GRIN

Olivier Namet

Supervisor : Ian Mackie

April 24, 2007

Contents

Contents **1**

1 Originality Avowal **5**

2 Abstract **7**

3 Acknowledgements **9**

4 Introduction **11**

5 Background **13**
5.1 Interaction Nets . . . 13
5.2 Graphical User Interfaces . . . 14

6 Review **17**

7 Specification & Design **25**
7.1 Requirements . . . 25
7.2 Specification . . . 26
7.3 Design . . . 27

8 Implementation **29**
8.1 GRIN Environment . . . 29
8.2 GRIN Development and Evolution . . . 30
8.3 GRIN Implementation . . . 32

8.3.1 Interaction Net classes . . . 32
8.3.2 Graphical User Interface . . . 33
8.3.3 Save & Load System . . . 34
8.3.4 Exporting & Importing . . . 35

9 Validation 37

10 Evaluation 39
10.1 Functional Evaluation . . . 39
10.2 Usability Evaluation . . . 39
10.3 Environment Evaluation . . . 40

11 Conclusion & Future Work 41
11.1 Conclusion . . . 41
11.2 Future Work . . . 42

Bibliography 43

12 User Guide 45
12.1 Compiling & Running . . . 46
12.2 New, Save and Open File . . . 46
12.3 Canvas . . . 46
12.4 Cell Manager . . . 46
12.5 Cells . . . 47
12.6 Ports . . . 47
12.7 Exporting . . . 48
12.8 Importing . . . 48
12.9 Rules . . . 48

13 Appendix 51
13.1 Use Cases . . . 51
13.2 Class Diagrams . . . 53

13.3 Program Listings 54

Chapter 1

Originality Avowal

I verify that I am the sole author of this report, except where explicitly stated to the contrary

Olivier Namet
25/04/07

Chapter 2

Abstract

Interaction Nets are a type of computational modelling. Nets are graphical by nature but so far very few programs have attempted to allow the creation and editing of nets graphically. GRIN is a program that sets to provide an intuitive *graphical* interface to design interaction nets through a non-standard GUI. This report shows the research, specification, design, development and evolution of GRIN towards achieving this interface. It then looks its limitations and possible expansions.

Chapter 3

Acknowledgements

Ian Mackie, for taking on my project and always providing invaluable advice throughout the development of GRIN.

Maribel Fernandez, for expanding my understanding of Interaction Nets through her Computational Models lecture.

Chapter 4

Introduction

Interaction Nets is an emerging model of computation and offers built-in features that most other computational models don't offer.

A program is defined as a *net* built from a set of *agents*. Due to this concept of nets, a graphical representation emerged but to this day, very few programs tap into the graphical side of Interaction Nets, instead relying on text-based representations.

A graphical representation of Interaction Nets is a lot more intuitive than a text-based one. GRIN was born to allow programmers to code using Interaction Nets using a graphical interface.

Research needed to be done in two main steps:

- What expectations were there for a graphical representation of Interaction Nets?
  - What *are* Interaction nets?
  - What actions can be performed on these nets?
- What kind of graphical interaction system would best suit that representation?
  - What existing graphical interactions are there?
  - What features from these systems should be combined and put into GRIN?

This report shows the research done in those two main steps and then follows with the design and implementation of the features needed to program Interaction Nets graphically. It then finishes by looking at how GRIN could be further extended to provide more features to the user.

Chapter 5

Background

5.1 Interaction Nets

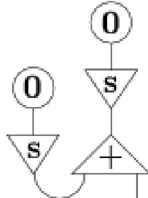
The notion of Interaction Nets came about in the last 20 years. It is a computational model first defined as such by Yves Lafont (CNRS). Comparable to Turing machines in concept, interaction nets can be defined as higher level due to the features they possess. Turing Machines can be represented by a transition table and considered sequential (and therefore linear) in their computation. Interaction Nets on the other hand are a set of cells that are linked together by one or more ports (one of those being a primary port). A set of rewrite rules are also given that can reduce two cells connected to each other by their primary port, these two cells are called an *active pair*.

Since each cell has only one primary port, all active pairs at any one time will never contain a common cell. This makes all the reductions independent of each other and therefore the latter can be done on different processors or systems. Interaction Nets therefore have built in parallelism. They also contain a built in memory management system with the ”erase” cell that will remove all cells that are no longer needed. For example: if we are multiplying a number x by zero, what ever the sequence of calculations to get to x, the result will be zero. Therefore a rewrite rule is written such that an “erase” cell is created to remove all the calculations that lead to x since they are redundant.

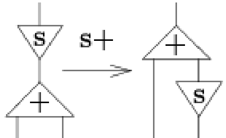
Rewrite rules take an active pair and replaces it by a net. It is governed by two main rules:

1. The reduction doesn’t affect the free ports of the active pair.
2. There can be no more than one reduction rule for every pair of agents.

How 1 + 1 is computed:



Example of a rewrite rule:



5.2 Graphical User Interfaces

Brief history of GUIs

The GUI first appeared in the 60s as a concept thought up by Douglas Englebart. In his essay "AUGMENTING HUMAN INTELLECT: A Conceptual Framework", he describes computers as a tool to enhance human intellect and not as something to replace the latter. In 1968, he publicly unveiled a system called NLS (short for oN-Line System) and demonstrated such features as full screen editing and a multi-window environment. The system had a 5 button keyboard (inputting letter by pressing chords) and a mouse. Such features were unheard of back then, and have paved the way to today's GUIs. Gradually, companies like Apple[1] and Microsoft[2] developed similar GUIs and extended them with extra features. Sadly, the priority for GUI development was more imitation than innovation which leads to today's set of GUIs that all look and function in essentially the same way and have the same base rules. Habit played a factor as well, people got used to this style of Graphical Interaction, and radical new designs would have alienated potential users due to a new learning curve.

[1]http://www.apple.com/
[2]http://www.microsoft.com

Standard GUI

When looking, at GUIs, I will only focus on application GUIs over Operating System ones. Applications have a standard set of rules for their GUI:

- A menu bar at the top of the application (containing standards such as File, Edit, Help).
- A status bar at the bottom of the application to display information.
- One or more toolbars (usually icon driven) that are either attached to one border of the application (top and left being the most common) or floating.
- Right-clicking on items within an application usually displays a pop-up list of possible actions for that item.

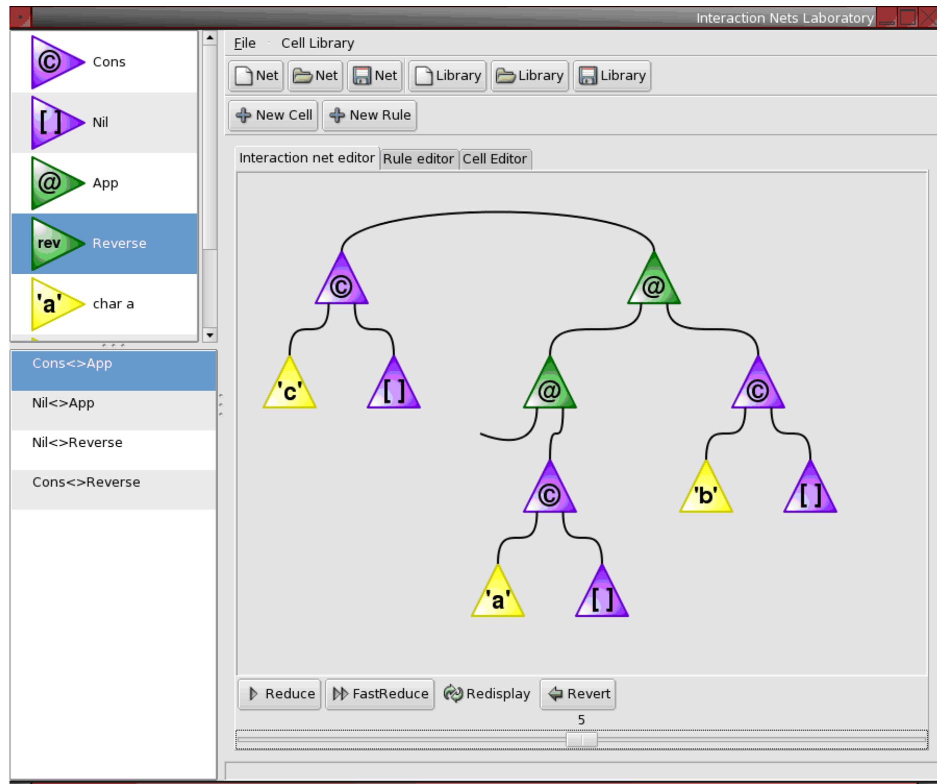
While these rules work for most applications, I feel that for more abstract data, a more organic feel is needed for user interaction due to the specific and unique nature of what is being represented.

