are different levels of undoing changes:

1. *Edit* > *Undo edit*: undoes an editing step. Undo operations can themselves be 'undone' by the *Edit* > *Redo* command.
2. *File* > *Open as draft domain…*: revert the whole domain to the last time it was saved to file by re-loading the file. *This change cannot be undone* – if unsure, first save the current draft domain to a different file.
3. *Edit* > *Revert to Published*: revert a construct to its state in the most recent published version, that is undo all changes to this construct since the domain was last published. Once again, *this change cannot be undone* – if unsure, first save the current draft domain to a different file.

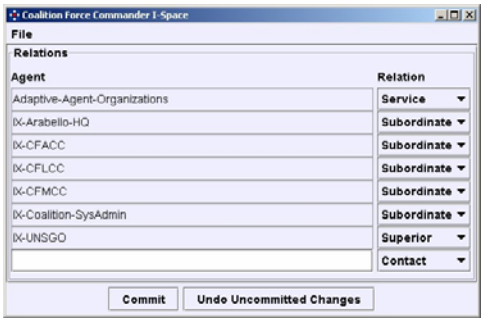
There is a fourth 'revert' function for convenience: *File* > *Discard domain changes* which reverts the *whole* domain (and not just the current construct) to its state in the latest published version, that is, it undoes all changes to all constructs since the domain was last published.

4.4 Use of XML and Text Editors

There is a further option for users without access to either the Simple Editor or I-DE, or who wish to make simple amendments to their models: the fact that the domains created by these tools are saved as XML format files means they can be modified using a dedicated XML editing tool - such as the freely available Microsoft XML Notepad (see `http://msdn.microsoft.com/xml/notepad/intro.asp`) - or even a standard text editor.



5 Using the I-Space Tool



The I-Space tool allows for the management of the organisational relationships of the current panel (referred to as "me") to other panels, agents and external services. New agent names can be typed into the text field at the bottom. Existing agents or panels can have their relationships altered. The *Commit* button is used to inform the process panel of any addition or changes to the set of relationships with existing entries. You can undo any changes made to the I-Space table that have not been committed already.

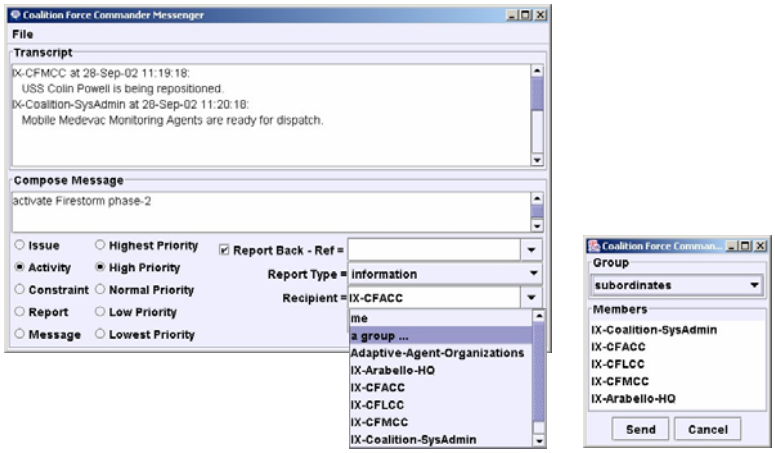
The relationships implicitly construct *Action* menu items for issues and activities on the parent Process Panel. The relationships and the corresponding *Action* menu items are as follows:

Relationship	***Action* Menu Item**
Superior	Escalate to (with report back)
Peer	Pass to (with report back)
Subordinate	Delegate to (with report back)
Service	Invoke (with report back)
Contact	None
None	None

Providing a description of the verbs associated with any agent (via the *Capabilities* tab) can be used to selectively show the agent in the Action menu only for the specified verbs. If no verb association is provided, it is assumed that all Superiors, Peers and Subordinates can handle any item (i.e., described using any verb). It is expected that an external-capabilities description is given for a Service or it will not appear on the menu at all.

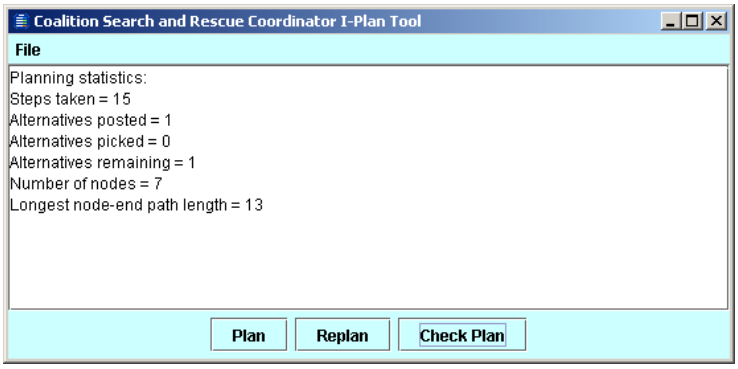
For some communications strategies, extra menu options are added into the I-Space tool to provide services and facilities relevant to the specific nature of the communications strategies in use.

