DECLARE Manual

February 18, 2008

Contents

Chapter 1

Introduction

1.1 Introduction

DECLARE is a prototype of a workflow management system that takes a declarative approach to business process modelling. It can be used for specification and enactment of loosely-structured processes.

The basic difference between DECLARE and traditional approaches is in its understanding of what a process model is. Traditional process modelling languages understand a process as a directed graph, where control has a clearly defined begin, end and flow between begin and end, i.e., they specify in detail the "HOW". DECLARE process models focus on specifying the "WHAT" by introducing constraints in models as rules that have to be followed during execution. Figure 1.1 shows the declarative approach. Because of their imperative nature, traditional languages focus in specifying to details of how to execute processes. Therefore, due to exceptions and dynamics it is often necessary to deviate from the prescribed model. On the contrary, DECLARE specifies what to do by introducing some constraints in the model that should not be violated.

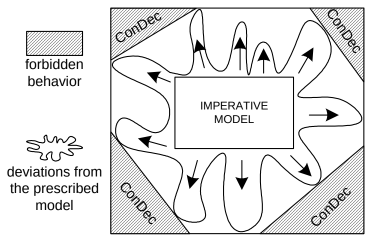


Figure 1.1: Declarative vs. imperative approaches.

DECLARE process models are constraint-based models, i.e., instead of explicitly modelling the control-flow, they model constraints on activities that represents some rules that should be followed. In this way, the control-flow is specified implicitly – as all executions that do not violate constraints are possible. Currently, DECLARE uses Linear Temporal Logic (LTL) [?] for constraint specification. Many algorithm were developed in the field of model-checking for translating LTL into finite state automata [?, ?]. DECLARE uses these automata for verification and execution of constraint models.

In simple cases, DECLARE can be used as a stand-alone workflow management system. However, it is also possible to use DECLARE for more complex applications together with YAWL and ProM (cf. Figure 1.2). YAWL is a workflow management system developed at Queensland University of Technology in Australia. It has a powerful process modelling language that supports all control-flow patterns. We propose to use YAWL for specification and enactment of high-level highly-structured operational processes, which are decomposed into loosely-structured DECLARE sub-processes. Both YAWL and DECLARE generate process execution logs that can be used in the ProM framework for process mining. Moreover, ProM framework already offers module for work with DECLARE logs. DECLARE language export can be used in ProM for process discovery, while DECLARE model exports can be used for conformance checking. Recommendation mechanism is a novelty which enables DECLARE process support by providing recommendations based on history. Recommendations are of significant importance in cases when users need advice based on past executions to help them choose between variety of options.

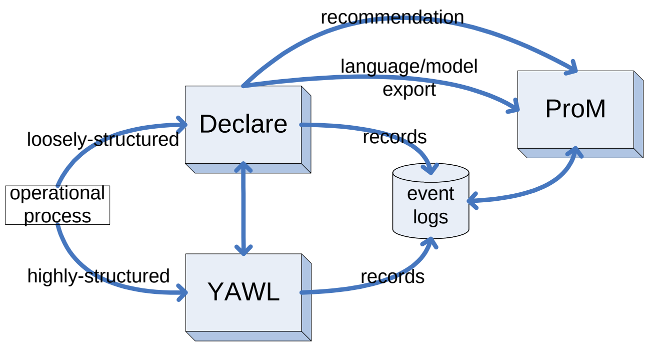


Figure 1.2: The architecture.

Chapter 2

Installing DECLARE

2.1 Installation

DECLARE is witten in `Java` and can be downloaded from `http://is.tm.tue.nl/staff/mpesic/`under terms of GNU licence[1]. Two modules are available for download: (1) (web) service module for connection with YAWL via its interface B, and (2) DECLARE itself. Note that core DECLARE system is not dependable on this service, YAWL or any other application - correct version of Java is the only requirement for running it.

2.1.1 Installing YAWL service for DECLARE

Requirements:

1. `declareService_executable.zip` (service itself) is obtainable at `http://is.tm.tue.nl/staff/mpesic/`;
2. YAWL system is obtainable at `http://yawlfoundation.org`;
3. Apache Tomcat is obtainable at `http://jakarta.apache.org/tomcat/`;
4. `Java 2` for running YAWL and declare service.

The service is contained in the file `declareService.war` that you can find in `declareService_executable.zip`. To use the service, you first have to have the Apache Tomcat `http://jakarta.apache.org/tomcat/`and YAWL system installed. YAWL system and related documentation can be obtained from `http://yawlfoundation.org`. Follow the YAWL documentation to install YAWL.

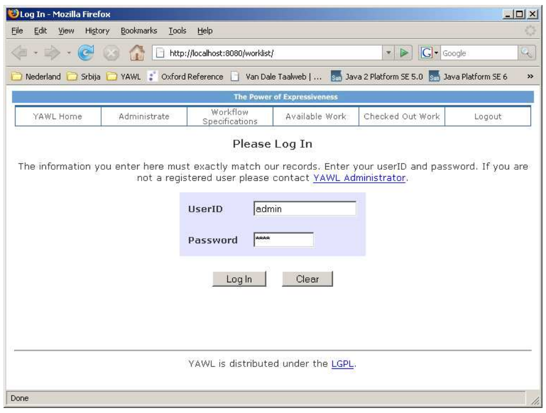
Launching the service To launch the service in your system, you have to install it as a web application in Apache Tomcat. To do this you need to:

[1]`http://www.gnu.org/licenses/gpl.html`

1. stop Apache server;
2. copy file `declareService.war` into the Apache folder for web applications, e.g., `C:\ProgramFiles\ApacheSoftwareFoundation\Tomcat5.0\webapps\`; and
3. start Apache server again.

If this process is completed successfully, there Apache should have automatically created a new folder for the service: `C:\ProgramFiles\ApacheSoftwareFoundation\Tomcat5.0\webapps\declareService\`

Registering the service in YAWL For the service to become operational it first has to registered in YAWL. Note that service registration is not persistent in YAWL - you have to register the service every time you restart YAWL. To access YAWL go to its worklist. Figure 2.1 shows YAWL worklist at log in. To register the service, log in and then go to link "Administrate". This page contains a list of registered YAWL services. To register DECLARE service enter `http://localhost:8080/declareService/ib`[2] as ServiceURI and click on the button "Add YAWL service" (cf. Figure 2.2). If this process is completed successfully, the new service should appear in the list of registered services.



Please Log In

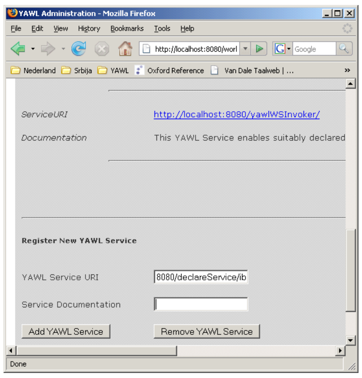
The information you enter here must exactly match our records. Enter your userID and password. If you are not a registered user please contact YAWL Administrator.

UserID: admin

Password: ****

YAWL is distributed under the LGPL.

Figure 2.1: Connecting to YAWL.


YAWL Administration - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://localhost:8080/worl
Register New YAWL Service
Add YAWL Service
Remove YAWL Service
Done

ServiceURI: http://localhost:8080/yawlWSInvoker/

Documentation: This YAWL Service enables suitably declared

YAWL Service URI: 8080/declareService/ib

Service Documentation:

Figure 2.2: Registering service in YAWL.