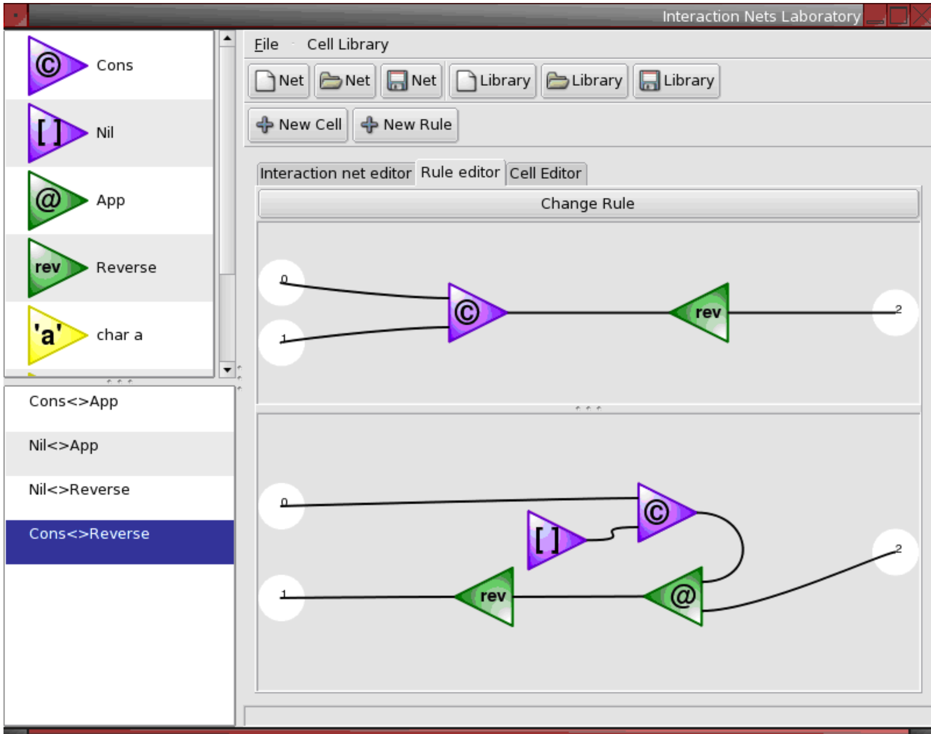
Chapter 6

Review

Interaction Net Laboratory

inl) http://inl.sourceforge.net/ (Marc De Falco)





Interaction Net Laboratory is one of the only graphical editors available for Interaction Nets. It has a rich feature set with a Net editor and a Rule editor. Cells can be created and have properties such as "title" and "colour".

Feature wise, INL is quite complete for creating and editing nets and rules but I have a few issues about the way the user interacts with program. Clicking on objects doesn't result in immediate visual feedback (clicking the "New Cell" button for instance). Also, most of the window is taken up by toolbars and lists which doesn't leave much space to actually draw and view Interaction Nets. Adding cells to the net feels more complicated than it should be, you need to click on the cell in the list to the left and then click somewhere on the net. If you want to add more than one of the same cell you need to keep on re-clicking on the cell that is in the list.

in

Graphical Interpreter for Interaction Nets - Sylvain LIPPI, lippi@iml.univ-mrs.fr

"in" is a program that imports a text based representation of a net and some reduction rules and creates a graphical net.

The user can then choose to reduce the net step by step or to perform all possible reductions in one go.

The following sequence of screen shots show the repeated use of the “Reduce” button on this input file:

symbol

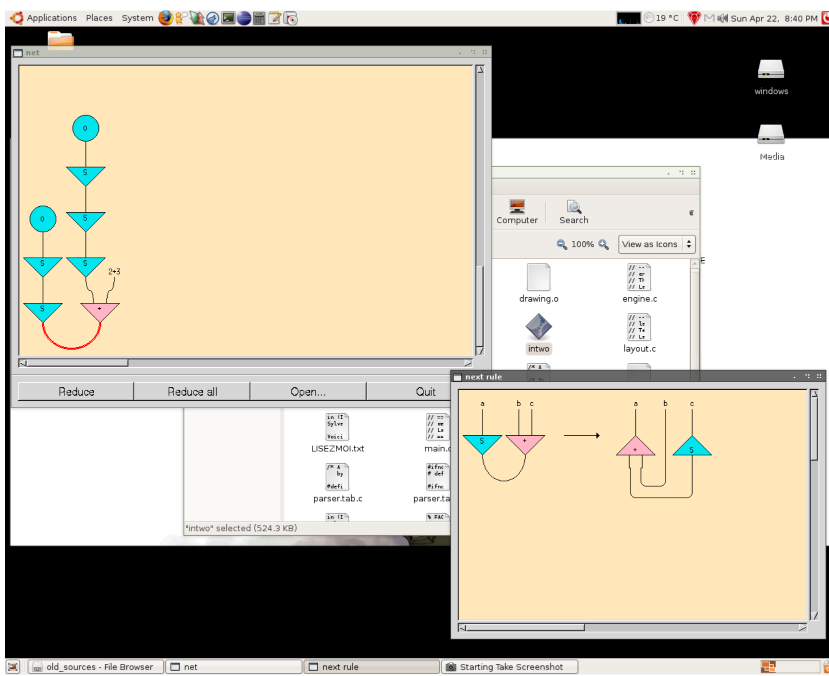
0, turquoise2 : 0
S, turquoise2 : 1
+, pink1 : 2