6 Using the I-X Messenger Tool



The I-X Messenger tool is used to compose and send messages to other panels and agents. Messages can be designated issues, activities (these with corresponding priorities) or constraints, or, less formally, as simple 'chat' messages. The tool also shows any chat messages received from other agents (in the Transcript window). You can send messages to your own panel ("me") and there is a simple group sending facility (which will be expanded in future releases).

7 I-Plan

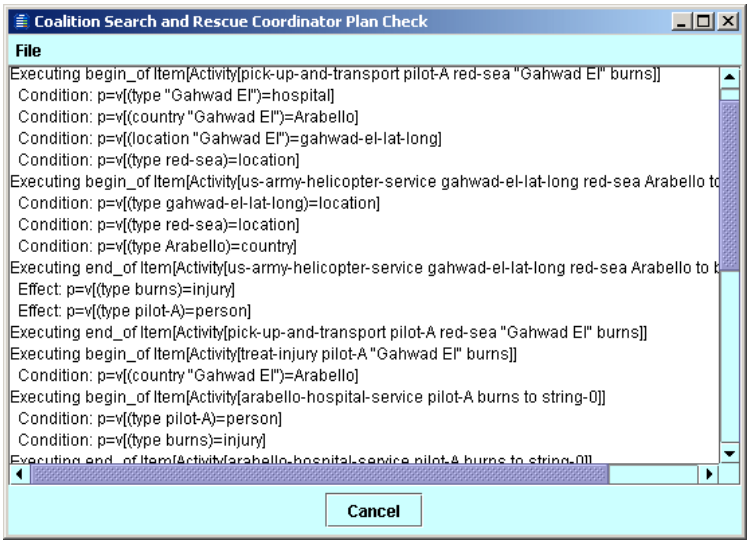


The facilities available in the I-X Process Panels provide context sensitive options for the handing of issues (such as the achievement of stated objectives), the performance of activities, and the satisfaction of constraints. A simple AI Planner (I-Plan) is available as a tool to propose alternative ways in which activities on the panel can be expanded.

I-Plan can perform hierarchical partial-order composition of plans from a library of plan schemas or Standard Operating Procedures. This library can be augmented during planning either using a simple 'activity details' interface to add in specific ways to expand a given activity (intended for users familiar with the application domain but not AI planning techniques) or with a more comprehensive graphical domain editor (I-DE – see section 4). Grammars and lexicons for the activities in the domain and the objects manipulated by them are built automatically during domain editing to assist the user.

I-Plan can check that conditions are satisfied or introduce activities to achieve them, it can select consistent bindings for a set of variables in the current plan, and can check or impose temporal constraints between activities. I-Plan cannot currently reason about resources, spatial constraints, or metric time constraints.

Future developments of I-Plan will provide more assistance with a 'How do I do this?' option under the *Action* menu which will be able to account for other concurrent items on the panel, and account for mutual satisfaction of open variables, unsatisfied world state conditions and other constraints. I-Plan will also be extended to provide a plan repair capability should activities fail during execution, or the environment dynamically change in unforeseen ways.



8 Creating your own I-X Application

Other tutorial and sample material is available. You can find pointers to these in the "doc" directory of the system. A file "doc/index.html" gives an overview of what is available.

A single Process Panel or a small cluster of panels in Superior, Peer or Subordinate relationships to one another can be quickly adapted to a new application. We will later support more dynamic and adaptable combinations of multiple panels in more complex organisational structures (which we call *I-Spaces*), but much of the support required for this is not yet sufficiently generic to provide in an easily altered form.

8.1 Configuring Process Panels

An example I-P^2 application is provided in the `apps\isample` directory, which can be copied and adapted as follows:

- Copy the whole `isample` directory to become a new directory with a name of your choice (e.g. `apps\myapp`).
- In the sub-directory `config` alter the `.props` properties file names and contents of these files to tailor display names and labels used on the Process Panels.
- You can modify the ways in which the panel *Actions* are set up using *I-Space* relationships such as Superior, Peer and Subordinate.
- In the directory `images` add in any logo or logos for panels as you wish. Replacing the logo `images\isample-logo.gif` will mean the default logo is amended without further changes.
- Tailoring a panel to a new application usually involves providing a suitable 'domain model' that describes ways in which activities can be refined into more detailed sub-activities for that application. Domain models can be represented in XML format (although a Lisp-oriented format is also available) and for I-Sample Process Panels are in the `domain-library` directory. Domain editors are provided to create or amend domain models (see section 4), and domain models can be augmented while a Process Panel is running.
- You can add appropriate *Test* menu entries (see below).

