I-X Domain Editor – A User's Guide to I-DE

Jussi Stader and Austin Tate
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 4.5 – 5 June 2007

1 Introduction - Enterprise Modelling and <I-N-C-A>

Enterprise Modelling refers to a set of activities carried out for capturing and describing contextual information within an enterprise that is relevant for supporting the running of that enterprise. It usually covers aspects of that enterprise as required based on some application or project goals in mind. It normally deploys semi-formal modelling techniques. That is to say that formal (visual or graphical) models are used in conjunction with informal textual descriptions (i.e. written in natural language) that provide additional detailed information to the formal model.

To create an Enterprise Model, the first decisions are: which information is relevant, how it should be captured and in which format it should be described (i.e. using a format that is provided in the chosen modelling methods). The information captured must both be accurate and open to changes. Arguably, capturing information is one of the hardest parts of Enterprise Modelling activities. The modeller needs good capture techniques and suitable notations so that the modeller can easily see what has been modelled and can communicate this to others. There are today many different techniques and notations to support an enterprise modeller in capturing information, many of them are informal methods [5, 8, 7, 9, 15, 2]. There are good reasons for this proliferation of techniques and notations: the modeller needs all the support available to describe the complex and multi-perspectives of information within a domain, and the better the technique and notations suit the modeller and the aspect of the enterprise that is being modelled, the more effective the modeller can be and the better she can understand and communicate the information. This leads to the first requirement for enterprise modelling support: any realistic enterprise modelling support will have to be able to provide and cope with different techniques for capturing information, and with different notations (or views) for the information using the chosen modelling method.

When carrying out activities of Enterprise Modelling, in practice, it is nearly impossible to ever state: "this model is now finished and it will never need any further changes". This is so because it is usually impossible to model every aspect of a running enterprise both accurately and in sufficient detail so that the models describe all aspects of the enterprise [4, 3]. Furthermore, even if this task was possible, the world within and outside an enterprise does not stand still, and changes will always need to be incorporated into the model to make the models current. Enterprise Modelling efforts can be compared to the painting of the Forth Rail Bridge, a large and intricate metal structure near Edinburgh. This bridge is painted periodically to prevent corrosion of the structure. The painters start at one end of the bridge and work their way to the other end. As soon as the painters reach the other end, it is time to paint the first end again, i.e. it is a continuous job. Our second and third requirements then are that it must not be a necessary condition for the models to be complete to be useful. In other words, we must be able to cope with incomplete information and

make use of it, and it must be possible (and easy) to change and update the models.

Once enterprise information has been captured, it should be used to support the running of the organisation. How much support the models can provide depends on their quality and their form. If the models are available in paper form only (e.g. printed documents of diagrams and descriptions), they can be used for documentation and communication ("this is what we do to achieve so and so"). This can be useful for stating best practice, for teaching and training purposes, etc. However, paper models are not easy to change and their availability is not great. If the models are available on-line in the form of documents, they are easier to change and (in most organisations) more readily available.

However, since Enterprise Modelling is such a difficult job, we should be able to provide more support based on the models, rather than just using them as paper-based documentation. The models should be used to support the running of the enterprise much more directly. This usually makes more demands on the already difficult task of modelling: more information has to be included in the models and the models need to become more formalised to support likely automation. However, the benefits of applying such models can be significant and are usually well worth the effort. For example, process models and related information can become enacted in workflow systems and thus directly support the running of business processes [11, 12, 6, 1]; other models can be used for skills management [10] and more generally knowledge management. This type of support puts an organisation in a good position to quickly react to changes.

In summary, to provide appropriate automated support for Enterprise Modelling activities, we list a few high-level requirements below for such tools:

- The tool must provide for and cope with different modelling techniques, e.g. to support different textual and graphical facilities for capturing and viewing information, and using different notations (or views) for the information, when appropriate.
- It must not be necessary for the models to be complete. The user should be able to make sense of and cope with incomplete information when carrying out their tasks. In addition, the tool should maximise the use of the information available.
- It must be possible (and easy) to change and update the models.
- Captured models should be used to their full capacity to support the running of the organisation.

The I-X Domain Editor, I-DE, is a tool for Enterprise Modelling that takes into account the high-level requirements above. I-DE supports models used by I-X technology. These models are based on the <I-N-C-A> (Issues - Nodes – Constraints - Annotations) constraints model, a high-level

approach to model specifications [14]. <I-N-C-A> models can be used to describe any synthesised artefact, e.g. results, models, plans, configurations, designs, etc. An <I-N-C-A> specification defines a set of "nodes" to be included in the design, along with "constraints" on how these nodes can be related to one another and the environment they exist in. It also includes a set of outstanding "issues" and “annotations” related to the artefact(s). By having a clear description of the different components within a synthesised artefact, the model allows for them to be manipulated and used separately from the environments in which they are generated.

At various stages of the development of the I-X research the typography for rendering <I-N-C-A> has varied as the components have received clarification. <I-N-CA> originally stood for *Issues, Node, Critical* and *Auxiliary Constraints.* The aspect of separating critical (shared communications) constraints from auxiliary (separately managed) constraints is still important within the I-X architecture, but is now considered all part of managing the "C" (constraints) component of a model. The annotations were always present in the <I-N-C-A> framework and can be attached to all components, but the top-level entity annotations capturing the rationale behind the synthesised product or the process/plan being described has required more prominence as the work has continued and as mixed-initiative and human communications aspects have become more important. Hence, the rendering <I-N-C-A> with the extra hyphen now stands for *Issues, Nodes, Constraints and Annotations.*

The *issues* in the specification describe the outstanding issue items to be handled and can be used to represent unsatisfied objectives, problems raised during analysis and operations that need to be addressed, etc. The *constraints* can be thought of as implying further constraints which may have to be added into the design in future in order to address the outstanding issues. The *nodes* in the specification describe components that are to be included in the design. Nodes can themselves be artefacts that can have their own design(s) associated with them.

Constraints define relationships between nodes and place restrictions on artefacts within the design space to meet requirements in the domain. Constraints are split into two categories: *critical* and *auxiliary constraints*, depending upon their criticality in the domain. Auxiliary constraints are *conditional constraints* that constraint managers (solvers) may return "maybe" as answer to indicate the fact that such constraints are acceptable so long as some other critical constraints are imposed by the model. The maybe answer is returned with a list of disjunctive conjunctions of critical constraints. Critical constraints must be obeyed by a model. Finally, annotations can be added to describe the rationale behind design choices and other useful information.

The choice of which constraints are considered critical is domain dependent and a decision for an application of I-X and I-Core. It is not pre-determined for all applications. A temporal activity-based planner would normally have objects/variable constraints (e.g. equality and inequality of

objects) and temporal constraints (e.g. before(time-point1, time-point-2) constraint) as the critical constraints. But, in a 3D design or configuration application, object/variable and some other critical constraints (e.g. spatial constraints) might be chosen. It depends on the nature of application goals to decide what needs to be communicated between constraint managers and the planner, or other support tools.