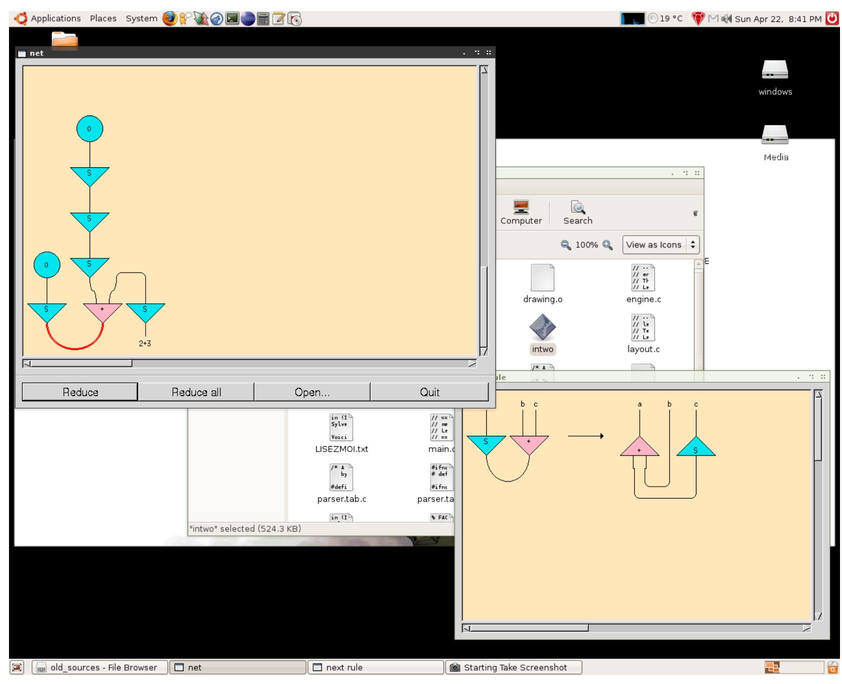
rule

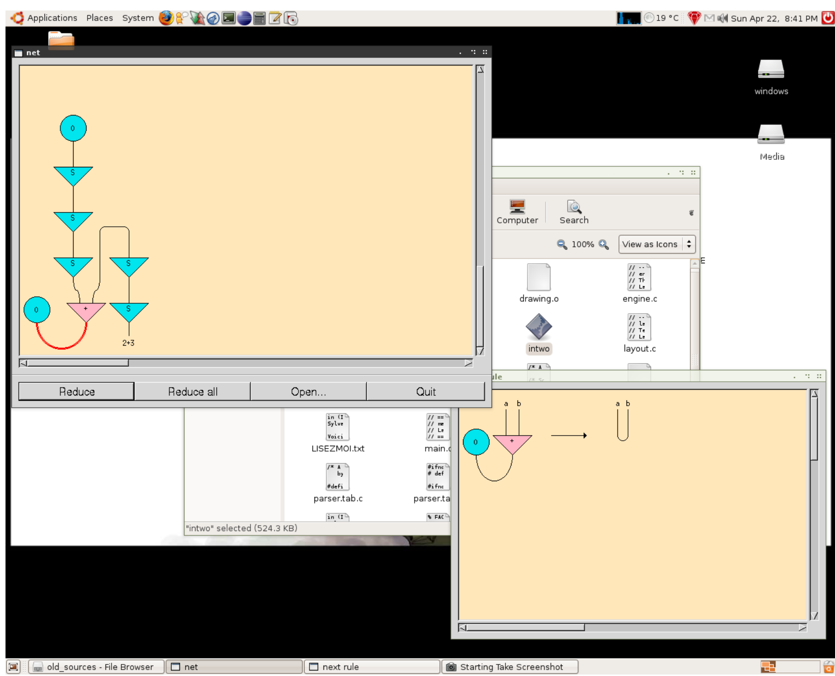
0 >< +(X,X) .
+(Y, S(X)) >< S(+(Y, X)) .

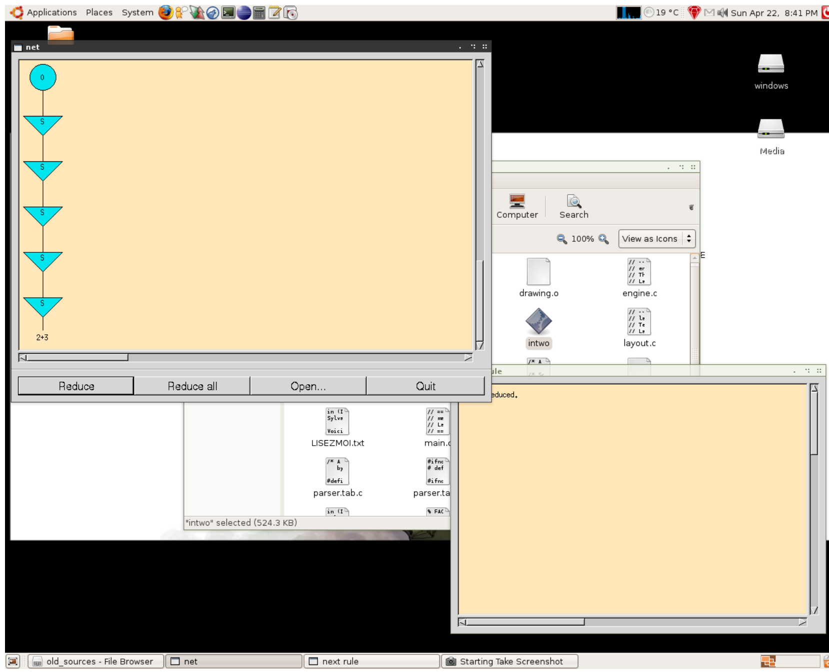
net

+(S(0), result) – S(S(0))




Applications Places System
Sun Apr 22, 8:41 PM
net
windows
Media
Computer
Search
100%
View as Icons
drawing.o
engine.c
intwo
layout.c
2+3
Reduce
Reduce all
Open...
Quit
LISEZMOI.txt
parser.tab.c
"intwo" selected (524.3 KB)
old_sources - File Browser
next rule
Starting Take Screenshot


Applications Places System
Sun Apr 22, 8:41 PM
net
windows
Media
Computer
Search
100%
View as Icons
drawing.o
engine.c
intwo
layout.c
2+3
Reduce
Reduce all
Open...
Quit
LISEZMOI.txt
parser.tab.c
"intwo" selected (524.3 KB)
old_sources - File Browser
next rule
Starting Take Screenshot



While this program is useful for displaying a net graphically and watching it reduce, the user still needs to input the nets in a text-based language. Also, there is no way of editing the contents of the net graphically: text-based input is the only way to create and edit nets.

GRIN will not only import nets from a text-based input, but will also allow the user to edit and create nets graphically.

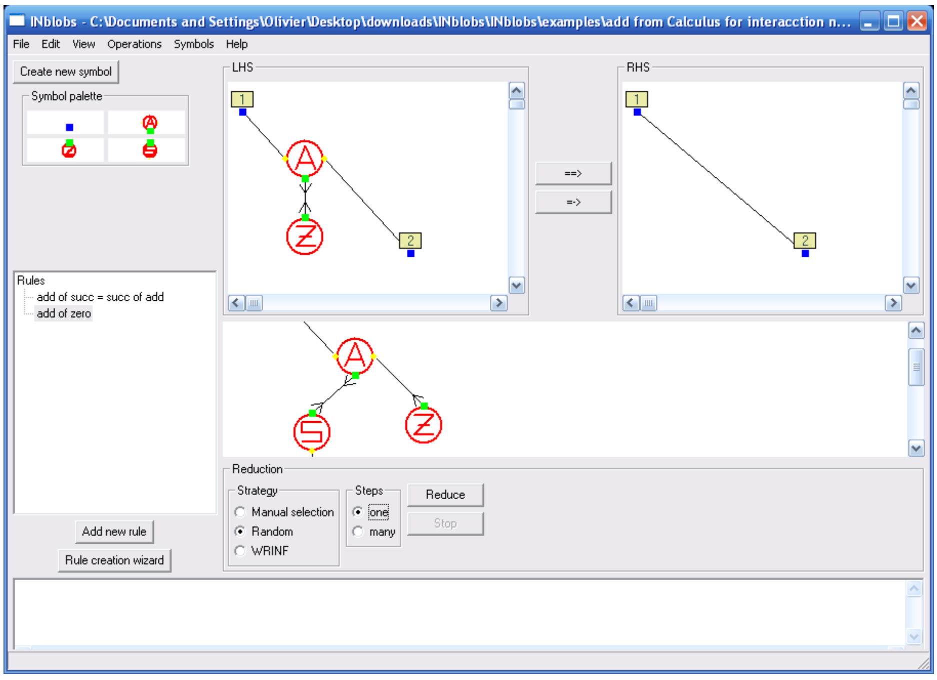
INblobs

http://haskell.di.uminho.pt/jmvilaca/INblobs/ - (Miguel Vilaa)

INblobs defines itself as *an editor and interpreter for Interaction Nets*. Feature wise, INblobs is on par with INL but has similar problems. The creation of cells is unintuitive since you have to enter a list of ports in text form.

Also due to the fact that there are three display windows, each suffer from being too small to view a complex net. It is also not possible to resize them.

