

# REPORT

For the NSERC Undergraduate Student Research Award - summer 2006

Department of Computer Science and Engineering

York University, Toronto

Project: Extending the OME requirements analysis tool to support formal  
analysis using ConGolog and CASL

Supervisor: Professor Yves Lesperance

Author: Gertruda Grolinger

## Contents

<b>Section</b>	<b>TITLE</b>	<b>Page number</b>
1	Introduction - Purpose of the project	3
2	Main components	4
3	Technical part	5
4	Conclusions and further research	41
5	Appendix A – ConGolog converter code	42

## Introduction – Purpose of the project

Requirements specification and analysis, as well as requirements linking to system/process design are becoming more and more important in software engineering and business process reengineering. Open OME (Organization Modeling Environment) is an open-source requirements engineering tool:

<http://www.cs.toronto.edu/km/ome/>. It is a goal-oriented and/or agent-oriented modeling and analysis tool that provides users with a graphical interface to develop models. It also provides a clear link between the requirements, specification and architectural design phases of development. OME primarily uses the i\* graphical notation: <http://www.cs.toronto.edu/km/istar/> for specifying models. i\* is a relatively informal notation with a limited expressiveness. This drawback limits the kinds of analysis that can be done. Nevertheless, i\* graphical notation can be employed together with the formal agent-oriented specifications language such as ConGolog:

<http://www.cs.toronto.edu/cogrobo/systems.html/>

<http://www.cs.yorku.ca/~lesperan/IndiGolog/>

<http://www.cs.yorku.ca/~lesperan/papers/AOIS01.pdf/>

to better specify the goals and processes that are being modeled and to perform formal analysis, verification, and simulation.

This project focuses on extending the Open OME to obtain a tool that can support better expressiveness of i\* and more varied and thorough analysis of the built systems.

## Main components

In this section I will explain on a high level what I was working on and what I accomplished.

The project consists of two components: extending the i\* notation and convertor that translates i\* diagrams into ConGolog code.

### Extending the i\*

i\* is an informal diagram-based notation for early-phase requirements engineering that supports the modeling of social dependencies between agents. It consists of two main components: the Strategic Dependency (SD) model and the Strategic Rationale (SR) model. The Annotated SR (ASR) model is an additional step added in order to be able to use i\* in combination with ConGolog framework. In this step the process details are filled out using annotations. The annotations were initially not implemented in the OME tool; this extension is the first part of the project.

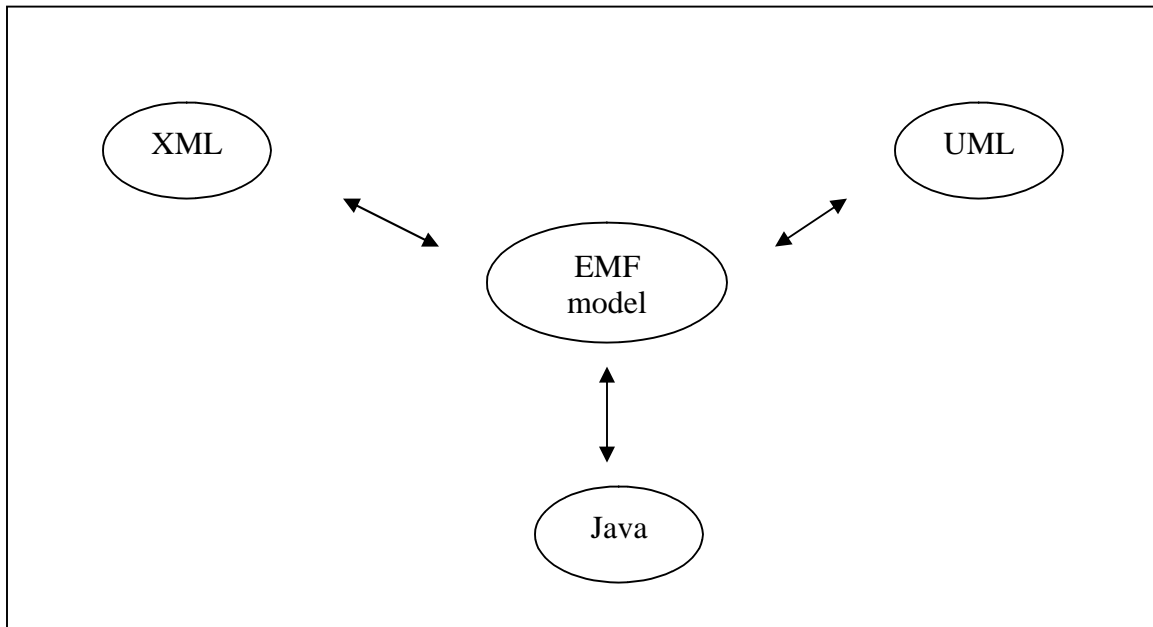
### Converter that translates i\* into ConGolog code

ConGolog is a framework for process modeling and agent programming. A ConGolog model of a domain includes two components: domain specification (initial state, actions to be performed, when these actions can be performed and what their effects are) and behavior specification (behavior of the agents involved in the domain). In order to be able to translate i\* into domain specification part of ConGolog code, I added further extensions to the OME. Additionally, I started developing a Java program that does the translation and generates an executable ConGolog code. This part of the project is considerably longer, more involving and requires more time, research and testing and therefore it is not completed at this point.

## Technical part

This part of the report is intended for someone who would want to continue this work or to familiarize herself/himself in detail with the changes that were made on the OME tool and with the way the converter idea is implemented.

To work on the project I used the EMF, Eclipse Modeling Framework. EMF is a modeling framework that relates modeling concepts directly to their implementations; it is a framework and code generation facility that lets a programmer define a model in EMF, UML or XML form, from which then the others can be generated and also the corresponding implementation Java classes. The following figure shows how EMF unifies the three important technologies: Java, XML, and UML.



EMF is a modeling framework and code generation facility for building tools and other applications based on a structured data model.

Once a programmer specifies an EMF model, the EMF generator can create a corresponding set of Java implementation classes. One can edit these generated classes to add methods and instance variables and still regenerate from the model as needed: the additions will be preserved during the regeneration. If the

code that was added depends on something that was changed in the model, one will still need to update the code to reflect those changes; otherwise, the code is completely unaffected by model changes and regeneration. More on EMF can be found under <http://www.eclipse.org/emf/>.

The model is represented by a representation called goalmodel that is contained in a .genmodel file. Anyhow, in order to add something, it is not possible to directly change this file; the changes are done on the corresponding .ecore file that contains the meta model. More about ecore can be found under: <http://download.eclipse.org/tools/emf/2.2.1/javadoc/org/eclipse/emf/ecore/package-e-summary.html#details/>. After changes are done on .ecore file, they will be reflected in .genmodel after reloading it.

### Extending the i\*

The extension of OME tool that includes changes and adding of the annotations and some other components is the first part of this project.

Goalmodel (name.genmodel file) structure consists of EClasses. Elements in goalmodel that are represented by EClasses are: actor, goal, contribution, dependency, property, topic, linkAnnotation, conditionAnnotation, appCondAnnotation, effect and primAction. What follows the outline of all mentioned classes together with explanations, comments and changes that were made in this project.

#### actor

EClass 'actor' has:

- name – this is an EAttribute and it is of type EString
  - the name of the actor.
- super – this is a single EReference and it is of type 'actor' (see further)

- reference to the parent actor (there can only be one such reference).
- sub – these are the multiple EReferences and they are of type ‘actor’
  - references to the children actors (there can be any number of such references).
- type – this is an EAttribute and it is of type ActorType; ActorType is of type EEnum and it can have values: actor (0), agent (1), role (2), position (3)
  - actor node can be: unspecified, agent, role or position.
- goals – these are multiple EReferences and they are of type ‘goal’ (see further)
  - refers only to root level goals (any number of them) of this actor.
  - reciprocal to ‘actor’ attribute in ‘goal’ class (see ‘goal’ class).

### goal

EClass ‘goal’ has:

- name – this is an EAttribute and it is of type EString
  - the name of the goal.
- mode – this is an EAttribute and it is of type ModeType; ModeType is of type EEnum and it can have values: hard (0), soft (1), task (2), resource (3)

**Comment:** The goal class does not represent only unspecified goal; it represents all 4 types of goals found in ASR diagrams: hardgoals, softgoals, tasks and resources. These are distinguishable by setting the ‘mode’ attribute of the ‘goal’ class. The values of ModeType’s type EEnum are: 0 for hard goal, 1 for softgoal, 2 for task and 3 for resource.

So for example, to set a goal to be a task goal, one would set its ‘mode’ attribute to 2.

- type – this is an EAttribute and it is of type DecompositionType; DecompositionType is of type EEnum and it can have values: or

(0), and (1), leaf (2)

- or – refers to or-decomposition.
- and – refers to and-decomposition.
- leaf – refers to the goal that is not decomposed.

**Comment.** There are 3 types of links in ASR diagram:

‘task decomposition links’ (that connect tasks with components needed for their execution: if  $t$  is a task that is decomposed into nodes  $n_1...n_k$  by task decomposition links, then individual task decomposition links are linking  $t$  and  $n_i$ );

it is similar for ‘means-ends links’ (that specify alternative ways of means to achieve goals) and

‘softgoal contribution links’ (that specify how process alternatives affect quality requirements i.e. softgoals).

- parent – this is a single EReference and it is of type ‘goal’
  - reference to the parent goal (only one such goal).
- goal – these are multiple EReferences and they are of type ‘goal’
  - references to the children (any number of those).
- label – this is an EAttribute and it is of type LabelType; LabelType is of type EEnum and it can have values: satisfied (2), denied (-2), partially Satisfied (1), partially Denied (-1), unknown (0), conflict (4)
  - used for NFR analysis
- rule – these are multiple EReferences and it is of type ‘contribution’ (see below)
  - correspond to softgoal contribution link in ASR diagram (refers to contribution rule). More on its type can be found under ‘contribution’ EClass topic below.
- system – this is an EAttribute and it is of type EBooleanObject

- indicates if the goal is achieved by this system or delegated outside of the system (this part is from another project and it is not relevant to this project).
- boundary – this is an EAttribute and it is of type EBooleanObject
  - indicates if the goal is in the interface of the system.
- input – these are the multiple EReferences and they are of type ‘topic’ (see below)
  - used for parameters (there can be any number of input parameters).
- output – these are the multiple EReference and they are of type ‘topic’ (see below)
  - used for parameters (there can be any number of output parameters).

**Comment:** The following 5 EAttributes: exclusive, sequential, parallel, alternative and prioritizedConcurrency are corresponding to composition annotations in ASR diagram.

