AGH University of Science and Technology
Computer Science Laboratory
Department of Automatics
Al. Mickiewicza 30
30-059 Kraków, POLAND

Design of XCCS models with Inez XCCS Editor

Marcin Szpyrka, Piotr Matyasik
AGH University of Science and Technology
Department of Automatics
Kraków, POLAND
{mszpyrka,ptm}@agh.edu.pl

Published online: 25.11.2008

CSL Technical Report No. 3/2008

CSL Technical Report No. 3/2008

Design of XCCS models with Inez XCCS Editor*

Marcin Szpyrka, Piotr Matyasik
AGH University of Science and Technology
Department of Automatics
Kraków, POLAND
{mszpyrka,ptm}@agh.edu.pl

Abstract. XCCS (eXtended CCS) is a graphical extension of Robin Milner's CCS process algebra. Equipped with a graphical modelling language XCCS process algebra is a tool for fast and flexible design of concurrent and real-time systems. The modelling language is supported by a computer tool called *Inez XCCS Editor*. currently Inez is equipped with a graphical editor and transformation algorithms that generate CCS scripts automatically.

The technical report presents a survey of main features of the tool and a detailed description of the XCCS modelling language. Moreover, several examples of XCCS models have been also included in the report.

Keywords: process algebra, XCCS, Inez XCCS Editor, graphical design

1 Introduction

Inez XCCS Editor is a CAD tool for modelling of concurrent and real-time systems with the XCCS language (eXtended Calculus of Communicating Systems, [8]). The tool provides a graphical editor for the design of XCCS diagrams and transformation algorithms for exporting XCCS models into CCS scripts. The current version of the *Inez editor* (v. 0.34) provides algorithms for generation CCS and TCCS scripts. The future versions are expected to support value-passing calculus too. Generated CCS scripts are compatible with the Edinburgh Concurrency Workbench [6], so the CWB tool can be used for a formal verification of XCCS models.

This report is intended to serve as a manual for *Inez* users. It contains a short description of the XCCS language and a detailed description of the *Inez XCCS editor* capabilities. However, the document is intended for readers already familiar with the CCS process algebra. Readers unfamiliar with these concepts are referred to one of the books: [5], [3], [2], [1].

The paper is organised as follows. Section 2 deals with the XCCS modelling language. The *Inez XCCS editor* is presented in Section 3. Section 4 deals with the algebraic layer of XCCS diagrams. A short description of the transformation algorithms is presented in Section 5. Some examples of XCCS models are discussed in Section 6. Some future plans and a short summary are presented in the final section.

2 XCCS modelling language

A model in the CCS language is designed as a sequence of algebraic equations. The basic calculus uses only five operators, and two extra ones are used in the timed version of CCS (TCCS):

*The paper is supported by the AGH-UST Research Project No. 11.11.120.767

- The prefix operator (`.`) specifies the ordering of actions and events.
- The choice operator (`+`) selects one option among several possible choices.
- The composition operator (`|`) indicates that two agents (processes) execute simultaneously.
- The restriction operator (`\`) hides some ports that are used only for internal communication among agents.
- The relabel operator (`[ ]`) changes names of some ports to make connections among agents possible.
- The strong choice operator (`++`) – the two choice operators differ in the manner they handle delays ([3]). Strong choice `A++B` allows a delay only if both `A` and `B` are capable of that delay, while weak choice `A+B` allows also a choice to be made in favour of the agent which can delay the longest.
- The delay operator (`$`) allows infinite delay before an action is performed.

The key idea of the XCCS approach is to provide a graphical layer for modelling concurrent systems in the CCS language. An XCCS model consists of two layers: an algebraic (textual) and a graphical one. All operators applied to define interconnections among agents (`|`, `\`, `[ ]`) have been moved to the graphical layer.

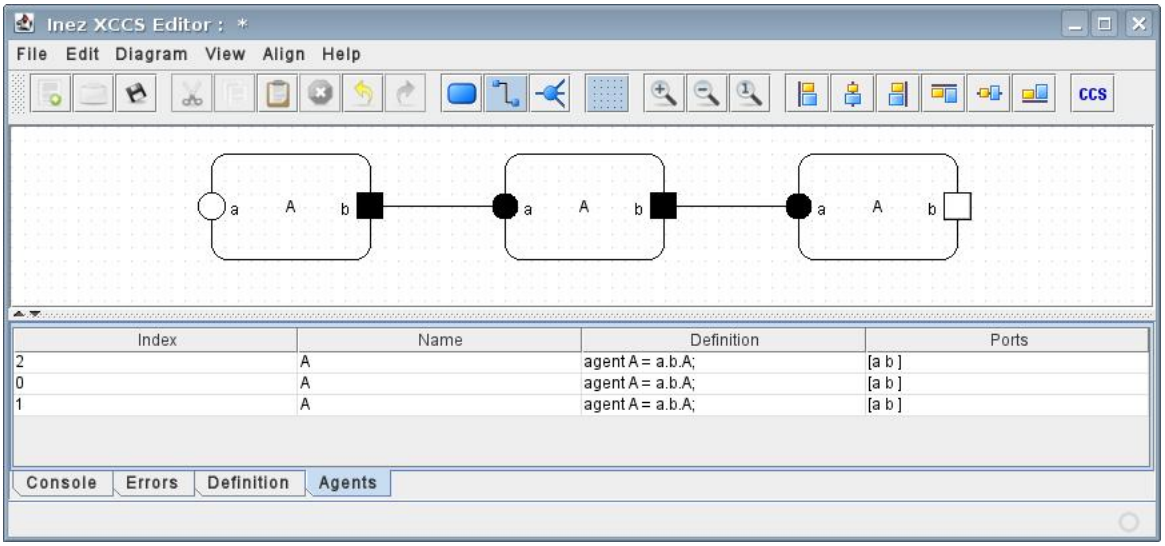

Figure 1: XCCS model of a three cell queue buffer

Let us consider the model of a three cell queue buffer shown in Fig. 1. Because the diagram contains three copies of the same agent `A`, the algebraic layer of the model consists of a single definition:

```
agent A = a.b.A;
```

In contrast to the Edinburgh Concurrency Workbench tool, *Inez Editor* does not use apostrophes to indicate output ports. Input and output ports are distinguished in the graphical layer. Input ports are represented by circles, while output ones are represented by squares.

The graphical layer of an XCCS model takes the form of an undirected graph. Vertices (ovals) represent agents and edges represent connections among agents. In the XCCS language all connections among agents are defined using the graphical layer. All connections must be defined explicitly and all necessary relabel functions are defined automatically. A communication channel can be defined between any two agents, if the channel connects ports of different type (an input and an output port).

The restriction operator is represented by different colours of ports. White (open) ports are ones that can be used to interact with the environment. On the other hand, black (closed) ports can be used only for the internal communication among agents.

More information about advantages of XCCS and our motivations for defining the language has been also presented in [7].

3 Using Inez XCCS Editor

The *Inez XCCS Editor* is being developed in the Java language with the use of the NetBeans IDE [4] as the development tool. *Inez* is a free software covered by the GNU Library General Public License. The tool is being developed at AGH University of Science and Technology in Kraków, Poland. *Inez home website*, hosting information about current status of the project, is located at *http://fm.ia.agh.edu.pl*. The current version of the tool can also be download from the website. It has been prepared using the Java Web Start technology, so the *Inez XCCS Editor* can be automatically download and run using any Web browser.

An example of *Inez* session is shown in Fig. 1. *Inez* is equipped with full-featured graphical editor that provides all features for creation and manipulation of XCCS diagrams. Moreover, the editor is equipped with several tabs and dialogs to manage the algebraic layer of XCCS models. For example, the *Agents tab* presented in Fig. 1 is used to preview definitions of agents used in the current diagram.

3.1 Creating diagrams

This subsection describes how to use *Inez Editor* to create an XCCS diagram. *Inez* starts with an empty diagram. If necessary, create a new diagram by clicking on *New* from the *File* menu. You will see an empty grid. To open an existing file, choose *Open* from the *File* menu. Double click the desired file in the *Open dialog* or select the file and click OK.