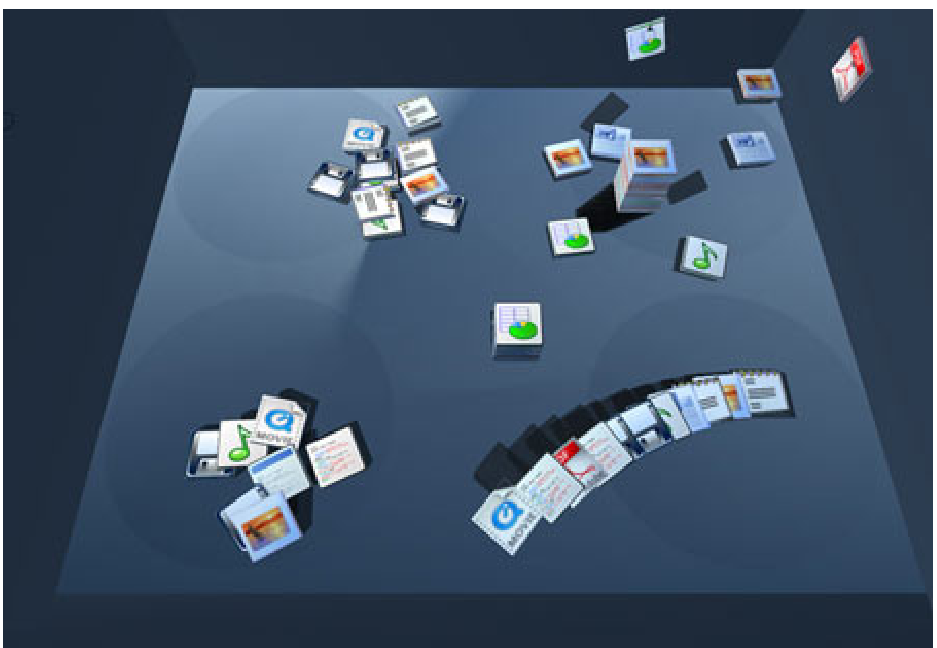
Creating links between ports also feels strange seeing as you are never really sure if the mouse is hovering over the cell or the port.



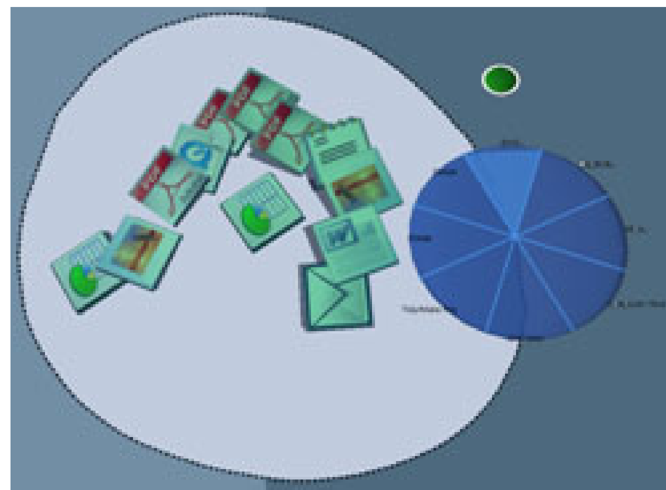
Bumptop

http://www.bumptop.com/

Bumptop is *a physically-based casual interface* with *pen-centric interactions*.



Bumptop takes the notion of computer *desktop* literally. Icons react like physical object and can be stacked and moved around in a pseudo-3D environment. Bumptop assumes users are interacting with the screen with a pen and allows the user to draw selection lassos round the object he would like to select. A wheel menu then appears showing the available actions:



Bumptop is still a prototype but a video demonstration can be viewed on http://www.youtube.com/watch?v=SWe-TIy2Lbs .

Chapter 7

Specification & Design

7.1 Requirements

1. Provide a graphical representation of an Interaction Net.
2. Provide an export and import feature.
3. Provide a save and open feature to and from .inet files.
4. Provide an easy to use and natural GUI to view and edit interaction nets.
5. Provide functions to view and highlight specific features of an interaction net.

1. This software will deal with displaying and editing Interactions Nets in a graphical manner. It will use a two dimensional representation. The software in no way will deal with computing reductions.
2. The program will feature an import and export function to an existing text-based representation of Interaction Nets. This representation will output *agent declarations*, *an agent list* and *a simplified list*.
3. The program will feature a save and open function that saves all the instances of cells, ports and links to a ".inet" file. These files will therefore also store position and rotation of cells.
4. The interface will not be a standard one. It will allow intuitive access to all the features and allow an easy way to navigate through interaction nets.
5. The interface will provide options to view specific features of an net such as "active links", empty ports...

7.2 Specification

A list of Use Case diagrams can be found in 13.1.

Non-standard interface:

The software will not contain any toolbars or standard menus. The lack of menus and toolbars mean that the entire application window is available to draw on, maximising the viewing space. GRIN consists of two windows : the main canvas and a Cell Manager window. The Cell manager is used to add/edit/delete Cell types from a list so therefore a "standard" interface is use with a List and Buttons. For the main canvas window the only interface immediately visible are six buttons (used for Application-wide actions like saving, opening, exporting...). The rest of the interface will dynamically appear as explained in the following paragraph.

Dynamic Icons:

Icons, representing the various actions a user can perform, will appear and disappear depending on what the user has selected on screen. For example : different icons will appear for a selected cell than for a selected port.

Solely Mouse Driven:

I aim to not need the use of a keyboard, unless for typing cell and port names in. Optional keyboard shortcuts will be added but the interaction method in GRIN is designed and optimised for just the mouse.

Visual Feedback:

Users will be able to see the information they need visually, without having to go looking for it. Selected cells and ports look different to unselected ones. Active links are automatically shown. Hovering over a port will display its name as a tooltip. All these visual cues may use colours but will also have other defining attributes (such as line thickness and style) so as not to handicap any colour-impaired users.

7.3 Design

Viewing Interface

With this program, Interaction Nets are represented on a 2D plane. Usually, two scroll bars would be used to do that but I feel that it would not be intuitive since you would have to move the mouse to either the right or bottom border of the window to move around. I have opted for clicking anywhere on the canvas and then dragging in the direction you want to move the view, as if you were ”pushing” it around.

Input Interface

Designing a mouse driven interface meant that the user only had two inputs: the right and left mouse buttons. To make this interaction intuitive, all the possible actions a user can make needed to be divided into two groups:

Left Mouse Button

The left mouse button is used for editing and creating: left-clicking will also activate buttons and select cells and ports. A double-click on the canvas, a cell or a port will execute default actions (adding a cell, adding a port, adding a link; respectively).

Right Mouse Button

The right mouse button controls the graphical side of the canvas, cell and ports. Dragging the right mouse button will move the view or cell (depending on which one is selected). Dragging the right mouse button over a port will allow the user to rotate the cell.

Optional Keyboard Input

Keyboard mnemonics have been created for the buttons in the Cell Manager window. Add to that, specific keys correspond to the available actions Add / Delete / Rename / Break Link / Copy. This is to speed up interaction for experienced users.

The main philosophy behind the interaction is that it should feel natural.

Chapter 8

Implementation

8.1 GRIN Environment

Java[1] will be used to create GRIN. Java will make GRIN cross-platform and will therefore be accessible to a wider audience. Personally, I have been coding in Java for over three years now meaning there will be less of a learning curve and downtime than if I used another language. Eclipse[2] will be used as the IDE of choice since it supports advanced features such as error detection and CVS (GRIN was on a CVS server to allow me to code from any computer) It is also multi-platform which will allow the development and testing of GRIN to happen on any operating system.