In an ASR diagrams, there is 4 composition annotations (sequence, concurrency/parallel, alternative, prioritized concurrency), while in goalmodel structure there is 5 attributes as listed below.

**Changes:** The first 3 annotations (exclusive, sequential, parallel) were implemented before and I added the rest (alternative, prioritizedConcurrency).

After discussion with my supervisor, it was concluded that this could lead to wrong representation since one can only chose one composition annotation.

Therefore, another attribute is added (see below under hasLinkAnnotation).

- exclusive – this is an EAttribute and it is of type EBooleanObject
- sequential – this is an EAttribute and it is of type EBooleanObject

- parallel – this is an EAttribute and it is of type EBooleanObject
- alternative – this is an EAttribute and it is of type EBooleanObject
- prioritizedConcurrency – this is an EAttribute and it is of type EBooleanObject
  
- property – these are the multiple EReferences and they are of type ‘property’ (see below)
  - used for extendibility and flexibility.
- actor – this is a single EReference and it is of type ‘actor’
  - reference to actor to which the goal belongs to – it can be set only for root (top) level goals i.e. only root level goals have this attribute.
  - used to determine actor of the goal (can be only one).
  - reciprocal to ‘goals’ reference in ‘actor’ class.
- dependencyFrom – these are the multiple EReferences and they are of type ‘dependency’ (see below)
  - dependency link that goes From the goal referenced here – outgoing dependencies.
- dependencyTo – these are the multiple EReferences and they are of type ‘dependency’ (see below)
  - dependency link that goes To the goal referenced here (end point) – incoming dependencies.

**Comment:** There are 3 types of annotations in ASR models. Annotations allow us to capture dependencies among goals much more precisely. The 3 types are: ‘composition annotation’, ‘link annotation’ and ‘applicability condition annotation’.

The ‘composition annotations’ are applied to task and means-ends decompositions to specify how the subtasks are to be combined in order to execute the supertask and achieve the goal.

There are 4 types of 'composition annotations': sequence (;) which is default for task decomposition, concurrency (||), prioritized concurrency (») and alternative (|) which is default for means-ends decompositions. In the OME another one was implemented before I started working on this; that is: exclusive composition.

The 'link annotations' are applied to subgoals to specify under what conditions they are supposed to be achieved. There are 6 types of link annotations: while loop, for loop, if condition, pick, interrupt and guard. The absence of 'link annotation' on a decomposition link means that there is no condition on the subgoal.

The 'applicability condition' applies to means-ends links used with goal achievement alternatives. They specify when the corresponding alternatives are applicable.

**Changes:** The following EAttribute was originally EReference and it was called 'annotation'. We wanted to be able to distinguish between 'link annotation', 'composition annotation' and 'applicability condition annotation' so I introduced a change to its name. The name was changed from 'annotation' to 'hasLinkAnnotation'. Its type was also changed because an addition was added to clarify it further. The hasLinkAnnotation attribute specifies if a goal has link annotation.

- hasLinkAnnotation – this is an EAttribute and it is of type EBoolean.
  - used to indicate if this goal has a link annotation.

**Changes:** The following EReference was added to be able to reference the objects of type linkAnnotation. This reference would point to an object of type linkAnnotation if the attribute hasLinkAnnotation is set to true.

- linkAnnotation – this is a single EReference and it is of type linkAnnotation; linkAnnotation is another EClass (see below).

- corresponds to link annotations in i\* (see below under linkAnnotation class).

**Changes:** The below EAttribute 'hasCompositionAnnotation' was added by me because existing representation (with no consistency check) could lead to wrong illustration since one should be able to choose only one composition annotation. The existing implementation was not removed; the new one was only added.

EAttribute hasCompositionAnnotation is of type EBoolean and it was added to be able to specify if a goal has composition annotation.

- hasCompositionAnnotation – this is an EAttribute and it is of type EBoolean.
  - used to indicate if this goal has a composition annotation.

**Changes:** The following EReference was added to be able to reference the objects of type compositionAnnotation. This reference would point to an object of type compositionAnnotation if the attribute hasCompositionAnnotation is set to true. If there is no composition annotation, then the corresponding goal would not have an object of this type.

- compositionAnnotation – this is a single EReference and it is of type compositionAnnotation; compositionAnnotation is another EClass (see below).
  - corresponds to composition annotations in i\* (see below under compositionAnnotation class).

**Changes:** The below EAttribute 'hasAppCondAnnotation' was added by me in order to add possibility of using 'applicability condition annotation'. It is type is EBoolean. If there is no applicability conditions specified, then this goal will have this attribute set to false, otherwise it will be set to true.

- hasAppCondAnnotation' – this is an EAttribute and it is of type EBoolean.
  - used to indicate if this goal has an applicability condition annotation.

**Changes:** The following EReference was added to be able to reference the objects of type appCondAnnotation'. This reference would point to an object of type appCondAnnotation if the attribute hasAppCondAnnotation'is set to true. If this goal does not have applicability condition, the it would not either have the reference to an object of appCondAnnotation type.

- appCondAnnotation – this is a single EReference and it is of type appCondAnnotation; appCondAnnotation is another EClass (see below).
  - corresponds to applicability condition annotations in i\* (see below under linkAnnotation class).

**Changes:** The following EAttribute 'isPrimAction' was added by me in order to be able to specify if this task is a primitive action. If it is, then it will be set to true; otherwise it will be set to false.

- isPrimAction – this is an EAttribute and it is of type EBoolean
  - used to indicate if this task is a primitive action or not

**Changes:** The following EReference 'primAction' was added by me in order to be able to reference an object of type primAction. If this task is not a primitive action, it will not have a reference to an object of this type.

- primAction – this is EReference and it is of type primAction
  - used to specify if this task is a primitive action; if it is, then this can also be used for its preconditions and effects (refer to the primAction class below).

**Changes:** This EAttribute is added in order to be able to specify the actual parameters for parameter binding).

- parameterBinding – this is an EAttribute and it is of type 'EString';
  - used for actual parameters that come in place of formal parameters.

### contribution

EClass 'contribution' has:

- type – this is an EAttribute and it is of type ContributionType;  
ContributionType is of type EEnum and it can have values: help(1), hurt(-1), make(2), break(-2)
  - ContributionType can have 4 different values: help, hurt, make and break while in ASR diagrams softgoal contribution links can have one of two values: help (+) or hurt (-).
- target – this is a single EReference and it is of type 'goal'
  - the goal that this contribution link is referring to.

### dependency

EClass 'dependency' has:

- dependencyFrom – this is a single EReference and it is of type 'goal'
  - dependency link goes From the goal referenced here.
- dependencyTo – this is a single EReference and it is of type 'goal'
  - dependency link goes To the goal referenced here.
- trust – this is an EAttribute and it is of type EFloat
  - the corresponding number indicates the strength of the dependency.

### property

EClass 'property' has:

- name – this is an EAttribute and it is of type EString

- name of the property .
- value – this is an EAttribute and it is of type EString
  - value of the property.

**Changes:** I added the EAttribute 'expression' to the class 'topic'. Since the EAttribute 'name' specifies the names of parameters (formal parameters) and the EAttribute 'type' specifies the type of these parameters, I decided to add the 'expression' attribute to be able to specify the values of the parameters (actual parameters).

#### topic

EClass 'topic' has:

- name – this is an EAttribute and it is of type EString
  - used for the names of the parameters.
- type – this is an EAttribute and it is of type EString
  - used for the types of the parameters.
- expression – this is an EAttribute and it is of type EString
  - used for the actual values of the parameters.

**Changes:** The values in the attribute 'type' of the following EClass 'linkAnnotation' that were there originally are: 0 (none), 1 (condition), 2 (while) and 3 (guard). The values for 'for', 'pick' and 'interrupt' were missing; so I added the 4 (for), 5 (pick) and 6 (interrupt).

#### linkAnnotation

EClass 'linkAnnotation' has:

- type – this is an EAttribute and it is of type linkAnnotationType; linkAnnotationType is of type EEnum and it can have values: none (0), condition (1), while (2), guard (3), for (4), pick (5), interrupt (6)

- corresponds to the representation of link annotations in i\* framework: if (condition in goalmodel), while loop (while in goalmodel), guard (guard in goalmodel), for loop (for in goalmodel), pick (pick in goalmodel) and interrupt (interrupt in goalmodel).
- details – this is an EAttribute and it is of type EString
  - used for parameters' details i.e. to store the condition and variables under which this annotation applies.

**Changes:** The following EClass 'compositionAnnotation' was added by me to complement the above changes with added compositionAnnotation EReference.

#### compositionAnnotation

EClass 'compositionAnnotation' has:

- type – this is EAttribute and it is of type compositionAnnotationType; compositionAnnotationType is of type EEnum and it can have values: sequential (0), parallel (1), prioritizedConcurrency (2), alternative (3), exclusive (4)
  - corresponds to the representation of composition annotations in i\* framework: sequential (;), parallel or concurrency (||), prioritized concurrency (»), alternative (|), exclusive.

**Changes:** The following EClass 'appCondAnnotation' was added by me to complement the above changes (added appCondAnnotation EReference).

#### appCondAnnotation

EClass 'appCondAnnotation' has:

- details – this is an EAttribute and it is of type EString;

- it is used to store the condition under which the corresponding alternatives are applicable (applicability condition is used with goal achievement alternatives).

**Comment:** The following class 'effect' is used for the Domain Specification of the ConGolog model of a domain.

It is used for successor state axioms that specify which fluents change after some actions are executed and how they change. It tells us that the value of the fluent changes to the new value after action is executed and provided that the condition holds.

**Changes:** Class 'effect' was added by me to be able to specify the successor state axioms in the Domain Specification of the ConGolog model of a domain.

It is used to indicate that the value of the given 'fluent' changes to the 'new value' (also given) after action (this task) is executed and provided that the given 'condition' holds.

### effect

EClass 'effect' has:

- fluent – this is an EAttribute and it is of type EString
  - the fluent whose value changes upon execution of the corresponding action (here, action is the task that is a primitive action).
- newValue – this is an EAttribute and it is of type EString
  - refers to the new value of fluent; the value of fluent changes to this new value after execution of some action (here, action is the task that is a primitive action).
- condition – this is an EAttribute and it is of type EString
  - the value of fluent changes provided that this condition holds.

**Changes:** The following class 'primAction' was added by me to be able to distinguish between the tasks that are primitive actions and those that are represented in ConGolog with a procedure and to specify the precondition axioms and successor state axioms of primitive actions in the Domain Specification.

It contains name, input parameters, precondition and effects.

One can have or not have a precondition; if there is precondition, it is specified in the given string and if there is none, it should be set to 'true'.

Class 'effect' (see above) is used as a type for the 'hasEffect' EReference; a primitive action can have multiple effects.