A wide range of parameters can be specified to simply customise a range of things about each panel. A properties file for a panel can specify most of these and can be set using, e.g.,

```
ix.ip2.Ip2 -load config/isample-supervisor.props
```

For example, the file `config/isample-supervisor.props` contains such things as:

```
symbol-name=Supervisor
display-name=Supervisor I-X Process Panel
logo-line-1=Supervisor I-X Process Panel
logo-line-2=Based on I-X Technology
logo-image=images/isample-logo.gif
domain=domain-library/isample-supervisor.xml
subordinates=Operator
```

You can also set one property individually when the Process Panel program is started using a command-line argument, such as

```
"-display-name=myapp <panel-identifier>"
```

The domain model, including process descriptions available to the panel, can be preloaded from a domain library file (e.g. as in the case above which loads the `isample-supervisor.xml` file describing sample processes that the panel is then made aware of and can use to 'expand' entries placed onto the panel).

The Test Menu

It can be convenient to provide some example issues, activities or other entries that can be added to a panel, which could have come from other systems or panels. It can also be convenient to provide messages that could be sent to other panels and agents. This allows simple demonstrations and testing to occur. The contents of the *Test* menu can be set using an XML file describing the entries, and informing the panel about this file using the

```
-test-menu=<pathname>
```

parameter – see Appendix A for details of all parameters. An example test menu file follows. (Note: using "me" for as the value of the `to-name` attribute means the message is sent to the current panel rather than externally. The value of `menu-text` is what actually appears as an entry in the *Test* menu. A `$to` item in the `menu-text` string (if present) is substituted by the `to-name` of the panel or agent for which the message is intended.)

```
<?xml version="1.0" encoding="UTF-8"?>
<list>

   <test-item
           menu-text="Send $to a request for transport"
           to-name="Supervisor">
      <contents>
         <activity priority="high"
                   report-back="yes">
            <pattern>
               <list>
                  <symbol>transport_by_helicopter</symbol>
                  <item-var>?wounded</item-var>
                  <symbol>field_hospital_a</symbol>
               </list>
            </pattern>
         </activity>
      </contents>
    </test-item>

    ...

</list>
```

More details of setting up a test menu are provided in Appendix C.

8.2 Advanced Tailoring

You can further tailor an I-X Process Panel to a specific application by renaming and amending the I-Sample Process Panel code as follows:

- You can rename the `java\isample` directory to be `java\myapp` and, in that directory, rename `Isample.java` to be `Myapp.java` or whatever you wish. Delete the compiled class files included there.
- Edit this renamed file to change the package name from `isample` to `myapp`.
- Change the class name `Isample` to `Myapp` wherever it occurs.
- Change any strings that refer to I-Sample to My-App as you wish.
- You can provide a customised 'state' viewer for panels. A description of how to do this is in the I-X Developer Guide.

- Recompile the `Myapp.java` code with the compile script provided.
- In the directory `scripts\win` (and `unix`) alter the script names and script contents as necessary to refer to the new name rather than the basic `ix.ip2.Ip2` class or the custom `isample.Isample` class.

8.3 Issue and Activity Handlers

The I-X Process Panels have a number of handlers that can process issues and activities in specific ways:

- ***Escalate*, *Pass* and *Delegate*:** One type of handling is to use the options in the *Action* menu reroute the issue or activity to other users or panels. These handlers are defined using information supplied in the *I-Space* description of Superiors, Peers and Subordinates for the Process Panel (this may be done using the *I-Space* tool (see section 5), via the command line or using the `.props` file for the panel).
- **Invoke:** External agents defined as being a Service will appear on the *Action* as possible handlers when their capability-verb that matches the verb of the current issue/activity. (Services are specified using the *Capabilities* tab of the *I-Space* tool or specifying their `external-capabilities` in the panel `.props` file.)
- **Connect:** Activities of the form `connect <agent-name> <relationship-type>` (e.g. `connect john peer`) have a connect handler that sets the appropriate entries in *I-Space* automatically. See section 5 for details of the available relationship types.
- **Others:** may be available. You can check this via a panel's menus using *Help* > *About Syntax*.

So, in order to impose some organisational structure on the domain and state the existence of Service agents, the application-builder may wish to declare the appropriate relationships within a `.props` file, and distribute this to all panel users within the environment. In addition, advanced I-X users can code their own handlers for specific applications.

8.4 Creating your own I-X Domain/Process Library

Each I-X Process Panel can make use of a domain model or process library which describes ways in which issues can be handled or high level activities can be broken down into more detailed activities which may then be performed. A panel can operate without such process descriptions, but becomes more useful and helpful if it has such knowledge. A process library can be loaded when a panel is started up, and additional process descriptions can be provided while it is running: indeed, they can be saved at any stage to amend the stored version for later preloading.