2.1.2 Installing DECLARE

Requirements:

1. `declare_executable.zip` (service itself) is obtainable at `http://is.tm.tue.nl/staff/mpesic/`;

[2]Note that the first part of the URI "localhost:8080" depends on the settings of your Apache Tomcat

2. `Java 5` for running DECLARE.

To install DECLARE unzip `declare_executable.zip`. You can start DECLARE applications by double-clicking on `designer.bat`, `framework.bat` or `worklist.bat`.

Chapter 3

Designing process models

The *Designer* can be used for designing DECLARE process models. It can also be used to chose a set of constraint templates that will be available for design. Moreover, it is possible to create new constraints and to add them to a template set. In the *Designer* you can also create end users and roles for the system and link roles to the user.

Launching the Designer To launch the Designer, double-click `designer.bat` in the declare folder.

Launch the Designer
1. Open Declare folder
2. Double-click `designer.bat`

Table 3.1: Launching the Designer

This will open the Designer as depicted in Figure 3.1. Initially the designer is empty. The designer contains two menu items, namely *Assignment model* and *Design*.

To create a new model (see Section 3.1), choose *Assignment model→new*, followed by the choice of template. To open or edit an exisisting model (see Section 3.1), choose *Assignment model→open.* CHECK WHEN FINISHED. To manage users, select *Design* and then *Organization*, or alternatively `Alt-R` (see Section 3.2.3). For opening the constraint templates, select *Design* followed by *Constraint templates*, or just press `Alt-C`(see Section 3.2.3).

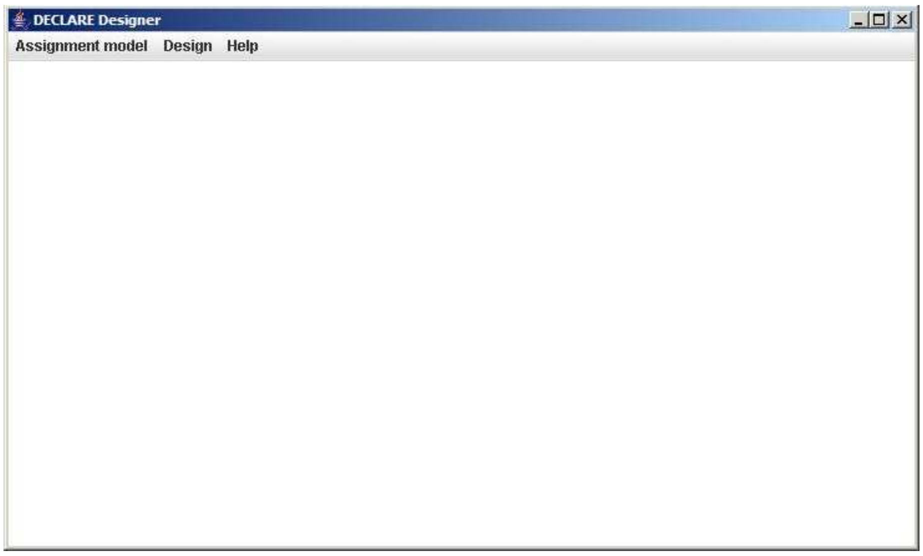


Figure 3.1: The designer after opening.

Alt	**Designer Menu**	
-O	Assignment model → open	open and edit an existing model
	Assignment model → new	create a new model
-R	Design → Organization	manage roles and users
-C	Design → Constraint Templates	manage constraint templates
		manage constraint groups

Table 3.2: The menus for the Designer

3.1 Creating a basic model

In this section we will explain how to create and verify a very simple DECLARE model that can be used for execution. This basic model will not contain any data or organizational aspects. In Section 3.2 we will explain how to extend a basic model with these aspects.

The actual process model is a set of constraints on activities. We consider the set of constraints on activities as work. DECLARE facilitates definition of people that can be involved in the execution of the process. Finally, data can be defined and initialized for the process.

To open an existing model click (*Assignment model→Open*), or alternatively, press `Alt-O`. To create a new model, click (*Assignment model→New*). Then choose one of the displayed constraint templates sets. Only constraint templates from this set will be available during design. Figure 3.2 shows the window for defining work in the model. The tabs of this window are explained in the remainder of this section.

Start a new model
1. Click `Assignent model`
2. Click `New`
3. Choose a template

Table 3.3: Start a new model

Opening an exisiting model
1. Click `Assignent model`
2. Click `Open` (or `ALT-O`)
3. Browse for the (XML) model

Table 3.4: Opening an existing model

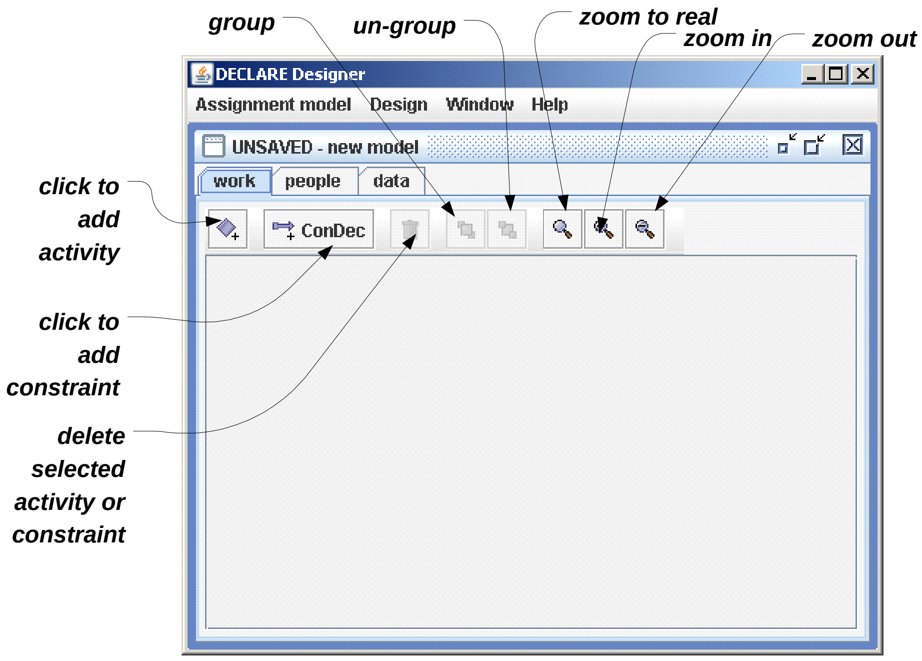


Figure 3.2: Creating a new model.

3.1.1 Defining activities

Activities can be created with the *create activity* button, the leftmost button in the window depicted in Figure 3.2. Clicking on this button will create a new activity in the window, as shown in Figure 3.3. Activities can be renamed in the *activity properties view*. To open this view, (*right-click activity* → *edit*). The name in the name field can now be retyped. Press `Ok` to apply the settings. The authorization and data properties are explained later.