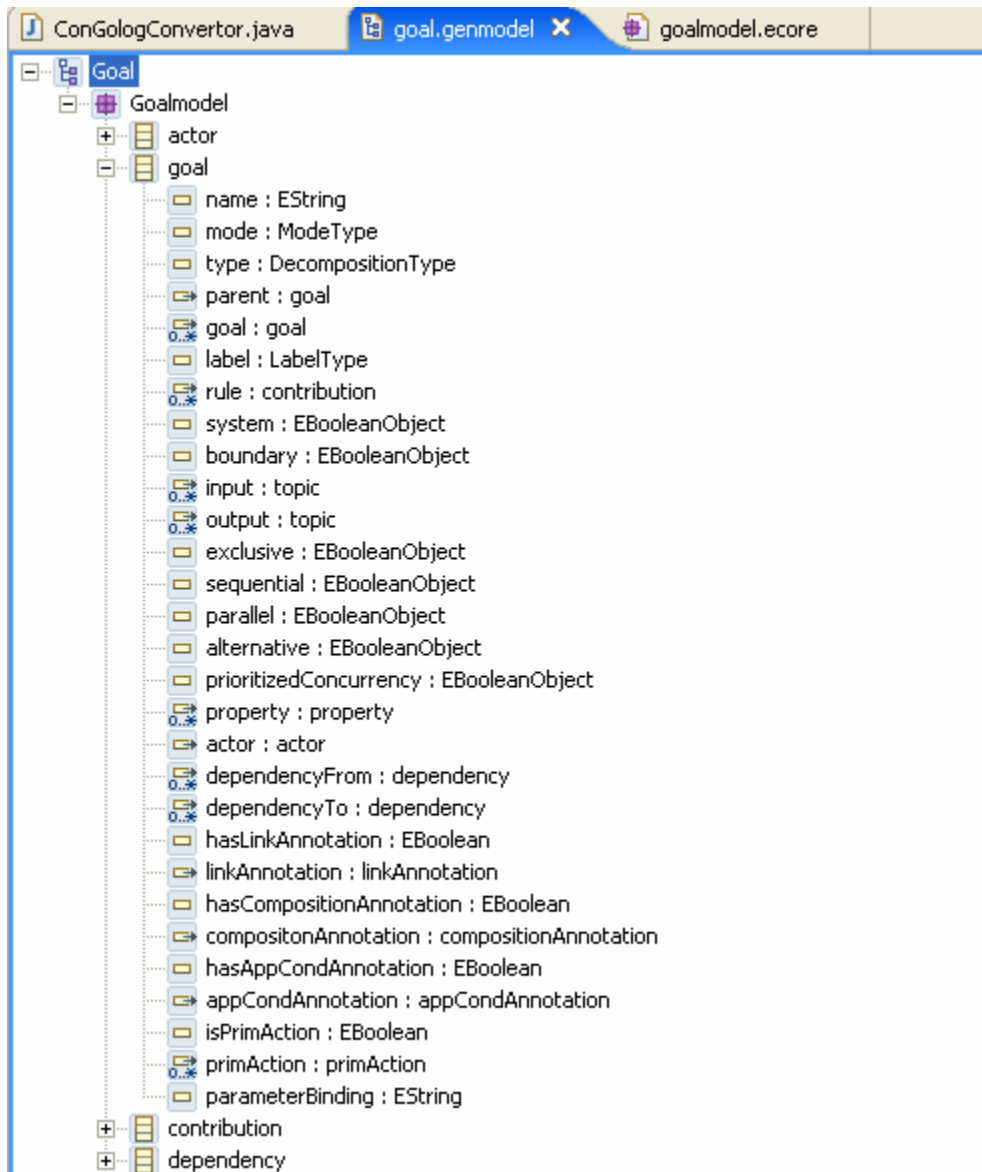
### primAction

EClass 'primAction' has:

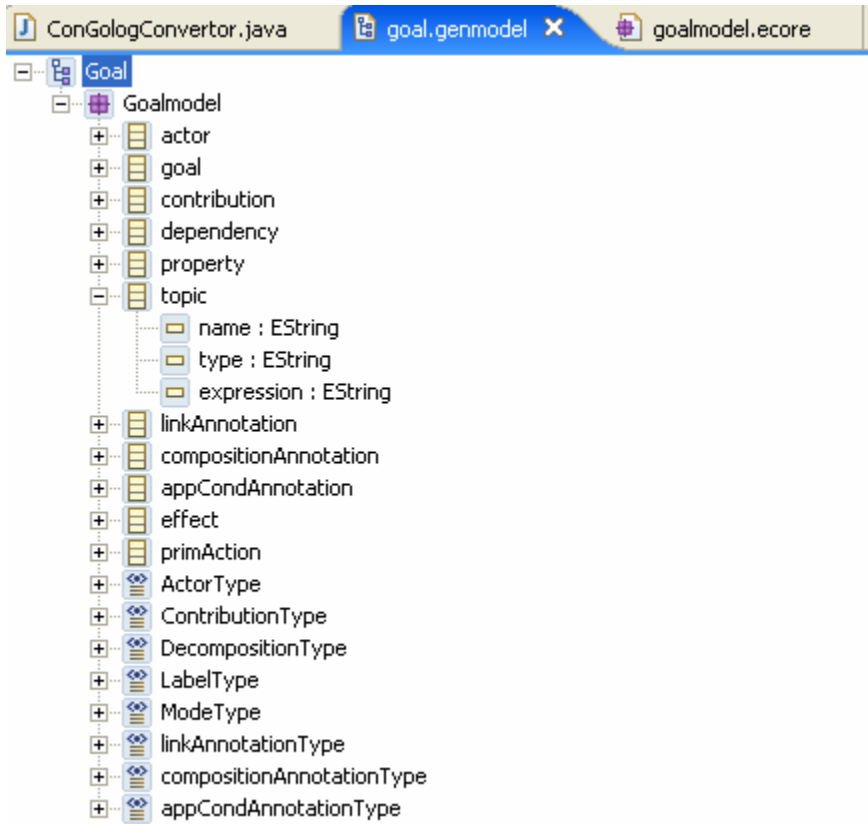
- name – this is an EAttribute and it is of type EString
  - the name of the primitive action.
- input – these are the EReferences and they are of type 'topic'
  - refers to the input parameters.
- hasPrecondition – this is EReference and it is of type EString
  - refers to the precondition axiom of this action.
- hasEffect – these are the EReferences and they are of type 'effect'
  - refer to the successor state axioms of this action (multiple effects are possible).

The majority of the changes can be found in the pictures on the following pages.

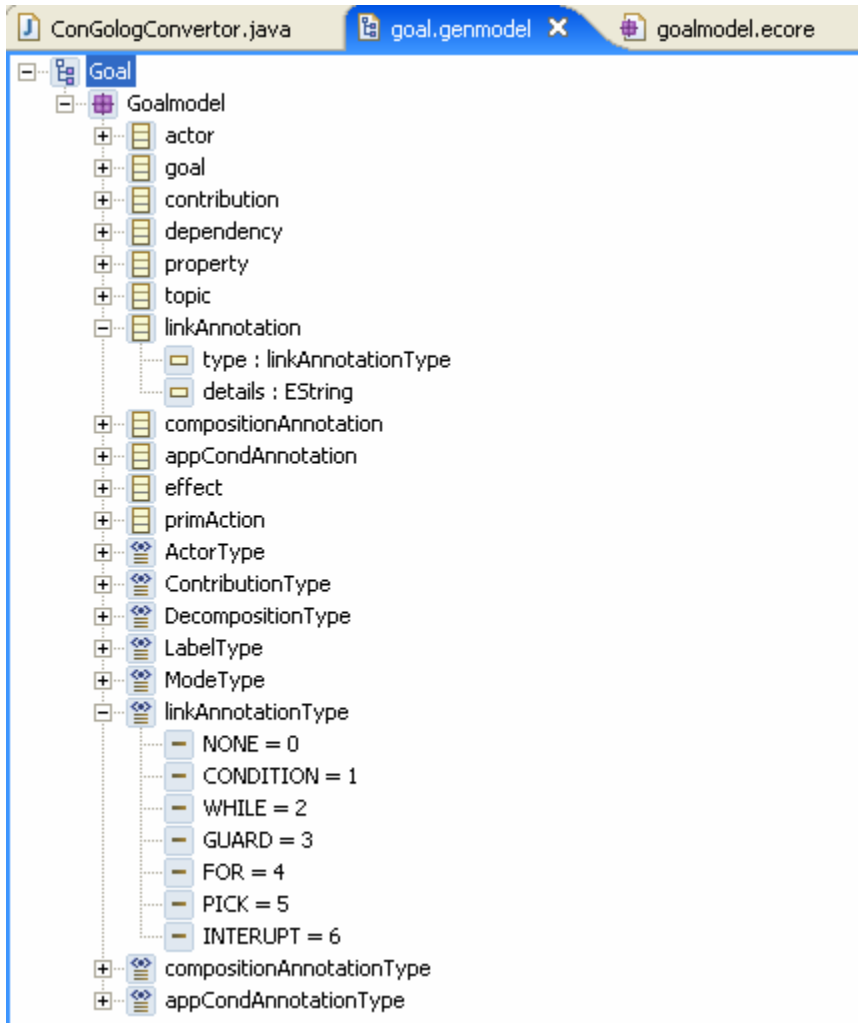
## 1. The changes made on the 'goal' class



