Deliverable number:	D3.1
Deliverable title:	Multi use level workbench

Delivery date:	2009/11/30
Submission date:	2010/02/28
Leading beneficiary:	University of Amsterdam (UVA)
Status:	Version 07 (final)
Dissemination level:	PU (public)
Authors:	Jochem Liem, Wouter Beek and Bert Bredeweg

Project number:	231526
Project acronym:	DynaLearn
Project title:	DynaLearn - Engaging and informed tools for learning conceptual system knowledge
Starting date:	February 1st, 2009
Duration:	36 Months
Call identifier:	FP7-ICT-2007-3
Funding scheme:	Collaborative project (STREP)

Abstract

The DynaLearn software allows learners to capture their ideas and investigate their logical consequences. By building causal models and simulating these, students develop an understanding of how systems behave. DynaLearn introduces 6 use levels at which the software can be used. Furthermore, by working at a particular use level, teachers can emphasise particular aspects of modelling a system (e.g. causality, conditional knowledge). The DynaLearn software is based on the Garp3 qualitative modelling and simulation workbench, but integrates the interface into a single screen, adds use levels, and incorporates significant improvements such as allowing multiple simulations, storing selections in state graphs, and saving simulations to a model.

Acknowledgements

The authors would like to thank Anders Bouwer, Floris Linnebank, and Jan Wielemaker for their insightful suggestions and support in the implementation, Paulo Salles for creative discussions on use levels and qualitative system dynamics and for providing feedback on an earlier draft, Dirk Bertels for support with the design of the interface, and Jorge Gracia del Río for giving feedback on an earlier draft.

Document History

Version	Modification(s)	Date	Author(s)
01	Basic outline + Initial content	2010-02-21	B. Bredeweg
02	First draft of 10, 12 and 13 and more	2010-02-23	J. Liem & W. Beek
03	Chapters 3, 4, 5, 6, 7, 8, 9	2010-02-24	W. Beek & J. Liem
04	Chapters 1, 2, 15 and 16.	2010-02-25	J. Liem & W. Beek
05	Global editing + fine tuning	2010-02-26	B. Bredeweg
06	Processing internal review	2910-02-27	W. Beek & J. Liem
07	Further editing + Finalising	2010-02028	B. Bredeweg

Contents

1. Introduction

Qualitative Reasoning (QR) provides means to express conceptual knowledge such as system structure, causality, conditions for processes to start and finish, assumptions and conditions under which facts are true, qualitative distinct behaviours, etc. Qualitative models provide formal means to externalise thought on such conceptual notions. However, building qualitative models is difficult and hampered by the lack of easy to use tools.

This document describes the DynaLearn Interactive Learning Environment (ILE). The functional design of this system was presented in [3]. DynaLearn is specifically created for students to develop their conceptual knowledge on systems. The DynaLearn ILE offers diagrammatic presentations for learners to express their ideas, and test these by running simulations.

The DynaLearn ILE is based on the qualitative modelling and simulation workbench Garp3 [1,2]. However, in DynaLearn the aim is to accommodate different groups of students. As such, the tool can be used on different use levels, depending on the student's level of expertise, and the kind of phenomenon dealt with. Furthermore, in DynaLearn the entire interface has been integrated into a single workspace (single screen).

The content of the report is as follows. Chapter 2 describes the ideas underpinning each of the use levels. Chapter 3 describes the main features of the integrated interface. Chapters 4 through 9 describe each of the use levels of the ILE. Chapter 10 describes the new multiple simulation (and saved simulation) functionality. Chapter 11 describes the editors for meta-data. Chapter 12 describes a set of special features available in the DynaLearn ILE. Chapter 13 is devoted to implementation details that explain how certain functionalities were established. Finally, Chapter 14 and 15 discuss the results and conclude.

The following details have been added in the appendix for completeness. Appendix 1 shows the relevant screens for each use level as they have been developed as an intermediate design step from D2.1 [3] towards the current implementation. Appendix 2 enumerates in detail how the functionality available in Garp3 was migrated to the DynaLearn ILE context, particularly emphasising how to remove, hide or merge Garp3 complexity at the lower use levels in the DynaLearn ILE. Finally, Appendix 3 holds the user manual of the GIT version management software used for the development of the DynaLearn ILE. GIT is also used for the other software development in DynaLearn.

2. Learning spaces – Use levels

The learning spaces in the DynaLearn software are organised as a set of use levels with increasing complexity in terms of the modelling ingredients a learner can use to construct knowledge. Six use levels have been designed and implemented (Figure 2.1 and Table 2.1):

- Concept map
- Basic causal model
- Basic causal model with state-graph
- Causal differentiation
- Conditional knowledge
- Generic and reusable knowledge

Each level is a self-contained interactive workspace useful for learning specific details about system behaviour. Self-contained implies that the representational primitives available within a use level form a logical subset of all the primitives available. Hence, they allow for automated reasoning on behalf of the underlying software. It also implies that from a Qualitative System Dynamics perspective, learners are able to create sensible representations of the phenomena they perceive when observing the behaviour of a real-world system. Moreover, what learners express using the software, will have consequences for what the software can infer. Hence, learners can be confronted with the logical consequences of their expressions, which either may or may not match the observed system behaviour or their expectations thereof. Particularly in the case of mismatch there is ample room for interactive learning. Progression between use levels happens by augmenting the current level with the smallest subset of possible modelling ingredients, again ensuring that the next level is self-contained. The formal context and starting point for developing the use levels is Garp3 [1,2].

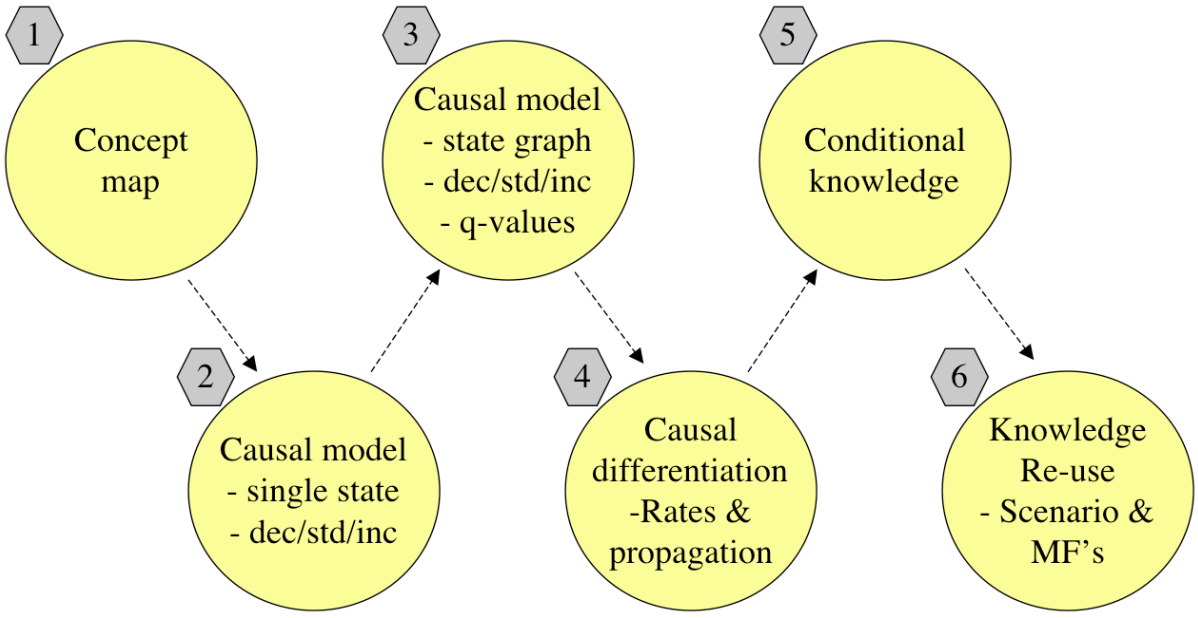


Figure 2.1: Schematic representation of the six use levels in DynaLearn (Notice, dec: Decreasing, std: Steady, inc: Increasing, q-values: Qualitative values. MF: Model fragment).

Table 2.1: Overview of modelling ingredients in the DynaLearn software. The levels are cumulative; details from lower levels are also present at higher levels. Learners create a single expression at use levels 1 to 4. At use level 5 one or more conditional expression can be added to that. At use level 6 learners may create multiple expressions, including a scenario.

Nr	Use level	Introduced ingredients
1	Concept map	• Entities • Configurations
2	Causal model	• Attributes • Quantities • Value assignments • Derivatives • Causal relationships (+ and –)
3	Causal model with state graph	• Quantity spaces • Correspondences

4	Causal differentiation	• Causal relationships (+ and – refined)
		  o Influences
		  o Proportionalities
		• Inequalities
		• Operators (+ and –)
		• Agents
		• Assumptions

5	Conditional knowledge	• Conditional expressions

6	Generic and reusable	• Multiple expressions
		  o Model fragments
		  o Scenarios

Below, a summary of the characteristics of each use level is presented (see [3] for further details).

A *concept map* (sometimes also referred to as an entity-relation graph) is a graphical representation that consists of two primitives: nodes and arcs. Nodes reflect important concepts, while arcs show the relationships between those concepts (cf. [7]). From a qualitative reasoning point of view, concept maps are less interesting because they do not capture dynamics. However, having this use level is useful from a learning point of view, as it is the root from which more complex knowledge representations emerge. Therefore a simple version of such a workspace is foreseen in the DynaLearn software.

The use level *basic causal model* focuses on quantities, how they change and how this change influences other quantities to change. Quantities represent behaviour. They are connected to entities, the structural units in the model. Simulation at this use level means to calculate for each quantity one of the following options (for its derivative): decrease, steady, increase, ambiguous (because of opposing influences), or unknown (because of missing information). This use level relates to the representational approach taken in [4].

The use level *basic causal model with state-graph* augments the basic causal model level with the notion of a quantity space, which can be assigned to one or more quantities. Adding this

feature has a significant impact on the simulation results and necessarily introduces concepts such as state-graph, behaviour path, and value history.

The use level *causal differentiation* takes all details defined for the basic causal model with state-graph and refines certain notions, particularly those related to causality. Different from the preceding use level is that the notion of exogenous quantity behaviour is included in the default setting. Also included in the default setting is the idea of an agent. The representational approach taken in [5] covers a subset of the vocabulary available at this use level.

The use level *conditional knowledge* is a refinement of the causal differentiation level. All representation details apply as they do for this preceding level. The main difference is the possibility to specify conditions under which specific set of details are true. In the preceding use levels all the expressions created by a learner are always true. However, some facts (e.g. the process of evaporation) only occur when certain conditions are satisfied. The conditional knowledge use level addresses situations like these.

The use level *generic and reusable knowledge* reflects Garp3 in its current status. The main difference with the other use levels is the focus on 're-usable' knowledge. That is, to capture essential details in a context-independent manor as much as possible, or otherwise to explicate the conditions under which the knowledge is applicable. At this level the notion of types and hierarchy become important. Also the idea of learners creating their own models by re-using (partial) solutions stored in a repository now viable.

3. Interface – Main features

This chapter presents the main features of the interface of the DynaLearn ILE. The main features are all accessible via the application window of the DynaLearn ILE, shown in Figure 3.1. The application window is divided in several regions that correlate to specific functionality. Each of those regions is discussed below.

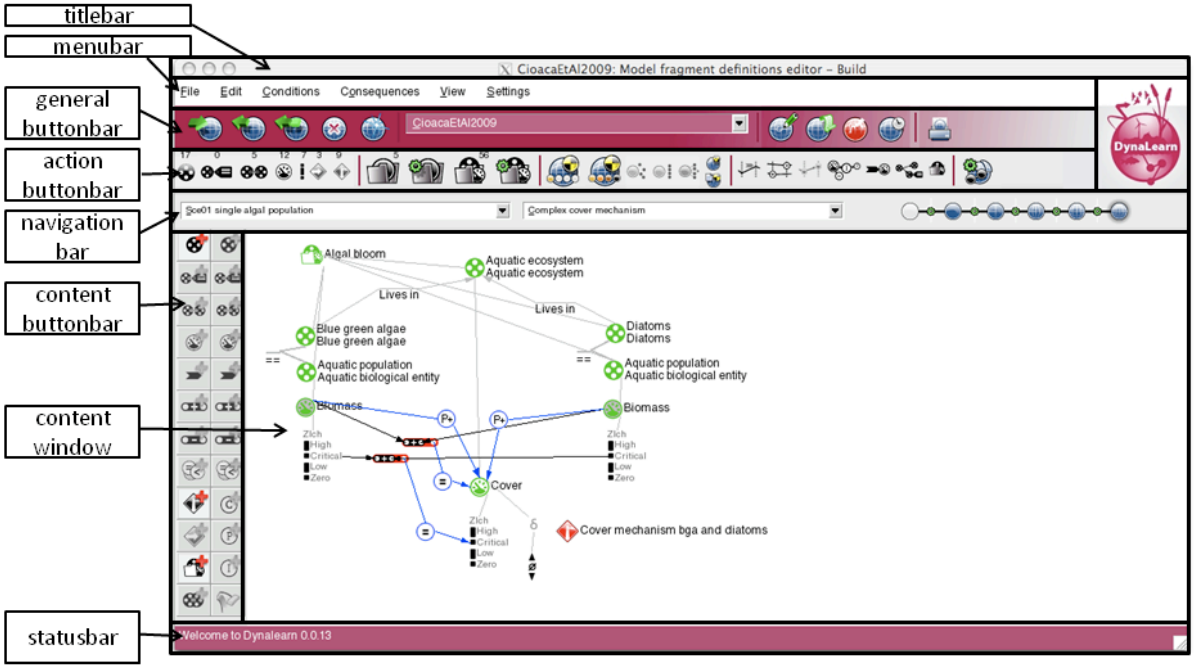


Figure 3.1: The functional regions of the DynaLearn application window.

3.1. Title bar

The title bar at the top of the screen shows the name of the currently active model, as well as a description of the current workspace (e.g. 'Model fragment definitions editor' in Figure 3.1). The text inside the title bar therefore depends on the current workspace inside the application.[1]

3.2. Menu bar

The menu bar is located right below the title bar. It comprises a context-sensitive menu with actions that can be applied in the current workspace.

3.3. General button bar

The general button bar is located below the menu bar. It is divided into several functionally distinct regions (Figure 3.2).

[1] Different views are displayed in the content button bar (section 3.7) and content window (section 3.8).

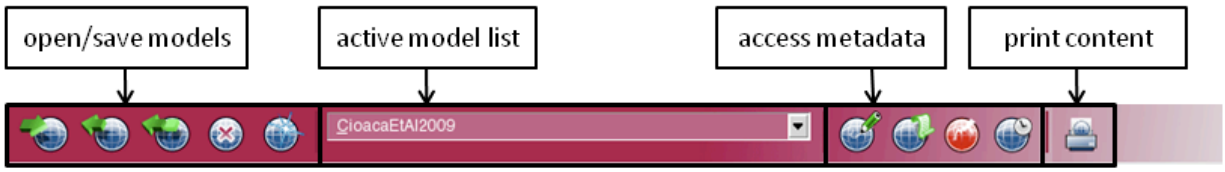


Figure 3.2: The general button bar.

The first five buttons (from left to right) are used for retrieving, storing, closing and creating new models. The functionality of these buttons is described in Table 3.1.

Table 3.1: General button bar, open/save models group.

Action	Icon	Description
Open Model		The purpose of this task is to open a model from a file. It is possible to open normal model files but also to open OWL formatted files.
Save Model		The purpose of this task is to save a model as a `.hgp` file (or in OWL format).
Save Model As		The purpose of this task is to save a model to a new filename.
Close Model		Closes the current model. This means that the current model will be removed from the dropdown list of active models. If there are unsaved changes, the user will be asked to either save or discard these changes.
New Model		The purpose of this task is to start a new model in the DynaLearn workbench. The new model button brings up a dialog in which a model name can be given and a use level can be chosen.