Due to the graphical nature of GRIN, a capable graphics system needs to be implemented. This system needs to be the powerful enough to implement all the features needed and yet not be too complicated by having many superfluous features. Here are the different systems looked at:

- OpenGL[3] describes itself as "*the premier environment for developing portable, interactive 2D and 3D graphics applications*". OpenGL is also free. Many video games use openGL and it can also be found in Quartz Extreme, the display engine used in Apple's OS X. Its main competitor is Microsoft's DirectX but since the latter isn't cross platform, it isn't on this list.

- Java2D[4] is the standard 2D graphics API that comes with the JAVA JDK. While not as powerful as other 2D APIs, it is built in to Java. Java2D allows

[1] http://java.sun.com/
[2] http://www.eclipse.org/
[3] http://www.opengl.org/
[4] http://java.sun.com/products/java-media/2D/

you to draw custom geometry and to import images from files to display to the screen.

- Torque Game Builder[5] is more than just a graphics system and encompasses features to created a fully fledged engine: Collision Detection, Physics Engine, TorqueNet, TorqueScript...

The Torque Game Engine, while being multi-platform, cannot be implemented into Java. This would require learning C++ and TorqueScript to create anything with TGE. Added to this, TGE's licence isn't free and a lot of its features (such as the Physics Engine) will never be needed.

OpenGL can be implemented into Java using JOGL[6] but much like TGE presents a lot of unnecessary features (such as 3D graphics and shader support).

I have therefore opted to use Java2D since its feature set meets the requirement of GRIN (which is to draw 2D nets so only needs to draw basic geometry), integrates naturally with Java and is cross-platform and free to use.

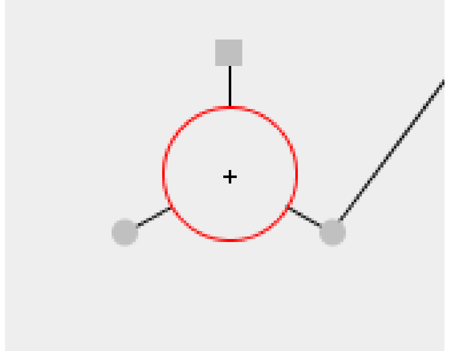
Java2D has certain drawbacks: there is no collision system, so a set of functions will need to be created to handle object selection. Also if in the future a more graphically complex system is needed to display nets, Java2D will not be able to follow and GRIN would need to be ported.

8.2 GRIN Development and Evolution

The development of GRIN will have to follow the evolution of my understanding of Interaction Nets. This means that some features might need to be added, removed or changed as my knowledge of Interaction Nets expand.

For instance, cells in GRIN didn't display ports and would only show a direct line from one cell to another to represent a link. Delving further into Interaction Nets, I realised it is quite common to have empty ports on a cell and the graphical representation currently used wasn't displaying this information to the user. The display of cells evolved to show ports "orbiting" round each cell.

[5]http://www.garagegames.com/products/torque/tgb/
[6]https://jogl.dev.java.net/



Different types of user interfaces would be tested and the most appropriate one would be kept and improved upon. At the beginning of the project, I thought of using a wheel menu similar to the one used in the Bumptop prototype (seen in 6). I soon realised that the system was complicated to implement, required more dexterity from the user than just clicking on a button and due to its size would obscure the cell and its surrounding area. This was less than optimal so another system needed to be used.

A standard GUI usually has toolbar icons somewhere on the screen. Most of the time, a lot of on screen icons are not usable since they don't correspond to an action possible by the currently selected item. Also, the user would have to constantly move the mouse from the canvas to the toolbar, wasting time. From this emerged the idea of dynamic icons, that appeared and disappeared depending on what was selected. They would also appear near the selected item, reducing the time it would take to get to them. Icons represent actions in a way that is less disruptive than text, making them ideal to use.

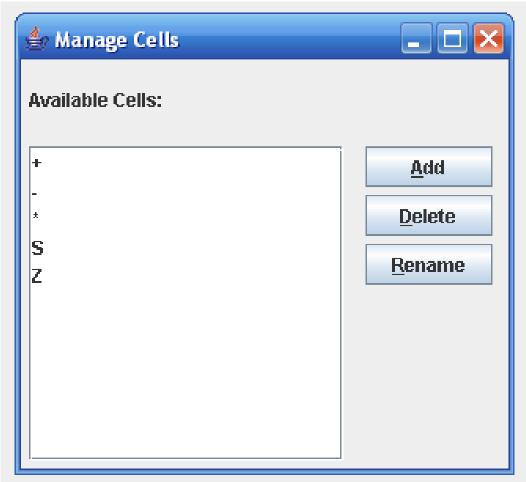
This system proved a lot less intrusive than a wheel menu while remaining as easy to use.

Each type of item in GRIN only has at most four possible actions:

- Add
- Delete
- Copy/Rename
- Break Link

Four consecutive keys on the keyboard were chosen to correspond to each of these four possible action: Z, X, C, and V. This allows the user to have one hand on the mouse and to use the other to perform all the possible actions, potentially speeding up interaction.

GRIN started off by dynamically creating and deleting MasterCell instances depending on the number of cells in existence at any one time. This meant the user had to enter the type of a cell each time one was created, even if one of the same type already exists. This method made the MasterCell creation vague and not fully under the control of the user. To remedy this, a Cell Manager window was created where the user could add, delete and rename MasterCells that would then show up in a drop down menu when the user tried to create a cell. An unforeseen feature emerged from this system allowing the user, by editing the MasterCell list, to delete and rename in one go all the cells of a certain type.



The Manage Cells window.

Subtle visual cues were added throughout GRIN to give the user more information without overloading the screen. A primary port ends in a small square while auxiliary ports end with a circle. Active pairs are linked by a dashed lined instead of a normal one. Tooltips were added when hovering over a port to show its name since constantly displaying the names of all the ports would complicate what the user sees on the screen.

Towards the end of the project cycle, rotation was added so as to make nets more manageable and easier to view. The integration of this feature adhered to the rules of the GUI: the right mouse button is used to drag the ports round the cell.

8.3 GRIN Implementation

8.3.1 Interaction Net classes

Four main classes are at work to represent Interaction Nets:

- MasterCell : defining types of cells.

- Cell : holding information about cells such as number of ports, position on screen and which MasterCell it is linked to.

- Port : containing names and positions of each port, as well as a pointer to the port that it is linked to, if any.

- Link : with two pointers to the pair of ports it links together.

For a more detailed view of these classes, class diagrams can be found in13.2

When a MasterCell, Cell or Link is created, they are added to a masterCellList, cellList or linkList LinkedList object respectively. This allows the program to iterate through all the cells and links in its draw phase to get data out of them to display to the screen.

Ports are added to a LinkedList within each cell instance. This proves more versatile than an array due to its dynamic size. Also, the first element of the cell's port LinkedList is always considered the primary port and is created when the cell is.

When a new port is added to a cell, an algorithm iterates through all the cells and adds an identical port to call cells that have the same MasterCell as that first cell. The same method applies to renaming and deleting Ports.

8.3.2 Graphical User Interface

Display System

Java2D is centred around the *Graphics2D* class. GRIN has an instance g of Graphics2D in its update loop.

Any shape or image to be displayed needs to be created or loaded and then passed onto g using *g.draw()* or *g.drawImage()*. It is also possible to assign g different attributes such as colour, stroke style and alpha-blend to change how the shape is drawn.

When launched, GRIN goes into a never ending loop where g is created and used to draw all the elements of a net as well as the GUI. It iterates through the cellList and linkList and displays every link and cell (and its ports) according to certain criteria. For example a selected cell is draw with a dashed blue outline instead of a normal red line, used for unselected cells.

All the cells, ports and links are drawn using the Java *Shape* class which allows you to create geometry on the fly as opposed to loading set images from files.

A custom class *Button* was created to load icons from *.png* files.