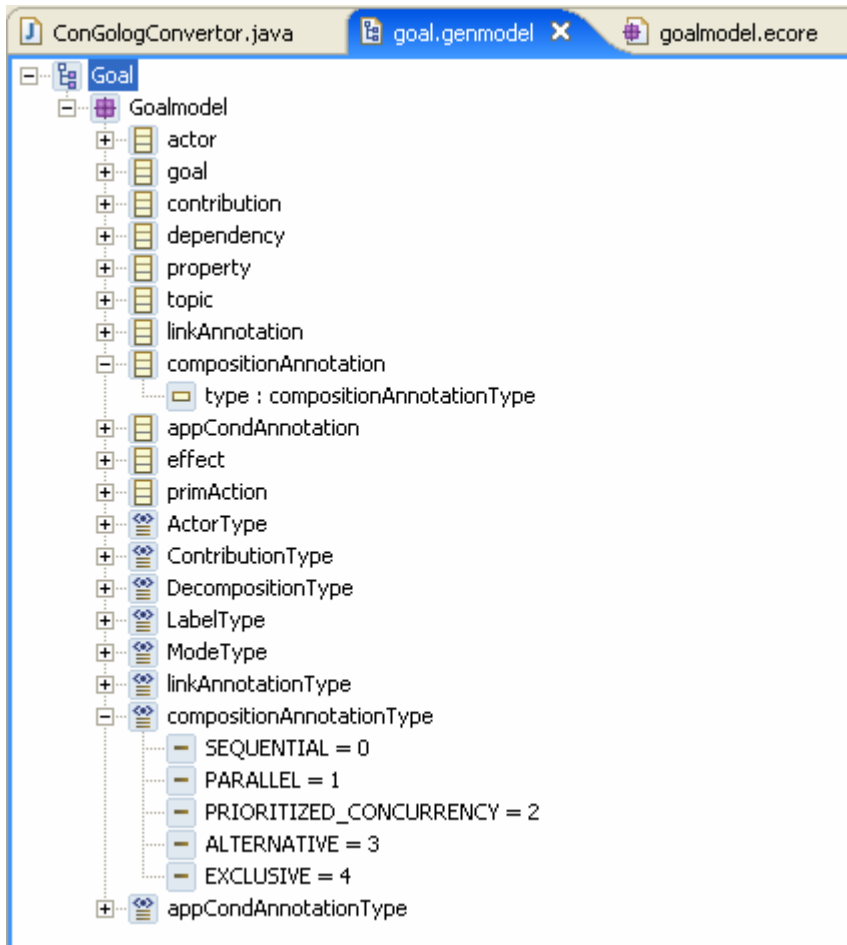
## 2. The changes done on the 'topic' class



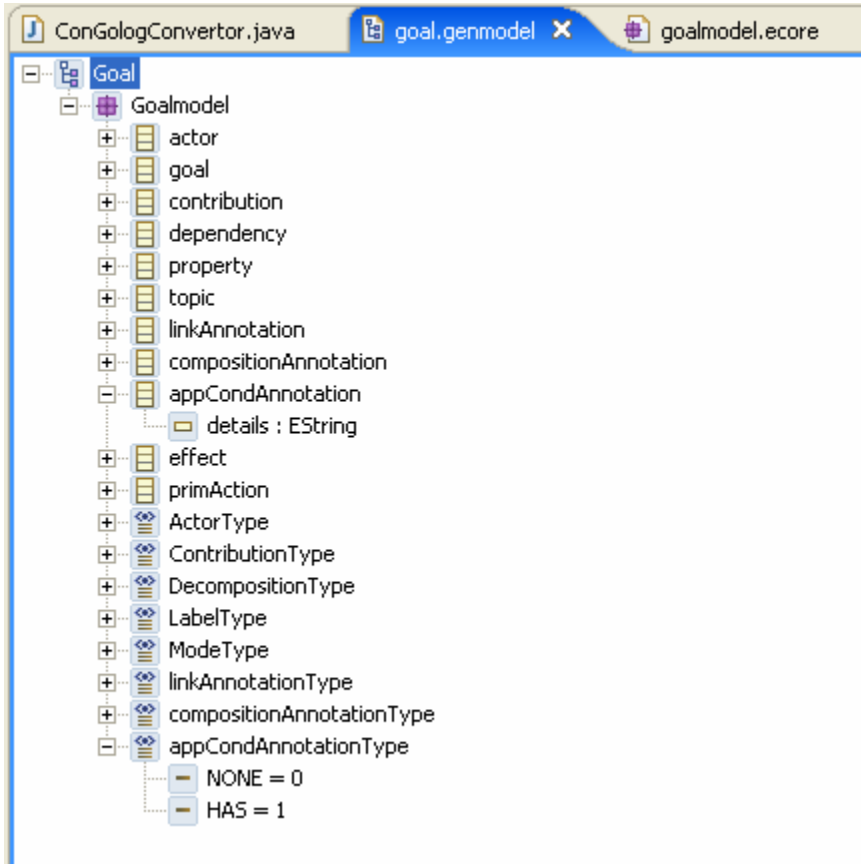
### 3. The changes done on the 'linkAnnotation class and on the linkAnnotationType



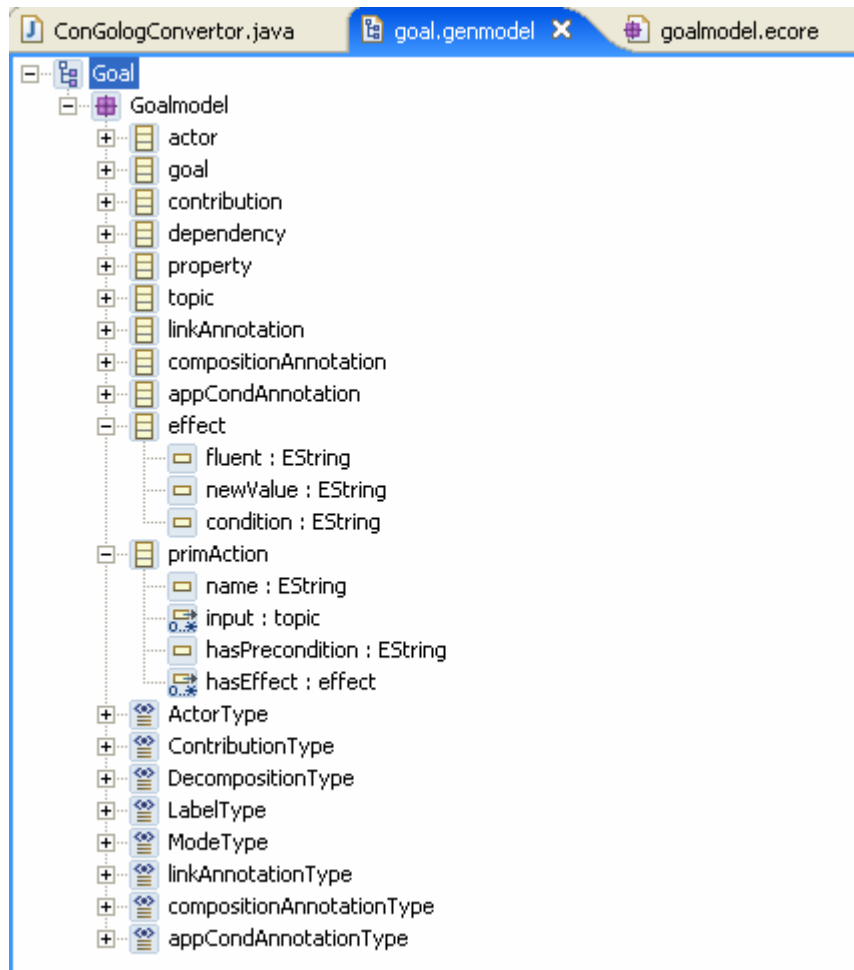
4. The changes done on the 'compositionAnnotation' class and on the compositionAnnotationType



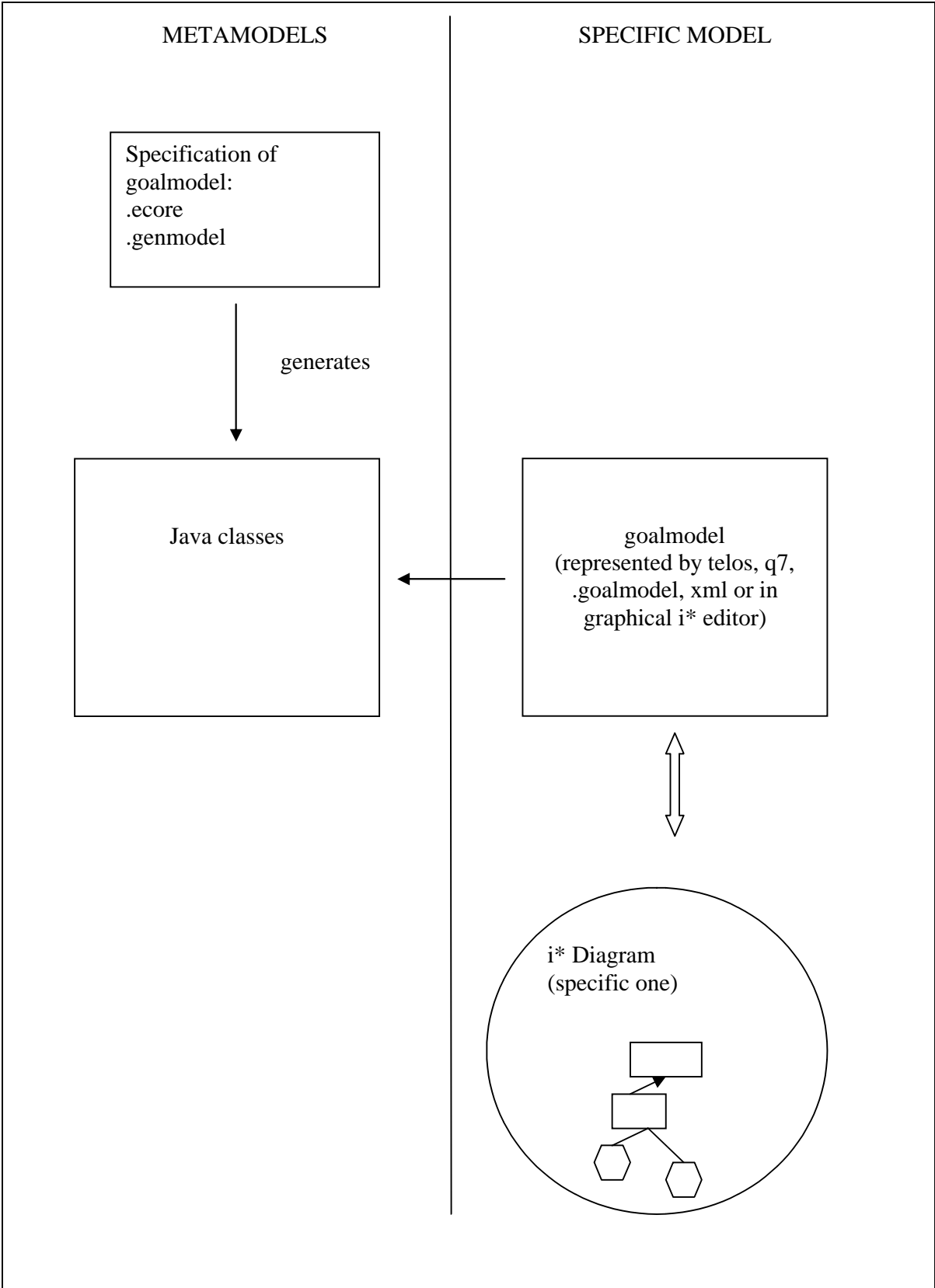
5. The changes done on the 'appcondAnnotation' class and on the appCondAnnotationType