The next element on the general button bar is the dropdown list displaying the models that are currently loaded inside the DynaLearn workbench. This allows the user to switch between models by selecting an item from this list. The list is updated whenever a model is created, opened, or closed.

The next group of buttons (to the right of the dropdown list) in the general button bar consists of four buttons. These buttons are used for accessing various sorts of metadata regarding the currently active model. The functionality of these buttons is described in Table 3.2.[2]

[2] The metadata views and features are described in chapter 11.

Table 3.2: General button bar: Accessing metadata group.

Action	Icon	Description
Abstract and General remarks		The purpose of this task is to add abstract and general remarks to the model metadata, such as information that explains something about the model goals and/or about the intended audience.
General Informal		The purpose of this task is to add general information to the model metadata.
Status and Bug Reports		The purpose of this task is to add status reports about the model in order to keep track of known problems and solving them.
Model Data		The purpose of this task is to add version information about the model to the metadata.

The last element on the general button bar is the right-most button that is used for printing the information that is currently displayed inside the content window (see Section 3.8). The general button bar is the same in all use levels.

3.4. Action button bar

Right below the general button bar, the action button bar is shown (see Figure 3.3). Globally, the buttons can be grouped into two categories: the ones used to build a model, and those used to run simulations with the model.

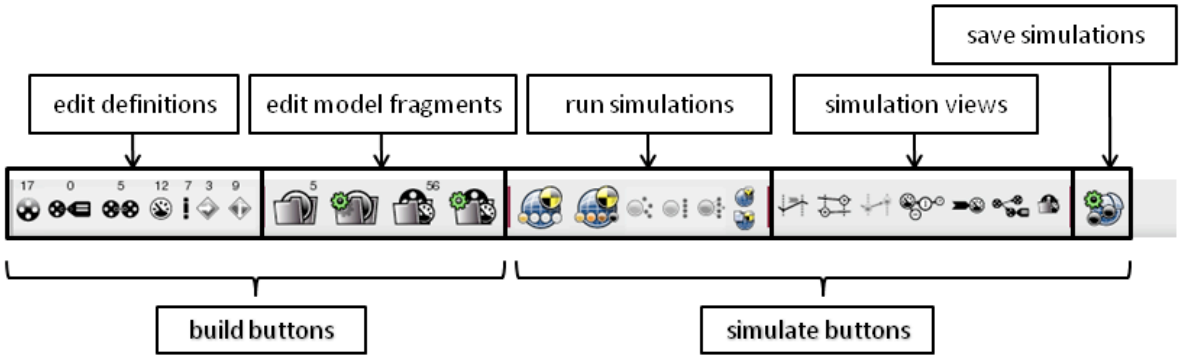


Figure 3.3: Action button bar, showing the buttons for building models to the left ('edit definitions' and 'edit model fragments') and the buttons for running, viewing, and simulations on the right.

There are two groups of build buttons: those used to edit definitions, and those used to edit model fragments.

The first group of build buttons, the edit build buttons, is used to access workspaces for editing definitions. In these workspaces, the user can add, remove, or edit definitions without adding instances of those definitions to a specific model. In other words, the buttons from the first group allow the user to create a palette of general elements that can later be instantiated as concrete

model ingredients that are used to create models with. The functionality of these buttons is described in Table 3.3.

Table 3.3: Action button bar, buttons for building definition.

Action	Icon	Description
Edit Entity Hierarchy		Entities are the physical objects or abstract concepts that play a role within the system and define the system structure. The entity hierarchy editor enables the user to define a hierarchy of entities with child-parent relations between them.
Edit Attribute Definitions		Attributes are static features of entities, often used to add descriptive details to the entity that are not used for calculations and do not change during the simulation. The attribute definitions editor enables the creation of attributes and the assignment of values that further specify these features.
Edit Configuration Definitions		Configurations are used to model relations between entities and agents. They are also referred to as structural relations. The configuration definitions editor enables the creation, deletion and modification of configurations that can be used in model fragments and scenarios.
Edit Quantity Definitions		Quantities represent changeable features of entities and agents. The quantity definitions editor enables the creation, deletion and modification of quantities that can be attached to entities and agents in model fragments and scenarios.
Edit Quantity Space Definitions		A quantity space specifies a range of qualitative values a quantity magnitude or derivative can assume. Each quantity has two associated quantity spaces: a definable one for the magnitude, and the default quantity space {min, zero, plus} for the derivative of the quantity. The quantity spaces definitions editor enables the creation, deletion and modification of quantity spaces that belong to quantity. The qualitative values in a quantity space form a total order. Each qualitative value is either a point or an interval, and within the quantity spaces these two types consecutively alternate.
Edit Agent Hierarchy		Agents are used to model entities outside of the modelled system. The agent hierarchy editor enables the user to define a hierarchy of agents with child-parent relations between them. Agents can have quantities influencing the rest of the system, which are sometimes called exogenous or external influences.
Edit Assumption Hierarchy		Assumptions are labels that are used to indicate that certain conditions are presumed to be true. They are often used to constrain the possible behaviour of a model. The assumption hierarchy editor enables the user to define a hierarchy of assumptions with child-parent relations between them. Because they describe neither structural nor behavioural aspects of a system, they belong to neither the structural building blocks nor the behavioural building blocks categories.

The second group of build buttons is used to access workspaces that allow modelling ingredients to be added to a specific model. A model is not created in one fell swoop, but instead consists of multiple components that together constitute the complete model. There are two kinds of model components: those that a simulation can starts from, called *scenarios*, and those that a simulation cannot start directly from, called *model fragments*.[3] In the second group of build buttons, there are two buttons for each of these types. These buttons are described in Table 3.4.

Table 3.4: Action button bar, editing scenarios and model fragments.

Action	Icon	Description
Edit Scenario Definitions List		Clicking this button displays the list of scenarios. New scenarios can be added to this list, and existing ones can be edited, copied or removed from the list. Scenarios describe a specific situation in which the system can be found. They can consist of all the ingredients that can be used as conditions in model fragments, except for other model fragments. Scenarios are used as input for the qualitative simulator. The qualitative simulator interprets the scenario (by finding applicable model fragments, incorporating their consequences, and deriving unspecified/implicit values for quantities) to generate one or more start states. These start states are then used to generate the rest of the behavioural graph.
Edit Scenario		In this workspace the contents of the last edited scenario can be edited again. Ingredients that can be added, changed or removed include instances of entities, attributes, configurations, quantities and quantity spaces, values, dependencies, agents, and assumptions. These are all conditional model ingredients.
Edit Model Fragment Hierarchy		Clicking this button displays the hierarchy of model fragments. New model fragments can be added to the hierarchy, and existing ones can be edited, copied, or removed from the hierarchy. In addition to this, model fragments can be deactivated (and reactivated again). Model fragments describe part of the structure and behaviour of a system in a general way. They are partial models that are composed of multiple ingredients. Model fragments have the form of a rule. This means that model ingredients are incorporated as either conditions or consequences. Model fragments themselves can be reused within other model fragments as conditions. Furthermore, subclasses of model fragments can be made, which augment the parent model fragment with new ingredients. The consequence ingredients of model fragments that match the actual system situation will be added to that scenario. In that case, the scenario fulfils the conditions specified in the model fragment (which describes a general situation).

[3] Scenarios must answer certain conditions that may trigger changes in the system, so that they allow simulations to start and run.

Edit Model Fragment		In this workspace you can edit the contents of model fragments, including entities, attributes, configurations, quantities and quantity spaces, values, dependencies, agents, assumptions, inequalities, values, calculations, influences, proportionalities, conditional model fragments, correspondences and identities.

Besides the build buttons, there are two groups of simulate buttons one related to running simulations and the other to accessing simulation results. The functionality of the first group of simulate buttons is described in table 3.5.

Table 3.5: Action button bar, running simulations.

Action	Icon	Description
Simulate Scenario		The simulate scenario button starts the simulation with an expression, the last edited scenario or the scenario that is currently selected in the scenario editor. It only performs the first step in the simulation, that is, determining the possible interpretations of the scenario (each becoming states) by matching model fragments and determining the values for each quantity.
Run Full Simulation		The run full simulation button simulates an expression, the last edited scenario, or the scenario that is currently selected in the scenario editor. It generates the full simulation of the scenario or expression in the simulate workspace.
Terminate Selected States		The terminate selected states button determines the possible ways a state can end. This results in a set of terminations.
Order Selected States		The order selected states button determines which terminations have priority and which terminations can occur together. This step removes and combines multiple terminations.
Close Selected States		The close selected states button either creates new states of each termination, or creates transitions to already existing states (if the resulting state already exists).
Open Simulation Preferences		The purpose of this workspace is to set the simulation preferences. These are settings that can influence the way in which the simulation is created.
Open Trance Window		The purpose of this workspace is to set the trace option, a functionality that allows for a description of the reasoning steps done by the simulation engine to be visualised.

The second group of simulate buttons that provides access the simulation results is described in Table 3.6.

Table 3.6: Action button bar, access simulation workspaces.

Action	Button Icon	Description
Show Value History		The value history shows the values quantities go through in a series of selected states. The value histories are shown per quantity.
Show Equation History		The Equation history shows a diagram of inequality/equation values in each of the currently selected states, and for each of the quantities.
Show Dependencies		The purpose of this workspace is to inspect all the relationships involving entities and quantities (dependencies) that hold in a selected state of the simulation state-graph.
Show Quantity Values		The purpose of this workspace is to inspect the values of all the quantities in a selected state of the simulation state-graph.
Show Modelling Ingredients		The modelling ingredients workspace lets the user inspect the structure of the entities, configurations and attributes in the currently selected state(s) of the simulation state-graph.
List Model Fragments		The model fragments list enumerates the model fragments that are active in the currently selected state(s) of the simulation state-graph.
Load/Save Simulations		The load/save simulations workspace allows for the current simulation to be saved, or a saved simulation to be loaded.

Depending on the use level that the user is in, a different collection of action buttons is displayed.[4]

3.5. Navigation bar

The navigation bar is displayed below the action button bar. It contains two dropdown lists. These dropdown lists display various modelling ingredients. What is displayed depends on the current workspace in the application (workspaces are discussed in the next section).

Besides the two dropdown lists, to the right, the use level switcher is displayed. The six use levels are there represented as six blobs that occur in a linear ordering. The use level with the shadow behind it indicates the current use level.[5]

[4] Specific buttons available in each use level are discussed in Chapters 4 - 9.

[5] Eventually, this use level switch will be used to traverse from one use level to another. See also section 12.6.

3.6. Workspaces

Before describing the features of the content button bar and of the content window, it makes sense to explain the notion of workspaces in the DynaLearn ILE.

These are the views that are currently present in the DynaLearn ILE:

- Build views:
    - Hierarchy views:
        - Entity hierarchy view: showing the ontology of entity definitions.
        - Agent hierarchy view: showing the ontology of agent definitions.
        - Assumption hierarchy view: showing the ontology of assumptions.
        - Scenario list workspace: showing the list of scenarios.
        - Model fragment list workspace: showing the list of model fragments.
        - Model fragment hierarchy workspace: showing the ontology of model fragments.
    - Content views:
        - Expression fragment content view: shows the expression fragment for editing purposes.
        - Scenario content workspace: shows a scenario for editing purposes.
        - Model fragment content workspace: show a model fragment for editing purposes.
- Simulation views:
    - State graph view: shows the behaviours the simulation derived from the specified model.
    - Value history view: shows the qualitative value changes throughout the simulation.
    - Equation history view: shows (in)equalities in a simulated state.
    - Dependencies view: shows the relations between entities and quantities in a simulation state.
    - Quantity values for state view: shows the values of the quantities in a simulated state.
    - Entity-Relation (ER) view: shows the dependency relations between entities and quantities in a simulated state.
    - Model fragments in state view: shows a list of model fragments that are active in a simulated state.
- Metadata views: show four different sorts of metadata (chapter 11).

Switching between views is effected by changes multiple interface items at the same time. Not only what is shown in the content window changes, but the menu options and content buttons change as well, in order to enable the user to interact with the new content. The following five interface elements change upon switching views:

1. The text that is displayed in the title bar (Section 3.1).
2. The actions that are displayed in the menu bar (Section 3.2).
3. The modelling ingredients that are displayed in the dropdown lists inside the navigation bar (Section 3.5).
4. The buttons that are shown in the content button bar (Section 3.7).
5. What is shown in the contents window (Section 3.8).

3.7. Content button bar

As discussed above, what fills the content button bar depends on the currently opened view. The items that are shown when the hierarchy view is active are described in Table 3.7.

Table 3.7: Buttons for the content button bar of the view editor.

Action	Icon	Description
Add New Definition		Adds a new definition. This is used for adding new entities, configurations, quantities, quantity spaces, assumptions, agents, scenarios, expression fragments, and model fragments.
Open Definition Properties		Displays the properties of the currently selected ingredient, showing its name, remarks, as well as definition type specific information (e.g. the parent type to which an entity definition belongs).
Copy Definition		Creates a copy of the selected definition.
Remove Definition		Removes the currently selected definition.

If the content view is active, then the items enumerated in Table 3.8 are shown in the content button bar. Which specific buttons are displayed, depends on the active use level. These distinctions will therefore be discussed in Chapters 4 through 9. In general terms, in an expression fragment and in a scenario only consequential model ingredients can be added (the blue ones). In a model fragment, both conditional (red) and consequential (blue) model ingredients can be added.

Table 3.8: Buttons for the content button bar of the view editor.

Action	Icon	Description
Add Entity		Adds an entity to a scenario or model fragment.
Add Attribute		Adds an attribute to a scenario, expression, or model fragment.
Add Configuration		Adds a configuration between two entities in a scenario, expression, or model fragment.
Add Quantity		Adds a quantity to an entity in a scenario, expression, or model fragment.
Add Quantity Space		Adds a quantity space to an existing quantity without one.
Add Value		Adds a value assignment to either a magnitude value, or to a derivative value in the quantity space.
Add Operator Relation (Plus or Minus)		Adds a calculus relation between two ingredients. There are multiple ways operator relations can be used: • Between two quantities • Between a quantity and a (point) value • Between two (point) values • Between an operator and a quantity • Between an operator and a point value
Add (In)Equality		(In)equalities can exist between different kinds of ingredients. There are five different combinations of arguments within an inequality relation: • An inequality between two quantities. • An inequality between two quantity values. • An inequality between a quantity and a point value. • An inequality between a derivative and a point value. • An inequality between two derivatives of two different quantities.

Add Positive/Negative Causal Relationship		Adds a positive or negative causal relationship between two entities.
Add Correspondence		Adds a correspondence. There are two kinds of correspondence: • Q-correspondences: between two quantity spaces. • V-correspondence: between two magnitudes / quantity values.
Add Proportionality		Adds a positive or negative proportionality between two quantities.
Add Influence		Adds a positive or negative direct influence relationship between two quantities.
Add Assumption		Adds an assumption to a scenario or to a model fragment.
Add Agent		Adds an agent to a scenario or to a model fragment.
Add Model Fragment as Condition		Adds an existing model fragment as a condition to the currently edited model fragment. The added model fragment must be part of the same library of model fragments.
Add identity		Identity links are a way to specify that two entities or agents refer to the same object. At least one of the items must originate from a conditional model fragment.
Delete Element		Deletes the currently selected element from a scenario or model fragment.

3.8. Colour coding

In most of the use levels in DynaLearn (UL1-4), the modelling ingredients in expressions are coloured blue. This colour indicates that these ingredients (except value assignments) are true in the entire simulation. However, in UL5 conditional knowledge is introduced via conditional expressions. The contents from the regular expression is incorporated the condition expression, and is visualised in green. In these conditional expressions it is possible to add modelling ingredients as conditions (red) or consequences (blue). The conditional expression can be read as a rule. If the conditions hold, then the consequences are also true. In this view, the representations on the earlier use-levels can be seen as rules without conditions.