3.1.1 Creating agents and ports

As it was said before, the graphical layer takes the form of an undirected graph of agents and connections among them. An agent is graphically represented by an oval with the agent name inside. To add a new agent to the current diagram select *Agent* from the *Diagram* menu. Alternatively, you can click the *Agent* icon in the toolbar. The new agent is located near top and left edges of the drawing area as shown in Fig. 2.

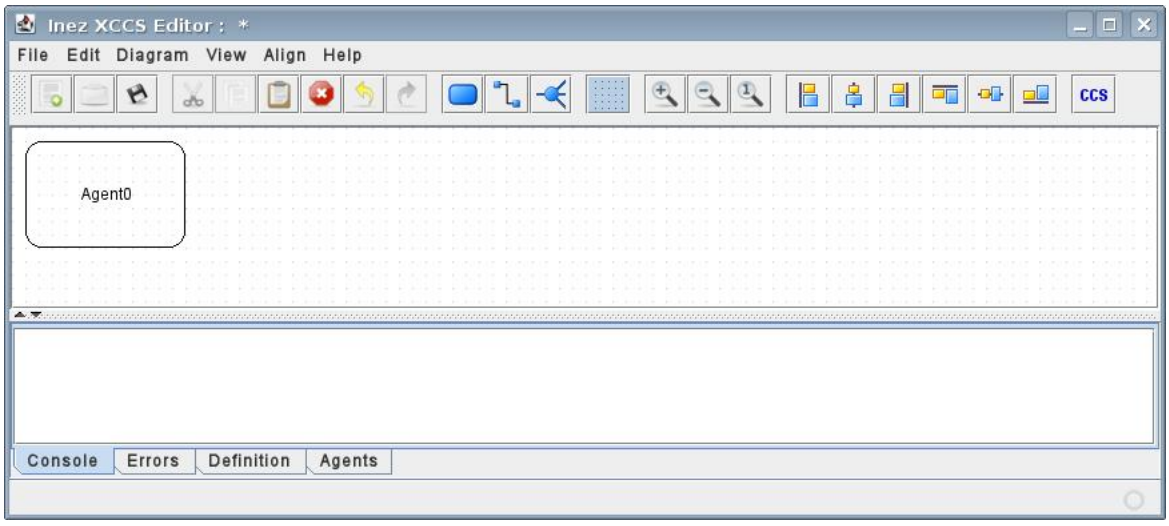

Figure 2: New agent added to the diagram

Select the agent by clicking and releasing the left mouse button. The agent appears with eight resize handles and green highlighting signifying that it is selected. Use the mouse the to move the oval to a desired position and/or resize it.

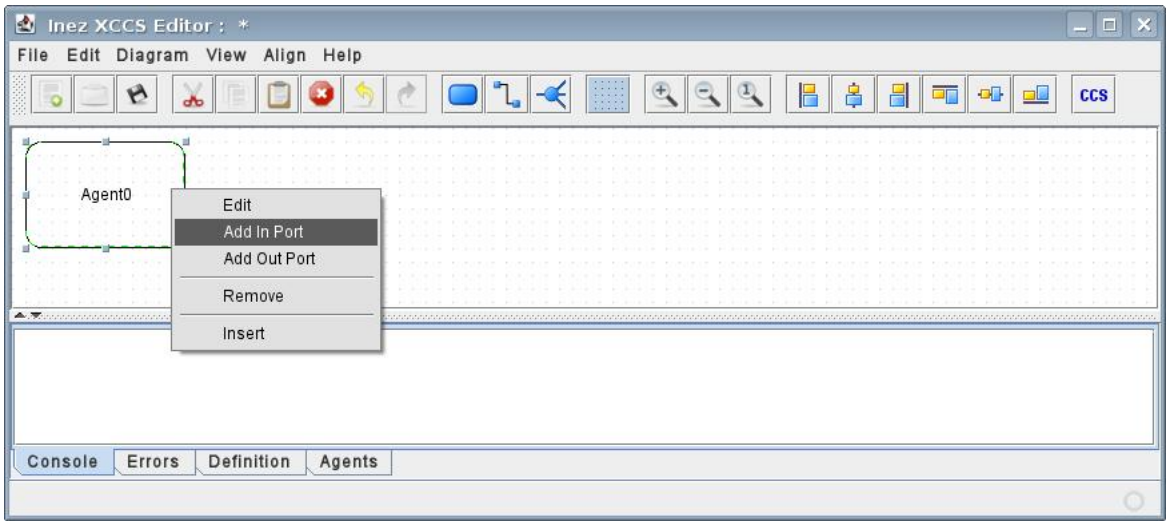

Figure 3: Pop-up agents' menu

Instead of overbars used in CCS to denote output ports, different shapes of ports are used in XCCS. Input ports are represented by circles, while output ones are represented by squares. The agents' pop-up menu (see Fig. 3) is used to manipulate agents' properties, e.g. adding new input and output ports. To add a new input port to an agent select *Add In Port* from its pop-up menu. Then, use mouse to move the port to a desired position. The port label (with default value) is moved automatically (see Fig 4).

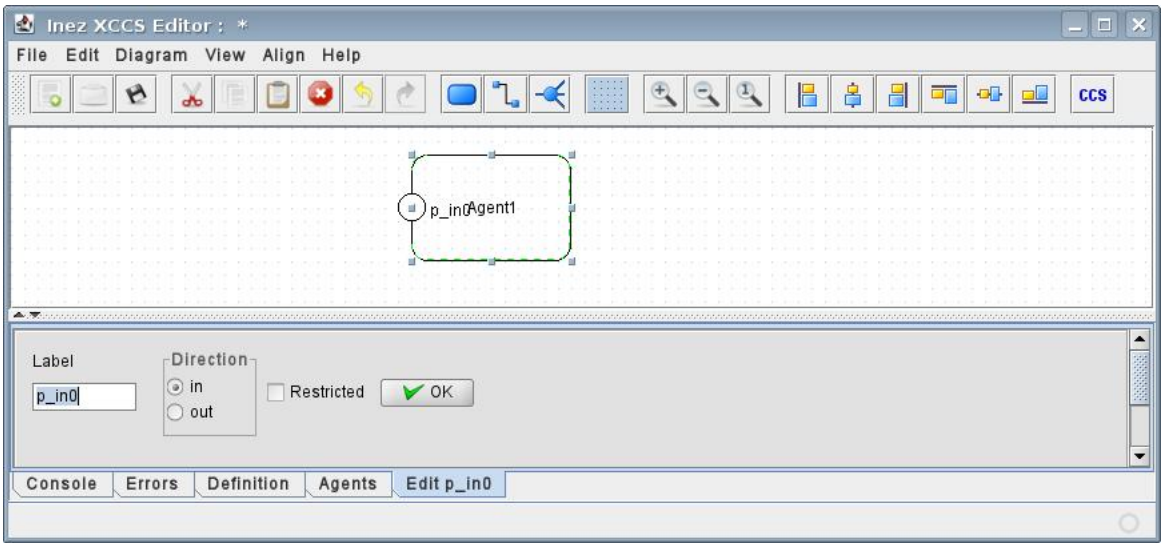

Figure 4: Edit port tab

Double click the new added port to open *Edit port tab* (see Fig 4). It is used to set ports' properties such as name, direction and restriction. The restriction operator is represented by different colours of ports. White (open) ports are ones that can be used to interact with the environment. On the other hand, black (closed) ports can be used only for the internal communication among agents. The fastest way to switch a port from open to close and vice versa is double clicking it while the *Ctrl* key is pressed.