## 6. The changes done on the 'effect' and 'primAction' classes



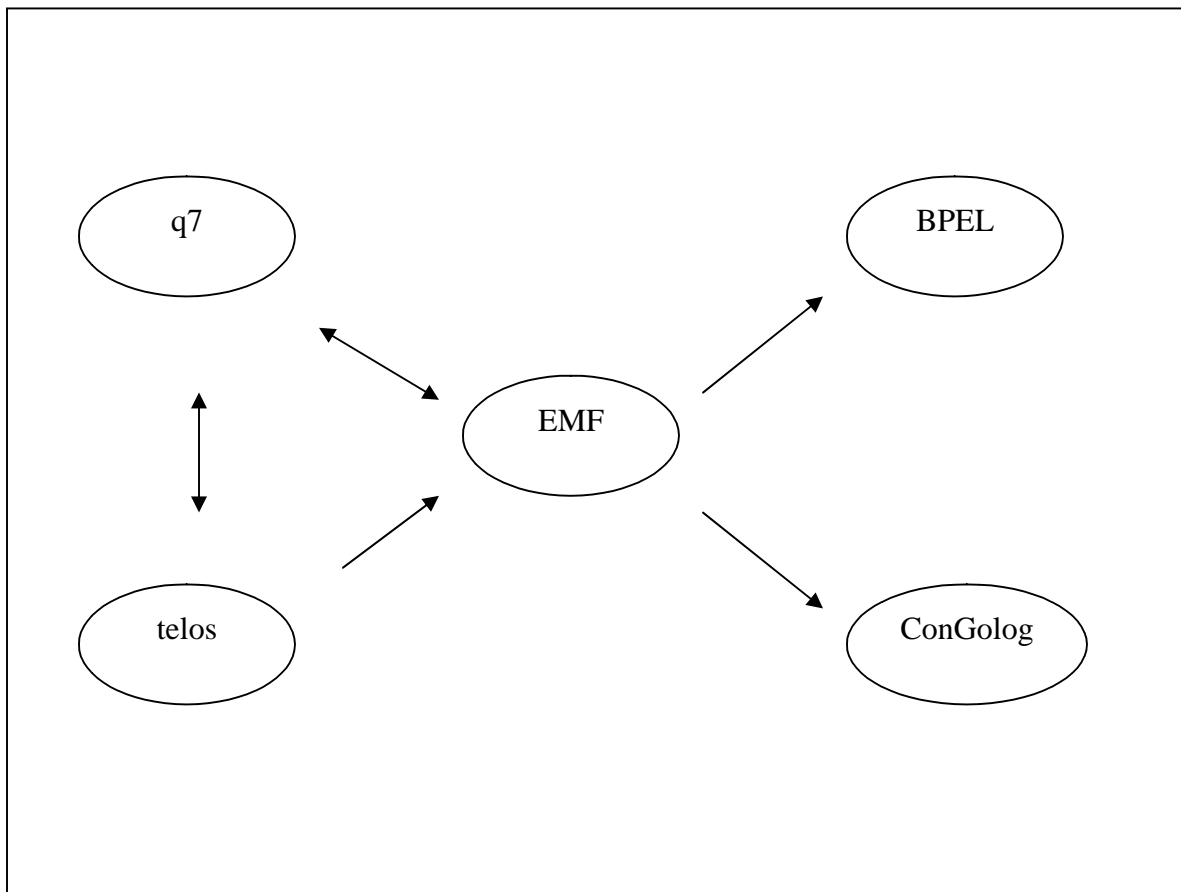
What follows is the rough diagram representation of the above described structure:



### Converter that translates i\* into ConGolog code

The second part of the project is the converter that translates the specific goalmodel (specified in telos, q7 or i\*) into the executable ConGolog code.

Several conversions are possible; they are represented by the following diagram:



The converter I am working on in this project is the one that translates to the ConGolog code. It can be found in the ConGologConverter.java file in the edu.toronto.cs.q7.load package.

The easiest way for me to start working on this converter was to first create examples that I would try to translate and then implementing the translator that would work for them. Further examples would be used for thorough testing and then many more to implement the rest of the converter. These examples I built initially in the i\* graphical editor and then I enriched the goalmodel in the corresponding .genmodel file.

The following steps outline the way a goalmodel example can be enriched and/or changed:

- ✓ Open the example in the Goalmodel Model Editor in OpenOME: right-click on the corresponding .goalmodel file in the left-most Navigator pane and choose Open With → Goalmodel Model Editor.
- ✓ After the example is opened in the middle pane, click on a feature and its properties will appear in the right-most Properties pane. Here, different properties can be changed as desired. If the Properties pane is not opened already, that can be done in the following way: click on Window → Show View → Other... Now the Show View window will open; click on the '+' sign beside Basic tag and choose Properties.
- ✓ If an additional attribute/reference needs to be added, this can be done in the following way: for example, if one needs to add another goal, right-click on the 'goal' attribute in the Goalmodel Model Editor and choose New Child / New Sibling (depending on if this new goal will be child or sibling of the existing one). After the new goal is added, its properties can be changed in the above described way.

The translator is written in Java and it uses the generated Java classes that get generated after creating the EMF model. These classes contain all the necessary 'get' and 'set' methods.

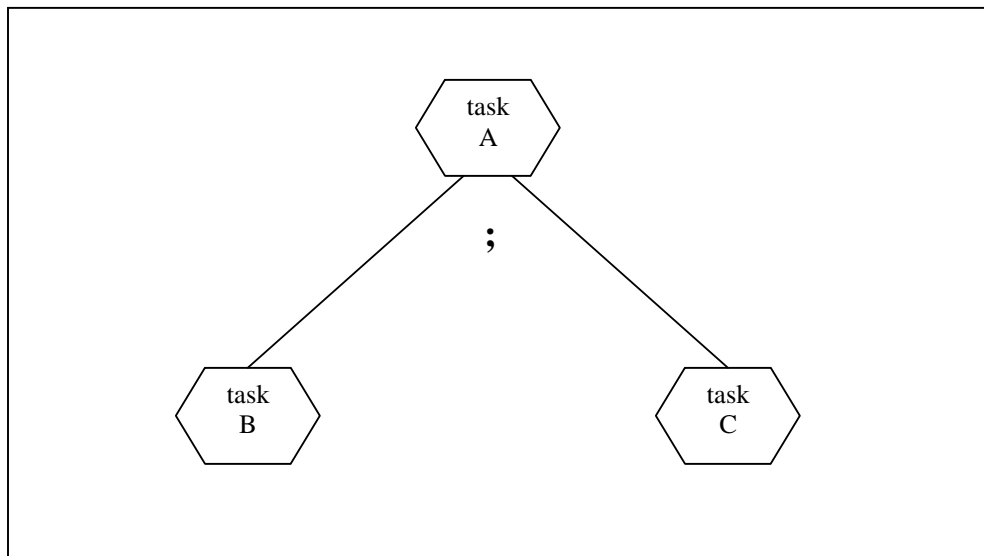
This is the example of the code of the ConGolog converter that contains the use of the get methods to get the information related to an example.

```
...
//CASE2: root is not a primitive action
EList children = task.getGoal();
//since it is not prim action, it has to have children/subgoals
String name = task.getName();
EList input = task.getInput();
...
```

In this example one can see the `getGoal()` method that is called on a task node. This method returns the list of the children of this task node. One can also see the use of the `getName()` and `getInput()` methods; both are called on a task node. The first one returns the name of this task node and the second one returns the list of inputs to this task node.

In the beginning of the translator development I consider only the simplest examples. For example, the first one does not contain any actors; it only has one task node that is decomposed into 2 other task nodes. Also, for the beginning I considered only the task nodes and intend to build on this.

The following diagram represents one such simple example:



The method that is called initially is the `convert()` method. This method calls the `outputGoal(goal g)` method and prints its output into a text file. This file will

finally contain the executable ConGolog code. It is important to mention that the initial situation and declaration of predicate fluents are meant to come from another file and therefore are not included in the code generated by ConGolog converter.

The method `outputGoal(goal g)` checks the mode of the goal and calls the appropriate methods accordingly. At this stage this method is able to handle the task nodes only.

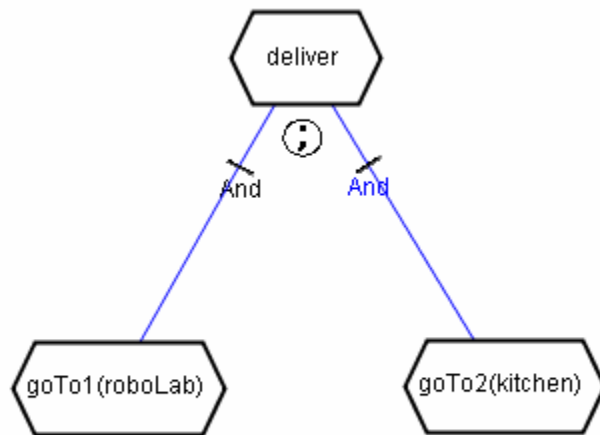
One of the issues that I encountered while working on the ConGolog translator comes with the formal and actual parameters for the ConGolog primitive actions and procedures.

This was solved in the following manner: All the primitive actions and procedures are stored in a hash table when encountered for the first time. They are stored with their name as the key and the primitive action or procedure definition as their value. For the definition, formal parameters are used while for the call, the actual parameters are plugged in.

I created six examples/test-cases that I tested my program on and that can also be used to show how it works.

The  $i^*$  diagrams and the emf-models are shown on the following pictures and the corresponding ConGolog code follows each. The  $i^*$  diagrams are incomplete due to the incompleteness of the graphical editor while in the emf-models the properties of the nodes cannot be seen from the pictures.