At use-level 6 model fragments and scenarios are introduced. Similar to expressions, scenarios can only contain modelling ingredients as consequences (coloured blue). In model fragments, both conditions and consequences can be added. However, it is also possible for a model fragment to inherit contents from its parent, which is coloured green. Furthermore, it is possible

to import complete model fragments of which the contents is coloured black (with the model fragment identifier coloured red).

3.9. Content window

Using the buttons in the content button bar, modelling ingredients are added to the content window, showing the currently edited scenario or model fragment. In the content window these elements have their own distinguishing icons that are, for obvious reasons, similar to the icons that are displayed inside the content button bar (see Table 3.8). Icons that are used in the content window are enumerated in Table 3.9. Also, the meaning of these modelling elements is explained below.

Table 3.9: Icons of the model ingredients in the content window of the view editor.

Ingredient	Icon	Description
Entity		Entities are the physical objects or abstract concepts that constitute the system. Their relevant properties are represented as quantities that may change under the influence of processes. Entities are arranged in a subtype hierarchy.
Agent		Agents are used to model entities outside of the modelled system. Agents can have quantities influencing the rest of the system, which are called exogenous quantities or external influences.
Assumption		Assumptions are labels that are used to indicate that certain conditions are presumed to be true. They are often used to constrain the system behaviour generated by a model. They can be associated to structural and behavioural aspects of a system.
Configuration		Configurations are used to model relations between instances of entities and agents. Configurations are referred to as structural relations.
Quantity		Quantities represent changeable features of entities and agents. They are represented by their quantity value, which consists of the pair magnitude (amount of stuff) and derivative (direction of change).
Quantity space		A quantity space specifies the range of possible values that magnitude and derivatives of a quantity can have. Each quantity has a user defined quantity space for the magnitudes, and a default quantity space for the derivatives, namely: {dec, std, inc}[6]. Values are ordered in a quantity space. Each qualitative value is either a point or an interval, and within quantity spaces these two types consecutively alternate.
Magnitude and derivative assigned value		A value assignment indicates that a quantity has or should have a particular magnitude or derivative value in a scenario or in a model fragment.

[6] dec = decrease; std = steady; inc = increase.

Direct influence		Direct influences are directed relations between two quantities, and are either positive or negative. Direct influences are the cause of change within a model, and are therefore said to model processes. Depending on the magnitude of the source quantity and the type of influence, the derivative of the target quantity either increases or decreases. The direct influence I+(Q2, Q1) causes the quantity Q2 to increase if Q1 magnitude is positive; to decrease if Q1 is negative; and remain stable when Q1 is zero (assuming there are no other causal dependencies on Q2). The source quantity is often referred as being a rate, as it expresses the amount of change within a certain time period. For a negative influence (I-) the effect of the source quantity in the target quantity is just the opposite.
Proportionality		Qualitative proportionalities are directed relations between two quantities. They propagate the effects of a process, i.e. they set the derivative of the target quantity depending on the derivative of the source quantity. For this reason, they are also referred to as indirect influences. Proportionalities can also either be positive or negative. A proportionality P+(Q2, Q1) causes Q2 to increase (get the derivative value plus) if Q1 is increasing (that is, it has already the derivative value plus); Q2 will decrease if Q1 is decreasing; and Q2 will remain stable if Q1 is also stable (assuming there are no other influences on Q2). For a proportionality P– this is just the opposite.
Correspondence		Correspondences are relations between qualitative values of quantity spaces belonging to different quantities, and provides the notion of simultaneity: if specific values of two quantities correspond, then when one occurs, the other one also occurs. Correspondences can be either directed or undirected. The former means that when value A of quantity space X corresponds to value B of quantity space Y, the simulator derives that quantity space Y has value B when quantity space X has value A. If the correspondence is undirected, it may also derive the value A of quantity space X when quantity space Y has value B. There are 6 correspondence types, 12 if directedness is included.
Inequality		Inequalities (<, ≤, =, ≥, >) specify an ordinal relation between two items, i.e. that one item is different from (or equal to) the other item. Because inequalities specify an order between items, they are sometimes referred to as ordinal relations. There are eleven ways to use inequalities, depending on the type of the two items related by them.

4. Concept map – Use level 1

The concept map use level (or use level 1) is the simplest way the DynaLearn software can be used. It allows the learner to create a network of nodes connected by relationships.

4.1. Representation

In terms of the DynaLearn knowledge representation the following ingredients can be used in the concept map (Figure 4.1):

- Entities
- Configurations

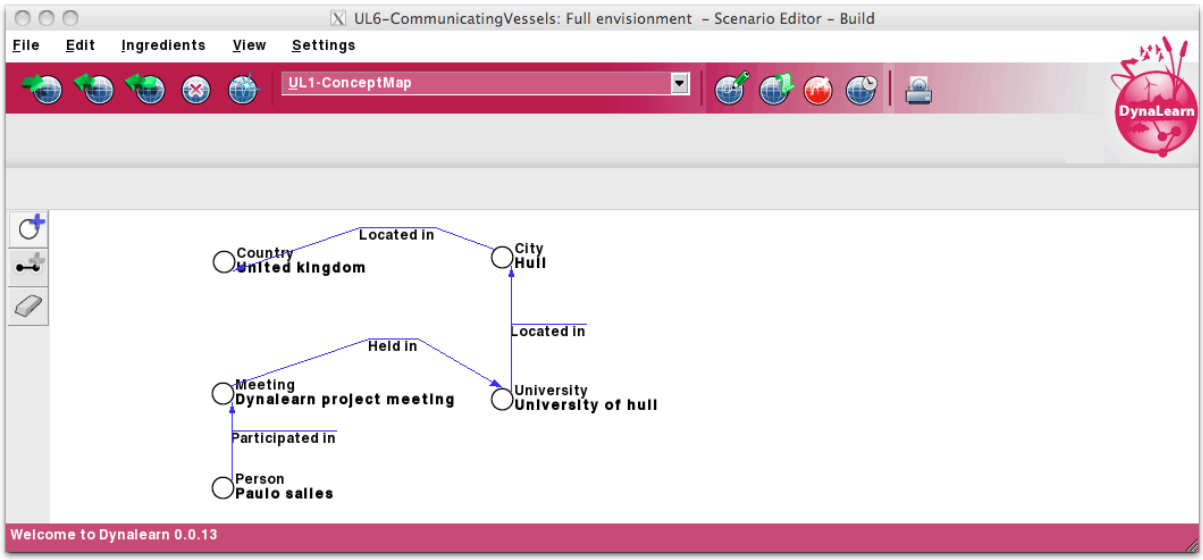


Figure 4.1: The concept map.

Important to note is the feature to make the knowledge representation even simpler. By default it is possible to add type information to concepts. For example, the United Kingdom is a Country.[7] However, for some group of student a teacher may choose to make the representation even simpler by removing the type information (this is done by clicking the *Hide Supertypes* options in the menu). The dialogs to add entities and configurations are adapted to not show the type information. The concept map without type information is shown in Figure 4.2. Note that this option is also available in user-levels 2 to 5.

At use level 1, no simulations can be run.

[7] In this example, 'United Kingdom' is the concept name and 'Country' is its type.

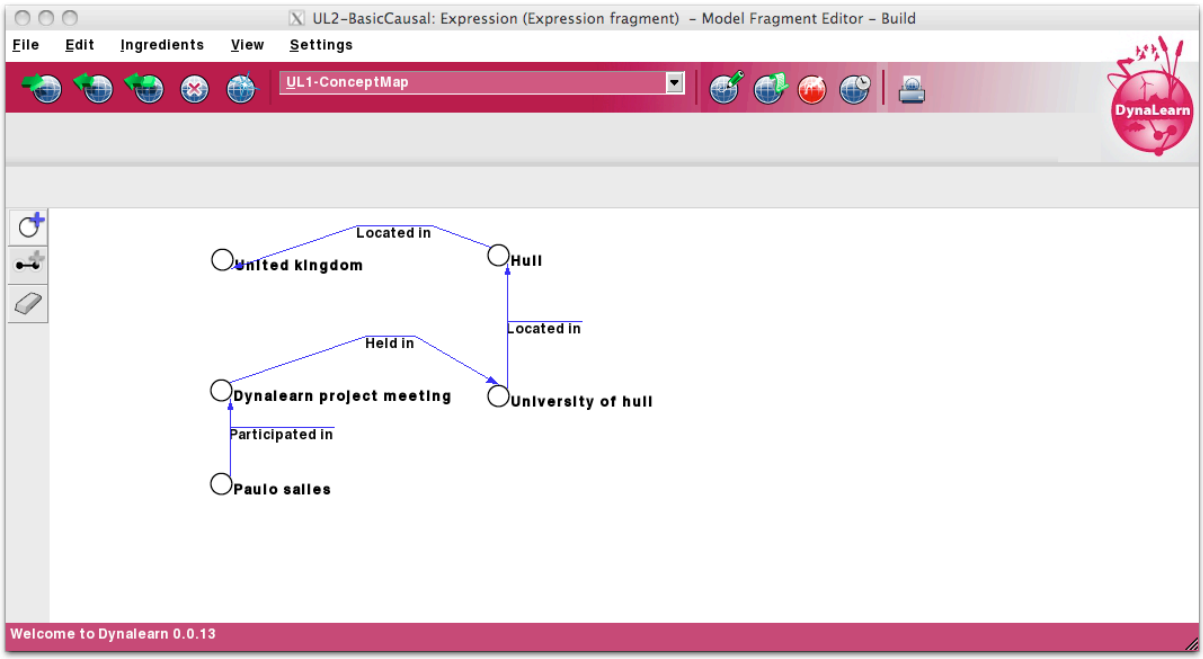


Figure 4.2: The concept map without type information.

5. Basic causal model – Use level 2

The basic causal model use level (or use level 2) is the first use level in which simulation can be run.

5.1. Representation

The Build workspace is shown in Figure 5.1.

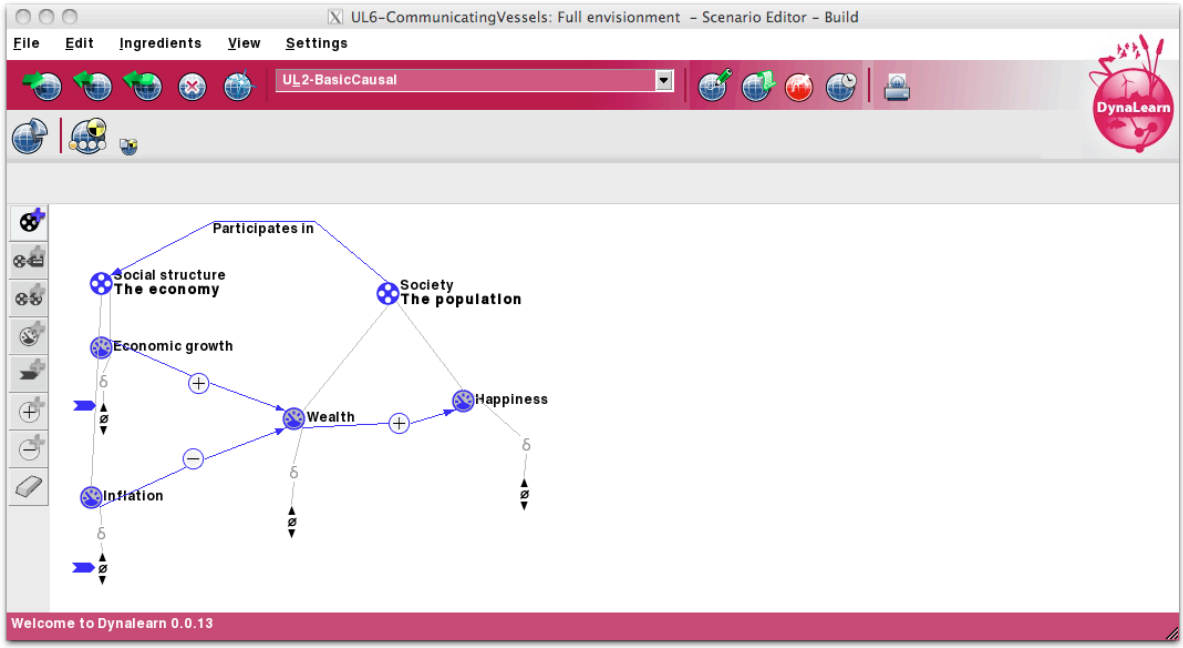


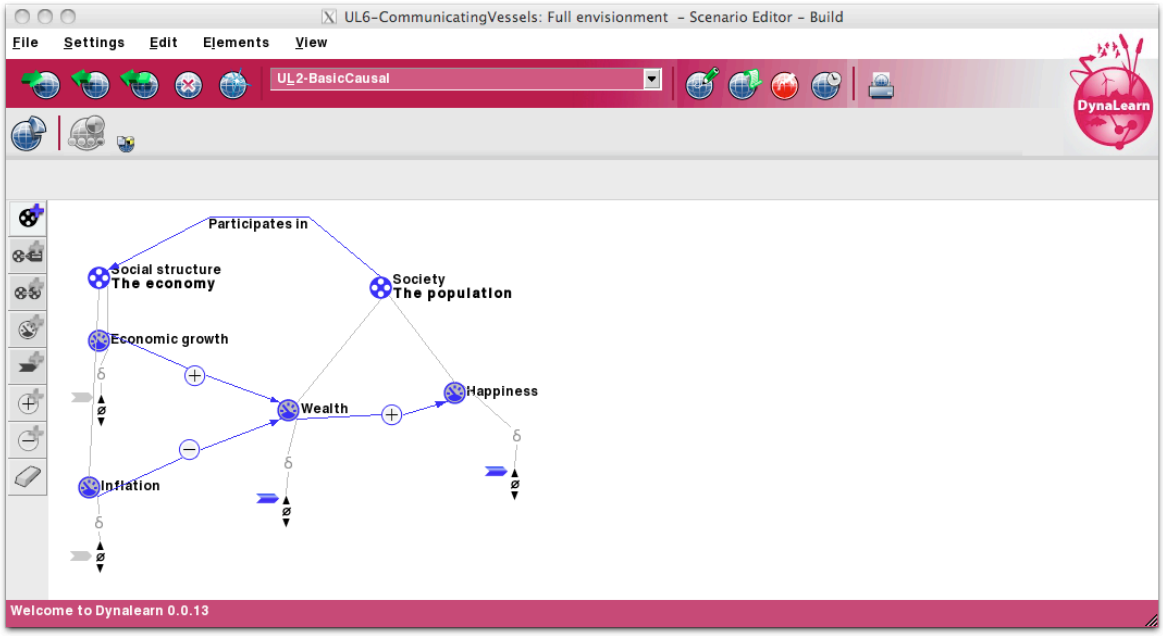
Figure 5.1: Use level 2 – Build workspace.