The types of constraints in an <I-N-C-A> model are usually specialised to a domain and described in a great level of detail. For example, the following types of constraints may be described in a process model within the <I-N-C-A> framework:

I - Issue constraints
- "include node" constraint
- "include node" constraint NOT allowed

N - Node Constraints
- "include node" constraints
- other domain-specific node constraints

C - Constraints; E.g., if the artefact is an activity plan:
- O - Ordering constraints
- V - Variable and other constraints
- X - eXtra constraints (such as):
  - Authority Constraints
  - World State Constraints
  - Resource Constraints
  - Spatial Constraints
  - Miscellaneous Constraints

A – Annotations, informal constraints, e.g., information about a graphical layout of the nodes in a GUI.

In our generic, conceptual model, an <I-N-C-A> construct has 5 basic components:

- name: an identifier for the construct,
- issues: a set of issues related to the use of the construct,
- nodes: a list of nodes that are part of the construct,
- constraints: a set of constraints that apply to the construct and its nodes,
- annotations: comments and other useful information about the construct.

The following sections first describe model specifications that can be created and edited using I-DE, before describing I-DE and its uses.

2 I-DE Models

In this section we describe the structures that can be specified, viewed, and edited with the help of I-DE. I-DE's concepts are:

- The Domain: a coherent set of specifications.

- Activity Specifications: details about an activity and how can be performed. Activity Specifications are also called Refinements.
- Activity Relatable Objects (AROs): specifications of classes of objects that are generated or manipulated within the domain. The object classes are organised into an object class (or type) hierarchy.
- Grammar and Lexicon: specifications of terms that can be used within the domain and how they can be combined to form specifications. Currently the lexicon is built automatically to reflect the patterns used in issues, nodes and constraints. It is envisaged that I-DE will eventually allow for managed grammars and lexicons to allow active assistance in modelling and provide active help in maintaining coherence of models.

All main construct specifications within I-DE conceptually follow the <I-N-C-A> model, except for the grammar and the lexicon. I-DE specifications make use of specialisations of <I-N-C-A> for constraints regularly used in process models and to add human-readable "comments" as part of their annotations. The keywords (and their specialisations) currently used by I-DE specifications are *issue, node, constraint (temporal, before, world-state, condition, effect, compute) and annotation (comments).*

2.1 Domain

The Domain is a coherent set of specifications that represent an area of expertise. The domain itself is an <I-N-C-A> object. It has a name and it can have issues, constraints, and annotations. Its nodes consist of the activity specifications and the activity relatable objects that are defined within the domain. Thus the domain serves to bind together sets of specifications that are relevant for an application. A domain can be loaded from file, saved to file, published, and reverted to a previous version.

2.2 Activity Specifications

Activity specifications follow the <I-N-C-A> model, but with the following specialisations:

- They have a pattern that describes the activity performed (conventionally starting with a verb).
- Nodes are sub-activities that are specified by giving their activity pattern. Each node has two node-ends: begin and end which are time points that can be referred to within constraints.
- Constraints take the form of type, sub-type, and other parameters such as node-end time point reference(s), "pattern = value", etc. The types and sub-types are keywords that can be shared between applications. There are three specific constraint types that are currently supported further. These are:
    1. Orderings, which are precedence relationships between node ends. An Ordering has the following specification: type is "temporal", sub-type is "before", and node references are node-end1 and node-end2. There is no pattern.

2. Conditions, which are descriptions of world state that have to be fulfilled before the activity can be considered for execution. A Condition has the following specification: type is "world state", sub-type is "condition", and node reference must currently be set to "self" but should in future also allow the activity's sub-nodes. Any form of pattern = value is allowed.
3. Effects, which are descriptions of world state that will be true when the activity has been performed. An Effect looks just like a Condition, except for its sub-type, which is "effect".

- Activity specifications can have variable declarations that determine which variables can be used in the specifications. The declarations can be “none” (no variables allowed), “any” (any variables allowed), or a given list of variables. In the latter case, no other variables are allowed.

2.3 Activity Relatable Objects

In I-X, Activity Relatable Objects can be supported by specifying a simple class system. An IX-ObjectClass has a name and may have sub-classes and properties. Classes can have more than one super-class, but cycles should be avoided. Classes inherit properties from their super-classes.

Properties of classes have a name and their values are typed based on a small set of predetermined types: string, symbol, number, object, list, or default. Currently, this type specification is only used to guide I-X parsers, but in future I-DE may use it to check constraints in the models.

2.4 Grammar and Lexicon

Currently this is built automatically to reflect the patterns used in issues, nodes and constraints. It is envisaged that this will eventually allow for managing grammars and lexicons to allow active assistance in modelling and provide active help in maintaining coherence of models.

3 The Domain Editor (I-DE)

I-DE is based on I-X Technology [13] from AIAI at the University of Edinburgh. The main window of the Domain Editor (the frame) contains several editor panels for editing different aspects (or constructs) of the domain. Currently the editors available are

- the Global Domain Editor, which contains information about the domain itself (e.g. the domain name);
- the Activity Editor, which edits “refinements” which contain information about activities and how they break down into sub-activities;
- the Grammar Editor, which currently only shows the patterns that are in use in the domain;
- the Objects Editor, which edits the object model which specifies object classes and their properties and how they break down into sub-classes.

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

1. Minimal View: a simplified version of the activity and its refinement. The main simplification is that no constraints are shown
2. Comprehensive View: a view that can display and edit all of an activity's specification
3. Graphical View: a graphical view that uses nodes and arcs to show an activity's sub-activities and the temporal relationships between them.

3.1 The Domain Editor Window

This window provides access to most functions of the overall domain editor via its menu bar, and access to the most commonly used functions via its tool bar. The window can display in one of three styles: simple, tabbed, and card style. The style can be changed using the preferences editor via the View menu.
Figure 1 shows the window in simple mode with the activity editor showing.