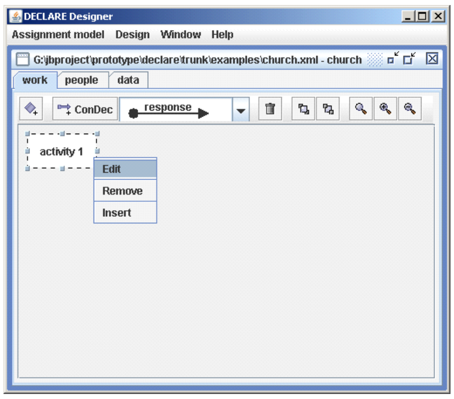
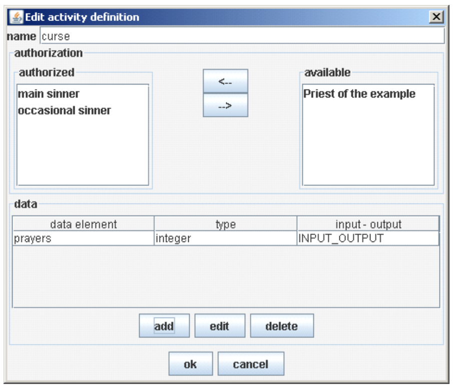


Figure 3.3: New activity in the model.



name: curse

authorized: main sinner, occasional sinner

available: Priest of the example

data

data element	type	input - output
prayers	integer	INPUT_OUTPUT

Figure 3.4: Editing one activity.

Creating and renaming an activity
1. Click the *create activity* button
2. Right-click the activity
3. Choose `Edit`
4. Retype the name
5. Click `Ok`

Table 3.5: Creating (1) and renaming (2-5) an activity

3.1.2 Defining constraints

Constraints can only be selected from the constraint set that was selected during creation of the model, e.g. ConDec or DecSerFlow). The choosen set is depicted in the work tab, e.g. in Figure 3.2, only constraints from the ConDec templates can be chosen.

To actually draw the constraint between two activities, first the *add constraint* button (the arrow with the template name) must be selected and then the constraint can be dragged from one activity to the other. Notice that the mouse pointer must be on the activity center to be able to do this.

Constraints can be renamed in the *constraint properties view.* To open this view, (*right-click constraint* → *edit*). The name in the name field can now be retyped. Press `Ok` to apply the settings. Additional constraint settings are explained later.

Creating constraints
1. Click the *create constraint* button
2. Move the mouse cursor onto the center of the source activity
3. Click and hold the left mouse button
4. Drag the constraint to the destination activity
5. Release the left mouse button
6. Right-click the constraint
7. Choose `Edit`
8. Retype the name
9. Click `Ok`

Table 3.6: Creating and renaming constraints.

SIMPLE CONSTRAINT FIGURE HERE

3.1.3 Removing activities and constraints

Select the activity, or constraint by left-clicking it and then press `delete` on the keyboard or click the *trash bin* button. Note that when an activity is removed, all its connected constraints are also removed.

Deleting activities and constraints
1. Click on the activity, or constraint
2. Click the *trash bin* button, or press `delete`

Table 3.7: Deleting constraints and activities.

3.1.4 Saving the model

To save the model, click *Assignment model* and then *Save*, or alternatively `ALT-s`, or declSave as.

Saving the model (`ALT-s`)
1. Click *Assignment model* in the menu
2. Click *Save*, or *Save as*

Table 3.8: Saving the model.

3.1.5 Validating the model

Models can contain many types of constraints and DECLARE offers a validation mechanism to check for errors before execution. During validation the model is checked for dead activities, i.e. activities that can never be executed and conflicting constraints. To validate the model, click *Assignment model* and then *Validate*, or alternatively `ALT-v`. Validation can be either successful, or unsuccessful and then errors are shown.

Validation (`ALT-v`)
1. Click *Assignment model* in the menu
2. Click *validate*,

Table 3.9: Validation of the model.

Successful validation

When there are no errors in the model, DECLARE will report this by prompting that "no errors were detected". A valid (and saved) model can be instantiated (Chapter 4.1) and executed (Chapter 5.1). Further extension of the model with data and organizational aspects is also possible and is described in Section 3.2. This section also describes how to extend template sets.

Unsuccessful validation

DECLARE finds and reports two kinds of modeling errors, namely dead activities and conflicts. A dead activity is an activity that can never be executed in the model. A set of constraints is conflicting if there exists no execution that would fulfill all constraints.

Dead activities Figure 3.5 shows a model with one dead activity.

This model contains four constraints: (1) the "0..1" constraint above activity "bless" specifies that this activity can be executed at mot once; (2) the "not co-existence" constraint specifies that one who curses can never be declared holy and that holy people never curse; (3) the "response" constraint between activities "curse" and "pray" specifies that after cursing one eventually has to "pray"; and (4) the "response" constraint between activities "pray" and "holy"

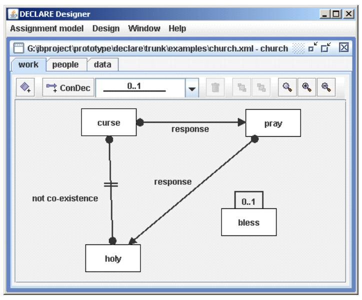
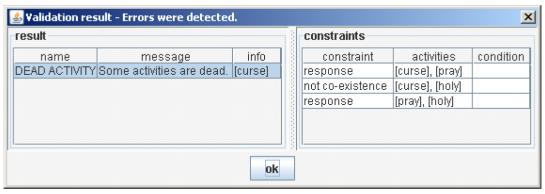

DECLARE Designer
Assignment model Design Window Help
G:\jbproject\prototype\declare\trunk\examples\church.xml - church
work
people
data
ConDec
0..1
curse
response
pray
response
not co-existence
0..1
bless
holy

Figure 3.5: Activity “curse” is dead.



name	message	info
DEAD ACTIVITY	Some activities are dead.	[curse]

constraint	activities	condition
response	[curse], [pray]	
not co-existence	[curse], [holy]	
response	[pray], [holy]	

Figure 3.6: Validation result for dead activity.

specifies that one is eventually declared holy after praying. During the execution, if the activity “curse” would execute the two “response” constraints would require execution of activities “pray” and “holy”, respectively. Execution of the activity “holy” would then violate the “not co-existence” constraint. Because activity “curse” would lead to an error, this activity will be permanently disabled in the system, i.e., this is a dead activity. The validation of this model reports this error as shown in Figure 3.6. On the left side of the screen the list of existing errors is shown. When one of the errors is selected, the sub-set of constraint that causes is shown in the table on the right side. In Figure 3.6, we can see that activity “curse” is a dead activity and that this is a result of a combination of three constraints in the model: (1) “response” constraint between activities “curse” and “pray”, (2) “not co-existence” constraint between activities “curse” and “holy”, and (3) “response” constraint between activities “pray” and “holy”. Note that the system detected that the fourth constraint (“0..1” constraint above activity “bless”) does not influence this error.

Conflicts A set of constraints is conflicting if there exists no execution that would fulfill all constraints. This causes the whole model to be dead, i.e., impossible to execute. To illustrate a conflicting situation we extend the example we used for dead activities. The conflicting model has one additional constraint specifying that activity “curse” has to execute at least once (the “1..*” constraint above activity “curse”). This constraint would enforce execution of the activity “curse”, which would then result (because of the two “response” constraints) in execution of activities “pray” and “holy”. Execution of activity “holy” would create an error, because it would violate constraint “not co-existence”. Shortly, it is not possible to execute this model in a way that would enable all constraints to be fulfilled. Validation detects this error and the result is shown in Figure 3.8. On the right side of the scree one conflicting error is shown. The set of constraints that causes it is shown on the right side of the screen. Note that the system detected that the fourth constraint (“0..1” constraint above activity “bless”) does not influence this error.