Event Handler System

GRIN implements a *MouseListener*, *MouseMoveListener* and a *KeyInputHandler* to get mouse and keyboard input. If the mouse is moved or clicked, a function will return the item it is currently hovering over. A click will make that item the selected item. Depending on the class of the item, different thing will happen:

- A cell or port is selected, and the appropriate buttons are displayed.
- A button is selected, and the *buttonActions()* method is called to take the appropriate action depend on the button.
- Nothing is selected, and all non essential icons disappear from the screen.

8.3.3 Save & Load System

Using *Object Serialization* in Java made it possible to save the *cellList*, *masterCellList* and *linkList* directly to file with a minimum of code. All that was needed was to have every class destined to be saved to implement the *java.io.Serializable* class. To actually save the objects the following lines of code were needed:

```
os = new FileOutputStream(fileName);
s = new ObjectOutputStream(os);
s.writeObject(cellList);
s.writeObject(masterCellList);
s.writeObject(linkList);
```

To load from a file was equally simple:

```
os = new FileOutputStream(fileName);
s = new ObjectOutputStream(os);
cellList = (LinkedList¡Cell¿)s.readObject();
masterCellList = (LinkedList¡MasterCell¿)s.readObject();
linkList = (LinkedList¡Link¿)s.readObject();
```

8.3.4 Exporting & Importing

Exporting

Exporting a net to a file would write three things into that file:

- Agent Declarations
- Agent List
- Simplified Agent List

Here is an example output file for a net that computes 2+1:

```
Agent Declarations:

+ : 2 ;
S : 1 ;
Z : 0 ;

************
Agent List:

+(x0,x5,x1)
S(x0,x3)
Z(x4)
S(x1,x2)
S(x3,x4)
Z(x2)

************
Simplified List:

S(S(Z)) = +(x5,S(Z))
```

Outputting the Agent Declarations is relatively trivial by finding each type of different cell and printing out its name and number of ports.

For the agent list, an algorithm iterates through the linkList, and replaces each pair of ports by a unique variable in their respective cells. It then iterates through

all the cells and replaces any remaining ports with a unique variable each. It then prints out all the cells.

For the simplified list a more complex algorithm was required. A recursive function is applied to a list of cells to create a tree:

If the first argument of an agent is also an argument of another agent, the first agent can be placed as the second's argument. The first cell would also lose its first argument. For example:

+(x0,x5,x1)
S(x1,x2)

Would become:

+(x0,x5,S(x2))

Once this happens, the function would recurse on that first agent.

The algorithm returns a tree and all that is needed is to read the tree out in preorder to get the penultimate state of the simplified list. We then need to take out the first argument in that list and to make it equal to what is left:

+(S(S(Z)),x5,S(Z))

Becomes:

S(S(Z)) = +(x5,S(Z))

Importing

Importing can import one net at a time when they are inputted in the form *+(S(S(Z)),x5,S(Z))*.

An algorithm parses through the input and creates a tree of strings. From that, a tree of cells is created. Links are added to linkList for each branch of this tree. Finally the tree is collapsed into a list that is added into cellList. Cells are given arbitrary positions along a grid. It is up to the user to then position the cells as he sees fit.

Chapter 9

Validation

A graphical interface for Interaction Nets is difficult to validate. Graphically, it isn't complex enough to warrant testing its speed. Also, since there is no "correct" net the user is free to create a net how he sees fit, even if it computationally it makes no sense.

The only elements that can have some form of validation are the Importing and Exporting features.

Importing

Testing the import function meant writing many different kinds of text-based nets and importing them into GRIN and comparing the resulting net to the one created by hand.

Exporting

Exporting is slightly more complex since one net can have more than one simplified form. To validate the function, I would export a net and then compare the output to what I had found by hand. If the simplified list didn't correspond, I would create a net from it and see if it returned the same net.

The Identity Test

A final validation test was:

- To export a net and then try import its output text and see if an identical net is created.
- To import a text-based net and then export the created net and see if the output is the same.

Reduction Rules

An unexpected features emerged from the export net feature: a reduction rule export. If the user:

1. Created the two nets of a reduction.
2. Linked the corresponding free ports to each other.
3. Exported the newly created new.

They would then find that the simplified list, after replacing the "=" for a "$\bowtie$", is in fact the reduction rule.

Chapter 10

Evaluation

Overall, GRIN and its feature set has stayed very close to the original idea: a program that allows the creation and editing of interaction nets with the possibility to import and export nets.

10.1 Functional Evaluation

First and foremost, GRIN was created to display and edit interaction nets. To this degree, any other feature like importing and exporting was left to last and aren't as powerful or versatile as other programs. The import string needs to be custom made in a certain way, and there is no evaluation of the inputted string. If the net is typed in incorrect, GRIN will still try to import it and at best will display a nonsensical net or at worst crash. Exporting could have been built from the ground up to accept plugins in case new textual based languages for nets appear.

Also, the reduction functionality was completely unintentional and to that degree could have been implemented better. A separate window where you can create rules to add to an net. Added to this, a reduction feature that would reduce the drawn net was left to the be done only if any spare time were to be found. Both INL (seen in **??**) and INblobs (seen in 6) feature versatile rule creation and the ability to reduce nets on screen.

10.2 Usability Evaluation

What GRIN may lack in interaction net features, it makes up for in usability. The learning curve for GRIN is a lot faster and easier than other programs. The inter-

action is natural and intuitive and the visual feedback shows a lot of information without blinding the user. It is also designed to be used by the colour impaired, something that is rarely thought about in GUIs.

With my knowledge in interaction nets, I still found programs like INL and INblobs to be unintuitive, even after ten minutes of using them. I tested GRIN on someone with only basic interaction net understanding and within minutes he was creating nets.

One regret I have about the usability is that I wish that the Manager Cell window looked more integrated with the rest of the program. Since it looks like a standard interface, it somewhat clashes with the canvas window.

My initial idea was for users to write onto the canvas and have a cell created around what they wrote. This method just seemed complicated to implement with Java2D and made MasterCell creation and editing vague. With more time and research, maybe a middle ground could have been found.

10.3 Environment Evaluation

Working with Eclipse was a dream. It is an incredibly versatile IDE that not only points out errors in your code, but also shows warning such as unused variables. Other features like auto-complete and displaying the java API information about a selected method or class greatly helped the coding of GRIN.

Java made coding and compiling easy to do on all three main operating systems (Windows, OS X and Linux) and different parts of GRIN were done on each of those systems.

Java2D was very simple to use but did have limitations. While extensively documented, the examples were usually confusing which meant a lot of time was spent figuring out why random things happened, or didn't happen. Once the main hurdle was jumped over, the system just worked.

There is no event handling on shapes and images displayed with Java2D which would have greatly simplified coding in the buttons and the selection methods.

Maybe a more powerful graphics engine would have helped speed some features along, and would have left GRIN open to develop into something more graphic intensive.

Chapter 11

Conclusion & Future Work

11.1 Conclusion

Java and Java2D have shown that they are both flexible and versatile systems, and while some limitations appeared within Java2D, there was always a way to work around them. It further shows that Java is capable of non standard GUIs and can hold its own graphically. With no extra work, GRIN is available to the three most popular operating systems. All these qualities also fit in a program that is under 300kb in size:this is including source code, compiled *.class* files and images for the icons.

I have also observed that so called "standard" GUIs only feel intuitive since we know what to expect from them. With a bit of thought and research, alternate GUIs can be created that suit a specific set of actions better. There is a misconception that keyboards and mice are natural, but again they only feel like that out of experience. A lot of my ideas about interacting with nets naturally were impossible to do with just a mouse and keyboard. Human-computer interaction is an important topic today since we now have the technology to back our ideas: multi-input touchscreens (as seen in Apple's[1] upcoming iPhone) and movement sensing controllers (proven a huge success with Nintendo's[2] Wii) are soon to become standard input devices and soon what we refer to as a "standard" GUI will need to be rebuilt from the ground up.