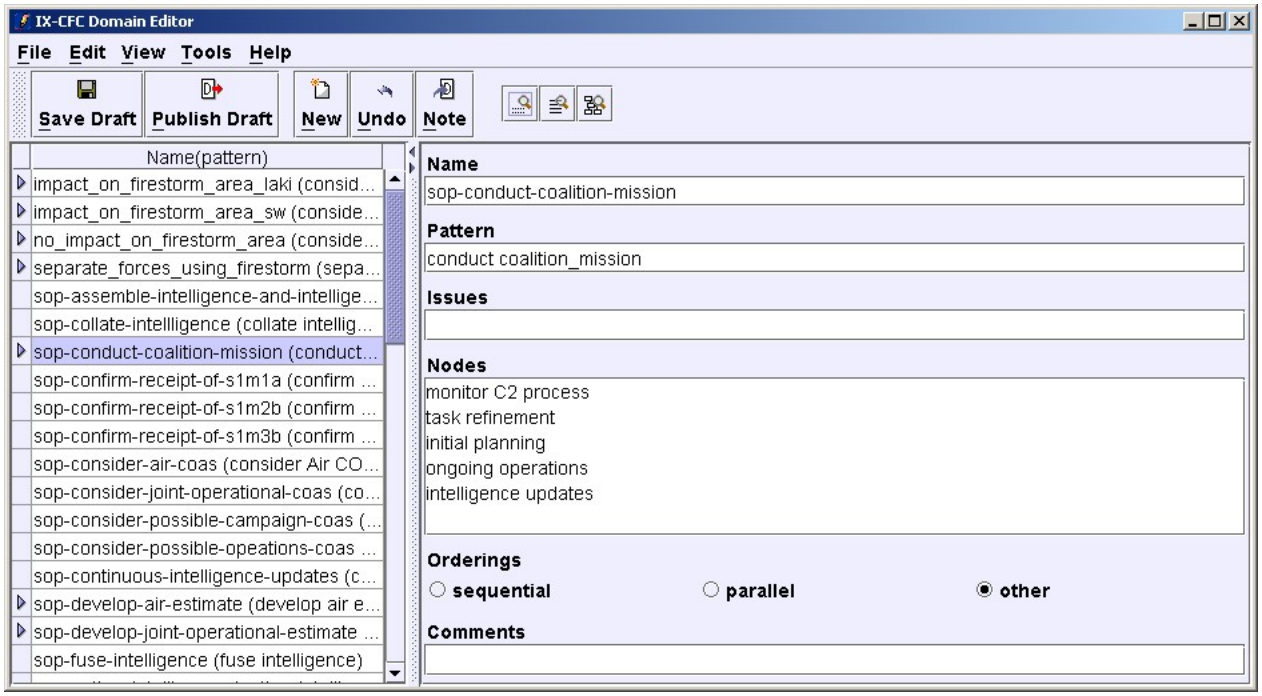

Figure 1: I-DE Window (in simple mode)

3.2 The Menu Bar

The menu bar has 5 standard menus:

1. File for closing the Domain Editor, for file access (open/save), publishing, and reverting. 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. Many of the edit menu options are common to all types of panels (new, modify draft, revert, ...)

but there are also more specific options that are only available in particular panels;
3. View for changing the visual set-up of the window, for changing which panel is shown in the window, for changing which view is shown in that panel (if applicable), for changing panel styles, etc.;
4. Tools for additional support like consistency checks etc. This menu also gives access to the preferences editor;
5. Help for access to this manual, other help, and information about the application.

3.3 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 (in most cases, the image on the toolbar button is shown in the menu next to the corresponding menu item. 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. The buttons themselves can either show just an icon or an icon with a short text underneath (determined by user preferences). The toolbar can be switched on and off via the View menu.

4 Working with the Domain Editor

This section gives an overview of how to work with I-DE. Construct specific editing (e.g. Activity editing) is described separately in more detail. The issues covered here are update levels, workflow, and preferences.

The Domain Editor maintains different levels of updates. The original domain model that the editor is started with is considered a public domain model, which other applications may be using for their own purposes (e.g. within an I-X Process Panel). This public domain model is kept as it is unless an updated model or a replacement model is explicitly "published" by the Domain Editor's user. (Note that this is true whether the Domain Editor is used in stand-alone mode or as part of another application). There is also a "draft domain model" which is the one that is being edited. The Domain Editor keeps track of any changes that are made to the draft domain model so that updates to the original domain model can be made explicitly.

4.1 Saving and Reverting

There are 3 levels of saving:
1. Modify Draft: When a construct has been edited in the Domain Editor Panel, initially these changes may be made only in the panel itself, not in the domain construct that is being edited. Such changes need to be transferred from the panel into the construct in the draft domain. The Domain Editor will perform this transfer (modify draft) when it is aware that this may be necessary, e.g. when a node has been added, when the user switches constructs, views or panels, or if the user decides to save or publish the draft domain. However, at any point the user can choose to explicitly modify the draft, i.e. note the changes into the draft domain via the toolbar button or the Edit menu.

2. Save to File: Modifying the draft (noting changes) does not save to file, so the next level of saving is to save the draft domain to file. As with all editing applications, it is recommended to do this 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 into the Domain Editor for further editing, or it can be accessed by other applications.
3. Publish: The underlying public domain is not changed by any of the above (simple editing, modifying draft, or saving the draft domain to file). The only way to update the public domain is to publish the draft domain via the toolbar button or the File menu. When this happens, all pending changes are transferred to the original domain and these changes will be seen by any application that has registered as listeners to this domain. Note that publishing is always done for a whole domain, 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 just before or straight after publishing. At that point the draft domain and the public domain can be represented by the same XML structures. It is a good idea to publish from time to time even if the Domain Editor is running stand-alone because it will make the editor more efficient.

There are undo functions that correspond to the 3 levels of saving:
1. Undo: revert a construct to the last time it was modified in the draft domain, i.e. undo an editing step. This function is reversible with a Redo function;
2. Revert to published: revert a construct to the public version (via Edit menu), i.e. undo all changes to this construct since the domain was last published. This function is not reversible;
3. Re-load: revert the whole domain to the last time it was saved to file by opening that file via the File menu. This function is not reversible.

Note that the first two undo functions apply to an individual construct, while the third applies to the domain as a whole. There is a fourth "revert" function that also applies to the domain as a whole: "discard changes to draft" which reverts the whole domain to the public version, i.e. undo all changes to all constructs since the domain was last published.

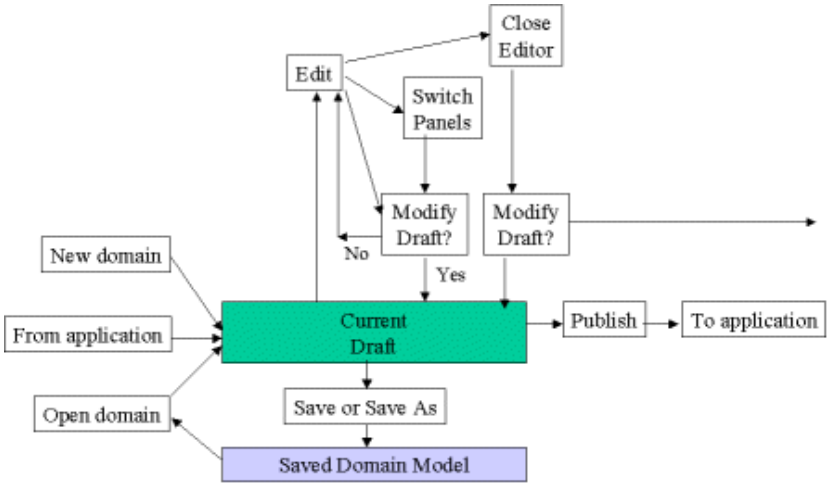

Figure 2: I-DE Workflow

The diagram shows the different states that the system may be in during an I-DE session, and it shows how a user can move between these states. After editing is done, the user may explicitly choose to modify the draft domain model. A modified draft domain model can be saved to file and/or published to be visible to the application (the I-X Process Panel) or both. When the domain editor is closed, the user is given the option to save the current draft to file and/or to publish the current draft to be visible to the application.