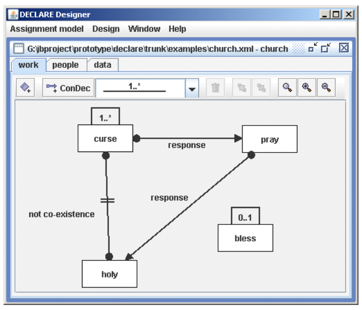
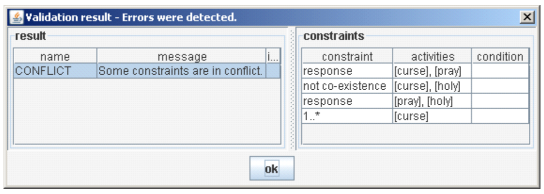

DECLARE Designer
Assignment model Design Window Help
G:\jbproject\prototype\declare\trunk\examples\church.xml - church
work
people
data
ConDec
1..*
1..*
curse
response
pray
response
not co-existence
0..1
bless
holy

Figure 3.7: Model with a conflict.


Validation result - Errors were detected.

result

name	message	i...
CONFLICT	Some constraints are in conflict.	

constraints

constraint	activities	condition
response	[curse], [pray]	
not co-existence	[curse], [holy]	
response	[pray], [holy]	
1..*	[curse]	

Figure 3.8: Validation result for conflict.

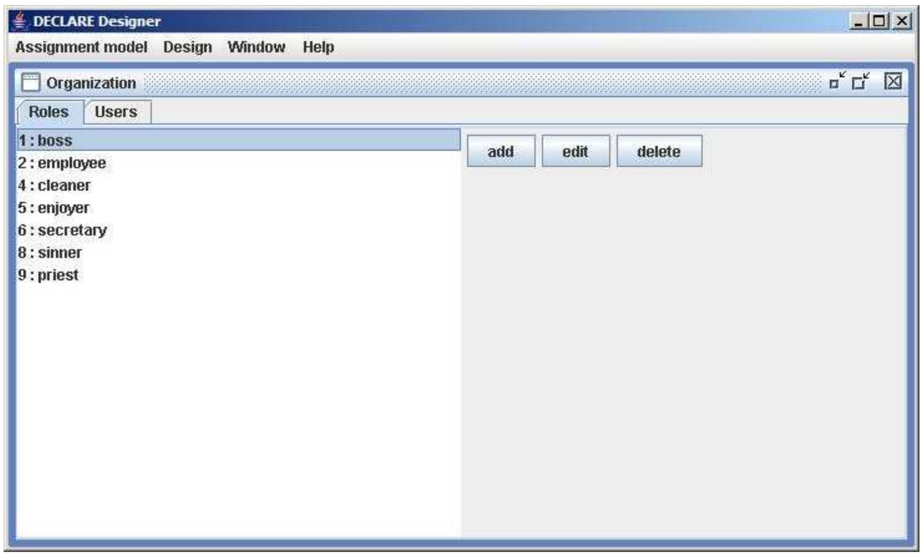
3.2 Creating an advanced model

3.2.1 Setting up the system level

At system level you can manage the users of the system and choose the convenient set of constraint templates that will be available for the design.

Defining the organization

The organization view of the designer is opened by choosing *Design* and then *Organization* in the menu of the Designer (cf. Table 3.2). This will give the "Organization" window with a tab for roles and a tab for users, as can be seen in Figures 3.9 and 3.10.



1 : boss
2 : employee
4 : cleaner
5 : enjoyer
6 : secretary
8 : sinner
9 : priest

Figure 3.9: Organization view in the Designer, roles tab

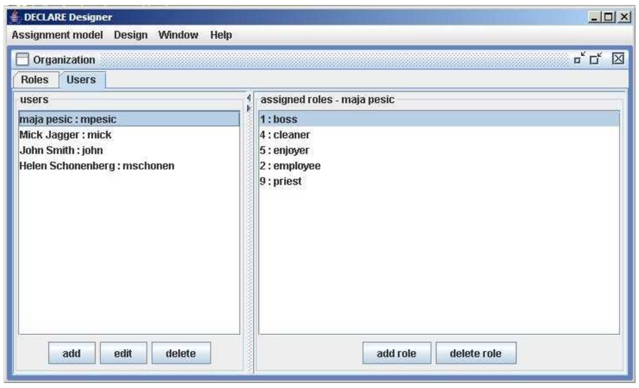
Roles The *Roles tab* allow the modeler to define roles for the execution of the model. ETC

Users In this tab of the the "Organization" window the users of the system can be managed. Four users are shown in the list in Figure 3.10. Each user has name (first and last), user-name and password (cf. Figure 3.11). Users can be added, deleted and edited via buttons "add", "edit" and "delete", respectively. At the left side of the list of users, a list of the assigned system roles can be seen for the selected user. For example, In Figure 3.10, we can see that the user "maja pesic" has several roles: "boss", "cleaner", "enjoyer", "employee",

Adding a role
1. Click `Add`
2. Enter a role name
3. Press `ok` (or `cancel` to cancel)
Editing a role
1. Click `Edit`
2. Re-enter a role name
3. Press `ok` (or `cancel` to cancel)
Removing a role
1. Select the role to be deleted
2. Click `Delete`
3. Press `ok` (or `cancel` to cancel)

Table 3.10: Launching the Designer

and "priest". Buttons "add role" and "delete role" are used to assign/disassign system roles to/from the selected user.



maja pesic : mpesic
Mick Jagger : mick
John Smith : john
Helen Schonenberg : mschonen

1 : boss
4 : cleaner
5 : enjoyer
2 : employee
9 : priest

Figure 3.10: Organization view in the Designer, users tab

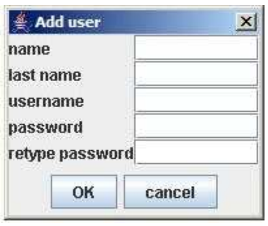


Figure 3.11: Adding a user

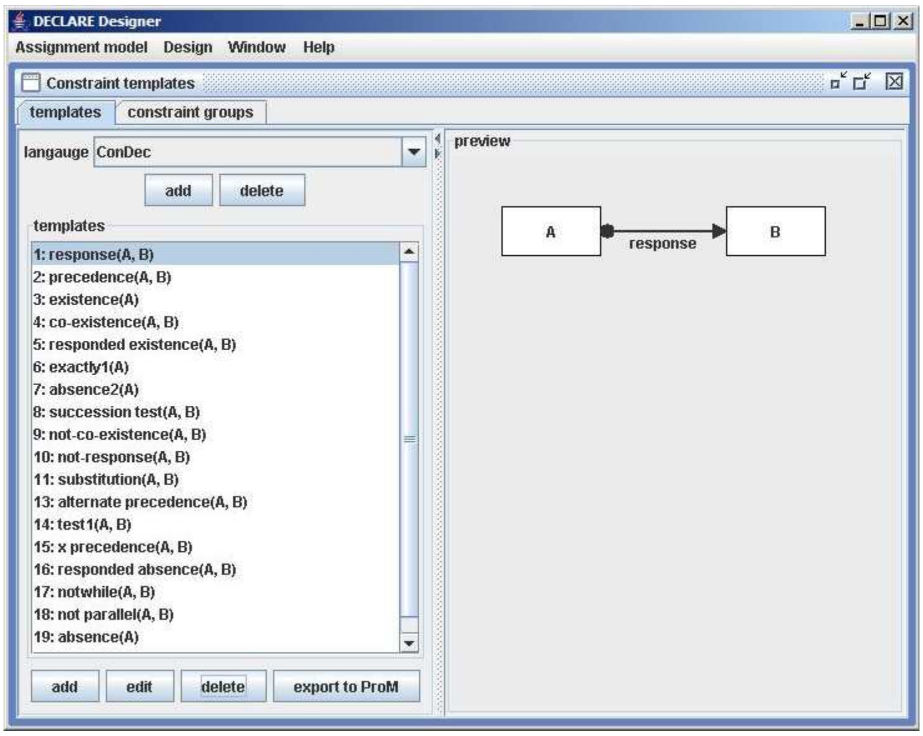
Defining constraint languages

It is possible to define customized process modeling languages in DECLARE. To get to the languages, choose the Design and then Constraint templates in the menu of the Designer. "Constraint templates" window has two tabs. On the firs tab - the "templates" tab - modeling languages are defined. The drop-down list of defined languages is shown in the top-left corner of the tab. Specifying a language is very simple - one just has to enter a language name. Use button "add" to add a new language and button "delete" to remove the selected language. Each language consists of a set of constraint templates – constructs that are used for developing process models in that language. When a language is selected in the drop-dow list of languages, its templates are shown in the list below it.