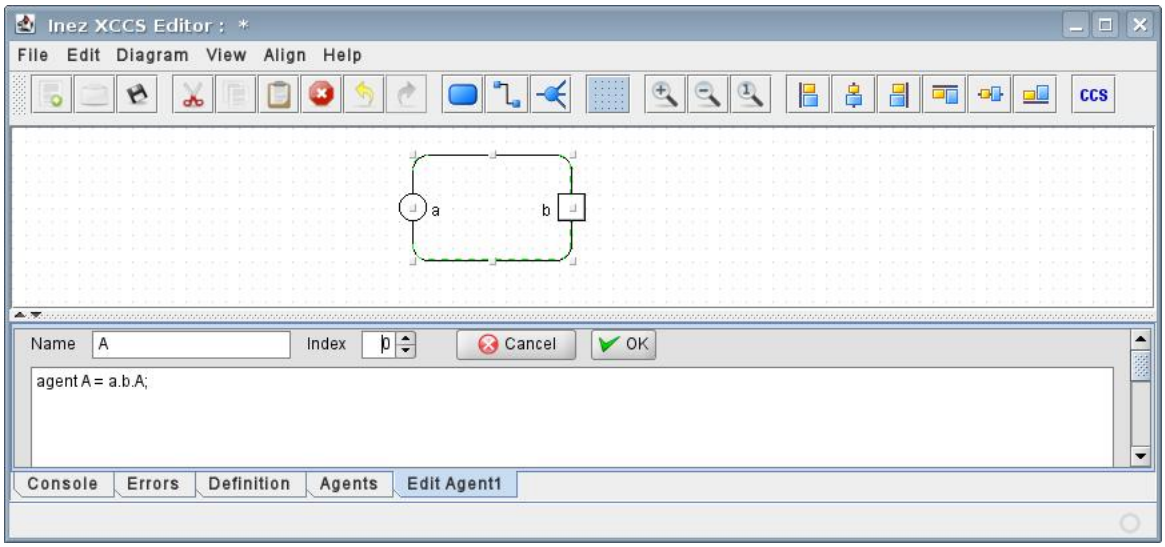

Figure 5: Edit agent tab

Double click an agent to open *Edit agent tab* (see Fig 5). It is used to set agents' properties such as name, index and definition. The index is used to put agents in order. The main part of the *Edit agent tab* is the multi-lines editor used to define the agent behaviour. A definition consists of a sequence of algebraic equations. See Section 4 for more details.

3.1.2 Basic editing commands

The *Edit* menu can be used among other things to copy, cut, paste or delete an selected agent or a selected part of a diagram. It is enough to click an agent to select it. Multiple selection can be made with the mouse alone or with the keyboard and mouse. To made a multiple selection with the mouse alone, click on an empty space of the diagram and drag to include desired shapes in the dashed-line rectangle. When the keyboard and mouse are used, click on the shapes to select them while the *Shift* or *Ctrl* key is pressed. After selecting a shape, you may click it again to deselect it. Select *Select All* from the *Edit* menu or press $Ctrl + a$ to select whole diagram.

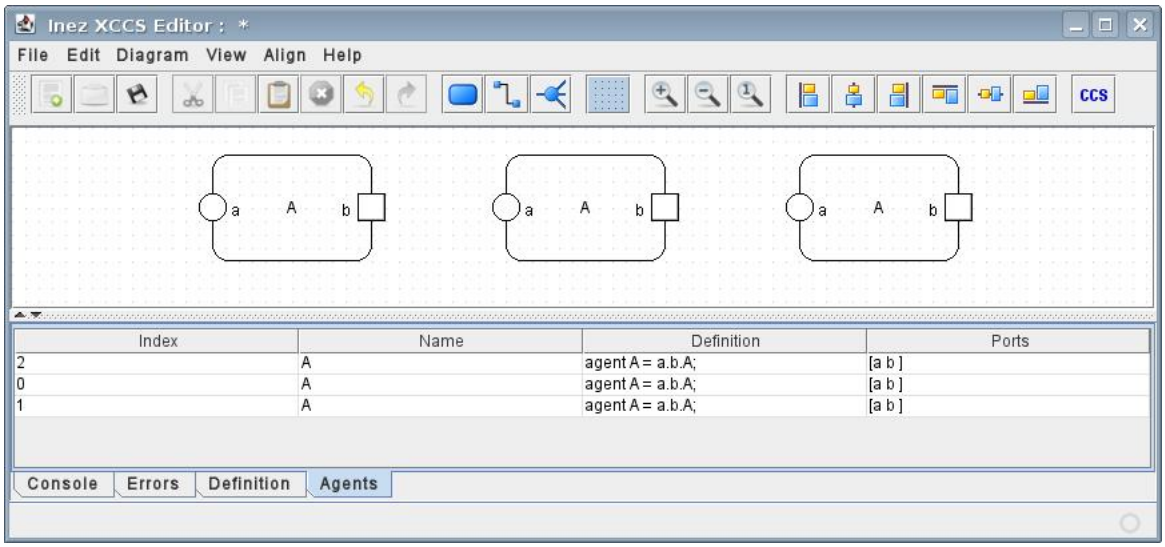

Figure 6: XCCS diagram with three copies of agent A

To copy the selected diagram elements to the clipboard, select *Copy* from the *Edit* menu. Alternately, you can use the *Copy* icon in the toolbar or press *Ctrl* + *c* or move the selected diagram elements whit the mouse while the *Ctrl* key is pressed. To cut the selected diagram elements to the clipboard, select *Cut* from the *Edit* menu or use the *Cut* icon in the toolbar or press *Ctrl* + *x*.

The clipboard contents can be inserted into the diagram. To do so, select *Paste* from the *Edit* menu. Alternately, you can use the *Paste* icon in the toolbar or press *Ctrl* + *v*. An XCCS diagram with three copies of agent A is shown in Fig. 6. If necessary, the algebraic definition can be changed for each of the agents regardless of the others.

Selected diagram elements can be easily moved with the mouse. When an element is moved out of the right or bottom border of the diagram, the size of the diagram is changed automatically and scroll bars are displayed if necessary.

To remove selected diagram elements, perform one of the following actions: select *Delete* from the *Edit* menu or use the *Delete* icon in the toolbar or press *Delete*.

When you create and edit a diagram, you may make mistakes like accidentally deleting a diagram element. You can use the *Undo* function to cancel the previous action. The *Inez XCCS Editor* can take back more than one command. On the other hand, you may re-perform the action using the *Redo* action. More than one command can also be re-performed.

3.1.3 Creating connections

To switch to the *Connection editting mode* select *Connection* from the *Diagram* menu or use the *Connection* icon in the toolbar. To create a connection click on the source port and drag the connector to the destination one (see Fig. 7). If the connection is valid it is placed onto the diagram.

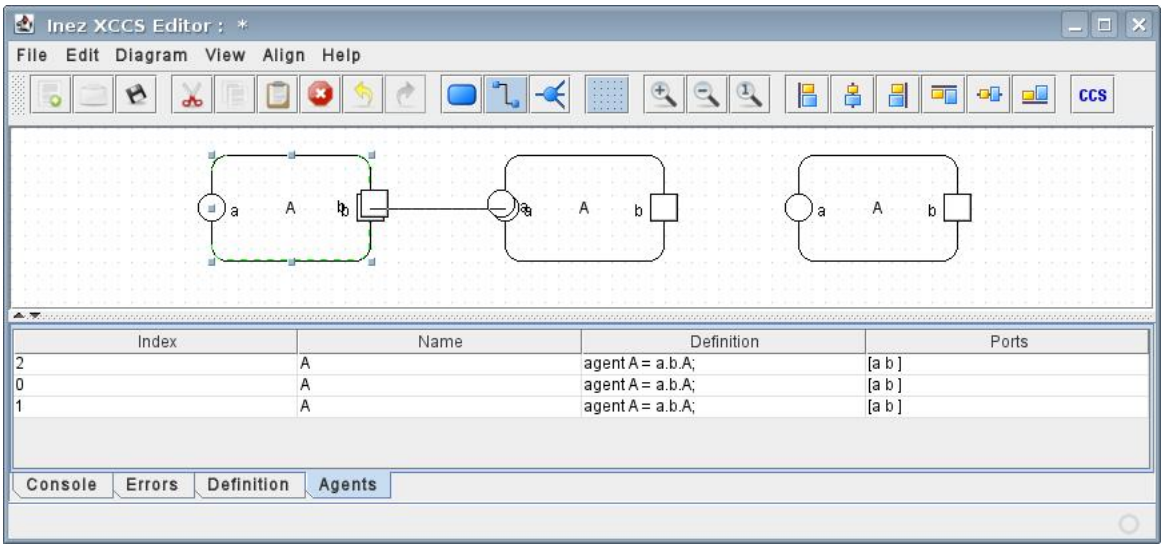

Figure 7: Inez XCCS Editor in *Connection editting mode*

A connection can take the form of a line or a Bezier curve. To add/remove a curve control point, click the line (curve) while the *Shift* button is pressed. Move the control points to manipulate the curve shape.