4.2 Preferences

There are several preferences and two modes of use that the user can choose from. There are default preferences for initial use of I-DE, but the user can change the preferences using a simple preferences editor. The preferences editor allows the user to change preferences, to apply the current preferences to the I-DE they are currently working with, to save preferences so that the editor starts up with the saved preferences, and to revert to previously saved preferences. Below, we describe the two modes of use and the preferences that are under the user's control.

The two modes of use are:

1. simple mode: a cut-down version of I-DE that shows only the essential features needed to quickly put together simple process models. Other user preferences are restricted in this mode, i.e. it can be seen as a quick way to set all preferences to the simplest option. Explicitly changing any of the restricted options will override the simple mode restrictions and result in advanced mode to be set;
2. advanced mode: the full version of I-DE that gives the user full control over preferences and access to all editing facilities.

If the user switches from advanced to simple mode, most preferences will be set and restricted. However their previous settings are kept and when the user switches back to advanced mode all preferences will go back to their previous values.

The preferences that are under the user's control cover the editor in general, sub-editors, and views. These include:

- mode of use: a flag that shows whether simple mode is set;
- button texts: a flag that determines whether text is shown on underneath the icons of toolbar buttons;
- lists as text: whenever there is a list of specifications that can be edited by the user (e.g. activity nodes), the user can choose whether to view these as a list and edit them with special-purpose dialogue-style editors, or to view them as lines of text that can be typed into directly;
- editable: to use the editor as a read-only viewer, its editing facilities can be switched off to ensure that no unintended updates are made. This is not yet implemented;
- panel style: the choice of panel style to use. The panel styles available are minimal (no visual queue for changing panels), tabbed (a tab is shown at the top of the panels that can be used to switch panel), and

card (a choice-box is displayed above the panel that can be used to change panel). Note that it is always possible to change panels by using the Windows option of the View menu;

- view: the view to use. The views available depend on the panel. For example, for the activity panel, the views available are minimal, comprehensive, and graphical.

There are additional choices that affect the display of constraints in the comprehensive view of the activity sub-editor. The three groups of constraints that can be displayed are "orderings", "conditions/effects", and the generic "other constraints". The user can choose to suppress the display of some of these; for example, if only orderings are of interest, all other constraints may be switched off. Note that, whenever the generic "other constraints" are displayed, all available constraints are shown. For example, if only the conditions/effects option is switched off, conditions and effects will be shown in their generic form under "other constraints". Constraint-related preferences are only available for the comprehensive view - the minimal and graphical views have their own, special purpose ways of displaying constraints.

In simple mode, the following preferences are restricted as follows:

- panel style is set to minimal,
- lists are displayed as text,
- activity view is set to minimal,
- only minimal information about orderings are displayed, so all preferences relating to constraints are disabled.

Other preferences are not affected by the choice of mode.

5 Construct Editing

Each construct editor is responsible only for manipulating the specification of the construct. In the Global Domain Editor Panel this is less intuitive than for the other panels: this editor only considers domain details as part of its editing remit (mainly name and comments), not the constructs within the domain. The only way to manipulate the domain as a whole is via the options in the File menu, and these are available for all panels.

I-DE provides several functions for all construct editors. The implementation of these functions may vary between different construct types, but they should all be available. These common functions are:

- new: create a new construct of this type;
- copy: make a new construct that holds the same details as the current one;
- delete: delete the current construct;
- edit: select a construct to be edited;
- modify draft: save the changes made in the panel into the draft domain;
- revert: revert the construct to the last time it was modified in the draft;

check: check the consistency within the current construct.

Figure 3 shows the global domain editor and Figure 4 shows the grammar viewer (currently the grammar cannot be edited).