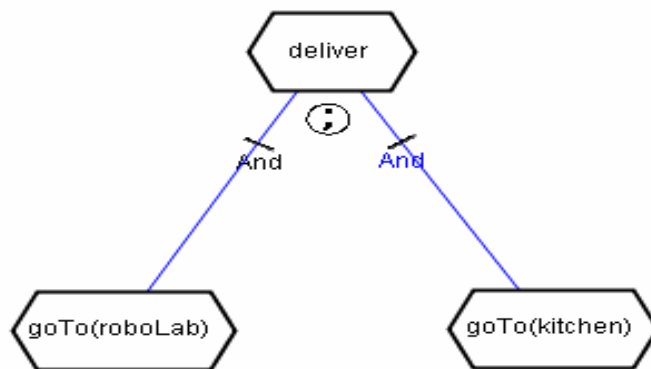
### Example 1

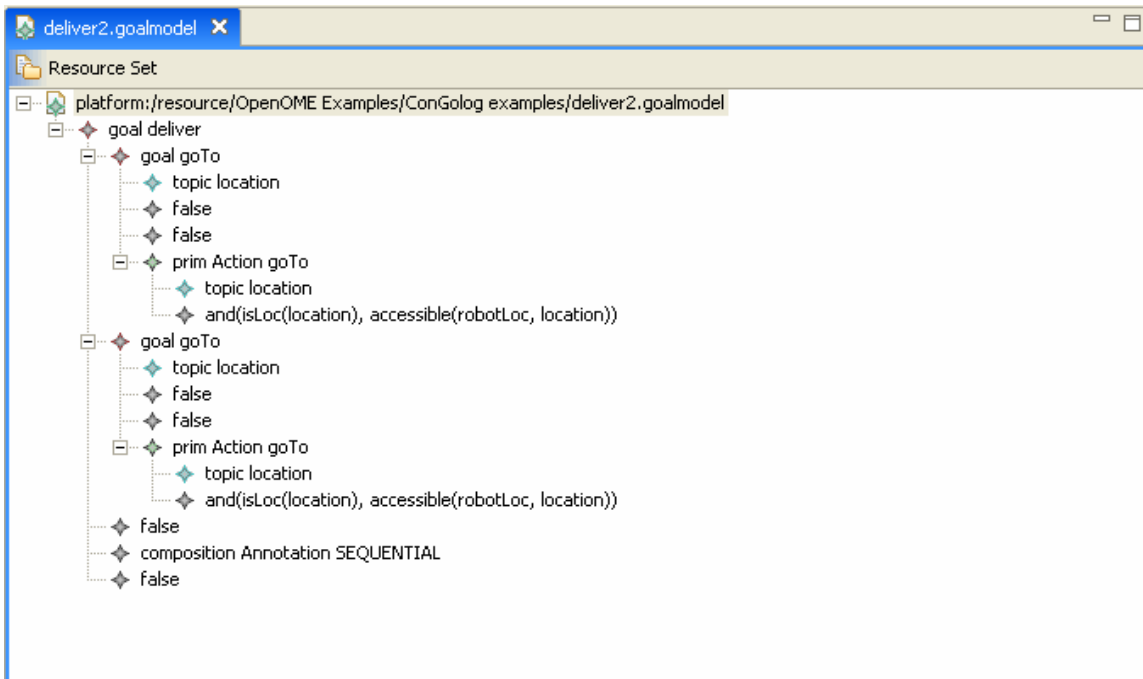


The screenshot shows a software interface for editing a goal model. The title bar reads 'deliver1.goalmodel'. Below it is a 'Resource Set' section with a tree view of the goal model structure. The root node is 'goal deliver'. It has two children: 'goal goTo2' and 'goal goTo1'. Each of these goal nodes has a 'topic location' property, two 'false' properties, and a 'prim Action' node. The 'prim Action goTo2' node has a 'topic location' property and an 'and(isLoc(location), accessible(robotLoc, location))' constraint. The 'prim Action goTo1' node has a 'topic location' property and an 'and(isLoc(location), accessible(robotLoc, location))' constraint. At the bottom of the tree, there are three more nodes: 'false', 'composition Annotation SEQUENTIAL', and 'false'.

```
deliver1.pl x
proc (deliver, [goTo2(kitchen), goTo1(roboLab)]).
    prim_action(goTo2(location)).
    poss(goTo2(location), and(isLoc(location), accessible(robotLoc, location))).
    prim_action(goTo1(location)).
    poss(goTo1(location), and(isLoc(location), accessible(robotLoc, location))).
```

*Example 2*





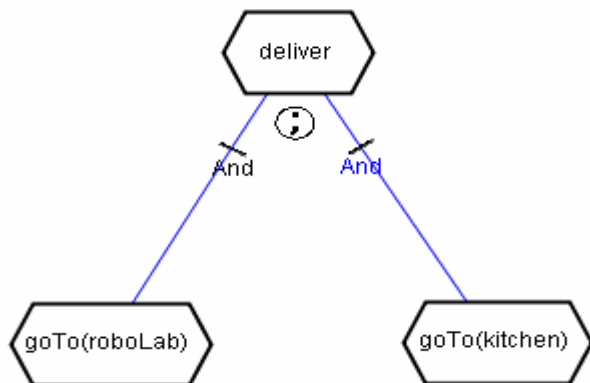
```

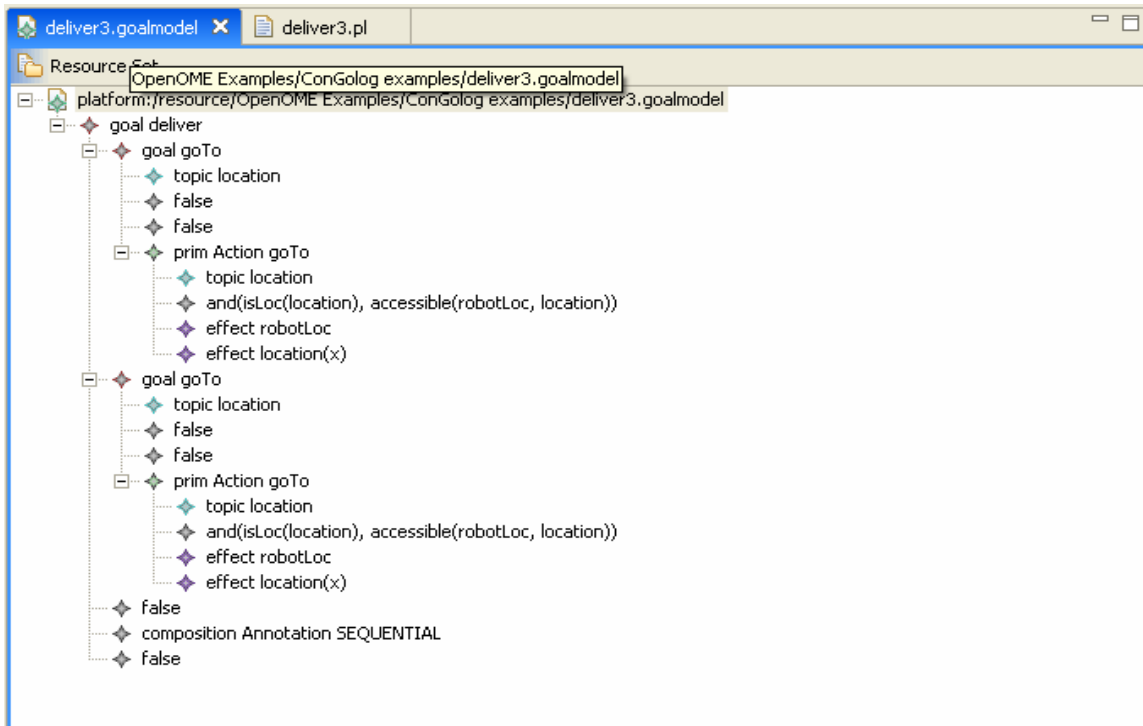
proc(deliver, [goTo( roboLab), goTo( kitchen) ] ).

prim_action(goTo( location) ).
poss(goTo( location), and(isLoc( location), accessible( robotLoc, location) ) ).

```

*Example 3*





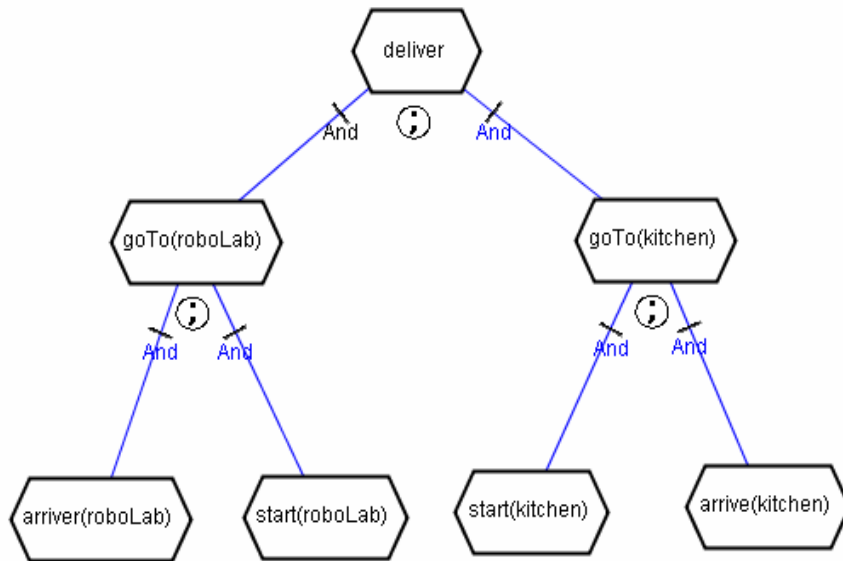
```

proc(deliver, [goTo( roboLab), goTo( kitchen) ] ).

prim_action(goTo( location) ).
poss(goTo( location), and(isLoc( location), accessible(robotLoc, location) ) ).
causes_val(goTo( location), robotLoc, location, true).
causes_val(goTo( location), location(x), location, onBoard(x) ).