A special graphical element called *junction* is used in XCCS to represent *one to many* or *many to one* connections. A junction is graphically represented by a small circle. To add a new junction to the current diagram select *Junction* from the *Diagram* menu. Alternatively, you can click the *Junction* icon in the toolbar. The new junction is located near top and left edges of the drawing area as shown in Fig. 8. Use the mouse to move it to a desired position.

You can switch to the *Connection editting mode* and create a connection from a port to a junction or from a junction to a port. An example of the one to many connection is show in Fig. 9. The

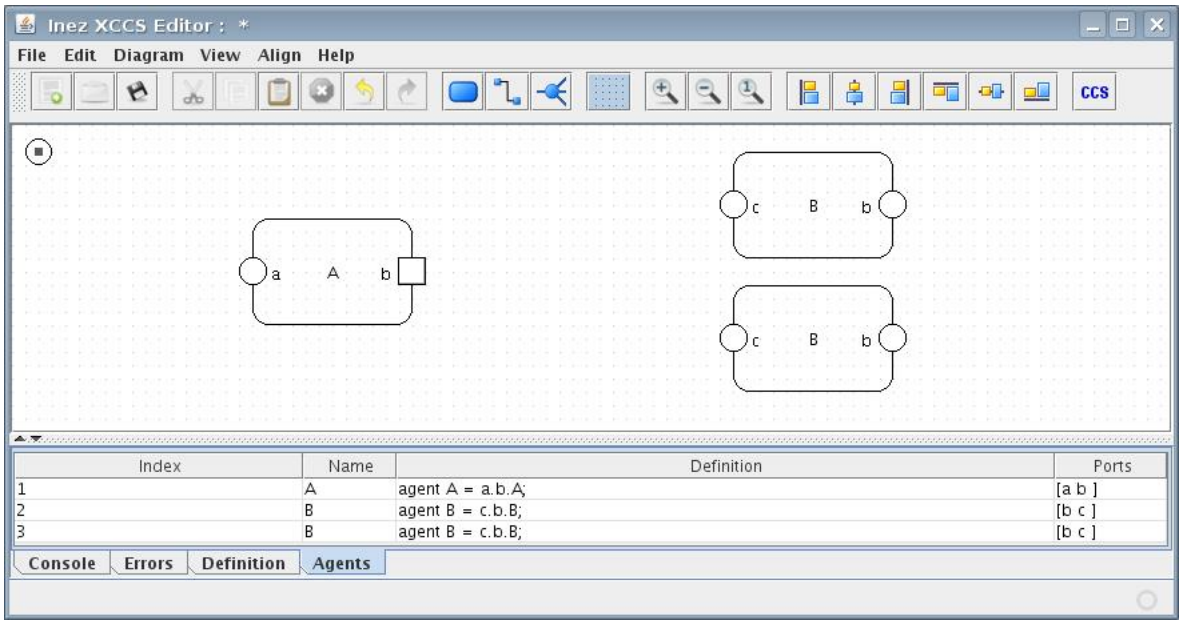

Figure 8: Junction element placed onto the diagram

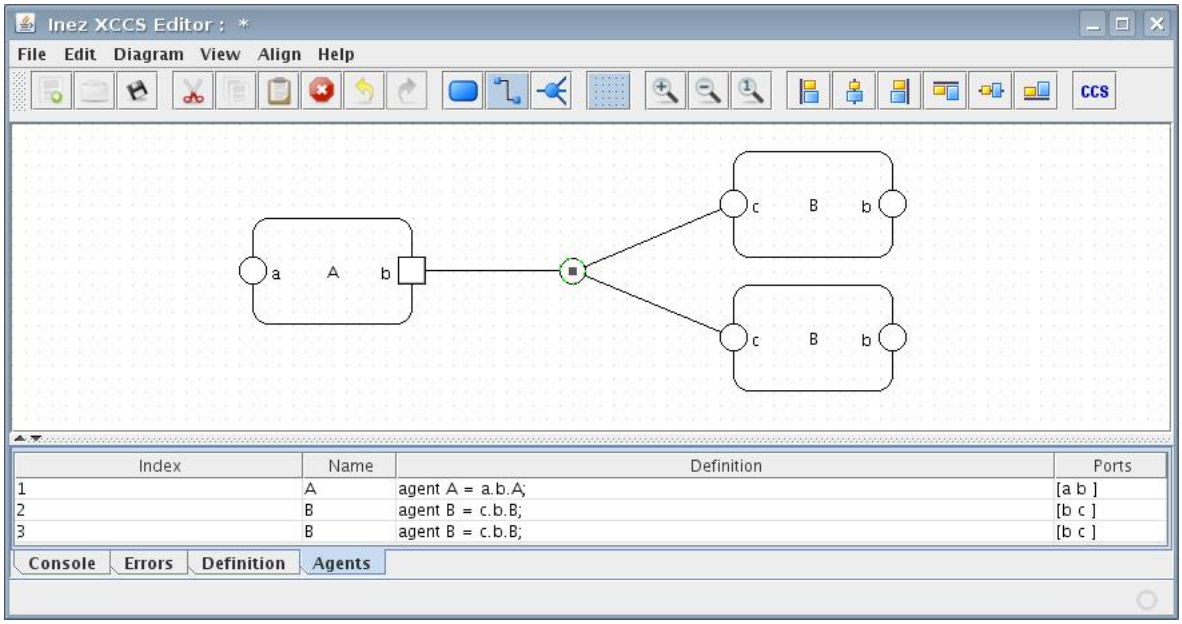

Figure 9: One to many connection

connection means that agent `A` sends some information through port `b` to both agents `B` at the same time.

On the other hand, when a many to one connection is used, an agent collects some information from two or more agents at the same time. Such a connection is presented in Fig. 10. Agent `A` collects information through port `b` from all agents `B` at the same time.

If a port is connected with a junction element, it cannot be connected with any other junction element or any other port. There is no *many to many* connections in XCCS. If a junction element is used, then there is exactly one input or exactly one output port connected to it. Moreover, a connection cannot be established between two junction elements.