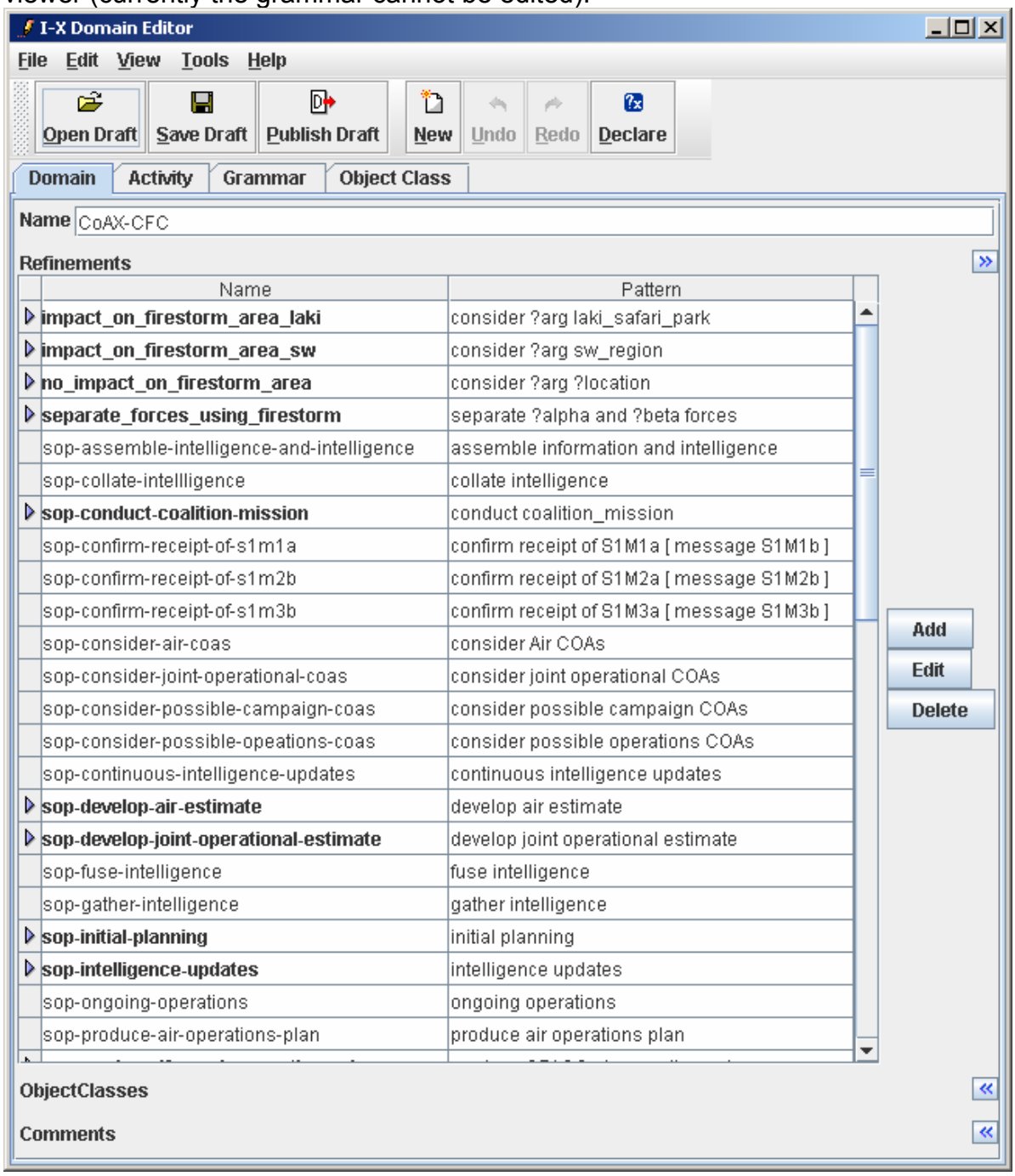

Figure 3: Global Domain Editor

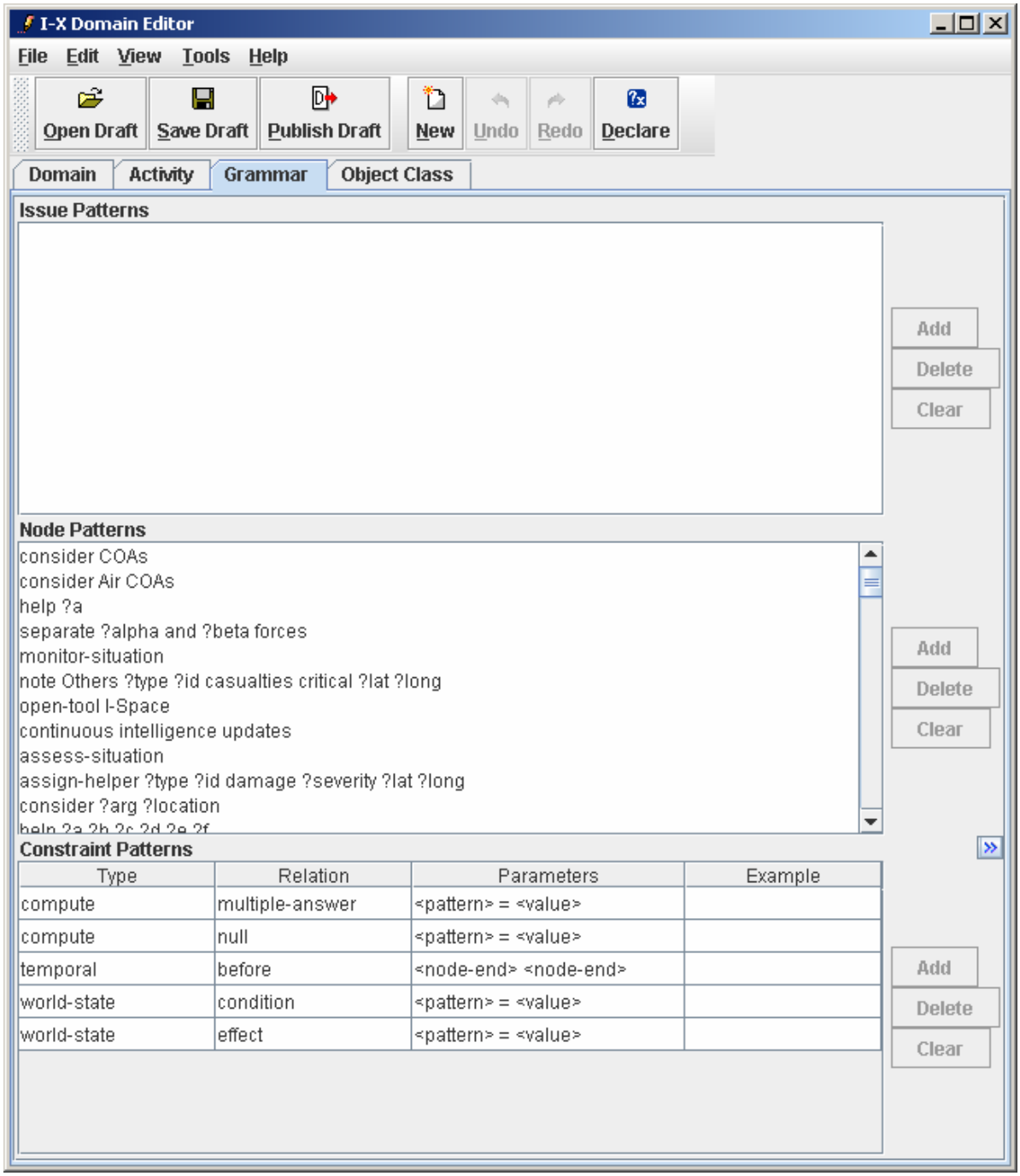

Figure 4: Grammar Editor

5.1 Activity Editing

The functions described above are common between editors for different concepts, but there are specific things that are available in I-DE to support activity editing. The Activity Editor has three different views that have different purposes:

1. the minimal view is for quickly defining or editing simple activity specifications.
2. the comprehensive view provides full access to all fields of activity specifications that are known to I-DE.

3. the graphical view for a more visual way to specify or edit sub-activities and ordering constraints between them.

The Minimal View (Figure 5**):** An activity's name and pattern can be specified, along with its list of issues and sub-activities. The sub-activities can be put into sequence or in parallel and the editor will indicate other ordering constraints if they are present, i.e. if they have been specified using a different view or a different editor. Specifications (and edits) can be made by simply typing into the respective fields on the screen.

The Comprehensive View (Figure 6**):** The comprehensive view shares some of the features of the minimal view, but provides additional ones. The name and pattern of the activity are presented and specified or edited in the same way as in the minimal view. For issues and nodes the user can choose whether to type into the fields directly or whether to see fields as lists and use structured dialogue-style editors for specifying and editing. Constraints are split into orderings, conditions/effects, and other constraints. The editor can show and edit these constraint types with specific support, but the user can also decide to suppress the display of specific constraint types, or to view constraints of all types (except orderings) in their generic form.