My delving into interactions net have shown how versatile and powerful a computational model it can be. While complicated and different from traditional models, the benefit it brings are not negligible. Having a graphical interface to

[1] http://www.apple.com/
[2] http://www.nintendo.com/

interact with nets will greatly help interaction nets become a standard model of computing: something that is taught as a language to code in, not a theoretical concept to lecture about.

11.2 Future Work

GRIN has shown that an alternative GUI is more functional than a standard one using the same input devices. I have often wondered how different GRIN could have been if the user could interact with it using a pen and touchscreen. The user would literally draw his net onto the computer that would then transform it on the fly into something it understands. With a multi-input touchscreen big enough, more than one user could collaborate on the same screen.

Speaking of collaboration, extending GRIN to work over a network where more than one user can edit a net in real-time would bring this model closer to becoming a standard to code in. After all, collaborating over an image is less confusing than over text.

Replacing Java2D by a fully fledged graphics engine, openGL for instance, would have allowed a more graphically complex GRIN. The user would be able to zoom in and out of the canvas and the view could dynamically change depending on what the user is doing. Also, more “eye candy” could be put into the program, making GRIN more visually pleasing. openGL’s 3D capabilities might have opened doors to some kind of 3D representation of interaction nets, whether such a representation is viable or not is to be seen.

With more time, a more powerful and versatile import and export feature could be implemented. Added to that, a rule creation system with real-time reduction would boost the usability of GRIN.

Bibliography

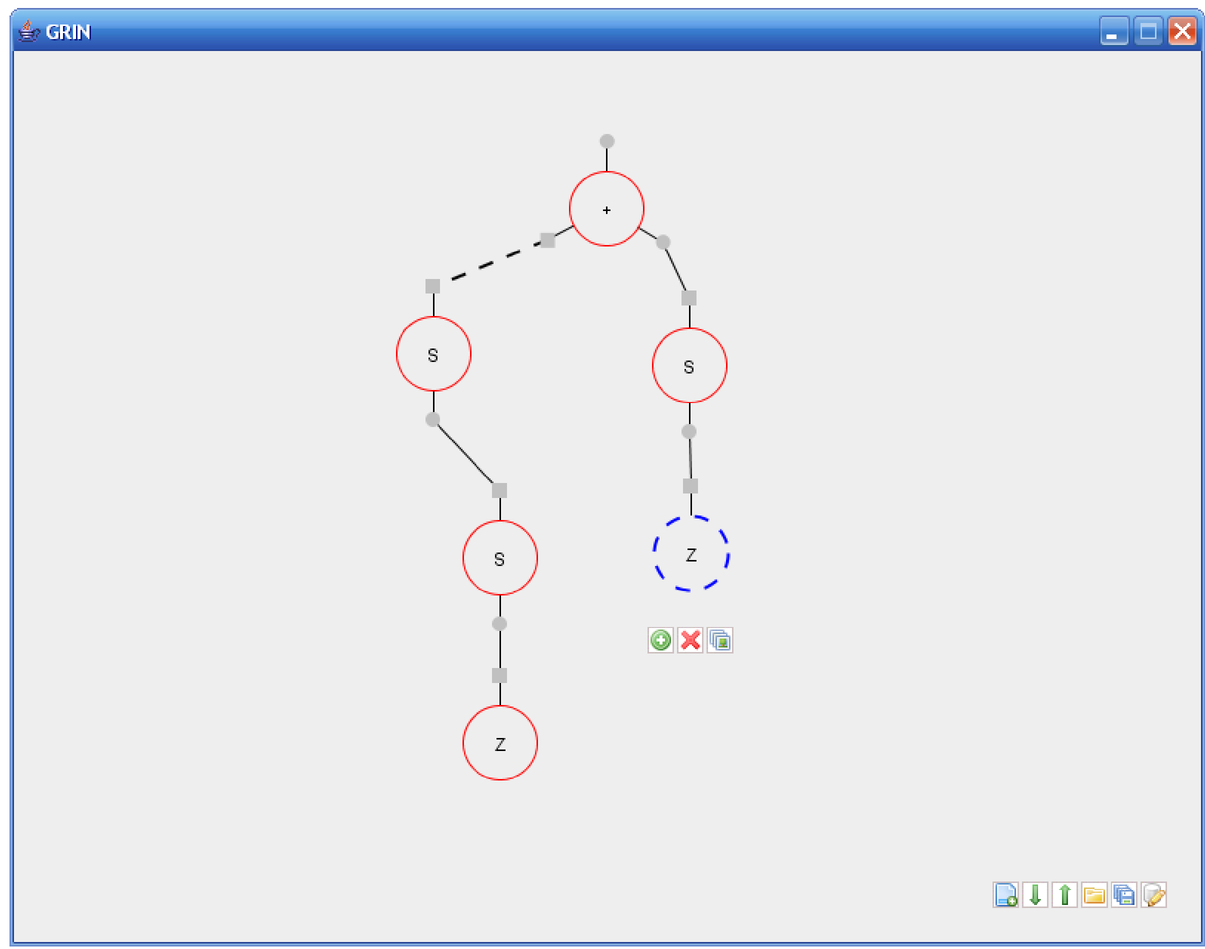
[1] Ian Mackie, *Towards a Programming Language for Interaction Nets.*

[2] Yves Lafont, *Interaction Nets.*

[3] Yves Lafont, *Introduction to interaction nets,* (CNRS Institut de Mathématiques de Luminy , 1998).

Chapter 12

User Guide



An example screen of GRIN in action.

12.1 Compiling & Running

To compile the GRIN package, go to the root directory are run the following command :

javac .\ grin\ $*$ .java

To run then GRIN, then type in:

java grin.Grin

12.2 New, Save and Open File

Reset the canvas removing all the nets previously on it.

Save current canvas to a *.inet* file.

Load a canvas from an *.inet* file.

12.3 Canvas

- Double-clicking on the canvas will create a new cell where the mouse is.
- Holding down the right mousebutton and moving will drag the canvas in the direction of the movement.

12.4 Cell Manager

Click on the icon on the canvas to bring up the Cell Manger.

In the window that appears, three actions are possible:

- Add : Opens a dialog box to enter the name of a new cell type.

- Delete : Deletes selected cell type and all existing cells of that type.
- Rename : Renames selected cell type and all the cell of its type.

Adding or renaming a cell type to an existing name will cancel the action.

12.5 Cells

Clicking on a cell will select it. A selected cell's border turns blue and is dashed. To move a cell, hold the right mouse button over it and move. To rotate a cell, hold the right mouse button over one of its ports and move.

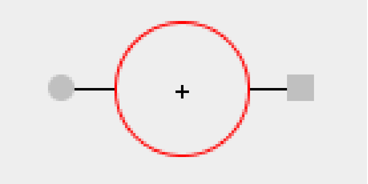
The possible actions on a cell are:

Icon	Description	Keyboard Shortcut
	Opens a dialog box to enter the name of a new port.	Z
	Deletes selected cell.	X
	Copies selected cell next to it.	C

Added to that, double clicking a cell will perform the same action as the icon.

12.6 Ports

There are two kinds of ports: primary and auxilliary.



A cell with its primary port on the right and an auxilliary port on the left.

Clicking on a port will select it. A selected port's border turns blue.

The possible actions on a port are:

Icon	Description	Keyboard Shortcut
	Click on another port to link it to. Click on same port to cancel linking.	Z
	Deletes selected port.	X
	Renames port. This is applied to all cells of the same type as the one the port belongs to.	C
	Breaks link if one exists.	V

Added to that, double clicking a port will perform the same action as the icon.

Note that the delete action is not available on primary ports.

12.7 Exporting

To export the current net, click on the icon. Then select the file you want to save to. This will replace all the data in that file.

12.8 Importing

To import a net, click on the icon. Then type the net into the dialog box.