Learners can express conceptual models using the following ingredients:

- Entity
- Attribute
- Configuration
- Quantity
- Quantity space
  - Derivative
- Value assignment
  - Derivative
- Causal dependency, plus and minus

5.2. Simulation

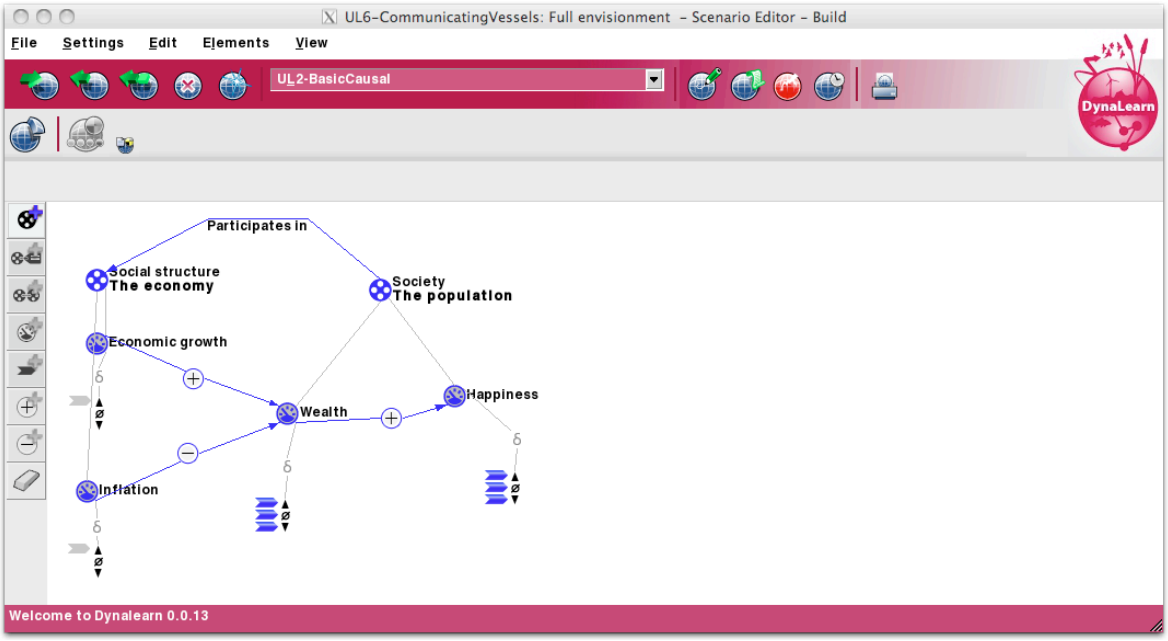
By pressing the simulate button in the expression workspace a simulation is performed. The visualisation is almost identical to the Build workspace. As shown in Figure 5.2 only the values assigned in the Build editor are greyed out (as in Economic growth and Inflation), and the derived values are added using a blue-coloured value assignment (as in Wealth and Happiness).



Fi

Figure 5.2: Simulate – The simulation on use level 2.

It is possible to have ambiguous results and inconsistent results for a simulation. Ambiguous means that there are multiple possible values for a quantity. This is visualised using multiple value assignments (Wealth and Happiness in Figure 5.3). Inconsistency means that the assigned values (Economic growth, Inflation and Wealth in Figure 5.4) contradict the values derived by the simulator. This is visualised using a large question mark.



Fi

Figure 5.3: Simulate – Ambiguous outcome on use level 2.

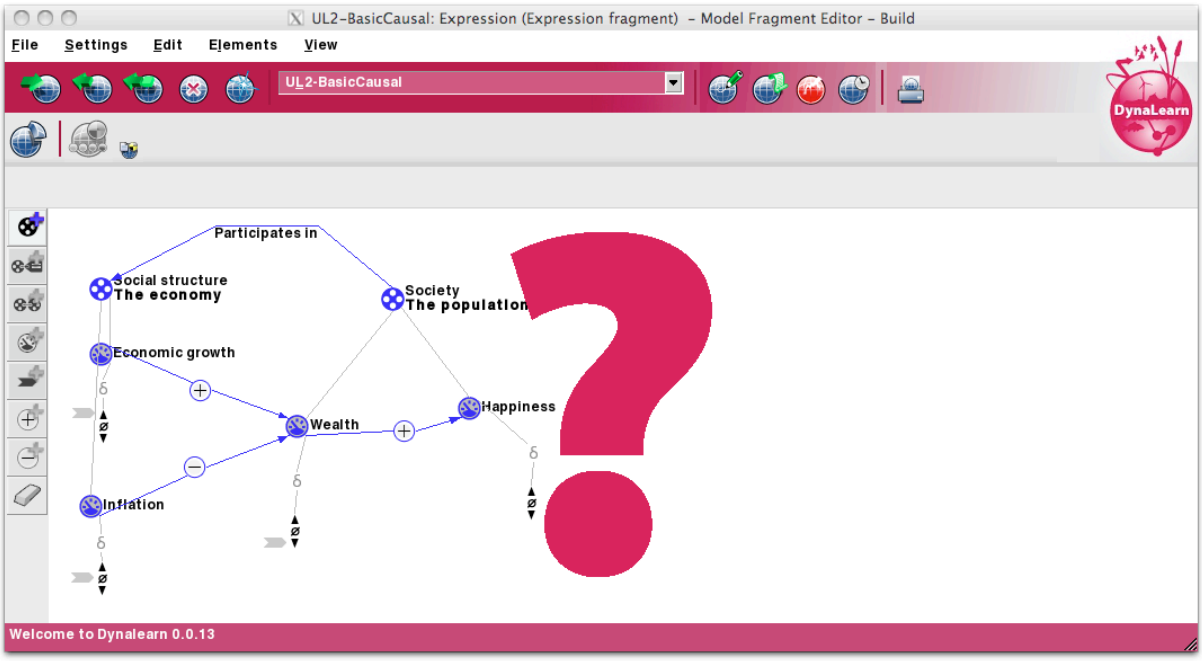


Figure 5.3: Simulate – Inconsistent outcome on use level 2.

6. Basic causal model with state graph – Use level 3

Use level 3 augments use level 2 by providing the option to add one or more quantity spaces to a quantity. These quantity spaces can be selected from a default list or created new by the learner. Adding this feature has a significant impact on the simulation results and necessarily introduces concepts such as state-graph, behaviour path, and value history.

6.1. Representation

The Build workspace is shown in Figure 6.1.

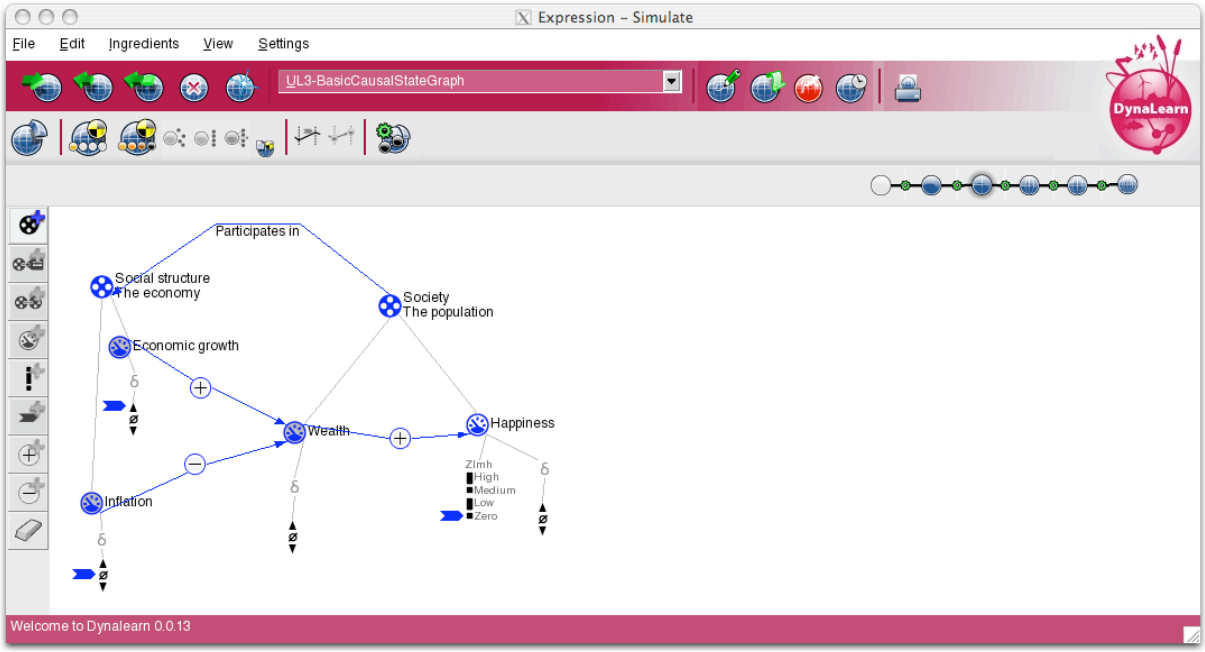


Figure 6.1: Use level 3 – Build.

Learners can express conceptual models using the following ingredients:

- Entity
- Attribute
- Configuration
- Quantity
- Quantity space
    - Magnitude
    - Derivative
- Value assignment
    - Derivative
    - Magnitude

- Causal dependency, plus and minus
- Correspondence

6.2. Simulation

Simulating the model generates a state graph (Figure 6.2), which visualises the following ingredients

- State
- State transition

States can be selected individually, as a set, or as a path.

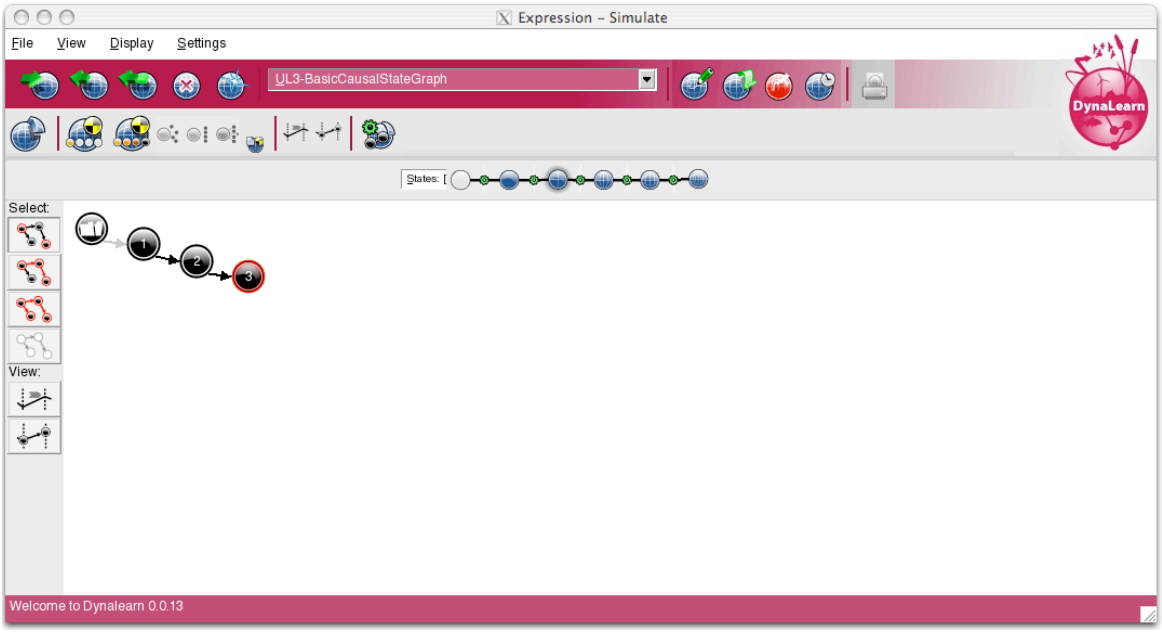


Figure 6.2: Use level 3 – Simulate: State graph.

From the simulate workspace the Value History (Figure 6.3) and the Transition History (Figure 6.4) workspaces can be opened. Both visualisations take the selected states as input. The Value history displays the following ingredients:

- Entity
- Quantity
- Quantity space (magnitude)
- States
- Current value (in each state)
    - Magnitude
    - Derivative value

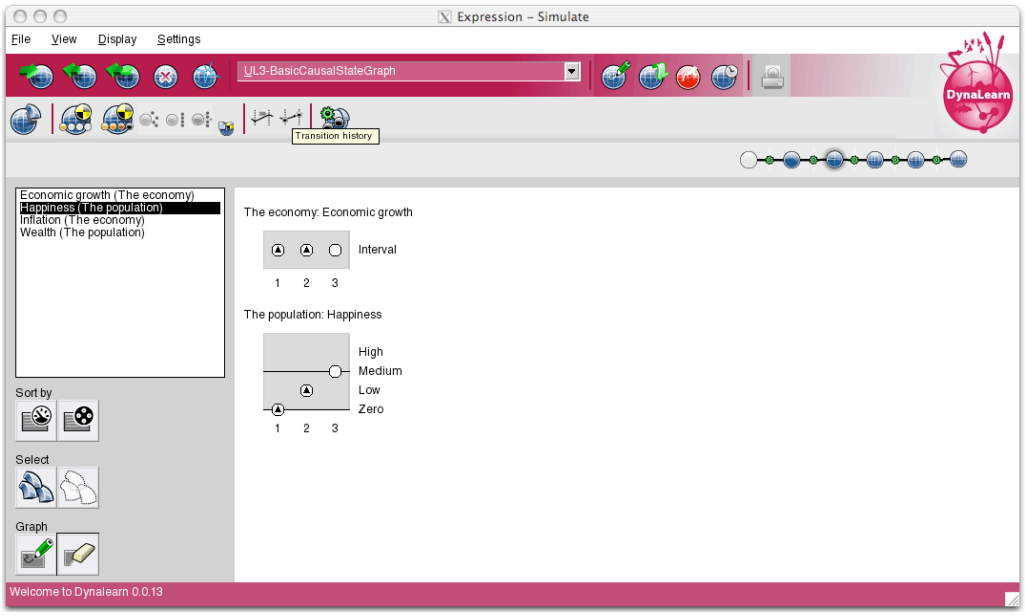


Figure 6.3: Use level 3 – Simulate: Value history.

The Transition history displays how states transition into each other. It shows the following ingredients:

- States
- Transitions

Furthermore, for each transition the Transition history shows what is causing the transition to occur. For example, the magnitude of a quantity changes into the interval above its current magnitude because it had a positive derivative in a previous state. Each of the transitions can be further analysed, which brings up a screen that provides details about the transition (Figure 6.4, front). The screen containing the transition details again shows what caused the transition to occur (a so-called transition rule), but it also shows the result of this transition rule on the current state (e.g. condition: *Happiness is zero and increasing*), and its successor (results: *Happiness derivative is greater than zero and Happiness is low*).

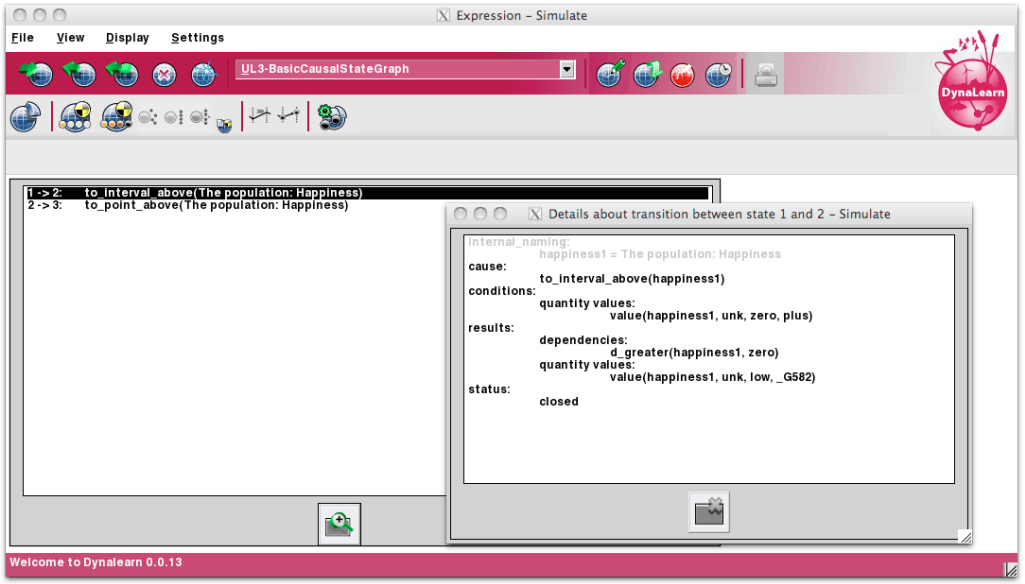


Figure 6.4: Use level 3 – Simulate: Transition history.

7. Causal differentiation – Use level 4

Use level 4 augments use level 3 in multiple ways. The main distinguishing feature is the inclusion of proportionalities and direct influences instead of the plus- and minus-signed relationships of the previous use level. Another difference is that every quantity has a quantity space. These quantity spaces are defined by using the quantity space editor. Whenever a quantity is being created in the quantity editor, a quantity space must be selected for it (in other words, in use level 4 quantities cannot exist without a quantity space).