The Graphical View (Figure 7**):** The graphical view uses nodes and arcs to show sub-activities and ordering constraints between them. There are also text fields for the activity name and pattern and (as in the other views) these can be edited by typing into the fields. Nodes can be added and deleted easily. They each show a node number that is assigned by I-DE whenever a sub-activity is specified (also in the other views), the pattern of the sub-activity, and the sub-activity's two node ends: begin and end. Arcs can be drawn between node ends to specify ordering constraints between the node ends.

In the graphical view, it is easy to move from a sub-activity to its possible expansions (all activity specifications whose patterns match that of the sub-activity) by clicking the right mouse button. This brings up a pop-up menu that lists all matching activity specifications that are currently in the domain. New expansions (activity specifications) can be defined using the same pop-up menu.

The minimal and comprehensive views have a summary panel on the left and an editing panel on the right. The summary panel lists all currently defined activities and their sub-activities as a tree. Clicking on an activity will put that activity's specification into the editing panel, clicking on a sub-activity will put that node's pattern into the pattern field in the editing panel, ready for the user to provide a new specification with that pattern. Note that the graphical view is mainly for illustration and visualisation purposes, and is not as stable as the other two views.

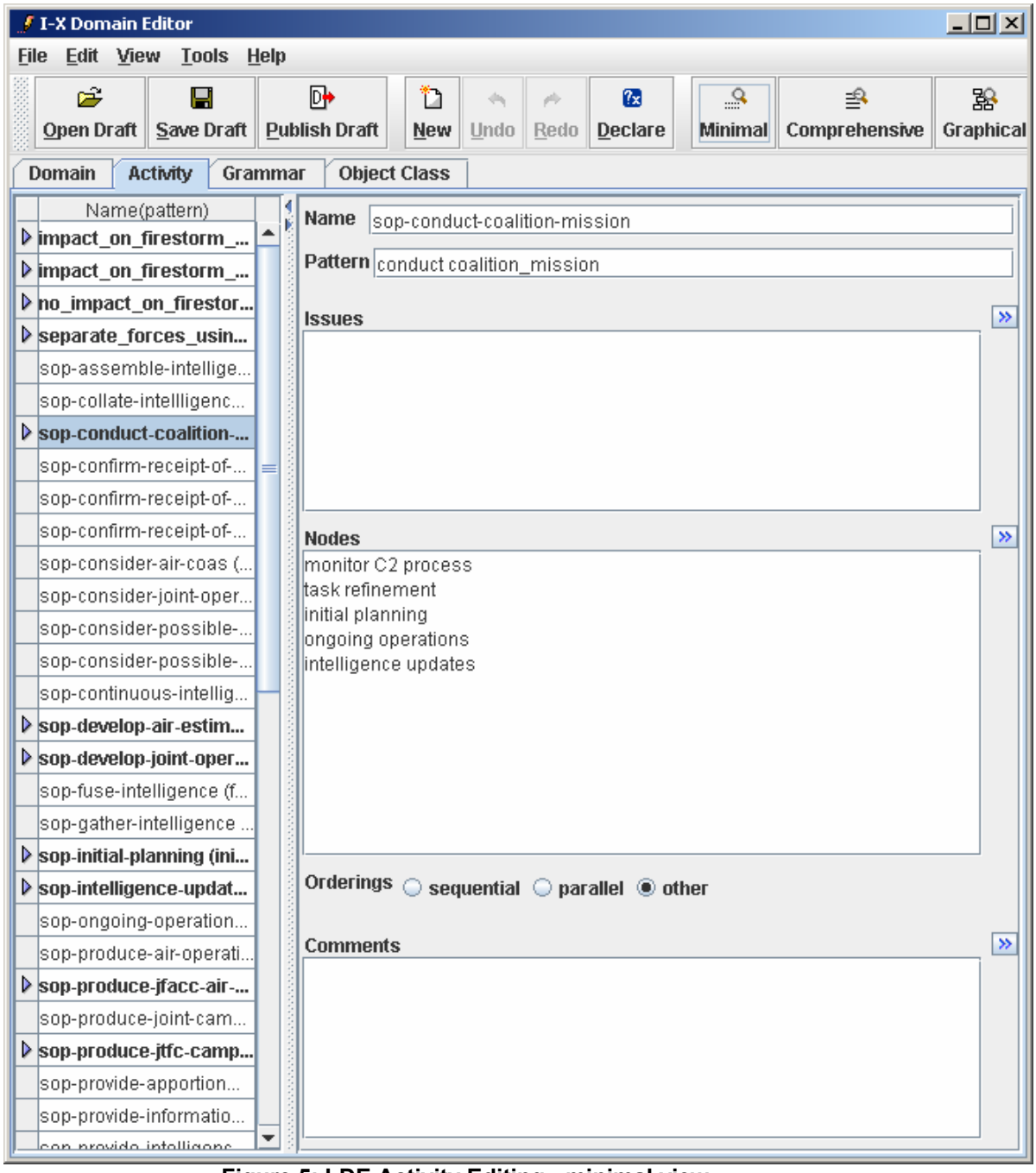

Figure 5: I-DE Activity Editing - minimal view

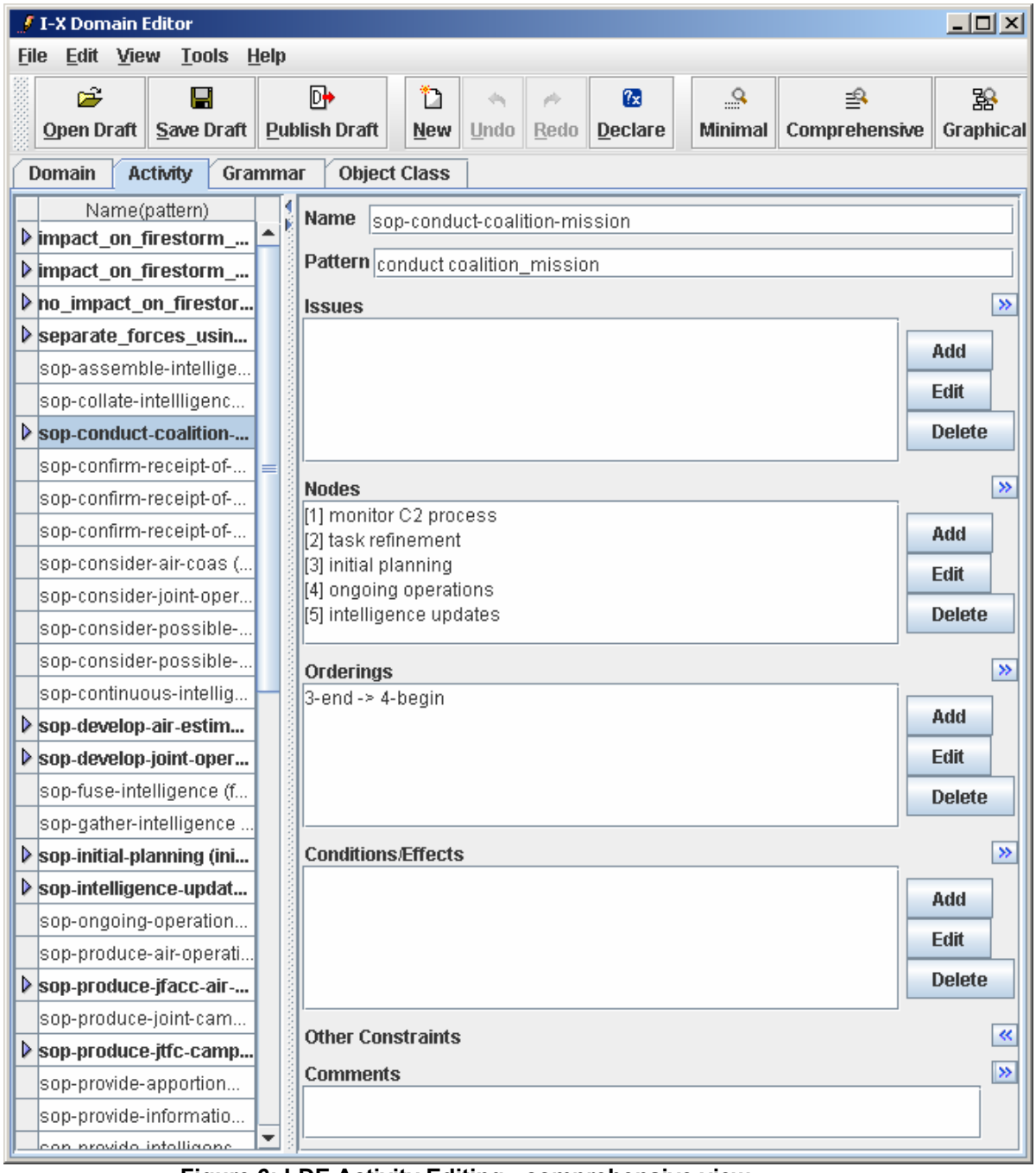

Figure 6: I-DE Activity Editing - comprehensive view

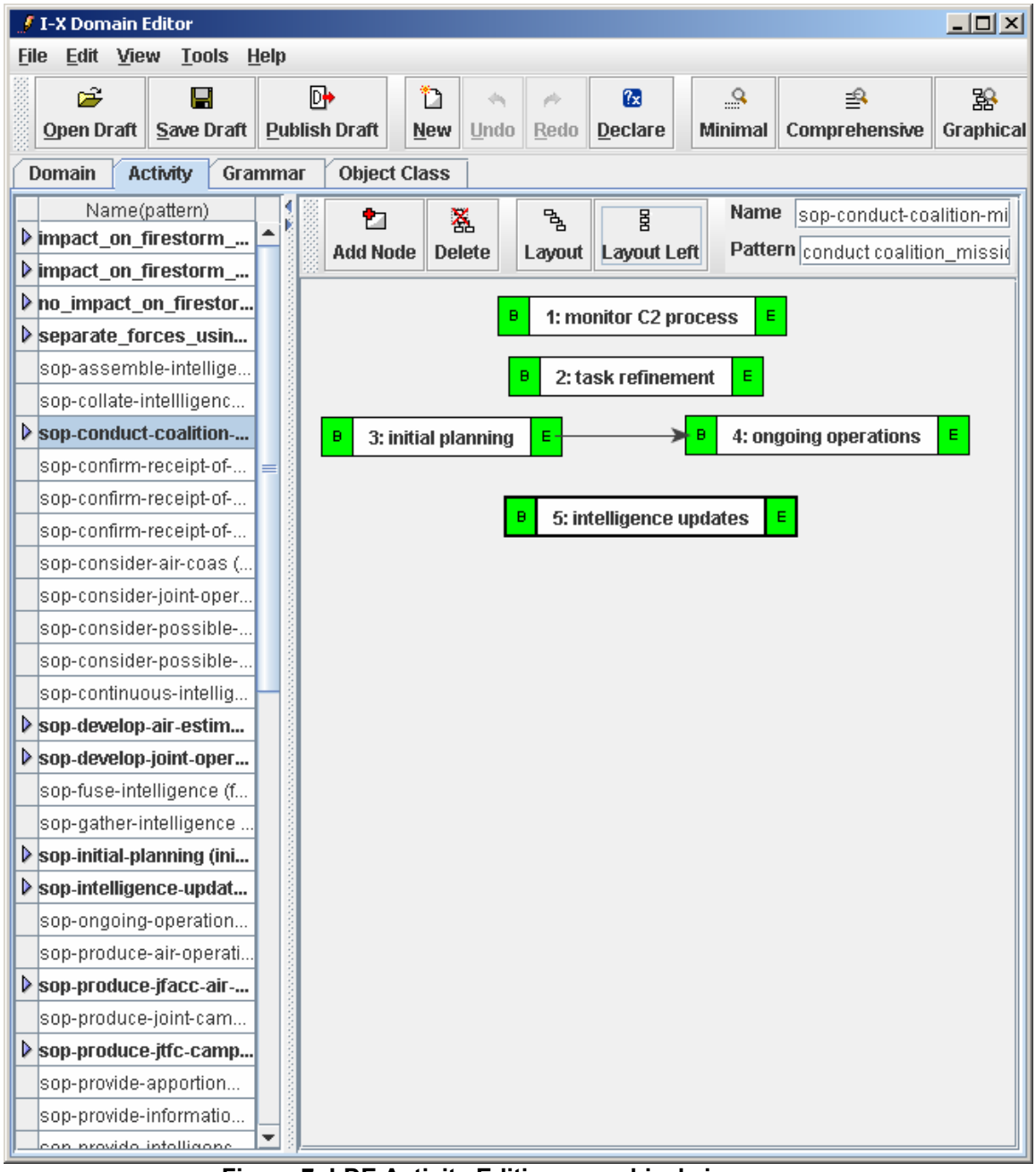

Figure 7: I-DE Activity Editing - graphical view

5.2 Variable Declaration

Variables are a way to refer to objects in the domain. I-DE supports the declaration of variables, linking these references to the domain's Object Classes. Once a variable has been given a class, information from the class specifications can be used to support the user to ensure consistent use of the variable, particularly when specifying conditions and effects: the class properties can be used to specify constraints consistently. A screen-shot that shows the variable declarations editor and its connection to constraint specifications is in preparation.