Then manually add constraints. To copy constraints from other languages it is better to copy from the file.

Defining constraint templates for languages

Constraint templates define possible relations between activities in the model. When a template is selected in the templates list, its graphical representation is shown on the panel on the right side. This graphical representation show how the template is presented to users in process models. For example, Figure 3.12 shows that template "response" between two arbitrary activities "A" and "B" is show as a single line with a circle symbol next to the activity "A" and the arrow symbol next to the activity "B".



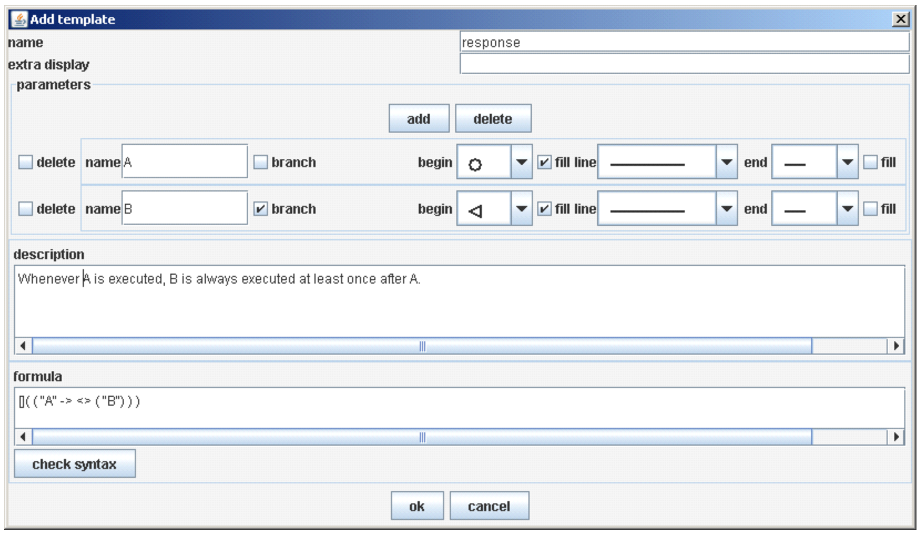
langauge: ConDec

templates
1: response(A, B)
2: precedence(A, B)
3: existence(A)
4: co-existence(A, B)
5: responded existence(A, B)
6: exactly1(A)
7: absence2(A)
8: succession test(A, B)
9: not-co-existence(A, B)
10: not-response(A, B)
11: substitution(A, B)
13: alternate precedence(A, B)
14: test1(A, B)
15: x precedence(A, B)
16: responded absence(A, B)
17: notwhile(A, B)
18: not parallel(A, B)
19: absence(A)

Figure 3.12: Defining constraint templates.

Figure 3.13 shows how a constraint template is defined. In this case, the name of the template is "response" and it is a binary template (i.e., the template defines relation between two activities). At the right of the template name the middle symbol and the style of the line can be selected in drop-down lists.

Since the template is binary, two formal parameters are defined. For each of them a name is given (in this case "A" and "B"), the symbol style and the branching option. So far, the graphical specifications of the "response" template are given. Next, a textual description of the template is given. Last, the template semantics is written in the formalization language. In this case, LTL formula for the template is given and by pressing button "check syntax", syntax of the formula is checked. In the formula, specified parameters "A" and "B". However, it is possible to specify parameters in more detail. Moreover, parameters should not be seen as activities, but rather as events. In DECLARE, there are three types of events: (1) activity.started, (2) activity.completd and (3) activity.canceled. If the parameter is used in its short version in the formula (e.g., "A"), DECLARE will assume that completed event is meant and replace this short specification with "A.completed". For illustration, our "response" template the specified formula is $\Box(``A'' \Rightarrow \Diamond(``B''))$. This is seen by the system as if it was specified $\Box(``A.completed'' \Rightarrow \Diamond(``B.completed''))$. Note that, for example, formulas $\Box(``A.started'' \Rightarrow \Diamond(``B.completed''))$ or $\Box(``A.canceled'' \Rightarrow \Diamond(``B.completed''))$ have different semantics.



name: response
extra display:

parameters

☐ delete	name A	☐ branch	begin ○	☑ fill line	———	end —	☐ fill
☐ delete	name B	☑ branch	begin ◁	☑ fill line	———	end —	☐ fill

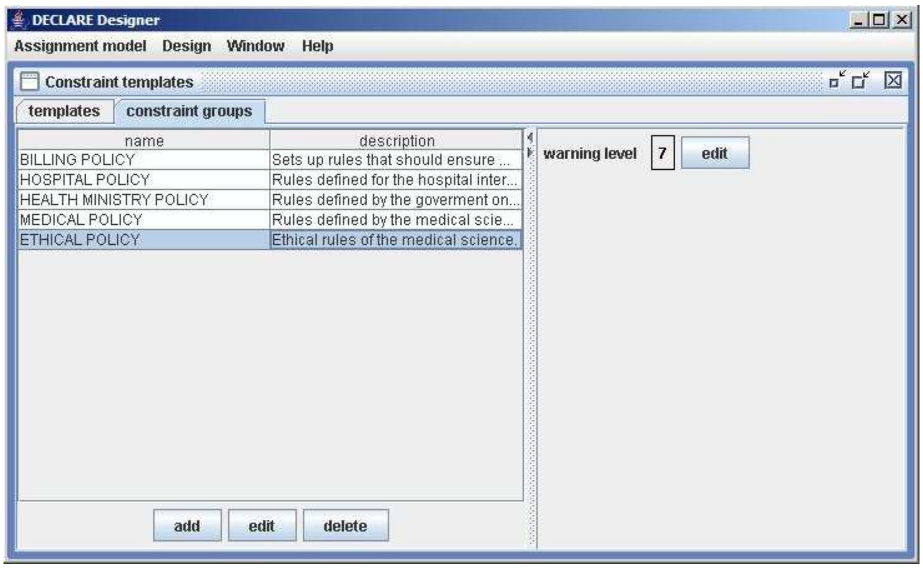
description
Whenever A is executed, B is always executed at least once after A.

formula
```
[]( ( "A" -> <> ( "B" ) ) )
```

Figure 3.13: Defining a constraint template.

Defining constraint groups

When designing constraints in process models, users can assign some constraint to predefined groups. These groups can be seen as policies and are used in cases constraint violation for warning users. Figure 3.14 shows an example of a DECLARE system with five policies. On the right side of the policies list, the warning level can be altered. The warning level is used to decide weather to earn the user when violating constraints.