You can create the process descriptions with the I-X Domain Editor provided (see section 4). It can be run on its own or can be called from within a Process Panel from the Tools menu. Since the process descriptions can be saved in a simple XML format, it is also possible to use any XML editor to change the descriptions if you wish. The format of the XML is described separately.

Example domain models and refinements can be found in the `apps\isample\domain-library` directory. Variables begin with `?` and can be used anywhere. Unbound variables appear in the process panel and can be bound by the user of the panel (and in later versions by external query capabilities). The representation used is based on the <I-N-OVA> constraint representation of activity (Tate, 1996).

9 Further Tailoring

An I-X Developer Guide is available with details on more ways to tailor I-X Process Panels and systems. Note that this usually involves programming extensions in Java.

9.1 Communications Strategy

I-X Process Panels can also be used with any of a number of "Communications Strategies". Example strategies are provided for the DARPA CoABS Grid (`grid`), Institute of Human and Machine Cognition (IHMC) KAoS (`kaos`) (Urzok et. al., 2004), Jabber XML framework (`jabber`), and the UK EPSRC-sponsored Advanced Knowledge Technologies AKTBus (`akt`). Also provided is an adaptor for a simple direct link between panels, possibly supported by a simple name server (referred to as the 'simple' or 'xml' communications strategy). Writing a suitable Communications Strategy can provide other message transport routes. More details are available in the I-X Developer Guide.

9.2 Custom World State Viewer and I-X Map Tool

It is possible to replace the simple table view used for the current world state.

`-state-viewer-class=StateViewTable` (default)

Viewers that show the state information clustered into the various objects or process products being handled, and giving their attributes and values in a convenient form can be provided. Graphical images of the process products can be added where required. This could include map-based data to show the position of the objects.

One example of a custom world state viewer is a Map Tool based on BBN's OpenMap (BBN, 2003). This is provided as an add-on that may be added into any I-X application that requires it.

10 References

Allsopp, D., Beautement, P., Bradshaw, J.M., Durfee, E.H., Kirton, M., Knoblock, C.A., Suri, N., Tate, A. and Thompson, C.W. (2002) "Coalition Agents Experiment: Multi-Agent Co-operation in an International Coalition Setting", Special Issue on Knowledge Systems for Coalition Operations (KSCO), IEEE Intelligent Systems, June 2002.

BBN (2003) OpenMap, http://openmap.bbn.com

Fraser, J. and Tate, A. (1995) "The Enterprise Tool Set -- An Open Enterprise Architecture", Proceedings of the Workshop on Intelligent Manufacturing Systems, International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Canada, August 1995.

Potter, S., Tate, A. and Dalton, J. (2003) I-X: Task Support on the Semantic Web, Poster Abstract, Second International Semantic Web Conference (ISWC-2003), Sanibel Island, Florida, October 2003.

Stader J., Moore J., Chung P., McBriar I., Ravinranathan M., Macintosh A.. (2000) "Applying Intelligent Workflow Management in the Chemicals Industries"; in “The Workflow Handbook 2001”, L. Fisher (ed), Published in association with the Workflow Management Coalition (WfMC), pp 161-181, Oct 2000.

Tate, A. (1996) "The <I-N-OVA> Constraint Model of Plans", Proceedings of the Third International Conference on Artificial Intelligence Planning Systems, (ed. Drabble, B.), pp. 221-228, Edinburgh, UK, May 1996, AAAI Press.

Tate, A. (1998) “Roots of SPAR”, in "Special Issue on Ontologies", Knowledge Engineering Review, Vol.13 (1), March 1998, Cambridge University Press.

Tate, A. (2000) “<I-N-OVA> and <I-N-CA> - Representing Plans and other Synthesized Artifacts as a Set of Constraints”, AAAI-2000 Workshop on Representational Issues for Real-World Planning Systems, at the National Conference of the American Association of Artificial Intelligence (AAAI-2000), Austin, Texas, USA, August 2000.

Tate, A. (2003) <I-N-C-A>: an Ontology for Mixed-initiative Synthesis Tasks, Proceedings of the Workshop on Mixed-Initiative Intelligent Systems (MIIS) at the International Joint Conference on Artificial Intelligence (IJCAI-03), Acapulco, Mexico, August 2003.

Tate, A., Dalton, J. and Levine, J. (1998) "Generation of Multiple Qualitatively Different Plan Options", Fourth International Conference on AI Planning Systems (AIPS-98), Pittsburgh, PA, USA, June 1998.