In addition, it is possible to use the editor to restrict which variables can be used in an activity specification. This is done with the help of variable

declarations that can be made via the "Declare Variables" option in the Edit menu. The user can:

- allow any variable to be used in the specification (default),
- allow no variables to be used in the specification,
- provide a list of all variables that can be used in the specification.

Once a variable declaration other than "any" has been given by the user, the activity editor will attempt to support the user in sticking to the declaration. Whenever the user types a "?" in a field that can contain a variable, the editor checks whether there are any variable declarations are present. If no variables are allowed, the editor complains and the "?" will not appear in the specification. If there is a list of allowed variables, the editor will present this list and allow the user to choose one of the declared variables. This should help the user to adhere to declarations. Note that the user can choose to ignore declarations and enter variables that are not allowed.

The editor also lets the user check explicitly whether the activity specification contains any violations via "check consistency" option in the Tools menu. On request, the editor will check whether there are any declarations. If there are, the editor checks whether any un-declared variables are used, and whether any of the declared variables are not used. Either of these two events is considered a violation and will be reported to the user.

The Variable Declaration Editor also allows the user to specify an object class for each variable. Once a variable has been assigned a class (or type), the properties of the variable's class are available for specifying constraints on the variable. This helps the user to keep track of object constraints and to be consistent about the use of property names.

6 Using the Models

A generic modelling framework, such as <I-N-C-A>, allows different kinds of models to be represented in a uniform style. The high-level, generic, multi-purpose structure combined with keywords that indicate the specific semantics of contents makes this framework a powerful representation tool. A shared ontology of keywords can ensure that the keywords are used consistently in the models. Different kinds of problem solvers that also share the ontology of keywords can then determine which specific parts of the models are relevant to them. Those parts of the models whose keywords are not part of the problem solver's domain can simply be ignored.

It is important to remember that the quality of the models and the amount of information present in the models will significantly affect their usefulness.

7 Evaluation and Conclusion

At the start of this paper we established a set of requirements for Enterprise Modelling support. We now take these in turn and check how I-DE relates to them:

- any realistic enterprise modelling support will have to be able to provide and cope with different techniques for capturing information, and with different notations (or views) for the information;
- I-DE illustrates how different viewers editors can be used in conjunction with each other (different sub-panels or sub-editors) and how different views can be used to highlight different types of information (form-based vs. graphical activity editor views).
- it must not be necessary for the models to be complete (we must be able to cope with incomplete information and we should make use of all the information that we have);
- I-DE looks for those things in models that it can display and update, ignoring the parts that it does not recognise. It also allows for incomplete models to be stored and published. However, generating and saving incomplete models is relatively easy. The main impact of this issue is on systems that use the models as a knowledge base, e.g. a workflow model.
- it must be possible (and easy) to change and update the models;
- We believe that I-DE is easy to use and that it provides good support for updating models. Its architecture and connection to the I-X framework also makes it easy to set up I-DE so that it can be used together with a workflow system in order to interleave process specification, planning, and enactment.
- models should be used to their full capacity to support the running of the organisation.
- The kinds of information that I-DE is designed to capture lend themselves well to being used as the basis of workflow.

Overall, I-DE provides good support for Enterprise Modelling. I-DE has been implemented in such a way that makes it easy to use the editor in different contexts and set-ups. It also makes it relatively easy to provide additional support. In the future, it would be interesting to see how much modelling support can be provided by using I-DE in conjunction with a workflow system running a modelling process model, i.e. to guide the use of I-DE with the help of a model that specifies Enterprise Modelling expertise.

8 References

1. Chen-Burger Y. and Stader J., “Formal Support for Adaptive Workflow Systems in a Distributed Environment”, to be published in Workflow Handbook 2003, Ed. Layna Fischer, 2003.
2. Dobson J. and Blyth A., Chudge J. and Strens M., “The ORDIT Approach to Organisational Requirements”, Requirements Engineering: Social and Technical Issues, London, ed. Jirotka and J.A.Goguen, Academic Press, 1994.
3. Fox M. and Gruninger M., “Enterprise Modelling”, AI Magazine, AAAI press, Fall 1998, pp.109-121.
4. Fraser J., “Managing Change through Enterprise Models”, Proceedings of Expert Systems '94, the 14th Annual Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK, 12-14 December 1994.
5. IBM, “Business System Development Method, Business Mapping, Part1: Entities; and Part 2: Processes”, 2nd ed, IBM England, May 1992.
6. Jarvis, P., Stader, J., Macintosh, A., Moore, J., Chung P., “What Right Do You Have To Do That?: Infusing adaptive workflow technology with knowledge about the organizational and authority context of a task”. First International Conference on Enterprise Information Systems (ICEIS-99), Setubal, Portugal,1999.
7. Mayer R, Cullinane T, deWitte P, Knappenberger W, Parakath B, & Wells S, “IICE IDEF3 process description capture method report (al/tr-1992-0057)”. Technical Report, Air Force Systems Command, Wright-Patterson Air Force Base, Ohio, 1992. See also http://www.idef.com/
8. NIST, “Integration Definition for Function Modelling (IDEF0)”, Federal Information Processing Standards Publication 183, National Institute of Standards and Technology (NIST), Dec 1993.
9. Ould M, “Business Processes: Modelling and Analysis for Re-engineering and Improvement”, John Wiley and Sons, 1995.
10. Stader J. and Macintosh A., "Capability Modelling and Knowledge Management"; Applications and Innovations in Expert Systems VII, Proceedings of ES 99 the 19th International Conference of the BCS Specialist Group on Knowledge-Based Systems and Applied Artificial Intelligence, Cambridge, December, 1999; Springer-Verlag; ISBN 1-85233-230-1; pp 33 – 50, 2000.
11. Stader J., Moore J., Chung P., McBriar I., Ravinranathan M., and Macintosh A., "Applying Intelligent Workflow Management in the Chemicals Industries", Workflow Handbook 2001, L. Fisher (ed), Published in association with the Workflow Management Coalition (WfMC), 2000, pp.161-181.
12. Stader J, “Results of the Enterprise Project”, Proceedings of the 16th International Conference of the British Computer Society Specialist Group on Expert Systems, Cambridge, UK, 1996.
13. Tate A, “I-X: Technology for Intelligent Systems”, www.i-x.info, AIAI, The University of Edinburgh, 2002.
14. Tate, A., “I-X and <I-N-C-A>: an Architecture and Related Ontology for Mixed-initiative Synthesis Tasks**”,** Proceedings of the Workshop on Planning/Scheduling and Configuration/Design at PuK 2001, Vienna, Austria, 2001.
15. Waern A. and Hook K. and Gustavsson R. and Holm P., “The Common-KADS Communication Model”, KADS-II/M3/SICS/TR/006/V2.0, Swedish Institute of Computer Science, Stockholm, Sweden, Dec 1993.