```

Example 4





```

deliver4.pl
proc(deliver, [goTo( roboLab), goTo(kitchen)]).

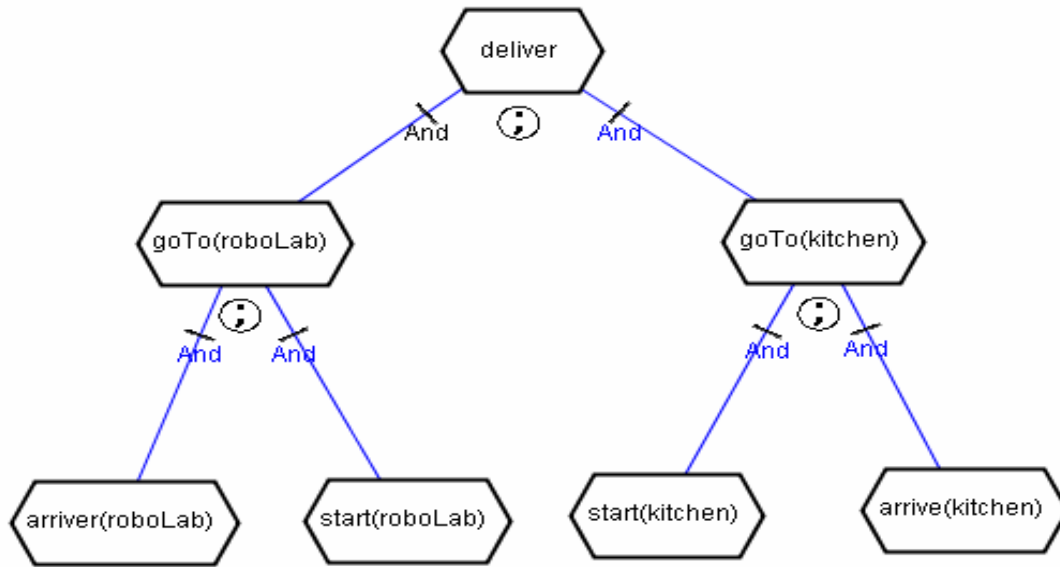
proc(goTo(location), [start(location), arrive(location)]).

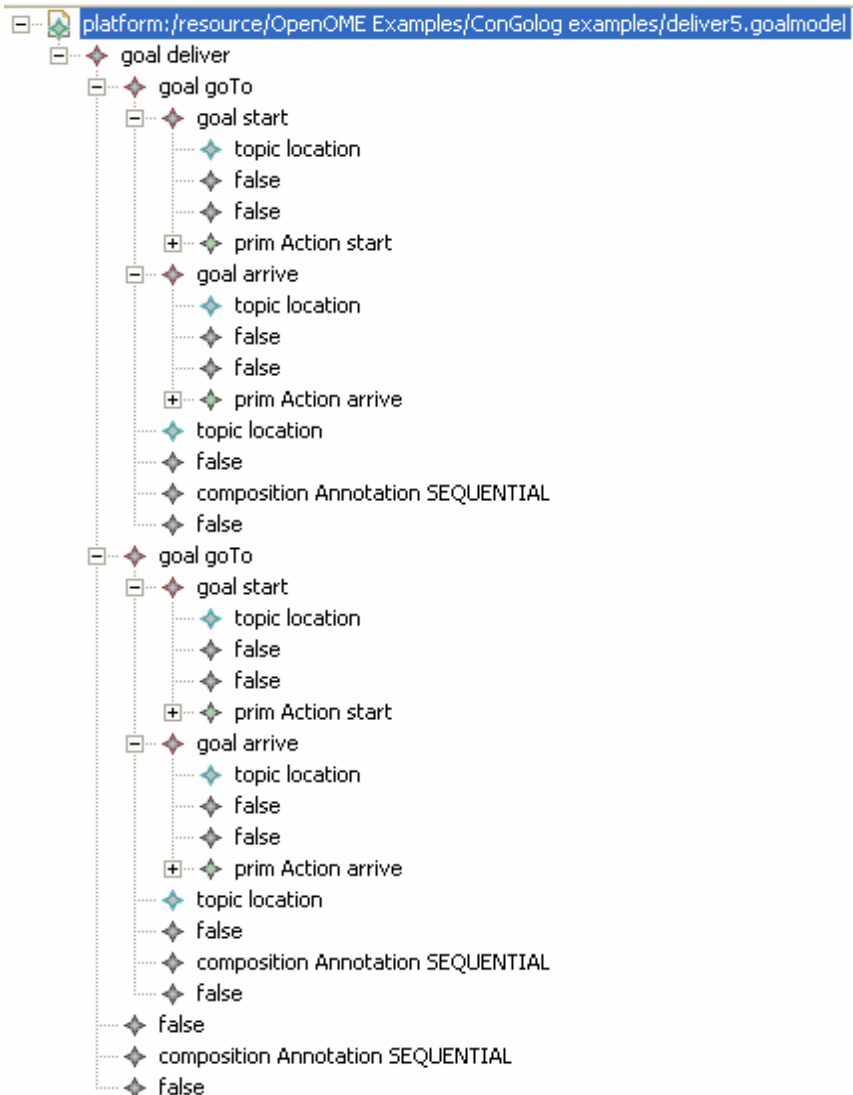
prim_action(start(location)).
poss(start(location), and(isLoc(location), accessible(robotLoc, location))).

prim_action(arrive(location)).
poss(arrive(location), and(isLoc(location), accessible(robotLoc, location))).

```

Example 5





```

deliver5.pl
proc(deliver, [goTo(robolab), goTo(kitchen)]).

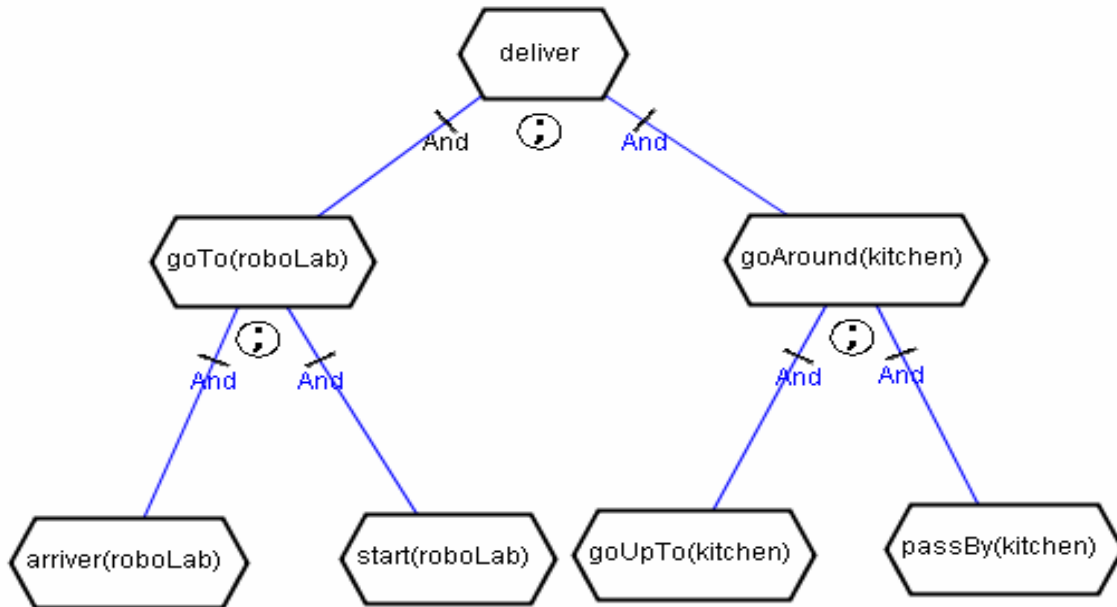
proc(goTo(location), [start(location), arrive(location)]).

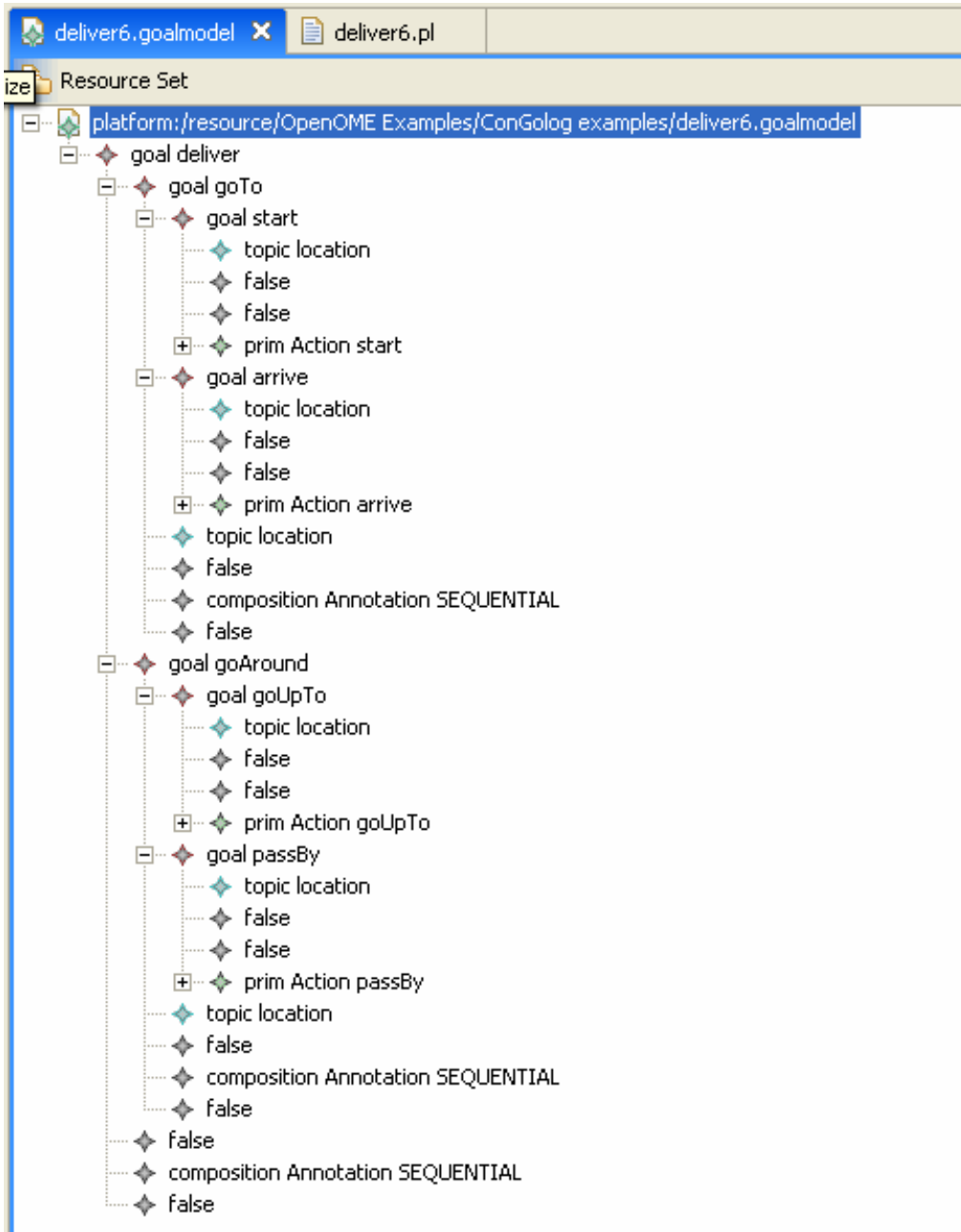
prim_action(start(location)).
poss(start(location), and(isLoc(location), accessible(robotLoc, location))).
causes_val(start(location), robotLoc, location, true).
causes_val(start(location), location(x), location, onBoard(x)).

prim_action(arrive(location)).
poss(arrive(location), and(isLoc(location), accessible(robotLoc, location))).
causes_val(arrive(location), robotLoc, location, true).
causes_val(arrive(location), location(x), location, onBoard(x)).