7.1. Representation

The Build workspace is shown in Figure 7.1.

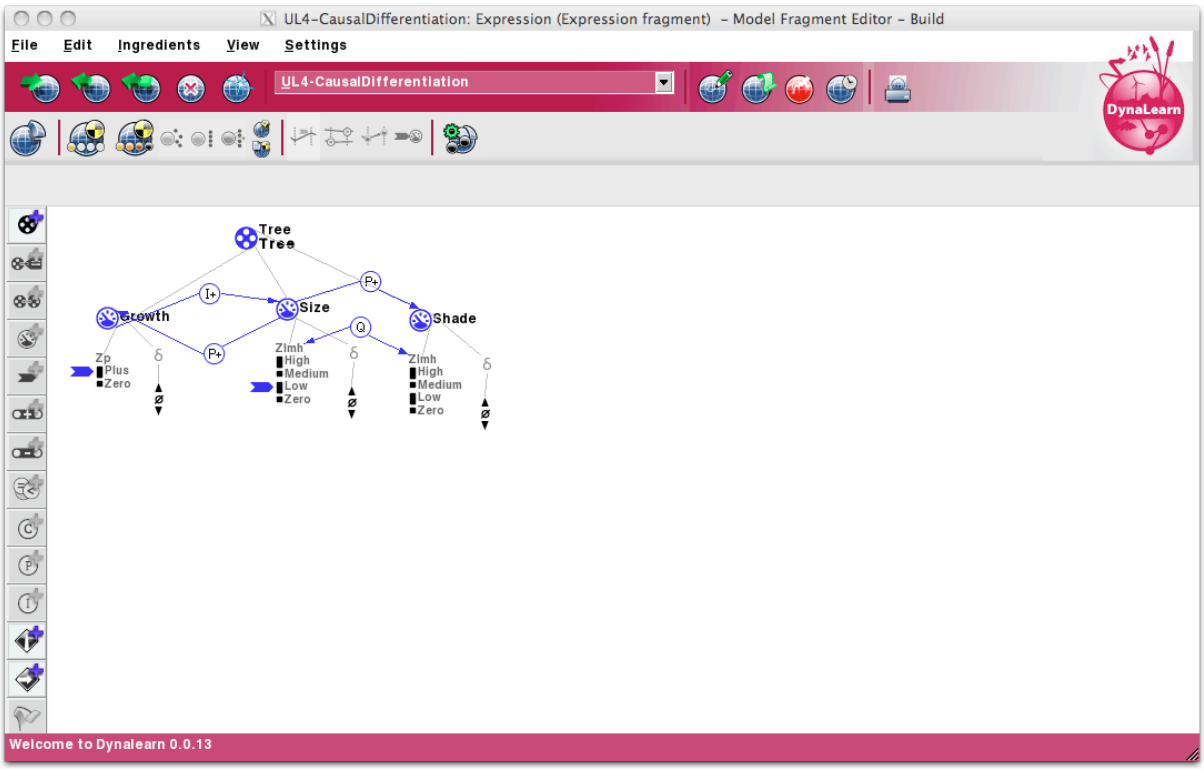


Figure 7.1: Use level 4 – Build.

Learners can express conceptual models using the following ingredients:

- Entity
- Attribute
- Configuration
- Quantity
- Quantity space
    - Magnitude
    - Derivative

- Value assignment
    - Derivative
    - Magnitude
- Causal dependencies
    - Proportionalities
    - Influences
- Correspondence

7.2. Simulation

As in use level 4, simulating the model generates a state graph (Figure 7.2, back). From the simulate workspace the value history and transition history views can be opened. New in this use level is the equation history (Figure 7.3), and the quantity values workspace (Figure 7.2, front). The quantity values workspace displays the following ingredients:

- Entities
- Quantities
- Magnitude
- Derivative, including second order derivative

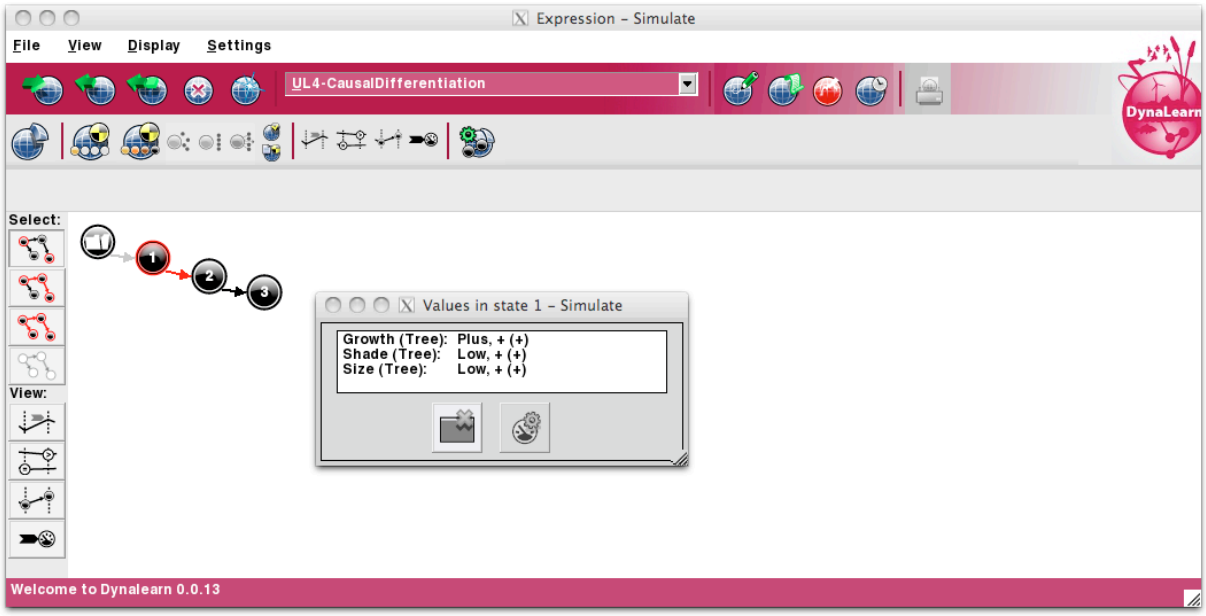


Figure 7.2: Use level 4 – Simulate workspace.

The equation history displays how equations (e.g. inequalities between quantities) change over time. It shows the following ingredients:

- Entities
- Quantities

- Magnitudes
- Derivatives
- Inequalities
- Operators

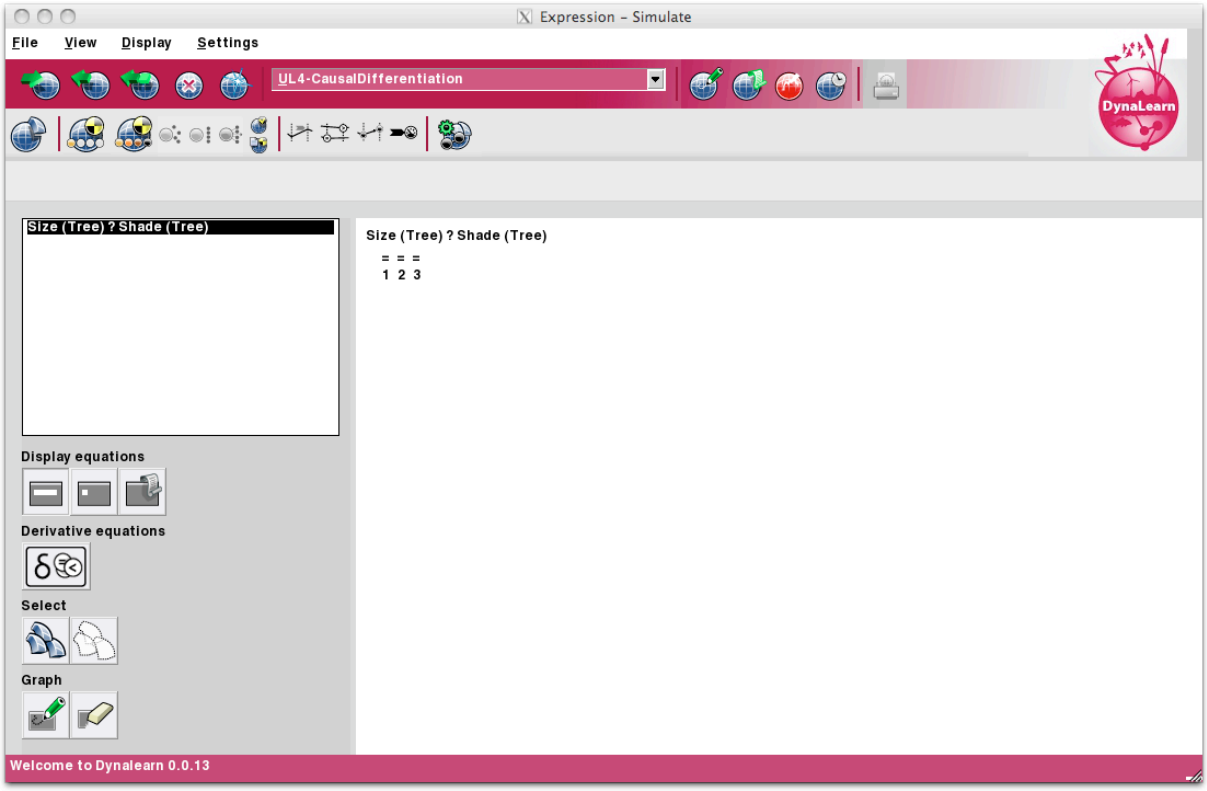


Figure 7.3: Use level 4 – Equation history workspace.

8. Conditional knowledge – Use level 5

Use level 5 augments use level 4 by allowing conditional expressions to be defined. These conditional expressions incorporate the expression fragment, but allow separate conditions and consequences to be added.

8.1. Representation

The Build workspace (Figure 8.1) is identical to the expression editor in use level 4. However, there are new buttons that allow switching to the conditional expression list and switching to the last edited conditional expression as well.

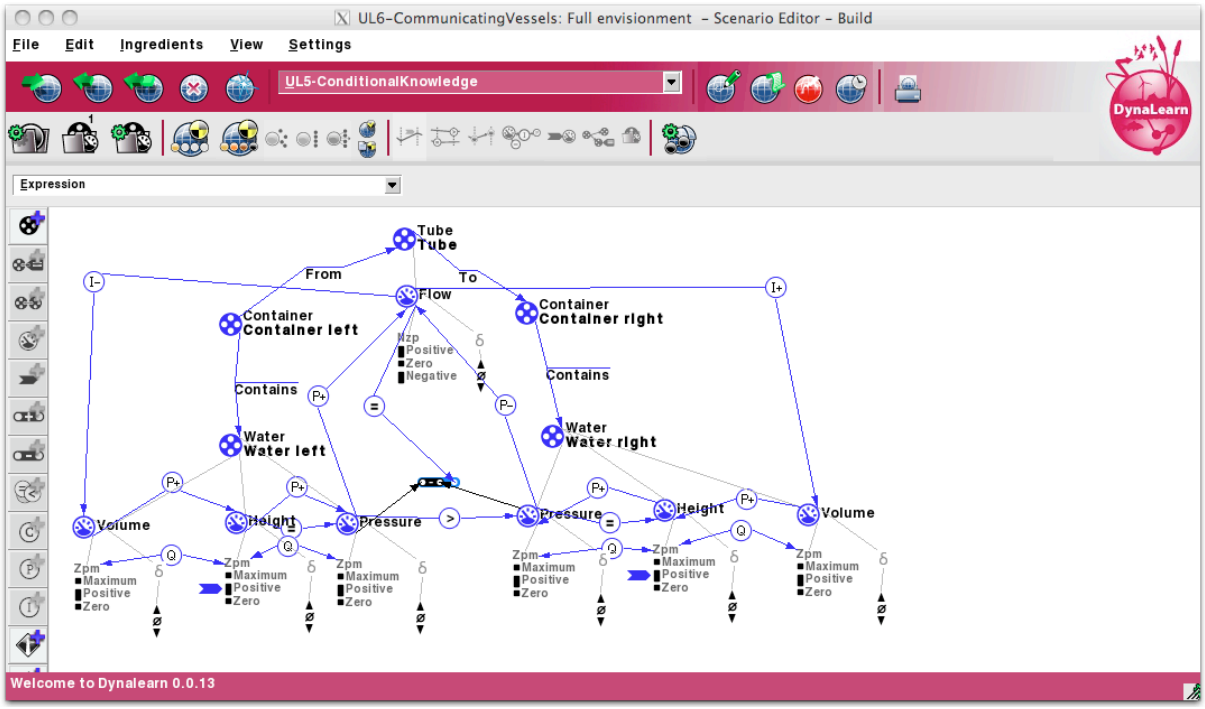


Figure 8.1: Use level 5 – Build workspace.

The conditional expression list (Figure 8.2) shows a list of all the conditional expressions a learner has created. The conditional expression workspace (Figure 8.3) allows the following ingredients to be created (as conditions and consequences, unless specifically specified):

- Entities
- Agents
- Assumptions
- Configurations
- Quantities, including quantity spaces
- Value assignments
- Operators, plus and minus

- Inequalities
- Correspondences (consequence only)
- Causal Dependencies (consequence only)
    - Proportionalities
    - Influences

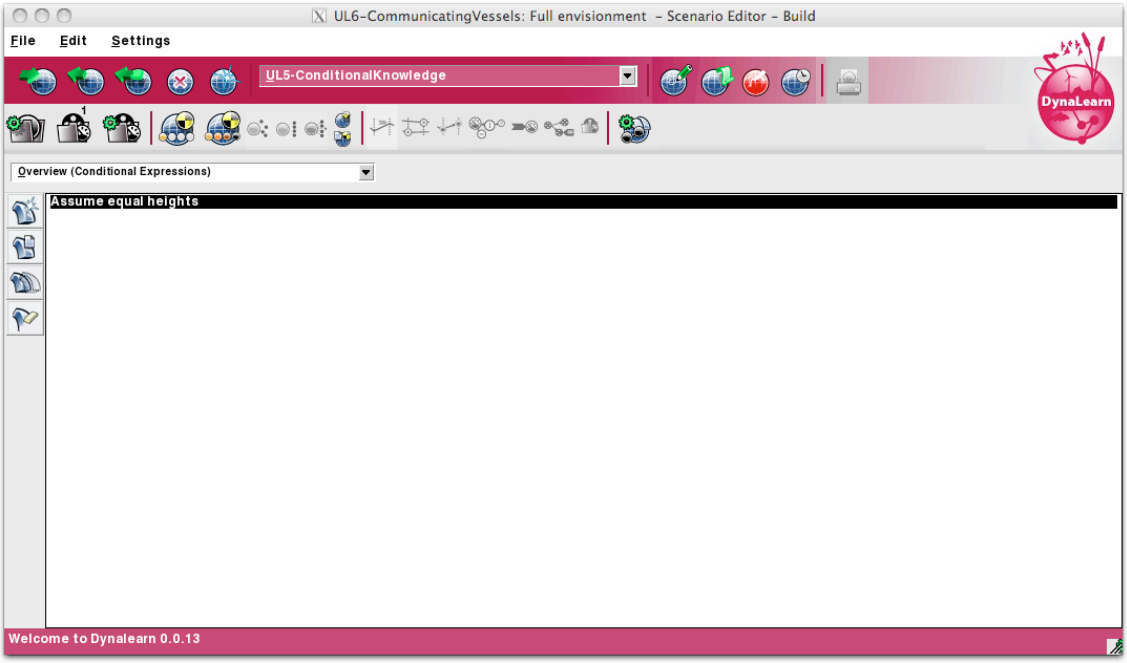


Figure 8.2: Use level 5 – Conditional expression list

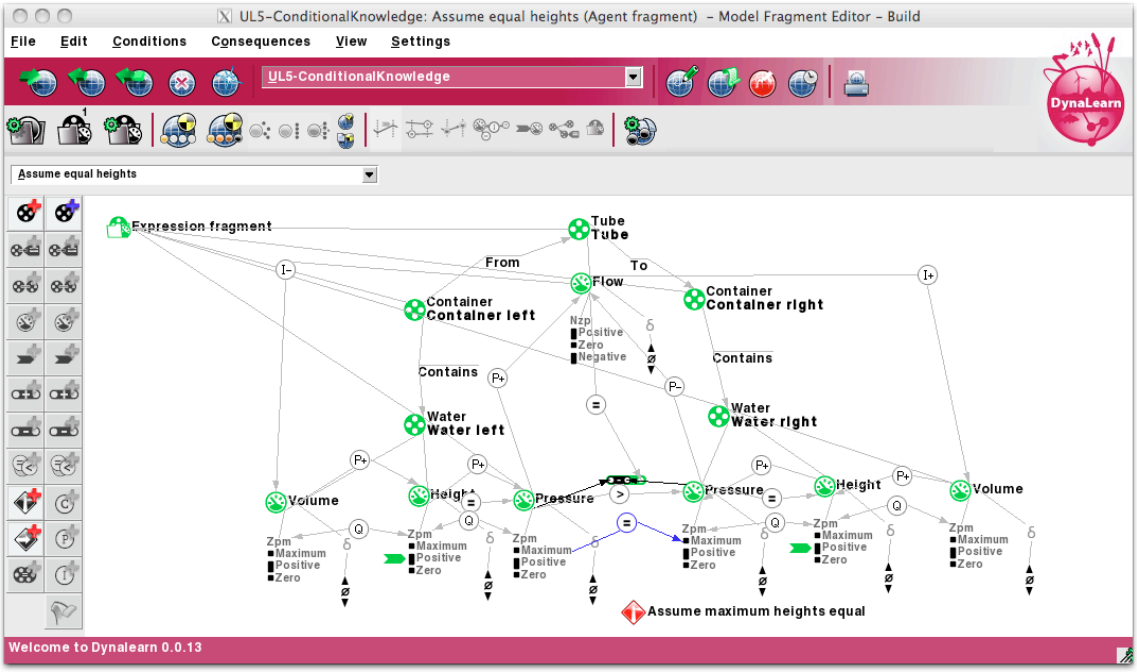


Figure 8.3: Use level 5 – Conditional expression workspace.