A connection can be established only between two ports of different agents (directly or with the use of a junction element) and an open port cannot be connected with a closed one. If a port is

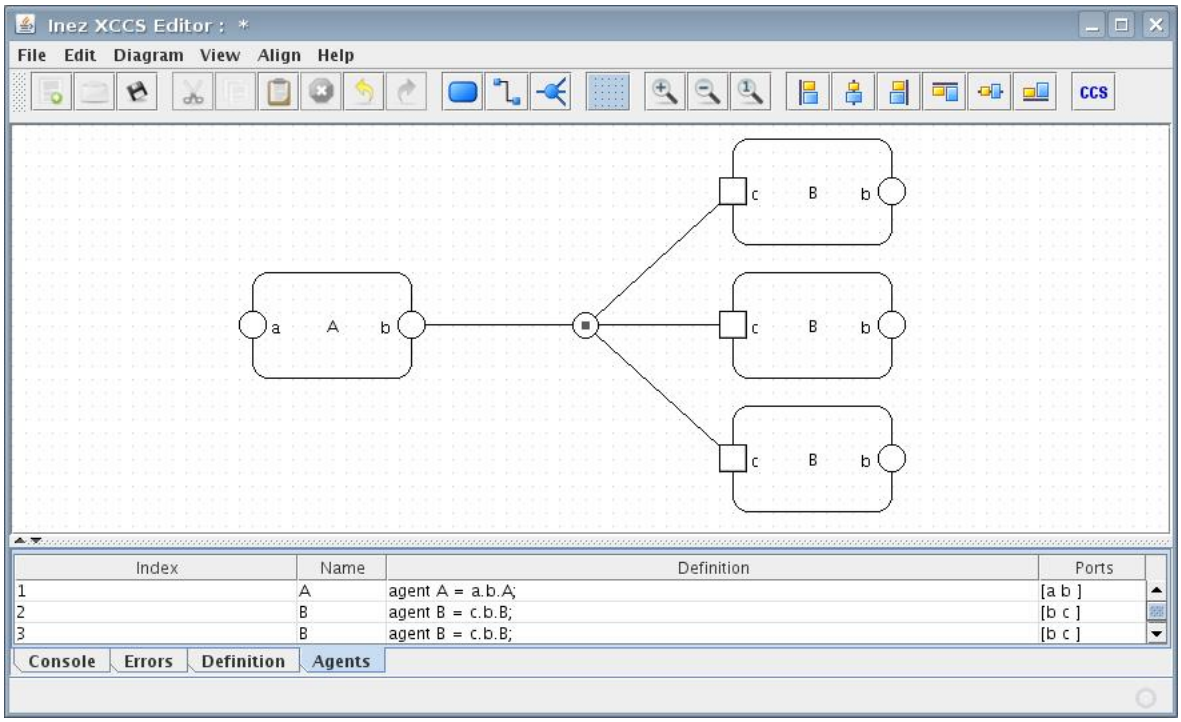

Figure 10: Many to one connection

connected with more then one complementary port, then each of them must belong to a different agent.

Connections among agents can be treated as a relation defined on the set of agents. The relation must satisfy the so-called *asymmetrical three-transitive condition*. It means that if four ports are connected in such a way that the connections form a path, then the first and the last port must also be connected. The condition is illustrated in Fig. 11. The connections between ports a and b, b and c, c and d form a path, so a connection between ports a and d (represented in the figure by the dashed line) must also be added to the diagram.[1]

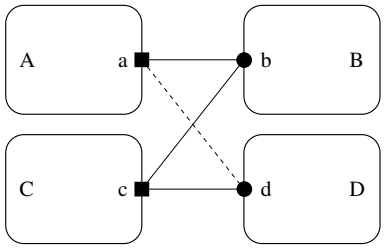

Figure 11: Asymmetrical three-transitive condition.

3.1.4 Zooming

The *View* menu and toolbar offer some tools to adjust the viewport of the current diagram. All of these tools have no effect on the geometry of the entities of the diagram. They only change the zoom factor and the visible area (viewport).

- The *zoom in* feature allows you to get a close-up view of the diagram. To perform zoom in, select *Zoom In* from the *View* menu or use the *Zoom In* button from the toolbar.

[1] Some of the remarks are thanks to Krzysztof Balicki from Rzeszów University, `kbalicki@univ.rzeszow.pl`

- The zoom out feature allows you to see more of the diagram at a reduced size. To perform zoom out, select *Zoom Out* from the *View* menu or use the *Zoom Out* button from the toolbar.
- The zoom to 100% feature allows you to view the diagram in its actual size (100%). To restore the zoom ratio to 100%, select *Zoom* $1:1$ from the *View* menu or use the *Zoom* $1:1$ button from the toolbar.

3.1.5 Aligning elements

The *Inez XCCS Editor* is equipped with *Show Grid* and *Snap to Grid* options. They allow for more precision in positioning elements on your diagram. When the *Snap to Grid* option is enabled, the elements can easily be positioned and resized along the grid interval. To toggle the visibility of grid lines, select *Show Grid* from the *View* menu. Alternatively, you can click the *Show Grid* icon in the toolbar. To turn the snap to grid option on/off select *Snap to Grid* from the *View* menu.

Aligning elements in the diagram can be also done with the use of alignment options in the *Align* menu. Alternatively, you can use the alignment icons in toolbar. The alignment options may be used to align elements in relation to other elements.

Left – The option aligns elements according to the left border of the first selected element.

Center – The option aligns elements according to the center of the first selected element (horizontally).

Right – The option aligns elements according to the right border of the first selected element.

Top – The option aligns elements according to the top border of the first selected element.

Middle – The option aligns elements according to the center of the first selected element (vertically).

Bottom – The option aligns elements according to the bottom border of the first selected element.

3.1.6 Export formats

The *Inez XCCS Editor* allows for exporting diagrams into *Xfig* format, which is helpful in documenting tested cases. To export your diagram into the format select *Export ...* from the *File* menu and then select *Export to Xfig*.

One of the most important advantages of the editor is the possibility of generation of CCS scripts automatically. To transformate the current diagram into the corresponding CCS script, select *Export ...* from the *File* menu and then select *Export to CCS* option. Alternatively, you can click the *CCS Export* icon in the toolbar. The CCS script is written into the file with the same name as the current XCCS diagram but with the *ccs* extension. More details about the transformation procedure can be found in Section 5.

3.2 Editor menu

The main window has six items: *File Menu*, *Edit Menu*, *Diagram Menu*, *View Menu*, *Align Menu* and *Help Menu*. Each menu and its submenus is described below.

3.2.1 File Menu

New – Creates a new empty diagram;

Open – Opens an existing diagram;

Save – Saves the current diagram;

Save As ... – Saves the current diagram with a new name;

Export ... – Exports the current diagram into various formats;

Export to Xfig – Exports the current diagram to Xfig format;

Export to CCS – Generates the CCS script;

Exit – Exits the application.

3.2.2 Edit Menu

Undo – Undoes the last operation;

Redo – Redoes the last undone operation;

Cut – Cuts the selected elements to the clipboard;

Copy – Copies the elements to the clipboard;

Paste – Inserts the elements from the clipboard;

Delete – Deletes the selected elements;

Select All – Selects all elements;

Preferences – Shows the application's preferences dialog.

3.2.3 Diagram Menu

Agent – Adds a new agent;

Connection – Creates a new connection;

Junction – Adds a new junction.

3.2.4 View Menu

Zoom In – Zooms in the diagram;

Zoom Out – Zooms out the diagram;

Zoom 1:1 – Resets the zoom to the default value;

Show Grid – Shows/Hides the grid;

Snap to Grid – Enables/Disables the Snap to grid option.

3.2.5 Align Menu

Left – Lays out the selected elements to the left;

Center – Center the selected elements horizontally;

Right – Lays out the selected elements to the right;

Top – Lays out the selected elements to the top;

Middle – Center the selected elements vertically;

Bottom – Lays out the selected elements to the bottom.

3.2.6 Help Menu

About ... – Shows the application's information dialog.

3.2.7 Pop-up Menu

Contents of the menu may change depending on the current context.

Edit – Opens *Edit agent tab*;

Add In Port – Adds an input port;

Add Out Port – Adds an output port;

Remove – Removes the agent;

Remove Port – Removes the port;

Insert – Adds an agent to the diagram.

3.3 Data format

Inez Editor uses XML format to store XCCS diagrams. The root element of such an XML file is the *xccsdiagram* tag, which contains sets of agents, junctions and edge elements. Each *agent* tag holds information about one agent:

- *name, size, position* – represented as attributes;
- *agent definition* – represented as separate tag;
- *set of ports* – represented as separate tags;

The port tag contains only attributes and provides following data:

- *name* – port label;
- *id* – used for reconstructing connections upon load;
- *position* – relative to the agent;
- *direction* – in or out;
- *open/restricted state*.