Tate, A., Dalton, J. and Levine, J. (2000) “O-Plan: a Web-based AI Planning Agent”, AAAI-2000 Intelligent Systems Demonstrator, in Proceedings of the National Conference of the American Association of Artificial Intelligence (AAAI-2000), Austin, Texas, USA, August 2000.

Tate, A., Levine, J., Dalton, J. and Nixon, A. (2002) “Task Achieving Agents on the World Wide Web”, in “Creating the Semantic Web”, Fensel, D., Hendler, J., Liebermann, H. and Wahlster, W. (eds.), MIT Press, 2001.

Uszok, A., Bradshaw, J.M., Jeffers, R., Johnson, M., Tate, A., Dalton, J. and Aitken, S. (2004) Policy and Contract Management for Semantic Web Services, AAAI Spring Symposium, Stanford University, California, USA, March 2004.

Uschold, M., King, M., Moralee, S. and Zorgios, Y. (1998) "The Enterprise Ontology", in "Special Issue on Ontologies", Knowledge Engineering Review, Vol.13(1), March, 1998, Cambridge University Press.

Appendix A: I-P^2 Parameters

IXAgent

`-debug=Boolean`

Set to true for more detailed diagnostics in the Java console window.

`-ipc-name=name`

ipc-name is essentially a synonym for symbol-name

`-symbol-name=text`

Note that symbol-name, by default, also becomes the ipc-name. If not provided, the default symbol-name is set to "<user-name>@<machine-name>".

`-display-name=text`

Note that display-name is set to "<symbol-name> Process Panel" if not explicitly set.

I-P2

`-classic=boolean`

Use alternative (simpler) interface for table views and other user interface elements.

`-descriptions-column-width=integer`
`-annotations-column-width=integer`

For classic viewers only.

IP2 Visual Appearance

`-metal-theme-secondary3=colour`

note `secondary3` not `secondary-3` (may alter in future)

`-font-increment=integer`

The relative font size for text, buttons and labels in I-X Process Panels: e.g., 2, 4 or –2. Odd numbers (e.g. 1 or 3) may not have bold fonts installed on all systems).

`-logo-line-1=text`
`-logo-line-2=text`
`-logo-image=pathname` (URL syntax)
`-frame-size=WIDTHxHEIGHT`

The initial width and height of the process panel in the form WIDTHxHEIGHT (e.g. 800x400)

I-Space

`-superiors=namelist`
`-subordinates=namelist`
`-peers=namelist`
`-contacts=namelist`

`-external-capabilities=name:verb,...`

Note that namelist is a list of names of other process panels or external agent resources. A namelist is comma separated, and the list cannot contain spaces.

`external-capabilities` specifies the verbs associated with any panel name or external agent name. If the panel or agent is not in a defined relationship, then it is considered to be a 'resource' that can be invoked through the *Action* menu when the specified verbs are encountered. If it is already in a defined relationship then it is treated as a restriction specification, so the relevant *Action* menu entries (i.e., *escalate*, *pass* or *delegate*) will only appear for those specific verbs – rather than for any pattern verb.

`-use-long-ids=boolean` (default `true`)
`-use-hash-ids=boolean` (default `false`)
`-allow-random-device=boolean` (default `false`)
`-send-received-reports=boolean` (default `true`)

Specify whether or not to send a “received” report when a panel receives an item. Can be turned off if this causes problems for the recipient (which may not be an I-X Process Panel).

IP2 Domain Library

`-library-directory=pathname` (URL syntax)

[for backwards compatibility with versions 3.0 and earlier this can also be called domain-library]

A directory can be specified which will be used as the location for saving or loading files via a file browser. By default this will be the current directory.

`-domain=resource(s)` (filename or URL syntax, comma separated)

One or more domain files can be loaded when a panel starts up. These can be in OWL (`.owl`), RDF (`.rdf`), XML (`.xml`) or Lisp (`.lsp`) formats. A resource can be a URL, a filename for an existing file, or the name of a resource that can be obtained from the agent’s classpath by a ClassLoader.

Initial Panel Contents (Initial Plan)

`-plan=resource(s)` (filename or URL syntax, comma separated)

One or more initial files can be provided to set up the panel contents and state. Panel contents can be given in OWL (`.owl`), RDF (`.rdf`), XML (`.xml`) or Lisp (`.lsp`) formats (typically saved from a previous run of the panel). Also files where each line is of the form `Attribute.ObjectName = Value` can be provided (with file suffix `.init`).

`-plan-state-to-save=namelist or * for all` (default is none)

State information is normally not saved into a saved plan file. You can save all state information by specifying `–plan-state-to-save=*` or else can select the initial keywords of those state patterns to save (a comma separated list).

Test Menu

`-test-menu=resource(s)` (filename or URL syntax, comma separated)

One or more XML files can be provided to set the Test menu entries that appear in the top right corner of a process panel. This can be convenient for testing and demonstrations.