8.2. Simulation

The simulate workspace (Figure 8.4) adds some new buttons in comparison to use level 4:

- Show Conditional Expressions for State
- Show Dependencies
- Show ER Diagram

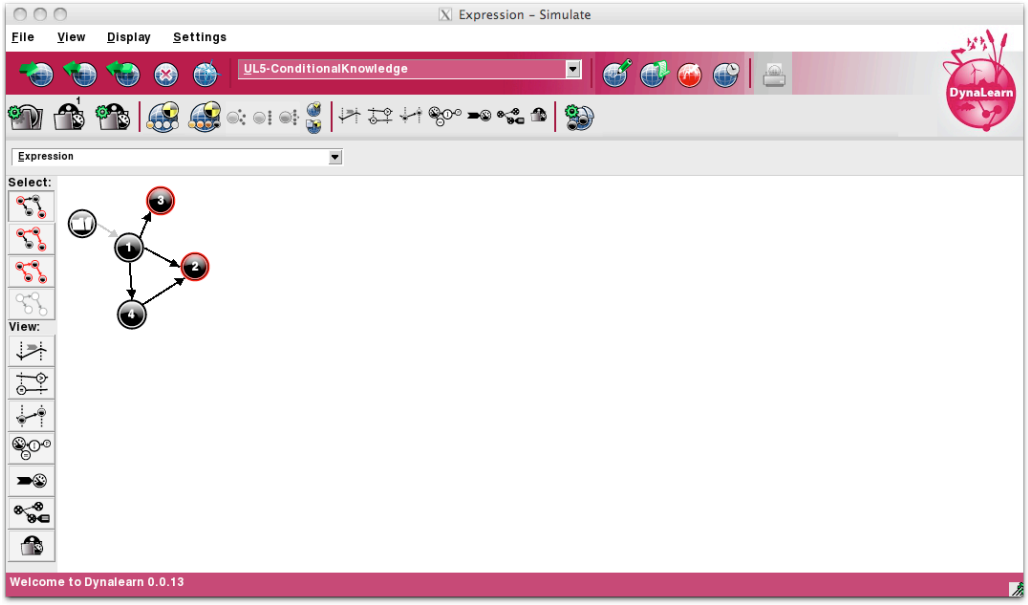


Figure 8.4: Use level 5 – Simulate workspace (State graph).

The *Show Conditional Expressions in State* workspace (Figure 8.5) shows the conditional expressions that have become active in a particular state.

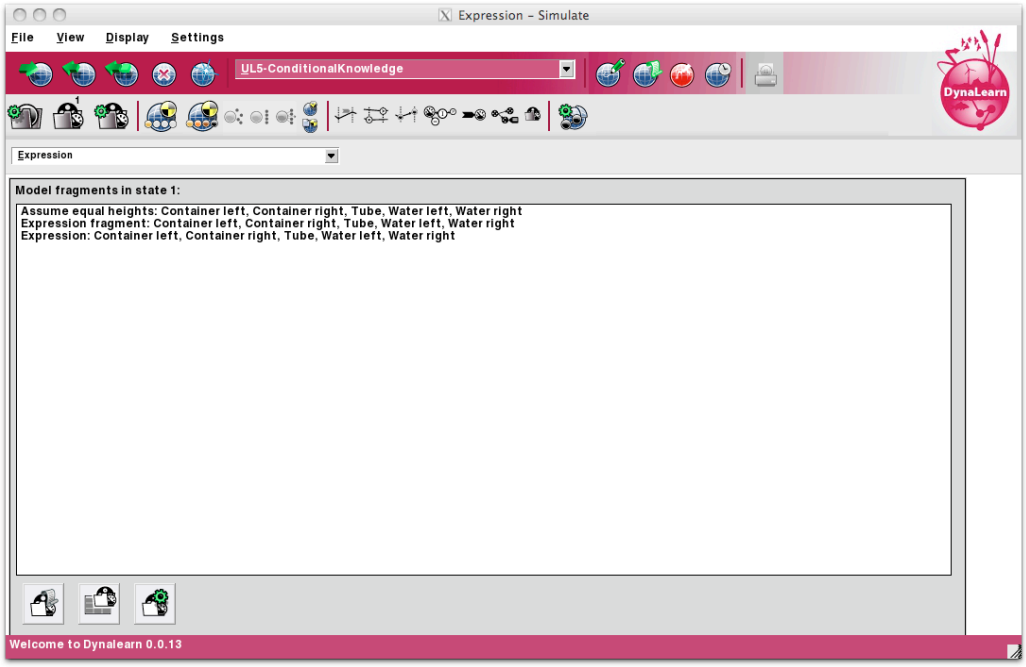


Figure 8.5: Use level 5 – Simulate conditional expressions.

The dependencies workspace (Figure 8.6) shows all model ingredients that are either in the expression, that are the result of the simulation (i.e. value assignments), or that have been introduced by active conditional expressions in a particular state. The workspace allows specific relationships to be hidden in order to make insightful images for reports.

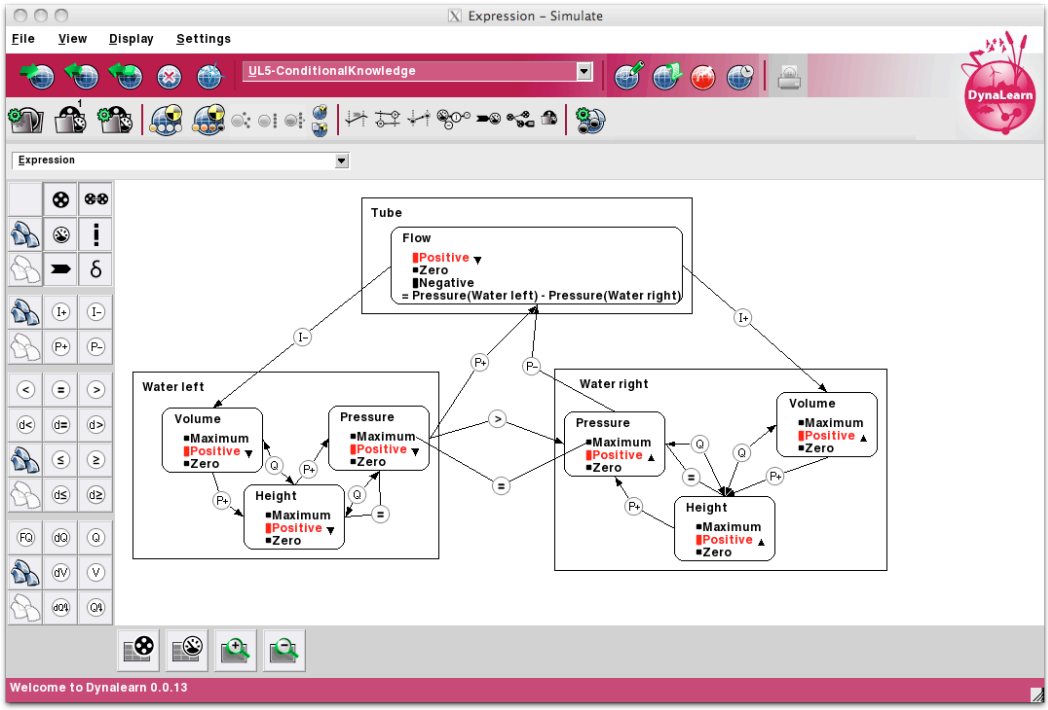


Figure 8.6: Use level 5 – Simulate dependencies workspace.

The Entity-Relation (ER) workspace (Figure 8.7) shows the structure of the system in a particular state. The conditional expression is shown with dashed lines.

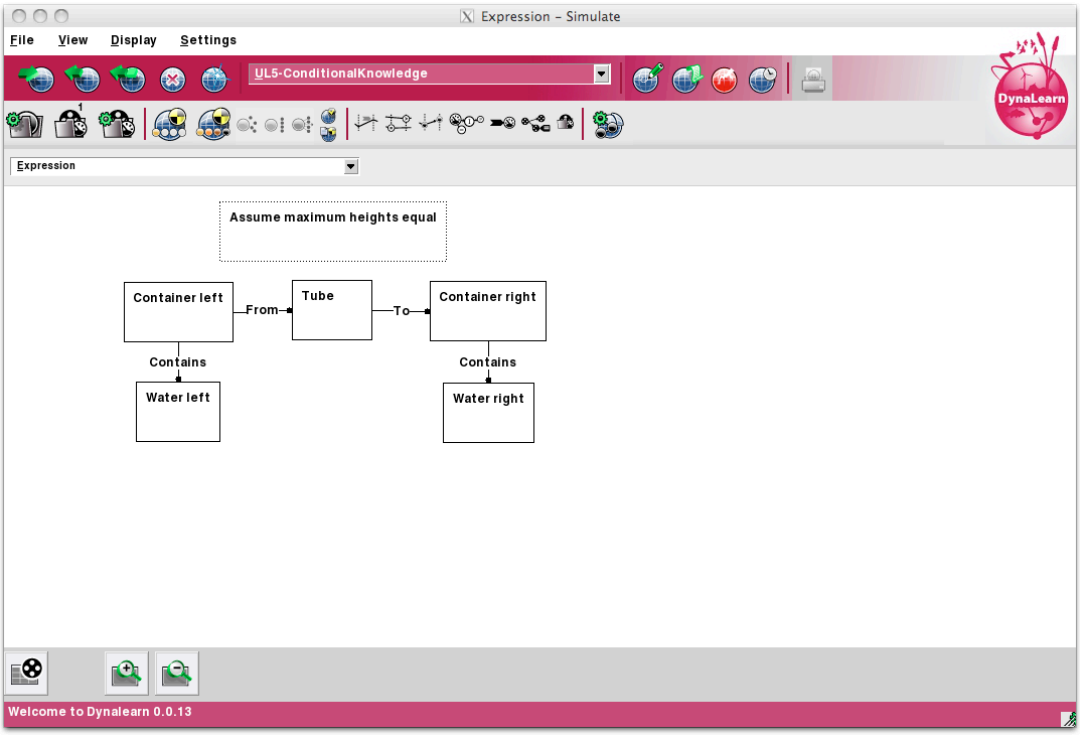


Figure 8.7: Use level 5 – Simulate ER diagram workspace.

9. Generic and reusable knowledge – Use level 6

The generic and reusable knowledge use level (or use level 6), allows the full range of representation and reasoning that is available in Garp3. Particularly, in contrast to use level 5, expressions (including conditional expressions) are no longer available. Instead, learners create scenarios and model fragments.

In contrast to expressions (which contain both scenario and model fragment aspects), which will always become active in a simulation, in use level 6 a simulation is based on a single scenario (which describes the initial state of the simulation). The model fragments (which contain both conditions and consequences) can be seen as rules (IF [*conditions*], THEN [*consequences*]). If the scenario fulfils the conditions of a model fragment, it becomes active, and the ingredients represented as consequences of the model fragment are introduced to the description of the initial state.

9.1. Representation

The scenario workspace is shown in Figure 9.1. In it the following ingredients can be used (all these can only be added as consequences):

- Entities
- Agents
- Assumptions
- Configurations
- Attributes
- Quantities, including quantity spaces
- Value assignments
- Operators, plus and minus
- Inequalities

In contrast to the conditional expressions in use level 5, which were based on the expression fragment, model fragments in use level 6 can be created independently. Another addition in use level 6 is the ability to organize model fragments into a subtype hierarchy (Figure 9.2). Moreover, model fragments can be imported into other model fragments. The model fragment workspace (Figure 9.3) shows the Liquid Flow model fragment using the imported Contained Liquid model fragment twice, and introducing a set of new ingredients. In the model fragment editor the following ingredients can be used (as both conditions and consequences unless specified otherwise):

- Entities
- Agents
- Assumptions (only conditional)
- Configurations

- Quantities, including quantity spaces
- Value assignments
- Operators, plus and minus
- Inequalities
- Correspondences (consequences only)
- Causal Dependencies (consequences only)
    - Proportionalities
    - Influences
- Model fragments (conditions only)
- Identity relationships

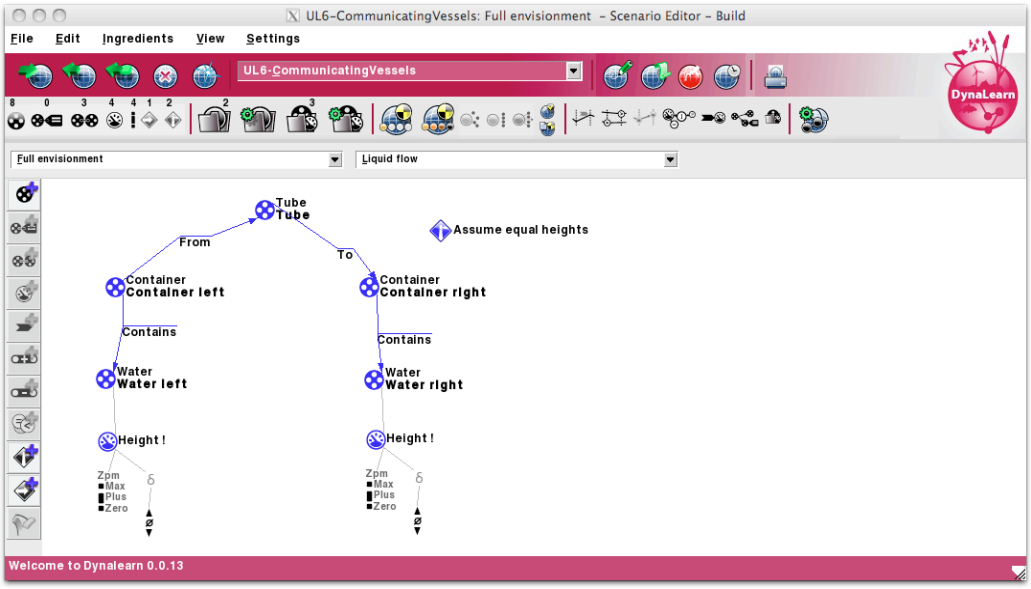


Figure 9.1: Use level 6 – Build: Scenario editor.