The net must be formatted like so : *+(S(S(Z)),x5,S(Z))*

12.9 Rules

To get the text-based version of a rule :

1. Create the two nets of a reduction.
2. Link the corresponding free ports to each other.

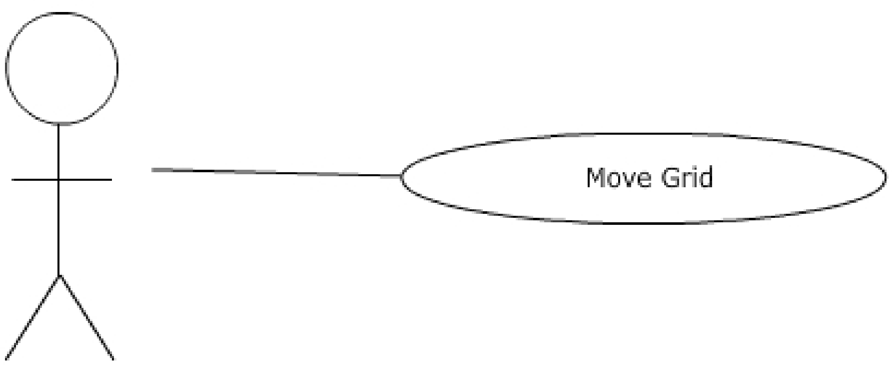
3. Export the newly created new.

4. Find the simplified list and replace the "=" for a "$\bowtie$" to get the rule.

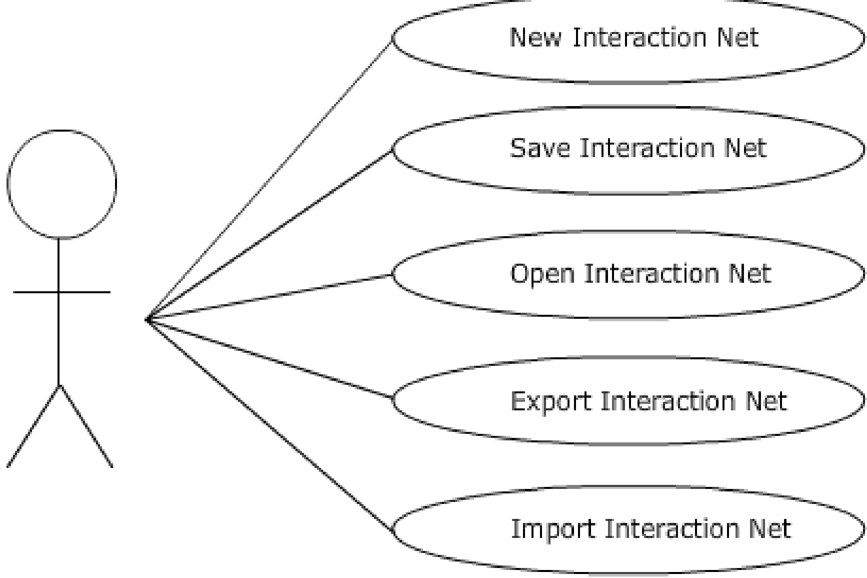
Chapter 13

Appendix

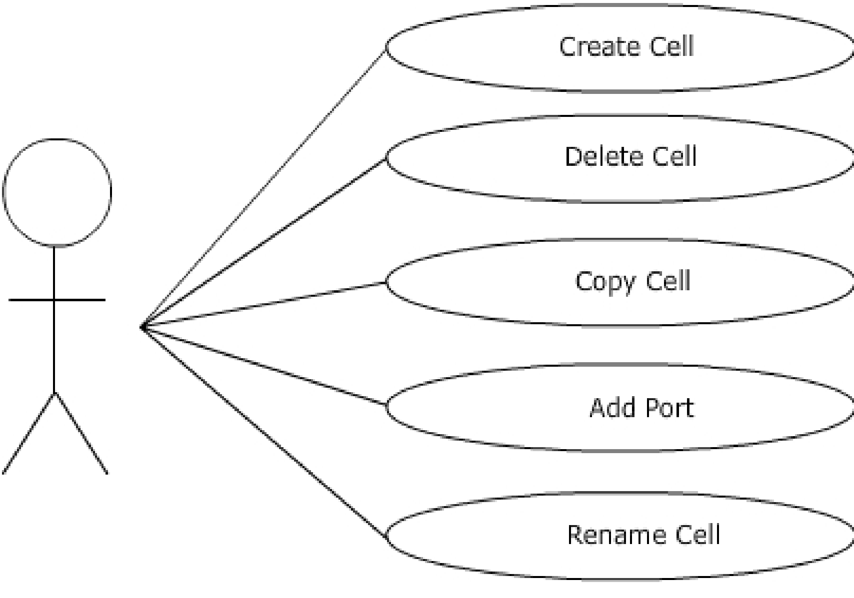
13.1 Use Cases



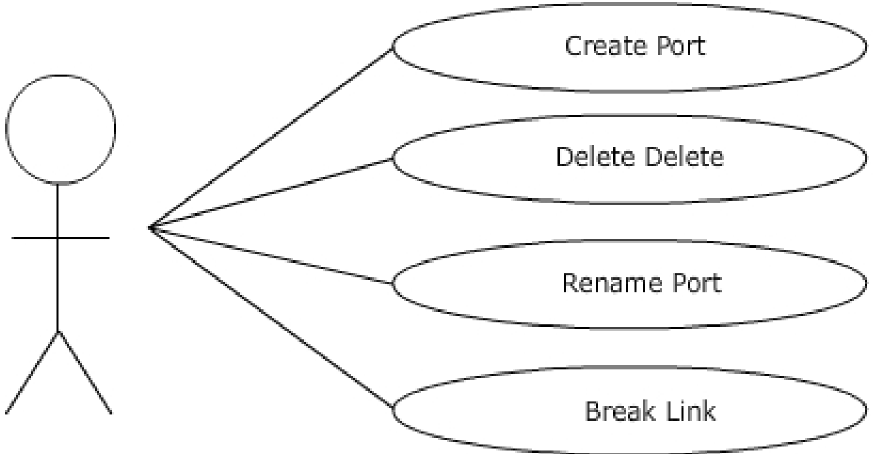
Inferface interaction



Interaction Net level functionality

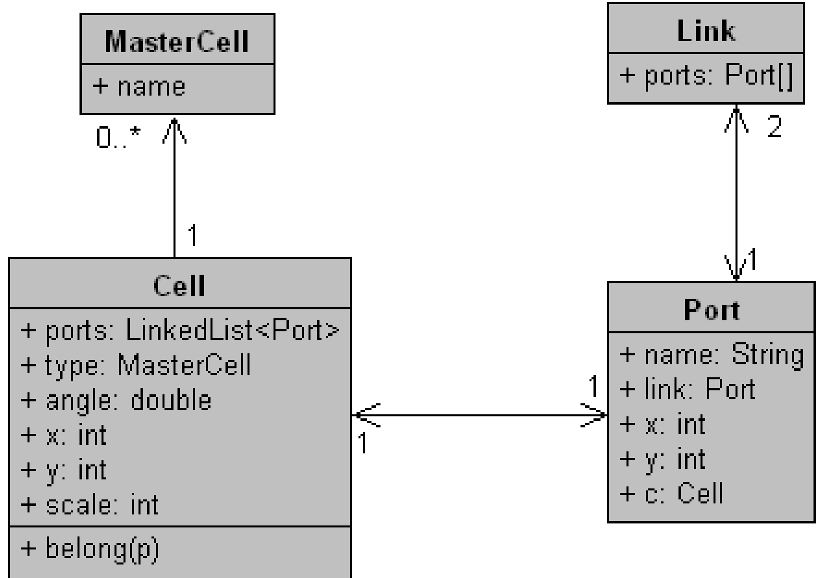


Cell level interaction



Port level interaction

13.2 Class Diagrams



13.3 Program Listings