The junction tag contains information about its *id*, *size* and *position*, while the edge tag holds *id* of *source* and *target* port. A piece of XML code describing a diagram (the three cell queue buffer model) is presented below:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE xccsgraph PUBLIC "inezPublicId-v0.1" "inezSystemId-v0.1">

<xccsgraph version="0.32">
  <agent height="80.0" index="1" name="A" width="120.0" x="360.0" y="30.0">
    <definition>agent A = a.b.A;</definition>
    <port direction="in" id="28259286" name="a" restricted="true" x="0.0" y="500.0"/>
    <port direction="out" id="13080585" name="b" restricted="true" x="1000.0" y="500.0"/>
  </agent>
  <agent height="80.0" index="2" name="A" width="120.0" x="570.0" y="30.0">
    <definition>agent A = a.b.A;</definition>
    <port direction="in" id="31365828" name="a" restricted="true" x="0.0" y="500.0"/>
    <port direction="out" id="4047035" name="b" restricted="false" x="1000.0" y="500.0"/>
  </agent>
  <agent height="80.0" index="0" name="A" width="120.0" x="140.0" y="30.0">
    <definition>agent A = a.b.A;</definition>
    <port direction="in" id="26530674" name="a" restricted="false" x="0.0" y="500.0"/>
    <port direction="out" id="27165481" name="b" restricted="true" x="1000.0" y="500.0"/>
  </agent>
  <edge source="27165481" target="28259286">
    <point x="259.0" y="69.5"/>
    <point x="360.0" y="69.5"/>
  </edge>
  <edge source="13080585" target="31365828">
    <point x="479.0" y="69.5"/>
    <point x="570.0" y="69.5"/>
  </edge>
</xccsgraph>
```

The appropriate DTD file is also provided for input file validation and graph reconstruction with Xerces XML parser.

4 Algebraic layer

The algebraic layer is used to define the behaviour of individual agents and takes the form of algebraic equations like in CCS. A limited set of CCS operators is used for this purpose. XCCS works both with the untimed and timed version of CCS. The operators: prefix (`.`), choice (`+`), strong choice (`++`) and delay (`$`) are used in XCCS in the same way as for the CCS language ([1], [3], [5], [6]). Moreover, the *interleaving operator* has been added to the XCCS language. The operator is denoted by the question mark.

The agent syntax supported by *Inez* is very close to the syntax used in CWB tool, however, some modifications or/and restrictions have been introduced.

4.1 Identifiers

Identifiers are used as agents' or ports' names. Identifiers for agents begin with a capital letter A-Z, while identifiers for ports begin with an lower letter a-z. The second and subsequent characters of an identifier may be: small letters, capital letters, the digits 0-9, and the underscore '_' character. The syntax diagrams for identifiers are shown in Fig. 12. The underscore character is used by the transformation algorithms, so it is recommended to use only alphanumeric characters.

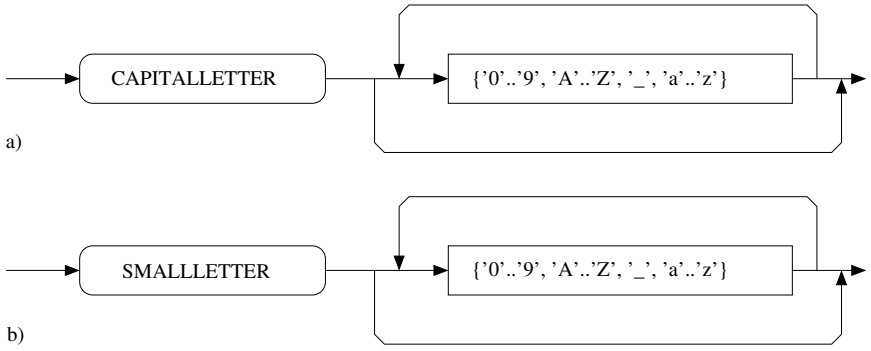

Figure 12: Syntax diagrams – identifiers: a) AGENTNAME, b) PORTNAME

Some identifiers are treated as *keywords* and cannot be used as identifiers. The set of keywords contains: *eps, tau, T, F, Internals, System, agent, set, label, type*. Please note that it is possible that the set of keywords may be extended in future, when the value-passing version of the *Inez XCCS Editor* will be developed. Moreover, 0 is used to denote the null agent that cannot perform any action.

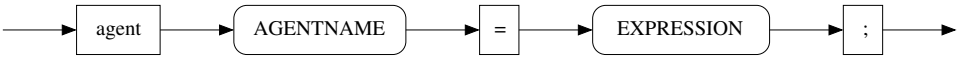

Figure 13: Syntax diagrams – agent definition

The syntax diagram for an agent definition is shown in Fig. 13. The EXPRESSION stands for an expression that describes the agent behaviour and will be presented later in this section. Using the already presented elements, we can write for example the following definitions:

```
agent A = 0;
agent B = C;
```

Agent `A` cannot perform any action, while `B` performs in the same way as `C`.

4.2 Time delays

Only discrite times are allowed in the XCCS modelling language. Time delays are represented by natural (positive) numbers. Moreover, the delay operator (see Subsection 4.3) can be used to denote the unlimited delay.

4.3 Operators

The *prefix operator* (`.`) specifies the ordering of actions and events. The left argument of the prefix operator must be an action or time delay. The right argument must be an expression that defines how the agent performs after the left side action or delay.

```
agent A = a.0;
agent B = a.3.b.B;
```

The agent `A` shown in the preceding listing performs action `a` and then stops, while `B` performs sequentially: action `a`, delays for 3 time-units and then performs `b`.

The XCCS modelling language (like CCS) provides two choice operators: weak `+` and strong `++` choice operator. The two choice operators differ in the manner they handle delays ([3]). Strong choice `A ++ B` allows a delay only if both `A` and `B` are capable of that delay, while weak choice `A + B` allows also a choice to be made in favour of the agent which can delay the longest. Let us consider the following examples:

```
agent A = a.B + b.0;
agent C = 5.a.C + 7.b.C;
agent D = 5.a.D ++ 7.b.D;
agent E = 5.a.E;
```

Agent `A` performs action `a` and then performs like `B` or it performs action `b` and then stops. Agent `C` delays for 5 time-units and then performs `a` or delays for 7 time-unit and then performs `b`. `C` repeats this behaviour an unspecified number of times. Agent `D` can delay only 5 time-unit and then it has to perform `a`. Thus, `D` performs exactly in the same way as `E`.

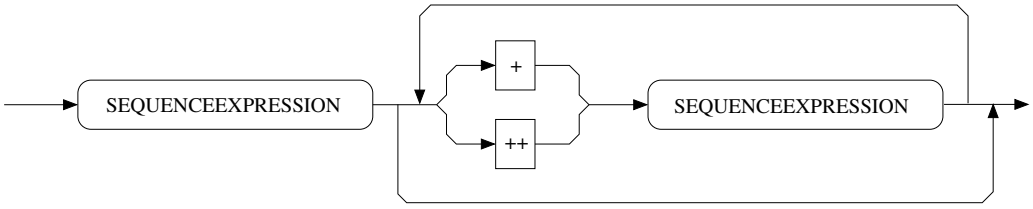

Figure 14: Syntax diagrams – EXPRESSION

As shown in Fig. 13 the main part of the right-hand side of the agent definition is an EXPRESSION. Such an EXPRESSION is defined as a finite sum (both choice operators can be used) of SEQUENCEEXPRESSIONs. The syntax diagrams for EXPRESSION and SEQUENCEEXPRESSION are presented in Fig. 14 and Fig. 15 respectively. As shown in Fig. 15 an EXPRESSION can be put in parenthesis.

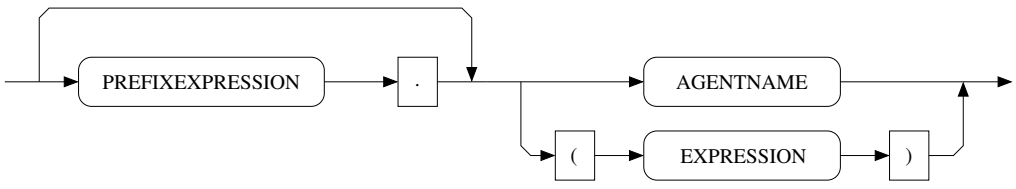

Figure 15: Syntax diagrams – SEQUENCEEXPRESSION

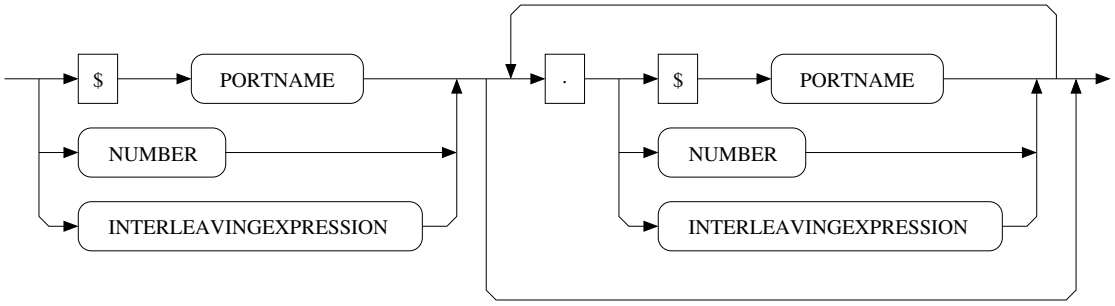

Figure 16: Syntax diagrams – PREFIXEXPRESSION

As said before, the left argument of the prefix operator must be an action or time delay. More precisely, it can be the so-called PREFIXEXPRESSION (see Fig. 16). To explain the PREFIXEXPRESSION in details two last operators must be introduced. Let us focus on the delay operator `$` first. It is used to indicate the possibility of an infinite delay before an action is performed.