IPC

`-ipc=strategyName`

`-ipc=class`

strategyName can be `simple` (default) or `xml` using built-in support. With suitable communications strategy add-ons other strategies can be specified such as: `grid`, `kaos`, `jabber` or `akt`.

See the javadoc for `IPC.getCommunicationStrategy(String strategyName)`.

`-enqueue-incoming-messages=boolean` (default `false`)

`-rename-variables-in-input=boolean` (default `true`)

Tells agent to rename any `?name` variables in patterns in incoming issues and activities so that they do not conflict with any existing variables in the receiving agent.

Default/Simple and XML Communication Strategies

`-port=number`

Tells the agent to use a specific port number rather than to ask the underlying operating system to allocate a free one. This is especially useful in environments with a firewall.

`-host=hostname`

Used to tell the agent what to call the machine it is running on when the default name will be incorrect. The default is the name returned by `InetAddress.getLocalHost().getHostName()`.

`-run-name-server`

Tells the agent to run a name-server.

`-name-server`

`-name-server=servername:port`

`-no name-server`

Tells the agent whether to use a name-server to look up the addresses of other agents, and if so, what host and port to connect to. The name-server `servername:port` defaults to localhost:5555.

Jabber Communication Strategy

`-jabber-server=hostname` (*eg.* `jabber.org` *or* `jabber.aiai.ed.ac.uk`)

`-jabber-username=username`

`-jabber-port=5222`

`-jabber-password=password`

`-jabber-resource=resource` (`I-X` by default)

`-jabber-priority=integer` (positive integer. 1 by default)

The `jabber-port` should not be altered lightly. Likewise, you should usually leave the `jabber-resource` at its default value. The assumption is made that all other panels with which this panel will communicate will share the same resource name. Hence, the resource name is used for two purposes:

- to distinguish I-X panel jabber clients from non-I-X panel jabber clients, and;
- to distinguish I-X panel clients belonging to a particular I-X 'cluster' from other I-X panels.

Caution should be applied when setting this parameter to anything other than its default value.

`-jabber-presence=keyword (e.g. Online)`

`-jabber-allow-queuing=boolean (default false)`

Can be set to `true` to allow asynchronous communications between panels that are not on-line at the same time (queuing is provided by jabber servers). The default mode (`false`) will indicate a communications failure if the target recipient resource is not on-line or, if no resource is specified, the target user has no on-line resources at all. (In the latter case, if no resource is specified and an I-X Process Panel is identified among the target user's on-line jabber clients, then this Panel will be the preferred destination for the message.)

`-jabber-autosubscribe=boolean (default true)`

If `jabber-autosubscribe` is set to `true`, any incoming request from another jabber user for subscription to the presence of the current I-X user will be automatically accepted, and a corresponding subscription request generated and sent in response. If `jabber-autosubscribe` is set to `false` then any incoming subscription request is ignored (and will be re-submitted the next time the current user starts a jabber client).

Parameterised Communication Strategies

Some communication strategies take other strategies as parameters. The syntax is

`<strategy> = <name> | <name>:<strategy>,<strategy>,...`

A parameterised strategy typically provides a 'wrapper' around 'inner' strategies specified by its parameters. The ones provided are:

`tracing:strategy`

Prints an XML description of the message contents when a message is sent or received by the specified inner strategy.

`separate:stategy`

Creates a dummy I-X agent for the inner strategy to use. This protects the main I-X agent from some operations a communications strategy might perform, such as changing the agent's symbol-name. `separate` is often used together with `dispatching`:

`dispatching:strategy,strategy,...`

Allows different inner strategies to be used for communicating with different agents. One of the arguments must be prefixed by `default:`. Strategies are assigned to agents as follows.

If a message is received from an agent A via strategy S, then A gets S as its strategy; otherwise a tab in the I-Space tool can be used to assign strategies explicitly.

Examples:

```
tracing:xml
dispatching:jabber,default:xml,tracing:xml
dispatching:jabber,xml,default:tracing:xml
dispatching:separate:jabber,default:kaos
```

The notation for communications strategy combinations is fairly general, but not quite to the point of allowing parentheses for grouping. For example,

```
dispatching:tracing:jabber,default:tracing:xml
```

works, but

```
tracing:dispatching:jabber,default:xml
```

does not, because the tracing strategy gets parameters `jabber` and `default:xml`.

Some strategy combinations won't work semantically despite being valid syntax. For example, you can't do

```
dispatching:xml,simple
```

because they both expect a name-server on the same port (but with different syntaxes for its messages).