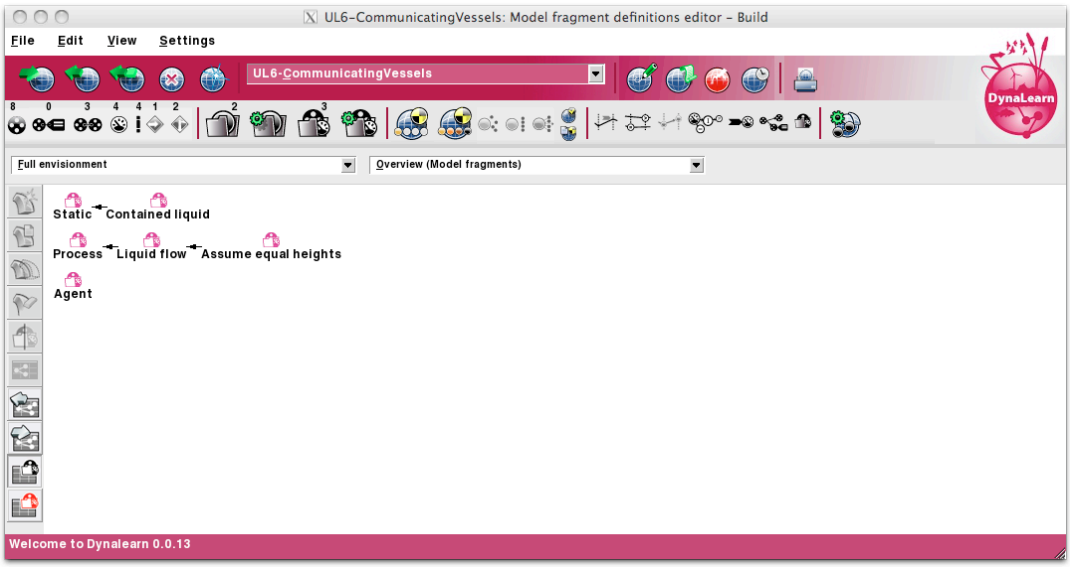


Figure 9.2: Use level 6 – Build: Model fragment overview.

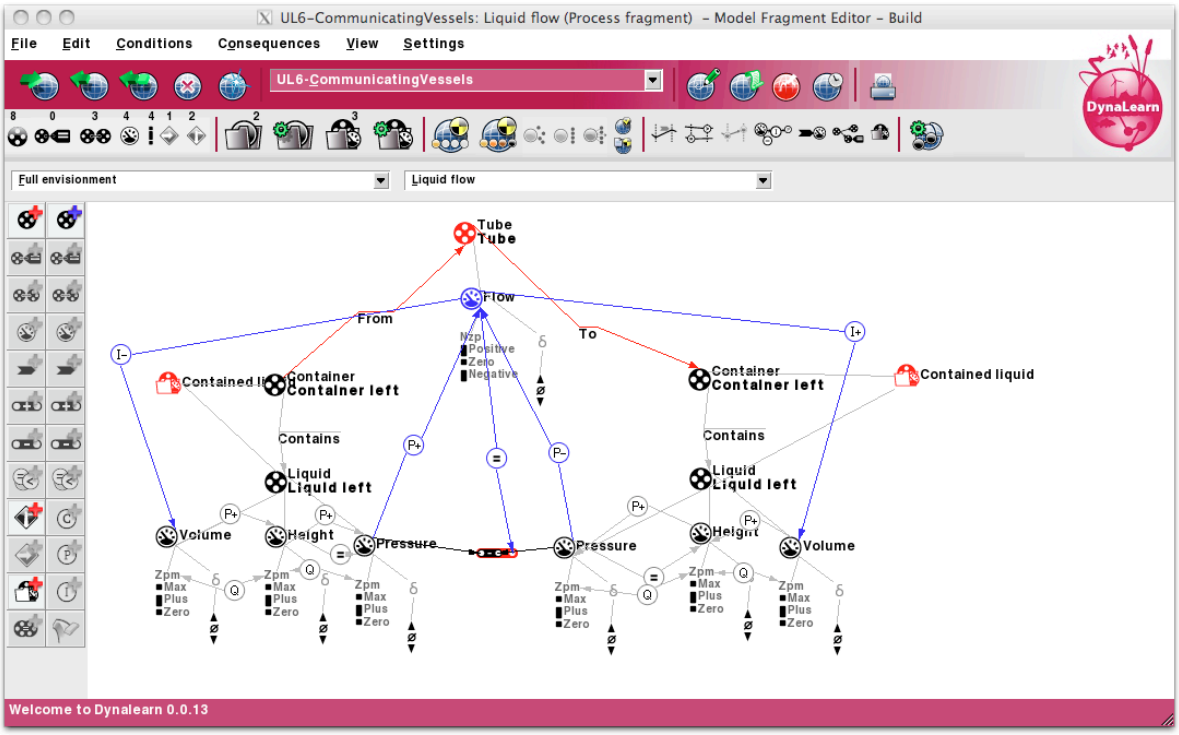


Figure 9.3 Use level 6 – Build: Model fragment editor.

9.2. Simulation

The simulate workspace (Figure 9.4) shows the state graph representing the predicted behaviour of the system. In use level 6, the complete range of simulate workspaces (introduced in earlier workspaces) is available for the learner to use.

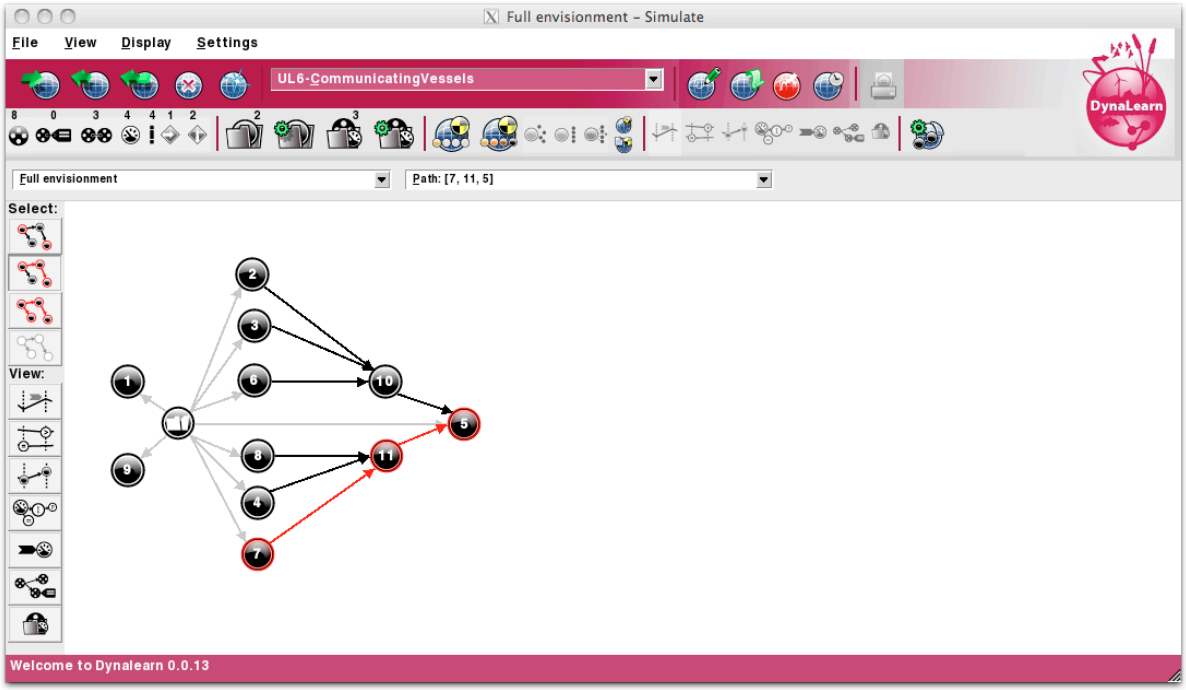


Figure 9.4: Use level 6 – Simulate: State graph.

10. Multiple simulations, saving simulations and path selection

In the Garp3 workbench, it is only possible to run a single simulation at a time. As such, in Garp3 all the screens related to a simulation are closed when a new simulation is run. It is therefore impossible to directly compare simulation results within the software.

In the DynaLearn ILE significant improvements have been made in how the software deals with simulations. As such, it is now possible to have *multiple simulations* running at the same time. This makes it possible to compare simulation results coming from different scenarios of the same model. It also provides the ability to run simulations of different models in the background. This particular feature is important in other parts of the DynaLearn project where, for example, simulation results of models created by students are compared to simulation results of models made by experts.

A further improvement upon Garp3 is the ability to *save state and path selections* made in the state graph. This feature has multiple educational purposes. Firstly, a domain expert creating an educational model can save specific paths in the state graph that reflect an important behaviour of the modelled system. Secondly, students and teachers can have easy access to the saved paths in the model. Furthermore, students can save behaviour paths that they do not understand, so they can ask for feedback from a teacher (or a Virtual Character). Thirdly, the stored selections can be used by the Virtual Characters to, for example, explain the model.

Finally, a feature that has been significantly improved since Garp3 is *saving simulations*. It is possible to save simulations in Garp3, but saved simulations can get out of synchronization with the model (i.e. a saved simulation may not reflect the correct simulation as it would be produced by simulating the model). Furthermore, the links between the simulation and the model are lost when a simulation was saved. In DynaLearn, these issues with saved simulations have been remedied. Saving simulations now also saves the links between the simulation and the model, and also saves the stored selections. As a result, in DynaLearn there is no difference between loading a saved simulation and running a simulation (except that there can be some stored selections).

There are multiple reasons why saved simulations are a key feature for an ILE such as DynaLearn. Firstly, this functionality is used to enable multiple simulation support. Secondly, it allows students to quickly inspect simulations without having to run the simulation themselves (which for complex models may take time). Thirdly, the saved simulations allow users to quickly switch between models made by learners and an expert model (running in the background), which, for example, can be important when the learner is being guided towards a more correct or more complicated model.

10.1. Multiple simulations

Multiple simulation support was implemented in such a way that there is no limit to the number of simulations that can run in parallel. From the perspective of usability, the question was how the functionality should be constrained in order to be understandable for the learner. In the design, we have chosen to have a single simulation per scenario. That is, the simulation buttons will either start a new simulation if it does not exist yet, or bring the user to the current simulation of the scenario. Once in the simulation environment, the learner can choose to restart the simulation. Multiple simulation support takes the multiple model support into consideration as well. It is only possible to investigate simulations that belong to the currently active set of models.

10.2. Storing selected states and paths

In the simulate environment, the right-hand side menu will show the current state selection (Figure 10.1). There are two types of state selections, states and path. In the state mode, both states and all transitions from each state will be selected (which is important for the state-transition view). In the path mode only paths can be selected (together with only the transitions between the selected states). If the current selection is not a path then the selection is shown as a state selection instead of a path selection. State selections are prefixed with the text 'States:', while path selections are prefixed with the text: 'Path:'.

Next to the menu there is a + and a – buttons. The + button stores the current selection in the simulation, while the – button removes the current selection. Selecting one of the stored selections makes this selection active in the state graph.

Note that storing a selection does not change the model itself, as a selection only makes sense in the context of a particular simulation. Simulating the same scenario in steps might result in a different numbering for the states (as it is controlled by the learner). As such in the new simulation the state selection might not make sense. The simulation should be saved in the model in order for the stored selections to be saved along with it.

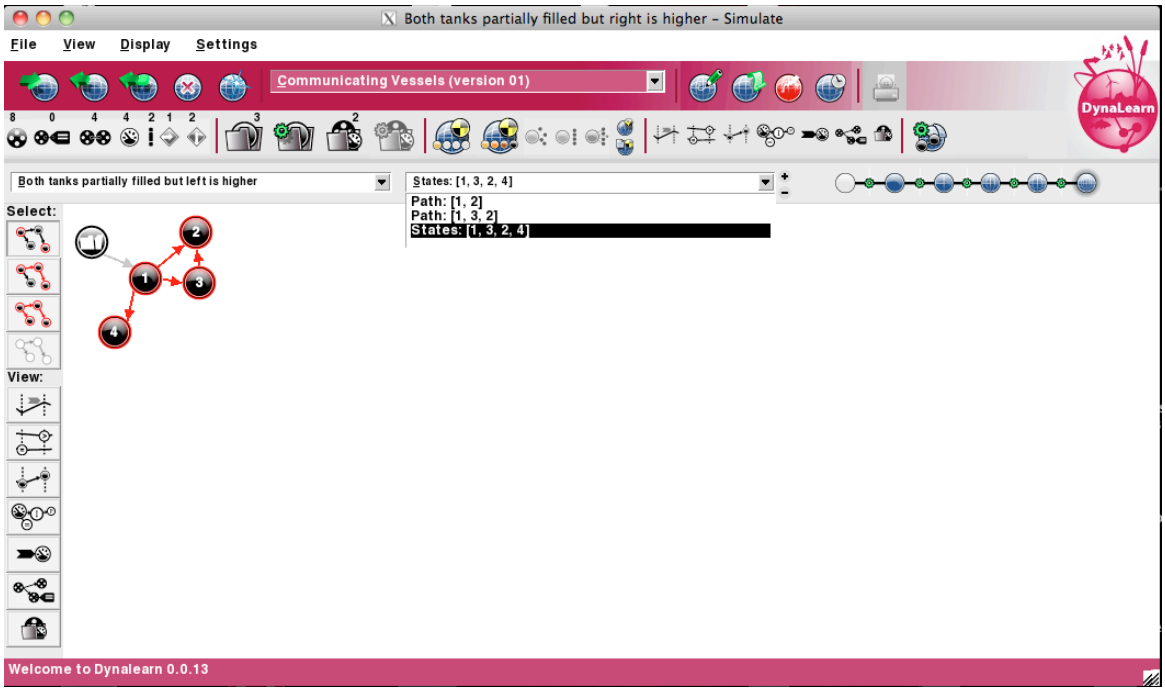


Figure 10.1: A number of saved state selections. State selections that make up a path in the state graph are identified as such in the list.

10.3. Saving simulation results

In the simulate environment, the 'save/load simulation' button creates a dialog in which simulations can be saved or loaded (Figure 10.2). If there is no active simulation, there is nothing to save, so the lower part of the dialog is not shown. In that case, saved simulations can only be loaded. When there is an active simulation, this simulation can be saved by giving it a name and pressing the 'ok' button. The 'delete' button allows saved simulations to be removed.

A new feature in DynaLearn is the ability to simulate all scenarios and save the resulting simulations to the model. Figure 10.2 shows the results of running and saving all simulations in a communicating vessels model.

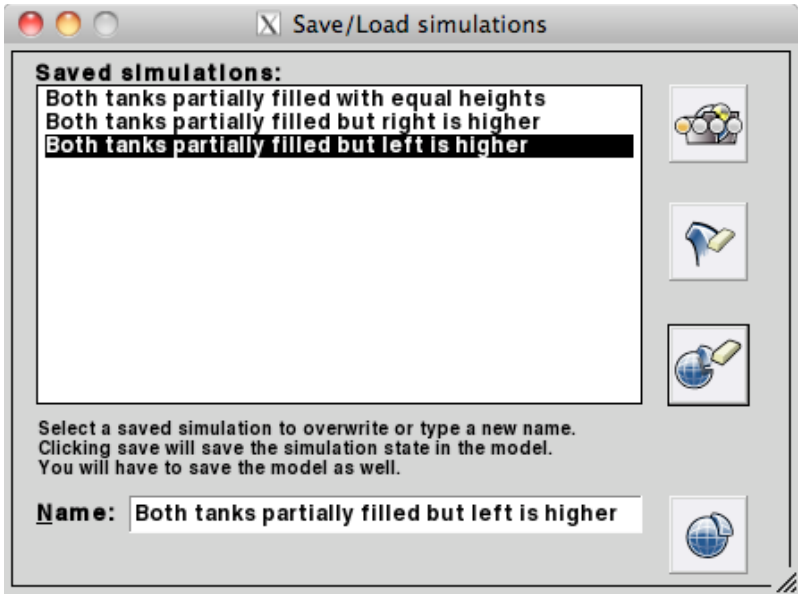


Figure 10.2: The save/load simulations dialogue.