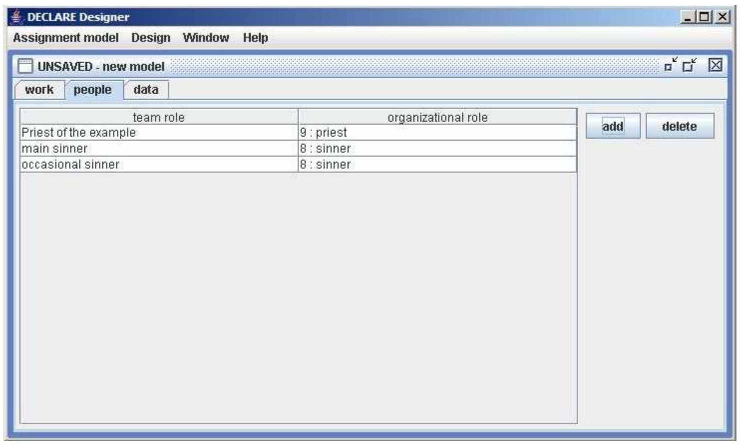
name	description
BILLING POLICY	Sets up rules that should ensure ...
HOSPITAL POLICY	Rules defined for the hospital inter...
HEALTH MINISTRY POLICY	Rules defined by the goverment on...
MEDICAL POLICY	Rules defined by the medical scie...
ETHICAL POLICY	Ethical rules of the medical science.

Figure 3.14: Creating a constraint group.

3.2.2 Defining data and people

In Section 3.1 we explained how to make a basic model while defining work. Now we will explain how to define data and execution roles for the model and in the next section we will explain how to link data and people to the activities.

Defining roles (people) in a model Figure 3.15 shows the window for setting people for the process. Initially this window is empty. Press `add` to add people to the process. Choose a team role for this process and choose the organizational role. Organizational roles are defined at system level and only roles from the defined set can be chosen.

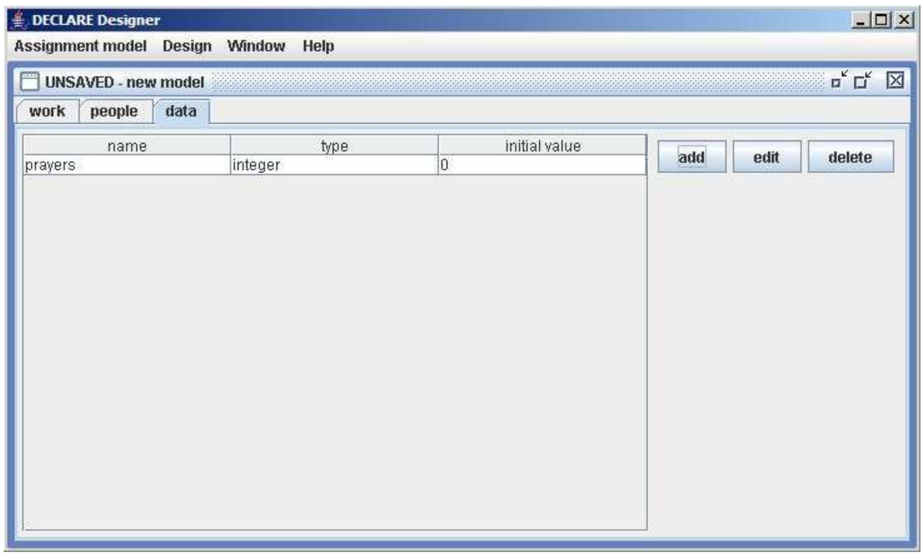


team role	organizational role
Priest of the example	9 : priest
main sinner	8 : sinner
occasional sinner	8 : sinner

Figure 3.15: Defining roles in the model.

Defining data elements in a model [1] Figure 3.16 shows the window for setting people for the process. Initially this window is empty. Press `add` to add data to the process and fill in the name, the type and an initial value. Existing data can be edited and deleted.

[1]DATA IS USED DURING COOPERATION WITH YAWL.



name	type	initial value
prayers	integer	0

Figure 3.16: Creating data for the model.

3.2.3 Linking work, data and roles

Activity properties

Figure 3.18 shows which properties can be set for activities. First the name of the activity can be changed into “curse”. Second, the authorizations for this activity are specified using the define team roles. If no authorization is set (the “authorized” list is empty) then any team member can execute this activity. If authorization is set (the “authorized” list is not empty), then any of the team members with any of the authorized team roles can execute the activity. Second, data elements available in the activity can be defined. In Figure 3.4, we can see that users with team roles “main sinner” and “occasional sinner” can execute activity “curse”. To define available data elements for the activity, one selects from the list of model-defined data elements (definition of model data is explained in Section **??**). In the case of the activity “curse”, we use model data element “prayers” as an “input-output” element. Further more, data elements that are available in the activity can be “input” (user can only read the value during execution), “output” (user has to provide a value during execution) or “input-output” (combination of the previous two) elements.

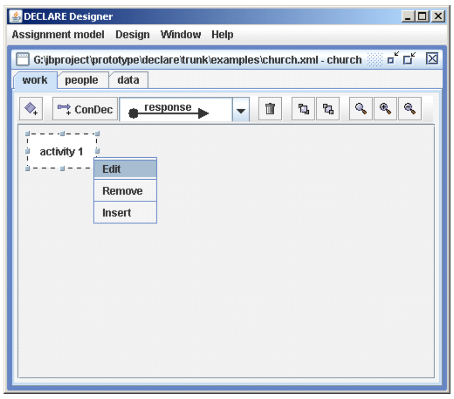
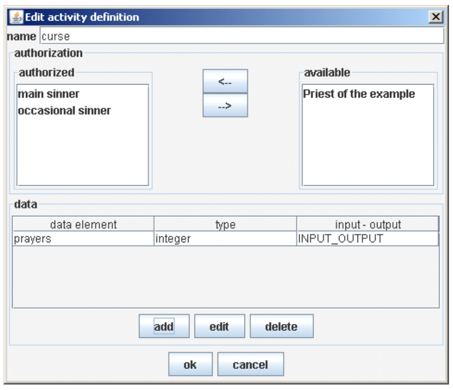


Figure 3.17: New activity in the model.



name curse

authorization

authorized	available
main sinner occasional sinner	Priest of the example

data

data element	type	input - output
prayers	integer	INPUT_OUTPUT

Figure 3.18: Editing one activity.

Constraint properties

After adding a constraint, its properties can be edited by clicking the item “edit” on the pop-up menu when the constraint is selected. The form that is used to edit constraints properties is shown in Figure 3.20.

Figure 3.19 shows that a constraint based on the template “response” is created between activities “curse” and “pray”. Note that, while the template uses formal parameters (e.g., “A” and “B”), constraint replaces these formal parameters with real activities in the model (e.g., “curse” and “pray”). This constraint (“response”) specifies that after every execution of the activity “curse”, activity “pray” will eventually be executed at least once.