Domain Editors, Item Viewers and Map Tool

```
-domain-editor-class=classname
```

Possible values are currently:

`ix.iview.DomainEditor` (default)
`ix.iview.SimpleDomainEditor`

Select domain editor to use.

```
-issue-viewer-class=class
-activity-viewer-class=class
-state-viewer-class=class
-annotation-viewer-class=class
```

These parameters allow the default Issue, Activity, State or Annotation Viewer to be replaced with a custom viewer class. For example the default State Viewer (StateViewTable) could be replaced with a custom class such as the Map Tool (StateViewMap).

`-map-properties=pathname` (URL syntax)
`-map-default-icon=pathname` (URL syntax)
`-map-object-icons=pathname` (URL syntax)
`-map-type-icons=pathname` (URL syntax)

The `map-properties` parameter (this parameter is required) is a Java properties file used to configure the BBN OpenMap tool with the map to use, the latitude/longitude it originates at and various other properties. The `map-initial-state` parameter allows for an initial world state to be specified which is loaded on start up of I-P2. It is equivalent to adding world state effects entries into the panel. The `map-type-icons` and `map-object-icons` parameters allow directories to be specified which contain icons to associate with various types of specific named objects that can be displayed. A default icon is used if no specific named or type icon is available. This default can itself be specified with the `map-default-icon` parameter.

Issue and Activity Handlers

It is possible to add new issue and activity handlers at run-time. As with plans and domains, there are 2 ways to do it: either via a command-line argument / `.props` file entry:

```
-additional-handers=className,className,...
```

Or through an activity with the pattern:

```
add-handlers className className
```

The reason for the language difference (`additional-handlers` vs. `add-handlers`) is that the former is a declaration, whereas the latter is an imperative. Here a class name can be a full Java class name (including package), but classes in the packages that are expected to supply handlers can be given in the 'dash-syntax' style used in the XML, e.g., `test-handler` rather than, say, `ix.test.TestHandler`.

One use for this is during testing. Some handlers will be only for testing purposes, and now they can be used without having to define an Ip2 subclass. But it also allows domain-specific handlers to be added for demos etc.

General Notes

- Filenames and pathnames are relative to the current directory when an application is run. This is usually the root directory for an I-X application using default start up scripts (i.e., `<I-X-base-directory>/apps/<app-name>/`).
- When providing command line arguments, the "-" is not part of the parameter name. It is just command line syntax.
- `-load` is not a parameter. It is syntax that says to load `name=value` lines from a file. More than one `-load` may be specified.
- `-no` and `-not` can negate the subsequent parameter (which is written without the initial "-"). It is equivalent to giving the parameter the value `false` but can be used in cases where it would seem odd to explicitly state `=false`.
- Resources for parameters like `domain`, `plan` and `test-menu` can be specified by URLs, file names, or names of resources accessible via a class loader (class-loaders use forward slashes for this).

Appendix B: <I-N-C-A> XML Message Formats

An I-X Process Panel can receive XML format messages from other agents or systems to give it issues to address, activities to perform and reports to note. A Test agent (called I-Test) is provided to give a simple way to try this out. The format of these messages is described separately.

Appendix C: Test Menu Setup

Test-menu Files

To use a test-menu file, have a command-line argument or `.props` file entry `test-menu=filename`, e.g., `ip2 -test-menu=somedir/test-sequences.xml`

The file contains a list; each element describes a single entry on the test menu. In outline, the file therefore looks like this:

```
<?xml version="1.0" encoding="UTF-8"?>
<list>
     ...
</list>
```

Three types of entry are allowed:

TEST-ITEM ::=

```
<test-item
     delay-before="INT">
     <menu-text>STRING</menu-text>
     <to-name>STRING</to-name>
     <contents>SENDABLE</contents>
</test-item>
```

TEST-SEQUENCE ::=

```
<test-sequence>
     <menu-text>STRING</menu-text>
     <test-items>
          <list>TEST-ITEM*</list>
     </test-items>
</test-sequence>
```

TEST-SEQUENCE-GENERATOR ::=

```
<test-sequence-generator
     initial-delay="INT"
     delay-between="INT">
     <menu-text>STRING</menu-text>
     <template>TEST-ITEM</template>
     [ <to-names>
          <list>...</list>
     </to-names> |
     <content-strings>
          <list>...</list>
     </content-strings> ]
</test-sequence-generator>
```

A **SENDABLE** is an issue, activity, constraint, report or chat-message.

Note that STRING-valued fields (such as menu-text) may be written as attributes instead of as elements if the string is sufficiently simple.

Test items, sequences, and sequence-generators do not have to come from files. They are ordinary Java objects that can also be constructed in Java. However, it is often more convenient to specify them in XML.

In each of the above syntaxes, the `menu-text` is a string that is displayed in the *Test* menu in the top right corner of an I-X Process Panel. For a `test-item` only, any occurrence of `$to` in the `menu-text` string will be replaced by the value of the same `test-item`'s

to-name. (This is not done for a sequence, because the messages in a sequence might be to different destinations.)