```
agent A = $a.A;
```

The agent `A` shown in the preceding listing can perform the action `a` immediately or it can delay for any number of time-units and then perform `a`.

The last operator used in the algebraic layer is called *interleaving operator* and is denote by the question mark. The operator can take any number of arguments. Let us consider the following examples:

```
agent B = ?(a,b).B;
agent C = ?(a,b,c).C;
agent D = ?(?(a,b),c).D;
```

Each presented equation can be written in equivalent form with the use of the choice operator:

```
agent B = a.b.B + b.a.B;
agent C = a.b.c.C + a.c.b.C + b.a.c.C + b.c.a.C + c.a.b.C + c.b.a.C;
agent D = a.b.c.D + b.a.c.D + c.a.b.D + c.b.a.D;
```

The interleaving operator is especially useful when an agent can perform a set of actions in any order. The syntax diagram for the INTERLEAVINGEXPRESSION is shown in Fig. 17.

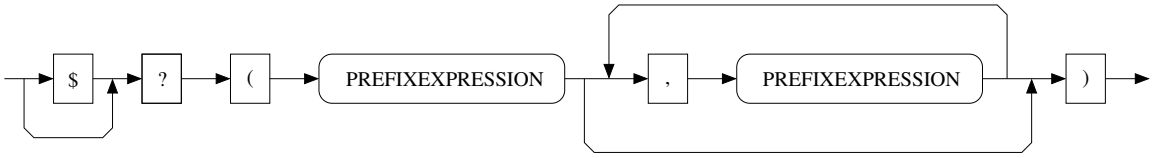

Figure 17: Syntax diagrams – INTERLEAVINGEXPRESSION

The behaviour of an agent can be defined as a set of equalities, e.g:

```
agent A = a.A1;
agent A1 = a.A2 + b.A3;
agent A2 = b.A;
agent A3 = a.A;
```

Labels `A1`, `A2`, etc. will be called *sub-names*, while `A` will be called (main) name. Such a definition must fulfil the following requirements:

1. The first equation must define the agent's name, i.e. the first defined name must be the same as the name of the agent in the graphical layer.

2. Definition of each agent must be independent of the others, i.e. the sets of sub-names of any two agents with different names must be disjoint.

It is also recommended to use the agent's name with an index as a sub-name. Moreover, the underscore sign should not be used by designers because it is used by the transformation algorithms.

5 Transformation algorithms

The *Inez XCCS Editor* is able to export XCCS diagrams to the corresponding CCS scripts, which can be executed via the Edinburgh Concurrency Workbench tool [6]. The transformation procedure consists of a few steps:

1. Elimination of junction elements – If a diagram contains junction elements, they are eliminated as presented in Fig 18. If agent `A` is defined as `agent A = a.A;`, then the algebraic definition takes the following form: `agent A = ?(a_1,a_2).A;`. It means that actions `a_1` and `a_2` can be performed in any order and there is 0 time units delay between them.

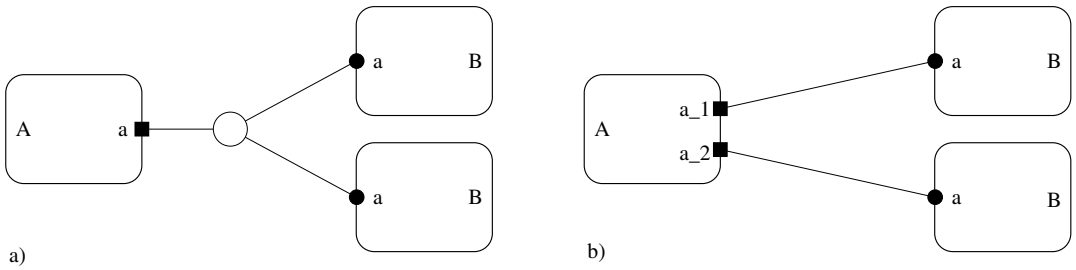

Figure 18: Elimination of a junction element

2. Numbering of agents – If a diagram contains a few instances of the same agent, each of them is assigned an individual number. If the algebraic definition of an agent consists of a few equations, then the same index is attached to each agent that appears in the definition.
3. Evaluating agents' priorities – If connected ports have different labels, then some of them are renamed. The transformation algorithm uses a priority function defined on the set of agents. Two different algorithms for evaluating agents' priorities are implemented at the moment. Firstly, agents' indexes can be used for this purpose – a lower agent index is the higher is its priority. Moreover, the priority of an agent A can be evaluated as follows:

$$P(A) = N_P + N_C - N_I \tag{1}$$

where:

- N_P is the number of ports of the agent A;
- N_C is the number of connections of the agent A;
- N_I is the number of instances of the agent A in the model.

4. Relabelling connected ports – Port labels of agents with a higher priority are inherited by ports of agents with a lower one. The relabel function works recurrently.
5. Elimination of undesired connections – The second relabelling stage deals with eliminating undesirable connections. If it is necessary, some ports are renamed (an extra index is added to a label) to avoid connections that were not defined explicitly.
6. Elimination of the interleaving operator – Each occurring of the interleaving operator is replaced with an equivalent expression with the choice operator (see Subsection 4.3 for details).

7. CCS script generation – First of all, the prime sign is added to each output port. Then, the `Internals` set is prepared. The set contains names of all restricted labels. The agent that represents the whole modelled system is called `System`. It is defined as the composition of all agents placed in the considered diagram.

A detailed description of the transformation algorithm can be found in [8].

6 Examples

This section presents three simple examples of XCCS models: the Readers/Writers problem, Producer/Consumer problem and Automatic Train Stop system. Other examples can be also found in [8].

6.1 The Readers/Writers problem

The problem consists of readers and writers that share a data resource. The readers can only read from the resource, while the writers can only write to it. In the considered example at most three readers can access the resource simultaneously, but a writer must have exclusive access to the resource.

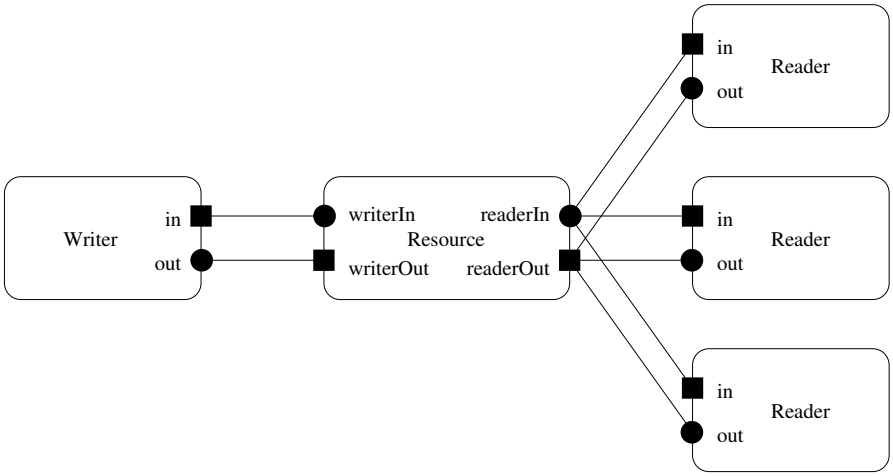

Figure 19: The Readers/Writers problem – XCCS diagram

The XCCS diagram of the Readers/Writers model is shown in Fig. 19. A state of agent `Resource` stores information about the current resource users, e.g. the number of readers using it. The algebraic layer of the model is presented in the following listing.