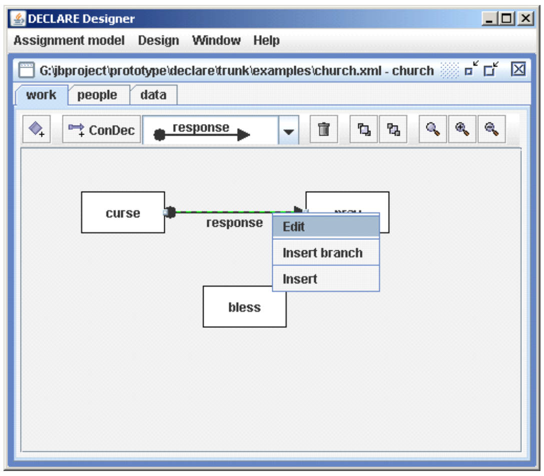
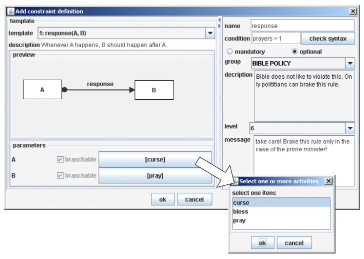

DECLARE Designer
Assignment model Design Window Help
G:\jbproject\prototype\declare\trunk\examples\church.xml - church
work people data
ConDec response
curse
response
Edit
Insert branch
Insert
bless

Figure 3.19: New constraint in the model.



template: 1: response(A, B)
description: Whenever A happens, B should happen after A.
A ✓ branchable [curse]
B ✓ branchable [pray]
name: response
condition: prayers > 1 check syntax
mandatory [✓ optional]
group: BIBLE POLICY
decription: Bible does not like to violate this. Only polititians can brake this rule.
level: 6
message: take care! Brake this rule only in the case of the prime minister!
select one item: curse bless pray

Figure 3.20: Editing one constraint.

Name First, the name of the constraint can be changed. By default, constraint gets the name of its template. For the readability of the model, it is possible to change this name and give a context-related new name. For example, we can change the name of the constraint "response" into "eventually pray after every curse" to make the model more understandable for users.

Condition Second, a condition can be specified using model data elements. In Figure 3.20 constraint "response" is valid only if the number of prayers is higher that one. Additionally, the correctness of the condition syntax can be checked using the button "check syntax". Finally, we can choose if the constraint is "mandatory" or "optional".

Constraint type MANDATORY, OPTIONAL
OPTIONAL: POLICY GROUP, DESCRIPTION, PRIORITY LEVEL, MESSAGE

Finally, we can specify wether the constraint is mandatory or optional. During the model execution, the system will impose mandatory constraints to users, i.e., users will not be able to violate them. On the contrary, optional constraints are not imposed during the execution, but rather used as warnings when about to be violated. However, to improve user support, the system generates informative warning when the user is about to violate the constraint. In this way, the user can make an informed decision weather to proceed and violate the constraint or to quit and leave the constraint fulfilled. The information that will be presented to the user when about to violate the "response" constraint is specified in the bottom part of the form in Figure 3.20.

Policy OPTIONAL: POLICY GROUP, DESCRIPTION, PRIORITY LEVEL, MESSAGE

First, the constraint group for the constraint is selected (e.g., "BIBLE POLICY") and its description is automatically displayed. Constraint groups are defined on the system level (cf. Section 3.2.1). Second, optional constraint should also have a (priority) level. This level is an integer on the scale from 0 to 10, and the higher it is the more dangerous it is to violate this constraint. Third, to complete optional information, we can specify a context-related message that will explain the constraint in more details to the user during the execution.

OLD
Each activity has properties that can be set by (*right-click activity* → *edit*). This will open the *activity editor window*, shown in Figure 3.3).

Chapter 4

Creating and adapting cases

4.1 Creating and adapting cases

Application Framework is used to create cases and change already running cases. Double-click on `framework.bat` to start this application. Figure 4.1 shows the Framework screen. First, there is the `available assignments` table where all pending requests from YAWL are shown. Second, there are two buttons: (1) click the `load available` button to load a pending YAWL request or (2) click the `load empty` button to load a independent assignment. At the bottom of the screen there is the `active assignments` table – this table contains all assignments that are currently being executed by users. Finally, the two left bottom buttons are used to close the selected active assignment and change the model of a selected active assignment.

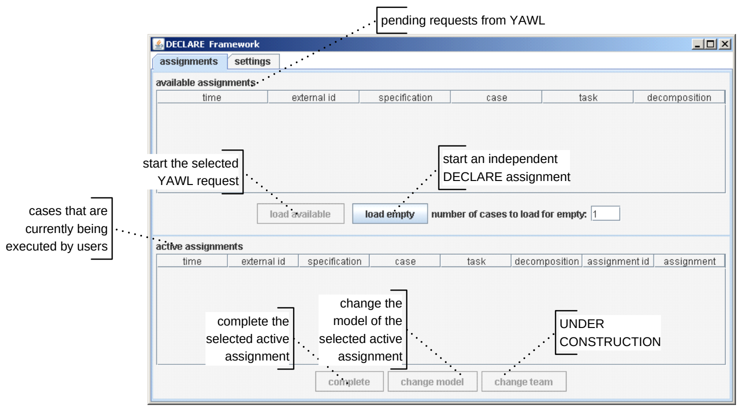


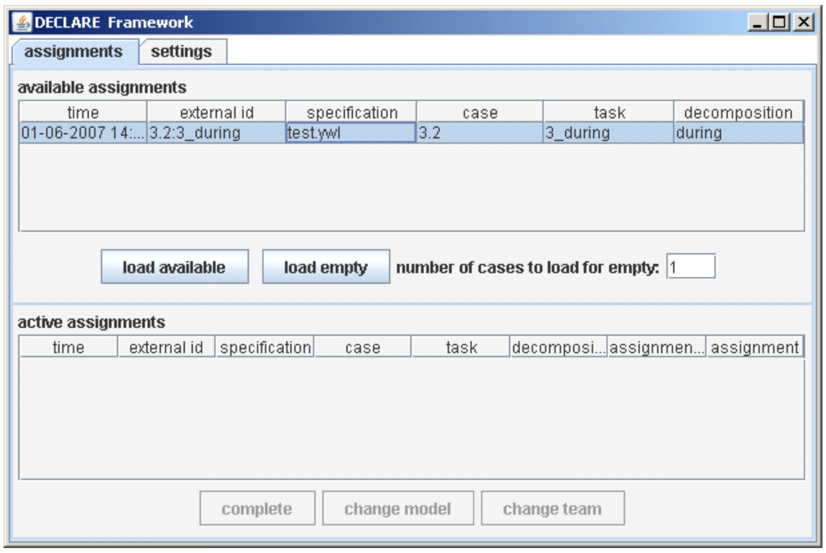
Figure 4.1: The framework screen.

4.1.1 Creating cases

There are two ways to create a case in the Framework. First, a case can be created for each task that arrives from the YAWL system. Second, an independent DECLARE assignment can be created.

Creating cases for YAWL tasks

If a task in a YAWL model should be handled as a DECLARE assignment, declare service for YAWL will send this request to the Framework. The pending YAWL requests are shown in the `available assignments` table on the Framework screen. Figure 4.2 shows one pending request for the YAWL task `during`. First column shows the time of the request arrival. The second column shows the identification number of the YAWL task. The third column shows the name of the task YAWL process. The fourth column shows the task name and the fifth column the YAWL decomposition identification.