```

Example 6





```
deliver6.pl x
proc(deliver, [goTo(robolab), go&round(kitchen)]).

proc(goTo(location), [start(location), arrive(location)]).

prim_action(start(location)).
poss(start(location), and(isLoc(location), accessible(robotLoc, location))).
causes_val(start(location), robotLoc, location, true).
causes_val(start(location), location(x), location, onBoard(x)).

prim_action(arrive(location)).
poss(arrive(location), and(isLoc(location), accessible(robotLoc, location))).
causes_val(arrive(location), robotLoc, location, true).
causes_val(arrive(location), location(x), location, onBoard(x)).

proc(go&round(location), [goUpTo(location), passBy(location)]).

prim_action(goUpTo(location)).
poss(goUpTo(location), isLoc(location)).
causes_val(goUpTo(location), robotLoc, location, true).
causes_val(goUpTo(location), location(x), location, onBoard(x)).

prim_action(passBy(location)).
poss(passBy(location), isLoc(location)).
causes_val(passBy(location), robotLoc, not(location), true).
causes_val(passBy(location), location(x), not(location), onBoard(x)).
```

## Conclusions and further research

I am very pleased with the project as a whole, as well as with the knowledge and experience that I acquired while working on it.

The first part was easier to manage but it certainly cannot be completed while other parts of the OME that depend on it are still being developed. While working on the other parts of the project, there is always a possibility that one will find that something is missing in the emf-model or that some part is better implemented in a different way. Anyhow, one should be careful not to change the model representation too much as other parts of OME depend on it and do not get adjusted automatically.

The second part of the project on the other hand is much more demanding and it also depends on the emf-model. As any program it has to be developed with care and so that it is able to handle all the possible inputs. The developer will also find the need to keep changing and adjusting the code with the changes in the other parts of OME.

In the future, the emphasis should be put on the development of the ConGolog converter while the emf-model can be considered mostly complete at this point. The converter still requires lots of research, work and coding before it can be completed.

Finally, I would like to thank my supervisor Dr. Yves Lesperance for his support, help and guidance. Also, I had a significant help that is greatly appreciated from Dr. Yijun Yu from University of Toronto.

## Appendix A

Code of the ConGolog Converter program:

```
C:\workspaces\openome-sdk\edu.toronto.cs.q7\src\edu\toronto\cs\q7\load
```

```
package edu.toronto.cs.q7.load;
```

```
import java.io.File;
```

```
import java.io.FileNotFoundException;
```

```
import java.io.PrintStream;
```

```
import java.util.Iterator;
```

```
import java.util.Hashtable;
```

```
import org.eclipse.emf.common.util.EList;
```

```
import org.eclipse.emf.common.util.URI;
```

```
import org.eclipse.emf.ecore.resource.Resource;
```

```
import org.eclipse.emf.ecore.resource.ResourceSet;
```

```
import org.eclipse.emf.ecore.resource.impl.ResourceSetImpl;
```

```
import org.eclipse.emf.ecore.xmi.impl.XMIResourceFactoryImpl;
```

```
import edu.toronto.cs.goalmodel.*;
```

```
import edu.toronto.cs.goalmodel.GoalmodelPackage;
```

```
import edu.toronto.cs.goalmodel.ModeType;
```

```
import edu.toronto.cs.goalmodel.actor;
```

```
import edu.toronto.cs.goalmodel.goal;
```

```
import edu.toronto.cs.util.Computing;
```

```
public class ConGologConvertor
```

```
{
```

```
    private String filename;
```

```
    private String output_file;
```

```
    //Hashtable to store all the definitions of
```

```
    //the procedures with name of the procedure as
```

```
    //the key and its definition as the value
```

```
    private Hashtable procedures = new Hashtable();
```

```

public ConGologConvertor(String file, String out)
{
    filename = file;
    output_file = out;
}

public String outputGoal(goal root)
{
    ModeType mode = root.getMode();
    if(mode == ModeType.get(ModeType.TASK))
    {
        // The type of this node is task
        return outputNodeTask(root);
    }else
    // TO DO: add the rest of mode types, for now return anything
    return "";
}

private String outputNodeTask(goal task)
{
    String out = "";
    if(task.getPrimAction().size() > 0)
    {
        //CASE1: root is primitive action
        //getPrimAction returns 0 if it is not a prim act
        //and 1 if it is, meaning it has one prim action object/node
        primAction primAct;
        primAct = (primAction) task.getPrimAction().get(0);
        String name = primAct.getName();
        EList input = primAct.getInput();

        //Hashtable checking
        if (name != null)

```

```

{
    boolean contains = procedures.containsKey(name);
    if(contains)
    {
        //do nothing - everything has been already done
    }else
    {
        //it's not in defined procedures, output it and store it in
        //hashtbl
        String value = "";
        if(input.size() == 0)
        {
            value = "prim_action(" + name + ")." + "\n";
        }else
        {
            value = "prim_action(" + name + "(" +
                getInputNames(input) + ").\n";
            out += "prim_action(" + name + "(" +
                getInputNames(input) + ").\n";
        }
        procedures.put(name, value);

        //get the precondition and output it
        String precond = primAct.getHasPrecondition();
        out += "poss(" + name + "(" +
            getInputNames(input) + "), " + precond +
            ").\n";

        //get the effects and output them
        EList effects = primAct.getHasEffect();
        if(effects.size() != 0)
        {
            for(int i=0; i < effects.size(); i++)
            {
                //get the effects one by one and output it
            }
        }
    }
}

```

```

        effect eff = (effect) effects.get(i);
        String fluent = eff.getFluent();
        String newVal = eff.getNewValue();
        String cond = eff.getCondition();
        if(i == effects.size()-1)
        {
            out += "causes_val(" + name + "("
                + getInputNames(input) + "),"
                + fluent + ", " + newVal
                + ", " + cond + ").\n\n";
        }else
        {
            out += "causes_val(" + name + "("
                + getInputNames(input) + "),"
                + fluent + ", " + newVal
                + ", " + cond + ").\n";
        }
    }
} else //no effects, output nothing but an empty line...
{
    out += "\n";
}
} // name != null
} else //CASE2: root is not a primitive action
{
    //CASE2: root is not a primitive action
    EList children = task.getGoal();
    //since it is not prim action, it has to have children/subgoals
    String name = task.getName();
    EList input = task.getInput();
    //Hashtable checking
    if (name != null)

```

```

{
    boolean contains = procedures.containsKey(name);
    if(contains)
    {
        //do nothing
    }else
    {
        //it's not in defined procedures, output it and store it in
        String value = "";
        if(input.size() == 0)
        {
            value = "proc(" + name + ", [" +
                getSubGoalsExp(children) + "]).\n\n";
            out += "proc(" + name + ", [" +
                getSubGoalsExp(children) + "]).\n\n";
        }else
        {
            value = "proc(" + name + "("
                + getInputNames(input) + ")"
                + ", [" + getSubGoalsName(children)
                + "]).\n\n";
            out += "proc(" + name + "("
                + getInputNames(input) + ")"
                + ", [" + getSubGoalsName(children)
                + "]).\n\n";
        }
        procedures.put(name, value);
    }
}

//go onto the children and process them one at a time
for(int i=0; i < children.size(); i++)
{
    out += outputGoal((goal)children.get(i));
}

```

```

        }
    return out;
}

// Helping methods
//For the names of the inputs (formal parameters)
private String getInputNames(EList list)
{
    String out = "";
    for (int i=0; i < list.size(); i++)
    {
        if(i == list.size()-1)
        {
            out += ((topic)list.get(i)).getName();
        }else
        {
            out += ((topic)list.get(i)).getName() + ",";
        }
    }
    return out;
}

//For the expressions for the inputs (actual parameters)
private String getInputExpressions(EList list)
{
    String out = "";
    for (int i=0; i < list.size(); i++)
    {
        if(i == list.size()-1)
        {
            out += ((topic)list.get(i)).getExpression();
        }else
        {
            out += ((topic)list.get(i)).getExpression() + ", ";
        }
    }
}

```

```

        }
    }
    return out;
}

```

//For the expressions of the subgoals (actual parameters)

```

private String getSubGoalsExp(EList list)
{
    String out = "";
    for (int i=0; i < list.size(); i++)
    {
        goal g = (goal)list.get(i);
        String name = g.getName();
        if(i == list.size()-1)
        {
            out += name + "(" + getInputExpressions(g.getInput()) + ")";
        }else
        {
            out += name + "(" + getInputExpressions(g.getInput()) + "), ";
        }
    }
    return out;
}

```

//for the names of the subgoals (formal parameters)

```

private String getSubGoalsName(EList list)
{
    String out = "";
    for (int i=0; i < list.size(); i++)
    {
        goal g = (goal)list.get(i);
        String name = g.getName();
        if(i == list.size()-1)
        {

```

```

        out += name + "(" + getInputNames(g.getInput()) + ")";
    }else
    {
        out += name + "(" + getInputNames(g.getInput()) + ")";
    }
}
return out;
}

```

```

private String getActorName(goal g)
{
    actor a = g.getActor();
    String n = "";
    if (a!=null)
    {
        n = Computing.quotation(a.getName()) + n;
    }
    else
    { // actor must be inherited from the ancestor goals
        goal p = g.getParent();
        while (p!=null)
        {
            actor pa = p.getActor();
            if (pa!=null)
            {
                n = Computing.quotation(pa.getName()) + n;
                break;
            }
            p = p.getParent();
        }
    }
    return n;
}

```

```

// Convert method
public void convert()
{
    ResourceSet resourceSet = new ResourceSetImpl();
    resourceSet.getResourceFactoryRegistry().getExtensionToFactoryMap().put
        (Resource.Factory.Registry.DEFAULT_EXTENSION,new MIResourceFactoryImpl());
    resourceSet.getPackageRegistry().put(GoalmodelPackage.eNS_URI,
    GoalmodelPackage.eINSTANCE);
    File file = new File(filename);
    URI uri = file.isFile() ? URI.createFileURI(file.getAbsolutePath()):
    URI.createURI(filename);
    File q7_file = new File(output_file);
    PrintStream os;
    try
    {
        os = file.isFile() ? new PrintStream(q7_file)
        : System.out;
    } catch (FileNotFoundException e)
    {
        os = System.out;
    }
    try
    {
        Resource resource = resourceSet.getResource(uri, true);
        for (Iterator i = resource.getResourceSet().getAllContents();
        i.hasNext(); )
        {
            Object o = i.next();
            if (o instanceof goal)
            {
                goal g = (goal) o;
                String an = getActorName(g);
                if (an.equals("") && g.getParent() == null)
                { // is a root goal

```

```

        os.println(outputGoal(g));
    }
}
//for now without actor
//TO DO: add the possibility of having the actor
/** else if (o instanceof actor)
{
    actor a = (actor) o;
    String an = "<" + Computing.quotation(a.getName()) + ">";
    EList l = a.getGoals();
    for (int j =0; j < l.size(); j++)
    {
        goal root = (goal) l.get(j);
        String output = outputGoal(root, 0);
        os.println(an + "::" + output);
    }
}**/
}
}
catch (Exception exception)
{
    System.out.println("Problem loading " + uri);
    exception.printStackTrace();
}
}
}

```