```
agent Reader = in.out.Reader;

agent Resource = writerIn.writerOut.Resource + readerIn.Resource1;
agent Resource1 = readerIn.Resource2 + readerOut.Resource;
agent Resource2 = readerIn.Resource3 + readerOut.Resource1;
agent Resource3 = readerOut.Resource2;

agent Writer = in.out.Writer;
```

6.2 The Producer/Consumer problem

In the Producer/Consumer problem, two processes share a fixed-size buffer. Both processes work concurrently, one process produces information and puts it in the buffer, while the other process consumes information from the buffer. The producer must not put an item into a full buffer and the consumer must not take an item from an empty buffer. In the considered example the buffer capacity is equal to four.

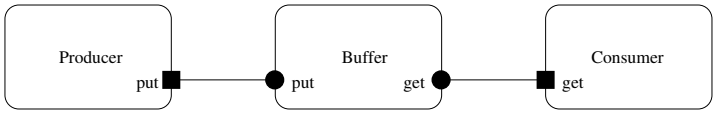

Figure 20: The Producer/Consumer problem – XCCS diagram

The XCCS diagram of the Producer/Consumer model is presented in Fig. 20. A state of agent `Buffer` stores information about the current buffer state, e.g. the number of items in the buffer. The algebraic layer of the model is presented in the following listing.

```
agent Producer = put.Producer;

agent Buffer = put.Buffer1;
agent Buffer1 = put.Buffer2 + get.Buffer;
agent Buffer2 = put.Buffer3 + get.Buffer1;
agent Buffer3 = put.Buffer4 + get.Buffer2;
agent Buffer4 = get.Buffer3;

agent Consumer = get.Consumer;
```

6.3 Automatic train stop system

The Automatic Train Stop system (ATS system [7]) is a railway technical installation to ensure safe operation in the presence of human failures. The ATS system uses cab signalling to check whether the driver controls the train. It turns on a light signal every 60 seconds. If the driver fails to acknowledge the signal within 6 seconds, a sound signal is turned on. Then, if the driver does not disactivate the signals within 3 seconds, using the acknowledge button, the emergency brake is applied to stop the train automatically.

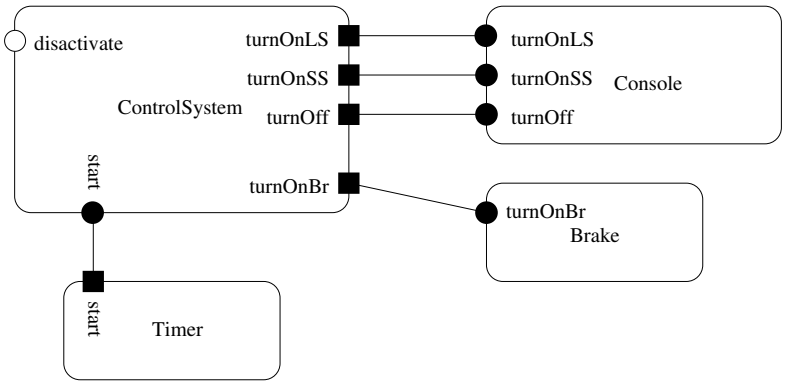

Figure 21: Automatic train stop system – XCCS diagram

The XCCS diagram of ATS model is shown in Fig. 21. It contains four agents: *ControlSystem* represents the control element of the ATS system, *Console* represents the cab console where warning signals are displayed, *Brake* represents a train emergency brake and *Timer* that is used to start the system cycle every 60 seconds.

The names of ports stand for the following actions: *turnOnLS* – turn on the light signal, *turnOnSS* – turn on the sound signal, *turnOff* – turn off signals, *turnOnBr* – turn on the emergency brake, *disactivate* – disactivate warning signals, and *start* – start a new cycle. Port *disactivate* is used to communicate with the system environment and represents disactivation of the warning signals by the driver.

The algebraic layer is presented below:

```
agent ControlSystem = $start.turnOnLS.ControlSystem1;
```

```
agent ControlSystem1 = $disactivate.turnOff.ControlSystem
                     ++ 6.turnOnSS.ControlSystem2;
agent ControlSystem2 = $disactivate.turnOff.ControlSystem
                     ++ 3.turnOnBr.0;

agent Console = $turnOnLS.($turnOff.Console + $turnOnSS.$turnOff.Console);
agent Brake = $turnOnBr.0;
agent Timer = start.60.Timer;
```

The ATS system ends in deadlock if the emergency brake is applied. To enable the system to restart after a deadlock, an extra supervisor agent should be included into it.

7 Summary

The *Inez XCCS Editor*, a tool for visual modelling with XCCS process algebra, has been presented in the report. Both a survey of main features of the tool and a detailed description of the XCCS modelling language have been presented. The report corresponds to 0.34 version of the tool. This version of *Inez XCCS Editor* is equipped with a graphical editor for the design of XCCS diagrams and transformation algorithms for exporting XCCS models into CCS scripts compatible with Edinburgh Concurrency Workbench. Both timed and untimed versions of CCS are supported.

The development of the tool is still in progress. Our future plans will focus on the development of the value-passing version of the XCCS language and algorithms for generation of VP CCS scripts.

References

[1] L. Aceto, A. Ingófsdóttir, K. Larsen, and J. Srba. *Reactive Systems: Modelling, Specification and Verification*. Cambridge University Press, Cambridge, UK, 2007.

[2] G. Bruns. *Distributed Systems Analysis with CCS*. Prentice Hall, 1997.

[3] C. Fencott. *Formal Methods for Concurrency*. International Thomson Computer Press, Boston, MA, USA, 1995.

[4] Netbeans Team. Netbeans. `http://www.netbeans.org`.

[5] R. Milner. *Communication and Concurrency*. Prentice-Hall, 1989.

[6] F. Moller and P. Stevens. Edinburgh Concurrency Workbench user manual. `http://www.dcs.ed.ac.uk/home/cwb`.

[7] M. Szpyrka and K. Balicki. XCCS – graphical extension of CCS language. In *Proc. of Mixdes 2006, the 14th International Conference Mixed Design of Integrated Circuits and Systems*, pages 688–693, Ciechocinek, Poland, June 21-23 2007.

[8] M. Szpyrka and P. Matyasik. Graphical modelling tool for CCS process algebra. In T. Hruška, L. Madeyski, and M. Ochodek, editors, *Software engineering techniques in progress. Proc. of the 3rd IFIP TC2 Central and East European Conference of Software Engineering Techniques: Brno, Czech Republic*, pages 81–94, Wydawnictwo Politechniki Wrocławskiej, Wrocław, Poland.