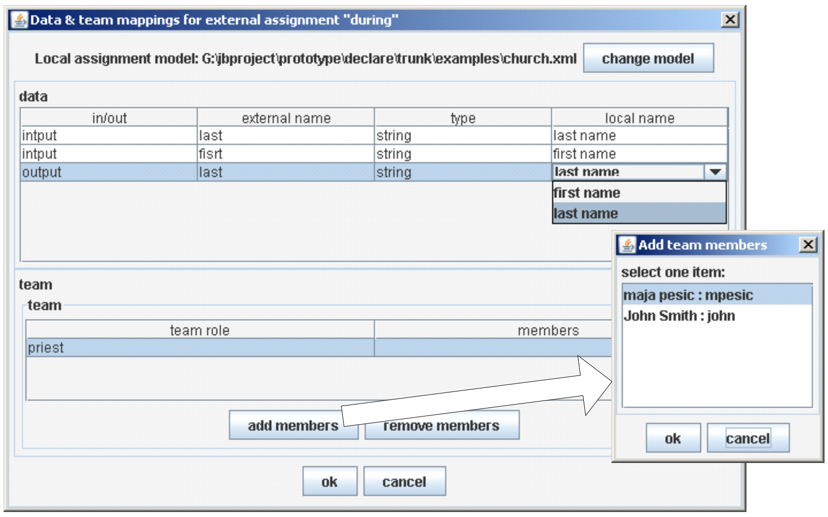
time	external id	specification	case	task	decomposition
01-06-2007 14:...	3.2:3_during	test.ywl	3.2	3_during	during

time	external id	specification	case	task	decomposi...	assignmen...	assignment

Figure 4.2: One pending request form YAWL.

To launch a case for a pending request you must select the request in the `available table` and click the button `load available` – you will need to select a DECLARE assignment model file you wish to load for this YAWL request. Next, the screen presented in Figure 4.3 appears. On top of this screen the selected DECLARE model file is presented. This file can be changed by clicking on the button `change model`.

Setting data mapping All data elements from the YAWL task request are listed in the `data` table. For each of the listed YAWL data elements one data element from the selected DECLARE model must be selected. The values of input YAWL data elements are copied into the referring DECLARE model data elements when the case is started. The values of the DECLARE model data elements will be copied into the referring output YAWL data elements when the case is closed.



Data & team mappings for external assignment "during"

Local assignment model: G:\jbproject\prototype\declare\trunk\examples\church.xml

data

in/out	external name	type	local name
intput	last	string	last name
intput	fisrt	string	first name
output	last	string	last name

first name
last name

team

team

team role	members
priest	

Add team members

select one item:

maja pesic : mpesic
John Smith : john

Figure 4.3: Data mapping for a YAWL request.

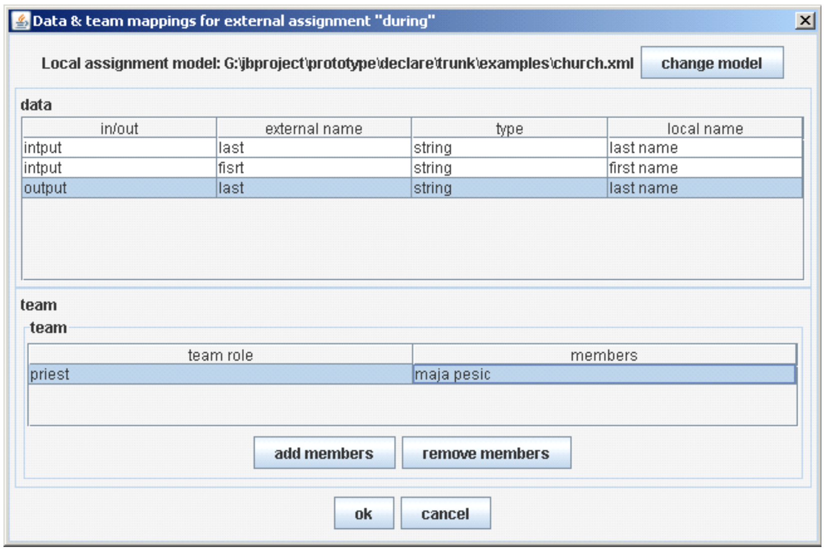
Assigning people to the case The model roles and the assigned people are presented in the `team` table. Use buttons add members and remove members to assign/disassign people to/from model roles. Figure 4.4 shows a completed data mapping and the assigned people to the tem role.

Creating independent cases

Click the button `load empty` to select a DECLARE assignment model file you wish to load as an independent case.

Setting initial values for the case data UNDER CONSTRUCTION !!!!

Assigning people to the case If there are no roles in the selected assignment this step will be skipped. By default, administrator is assigned to all roles in all assignments. If there are roles in the selected assignment, the screen in Figure 4.4 will appear: the `data` table will be empty and the people can be assigned to the case using the `add members` button.


Data & team mappings for external assignment "during"

Local assignment model: G:\jbproject\prototype\declare\trunk\examples\church.xml

change model

data

in/out	external name	type	local name
intput	last	string	last name
intput	fisrt	string	first name
output	last	string	last name

team

team

team role	members
priest	maja pesic

add members | remove members

ok | cancel

Figure 4.4: Completed data mapping and team assignment.

4.1.2 Changing cases at run-time

Changing the case model

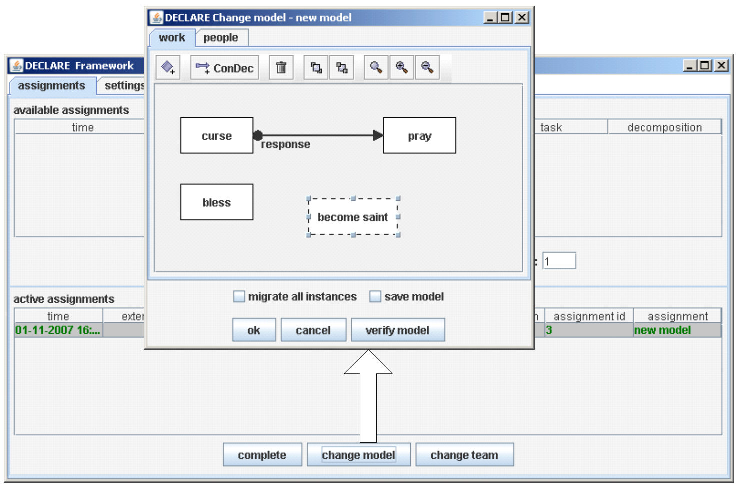


Figure 4.5: Changing the case model.

Changing case roles

Changing case work model

Verification

1. Dead activities
2. Conflicting constraints
3. History validation

Reassigning people to the case

UNDER CONSTRUCTION !!!!

Chapter 5

Executing cases

5.1 Executing cases

5.1.1 Selecting a case

5.1.2 Working on a case

Constraints

1. state (green, orange, red)
2. type (mandatory, optional)
3. conditional

Working with activities

1. Starting activity
2. Working on the activity
3. Completing the activity
4. Cancelling the activity
5. Receiving waringins about possible violations

Using the recommendation feature

5.1.3 Closing a case

Chapter 6

Adapting cases?

Chapter 7

Recommendation

7.1 Recommendation

Chapter 8

Glossary

Assignment Case

Case

Conflict Verification reveals a conflict when it is not possible to satisfy all constraints.

Constraint A constraint is a rule that has to be satisfied (at the end of execution).

Constraint Group

Constraint Template See template.

Data element

Dead activity A dead activity is an activity that can not be executed.

Declare Declare is a tool set for modeling in declarative language.

Designer The Designer is part of Declare that is used for defining

- the organization, for defining templates for languages
- languages,
- templates for languages,
- constraint groups.

Formal parameter

History violation A history violation is a possible validation result when changing the model. Validation gives a history violation if the execution trace permanently violates one or more new constraints, we

Mandatory Constraint A mandatory constraint is a constraint that must be followed during execution.

Optional Constraint An optional constraint is a constraint from which can be deviated during execution.

People

Policy list

Role Team role, organizational role

Template A (constraint) template is a graphical representation of a constraint (LTL formula).

Worklist The list of all started, but not yet completed activities of one case, for one user.

Syntax

Appendix A

LTL syntax

Appendix B

Constraint condition syntax