A to-name is the symbol-name of the agent the test-item's contents should be sent to.

The delay-before in a test-item is the number of milliseconds to wait before sending the contents. Delay-before defaults to 0.

Note that within the **SENDABLE** test item contents you can specify a sender-id if you wish it to look like an item came from that agent or panel. The syntax for **SENDABLE** items is included in an earlier section.

Here is an example.

```
<test-item
     menu-text="Give $to a report-back example issue"
     to-name="me">
     <contents>
          <issue priority="high"
               report-back="yes">
               <pattern>
                    <list>
                         <symbol>note</symbol>
                         <string>sample note text</string>
                    </list>
               </pattern>
          </issue>
     </contents>
</test-item>
```

That test-item would appear in the *Test* menu as *Give me a report-back example issue* and when selected would send the panel an issue with priority=high, report-back=yes, etc.

Here is a test-item that sends a report after a delay of 4 seconds:

```
<test-item menu-text="Send $to a single report with a delay"
     to-name="me"
     delay-before="4000">
     <contents>
          <report report-type="information"
               text="Here's some information"/>
     </contents>
</test-item>
```

A test-sequence contains a list of test-items. The menu-text of those items is ignored (and needn't be specified). The test-sequences' own menu-text appears in the *Test* menu.

When the test-sequence is selected from the *Test* menu, it processes the list of test-items in order. For each item, it waits for the item's delay-before milliseconds and then sends the item's contents to the agent named by the item's to-name. This allows a sequence to send messages to a variety of different destinations. By using delay-before values of 0, it is possible to get several messages to be sent (almost) at once.

Each item in a test-sequence may have a different type of contents. That makes it possible to send an issue to one agent, a report to a second, and so forth.

However, in some cases, all of the items in a sequence will have certain things in common; and then it may be possible to use a test-sequence-generator. A test-sequence-generator contains a single test-item that is used as a template. The menu-text of that item is ignored (and needn't be specified), but all other fields may be significant.

A test-sequence may contain either a list of to-names or a list of content-strings, but not both. If it contains a list of to-names, a sequence is constructed by making a copy of

the template for each of the `to-names`, replacing the copy's `to-name` each time. The resulting sequence will send essentially the same message to a series of agents.

If there is a list of `content-strings` instead, the sequence contains one copy for each of the `content-string`s, with the "main contents" of the copy replaced by the corresponding `content-string`, suitably interpreted. This sequence will send a series of similar messages of the same type (issue, report, or whatever) to a single agent: the agent specified by the `to-name` of the template.

The interpretation of a `content-string` depends on the class of the template's contents. If the template contains an issue or activity, the content string is treated as a pattern and parsed in the usual way (with `?names` being variables etc); if the template contains a report or chat-message, the `content-string` is placed in the object's text field. Constraints are not yet supported (although they may appear in ordinary test-sequences).

The `test-item`s in the generated sequence have `delay-before` values determined as follows: If the generator specifies an `initial-delay`, it becomes the `delay-before` of the first item in the generated sequence. If the generator specifies a `delay-between`, it becomes the `delay-before` of all subsequent items in the sequence. Otherwise, the template's `delay-before` is preserved. Here is an example that when selected will send a series of chat-messages:

```
<test-sequence-generator initial-delay="0"
     delay-between="2000"
     menu-text=”Send me some example chat messages”>
     <content-strings>
          <list>
               <string>Sample chat text 1</string>
               <string>Sample chat text two</string>
               <string>More chat</string>
               <string>This time there will be
several lines of text
     and maybe some indentation
     just for variety
and a last line</string>
          </list>
     </content-strings>
     <template>
          <test-item>
               <to-name>
                    <string>me</string>
               </to-name>
               <contents>
                    <chat-message />
               </contents>
          </test-item>
     </template>
</test-sequence-generator>
```

Menu separators can be specified in the `test-menu` file. The XML syntax is

```
<test-separator />
```

Just include it in the list between the test items you want to separate.

Using the *Test* Menu

Entries in the *Test* menu do not have to send messages. They don't even need to involve any of the objects described here. However, this section will describe only the cases that can be specified by a test-menu file.

If a single message is to be sent, without a delay, it is sent as soon as its *Test* menu entry is selected. That is so regardless of whether it came from a single `test-item` or from a 1-item `test-sequence`.

Otherwise, a new thread is created to supervise the message sending. While that thread is running, the corresponding menu entry is prefixed by “Stop: ” and, if selected, stops the thread. When the thread terminates, the entry reverts to its original form.

However, all messages are actually sent by the GUI event thread just as in the normal operation of a panel). The other thread exists only to control the timing.

Messages sent to “me” are given directly to the panel rather than going via the communications strategy.