10.4. Implementation

DynaLearn introduces two new classes to deal with multiple simulations, saved simulations and stored state selections: the `stateSelection` class and the simulation class. The `stateSelection` class is used to store particular selections in the state graph. The simulation class represents a single simulation.

The `stateSelection` class has the following variables:

- `selection mode`: either state or path mode
- `state`: the selected states
- `path`: the path that results from the selection
- `selectionText`: the text that should be shown in the menu
- `name`: a name of the selection
- `remarks`: the remarks belonging to the selection

The main methods of the `stateSelection` class are

- get and set functions for the variables
- equality check functions to compare a selection to other selections (used to check whether a particular visualisation of the simulation is already open or should be created)
- copy functionality, so that particular selections can be stored in a model

The simulation class has the following variables:

- `name`: the name of the simulation
- `remarks`: the remarks added to the simulation
- `currentSelection`: the current selection in the state graph
- `selections`: the set of stored selections in the state graph
- `engineState`: the representation of the simulation in the reasoning engine
- `exportedElements`: a hash table with references from the reasoning engine representation to elements in model fragments, scenarios and expressions
- `exportedObjects`: a hash table with references from the reasoning engine representation to the definitions of model ingredients in the model
- `scenario`: a link to the scenario that the simulation is based on
- `model`: a link to the model that the simulation is based on

The main methods of the `stateSelection` class are:

- get and set methods for some of the variables
- methods to store and delete state selections
- methods to load and save the engine representation (used when switching between simulations)
- methods to store references from the reasoning engine representation to the model, and methods to retrieve model ingredients from the model based on the engine representation

Simulations are saved by copying a simulation object to the `savedSimulations` slot in the model object.

11. Meta-data

The main reasons for adding metadata are

- Managing model versions:
- Embedding the model in its scientific or educational context. In scientific contexts: linking the model to researchers, research articles, and research projects. In educational contexts: linking the model to a course, educational program, or course assignment, mentioning the reviewer, etc.
- Referring to models: authors, year of publication, and other bibliographic citation information.
- Documenting known flaws and bugs.

The DynaLearn ILE offers four meta-data editors. In these editors, information can be stored that pertains to the model as a whole, but that does not involve the simulation process itself. Examples of such information categories are the title of the model, its authors, known bugs, the language in which the model is specified, etc.

11.1. Abstract and general remarks

The abstract and general remarks meta-data view embeds the model in its scientific or educational context (Figure 11.1). The fields are:

- Title
- Author
- Contributors
- Contact e-mail
- Keywords
- Domain
- Model version
- Known model limitation
- Bibliographic citation
- License

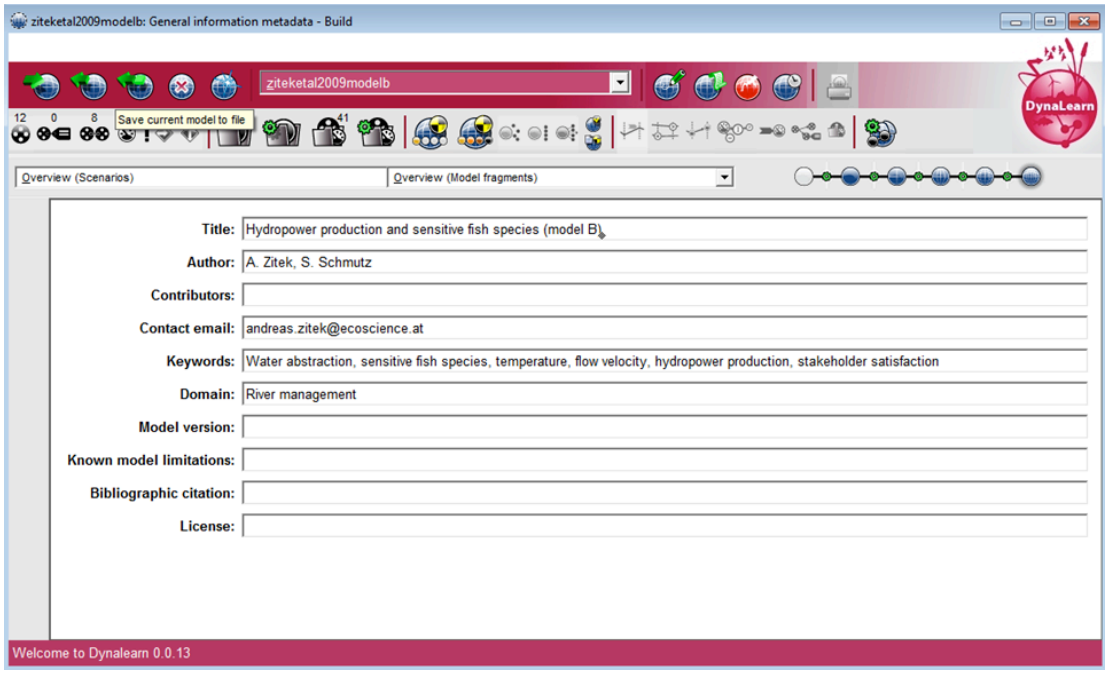


Figure 11.1: Meta-data, abstract and general remarks.

11.2. General information

This view shows general information about the model. See Figure 11.2. There are four categories:

- Abstract information
- Intended audience
- Model goals
- General remarks

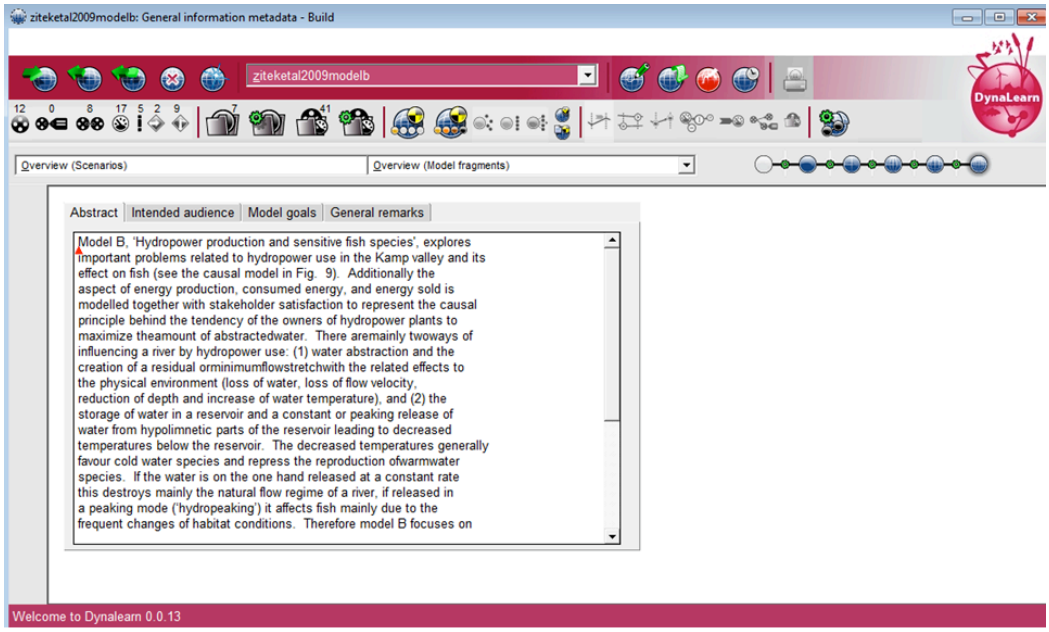


Figure 11.2: Meta-data, general information.

11.3. Status and bug report

The purpose of this view is to add updated status reports about the model in order to keep track of known problems and solving them. See Figure 11.3. A single model can have an arbitrary number of such reports, identifiable due to unique report IDs. The fields are:

- Report ID
- Name
- Priority
- Date
- Reported by
- Status
- To do
- Fixes
- Description

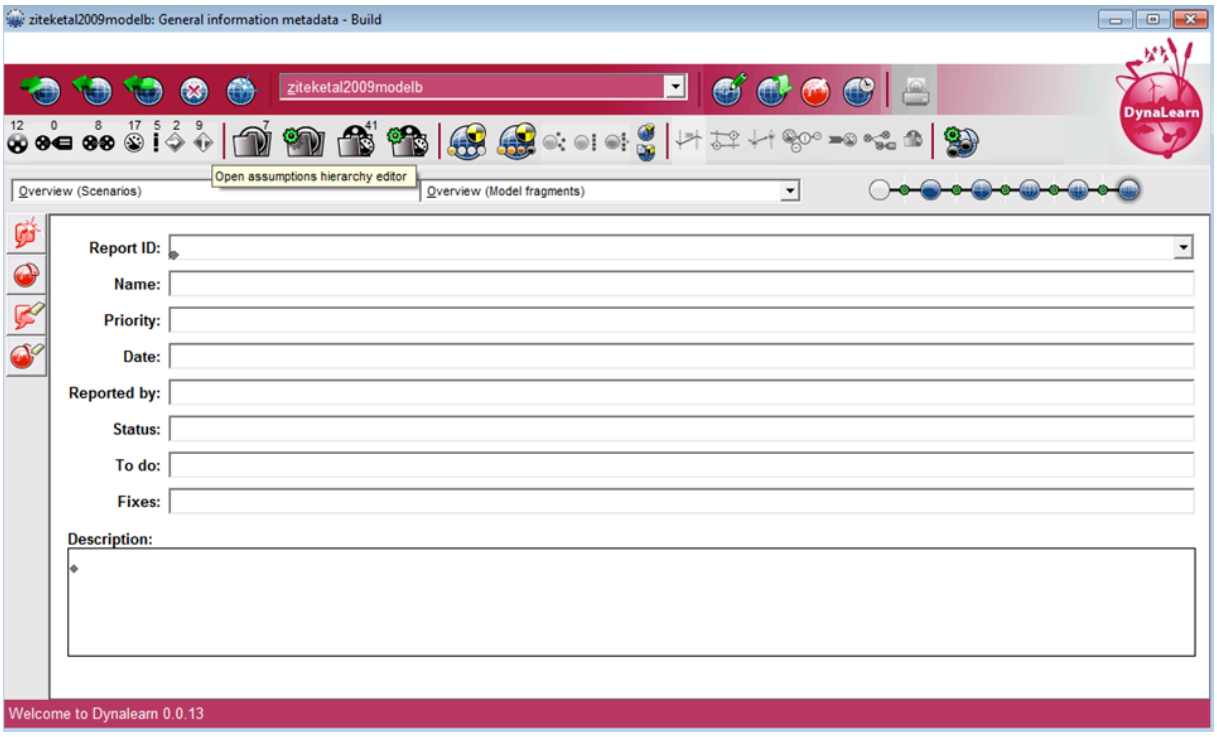


Figure 11.3: Meta-data, status and bug report.

11.4. Model data

The purpose of this view is to add version and update information about the model. See Figure 11.4. The fields are:

- Creation date
- Created in
- Creation definition version
- Last change
- Current program
- Current definition version

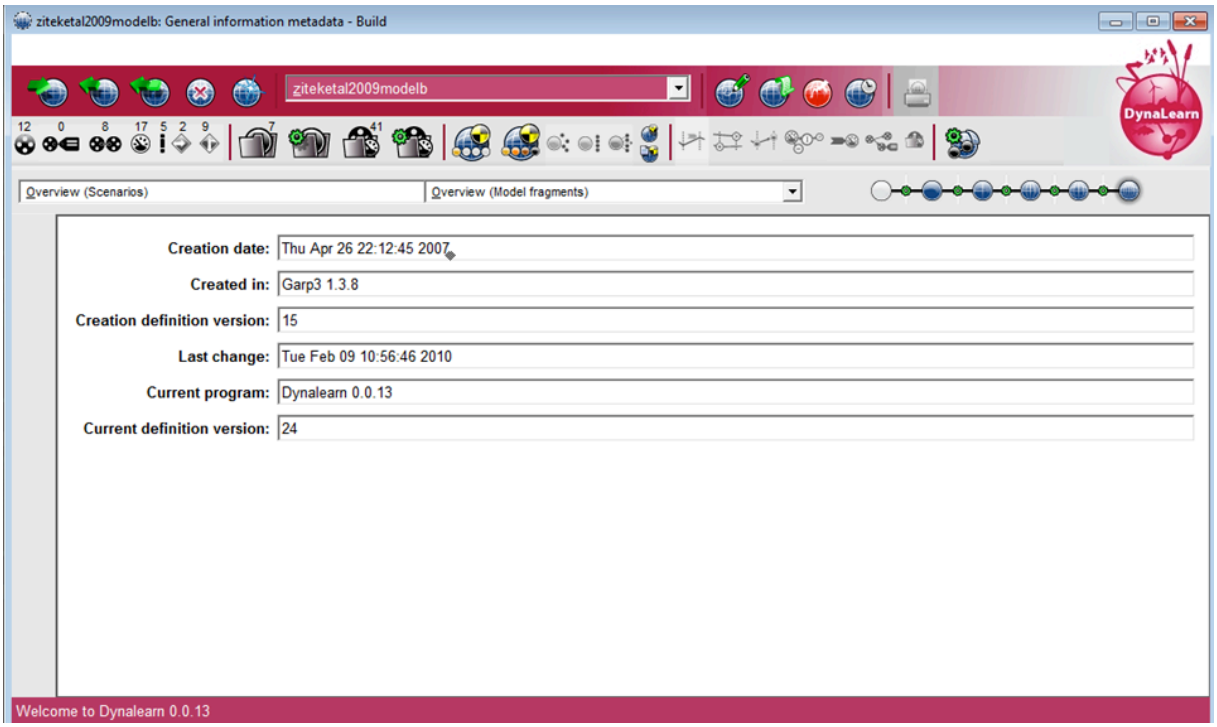


Figure 11.4: Meta-data, model data.

12. Special features

Over the years, Garp3 has gained multiple special features. These features have been incorporated in the DynaLearn software, and several other have been newly added to DynaLearn. Typical special features of DynaLearn are: multiple language support, OWL import/export, tooltips, Garp3 compatibility, EPS export, use level switching.

12.1. Multiple languages

Multiple language support allows models to be translated from one language into multiple others. This functionality is fully inherited from Garp3 and also works for the new ingredients introduced in DynaLearn.

12.2. OWL export/import

The Web Ontology Language (OWL) export and import functionality allows DynaLearn models to be exported as OWL files. This representation is used within the DynaLearn project to communicate models (and simulation results) between different components in the software. For example, the Semantic Repository stores the models in this format. The models in OWL format can also be imported back into the DynaLearn application again.

The OWL functionality has been adapted to a richer representation to capture the new knowledge representation used in DynaLearn. For example, information about use levels and new representation such as the + and – relationships are added to the implementation.

12.3. Tooltip – Basic explanation

In order to aid the user, tooltips have been added to the DynaLearn workbench in order to communicate information about the workings of the interface, as well as information regarding the specific model created by the user.

The former category, giving information regarding the interface, is provided whenever the user hovers over a UI element, such as a button. The tooltip text then communicates the functionality that this specific button allows. An example of this is given in Figure 12.1.

The latter category, giving information regarding aspects of the model, is provided whenever the user hovers over a model ingredient. The user can alter this text by editing the contents of the remarks text field inside the various editors. An example of this is given in Figure 12.2.

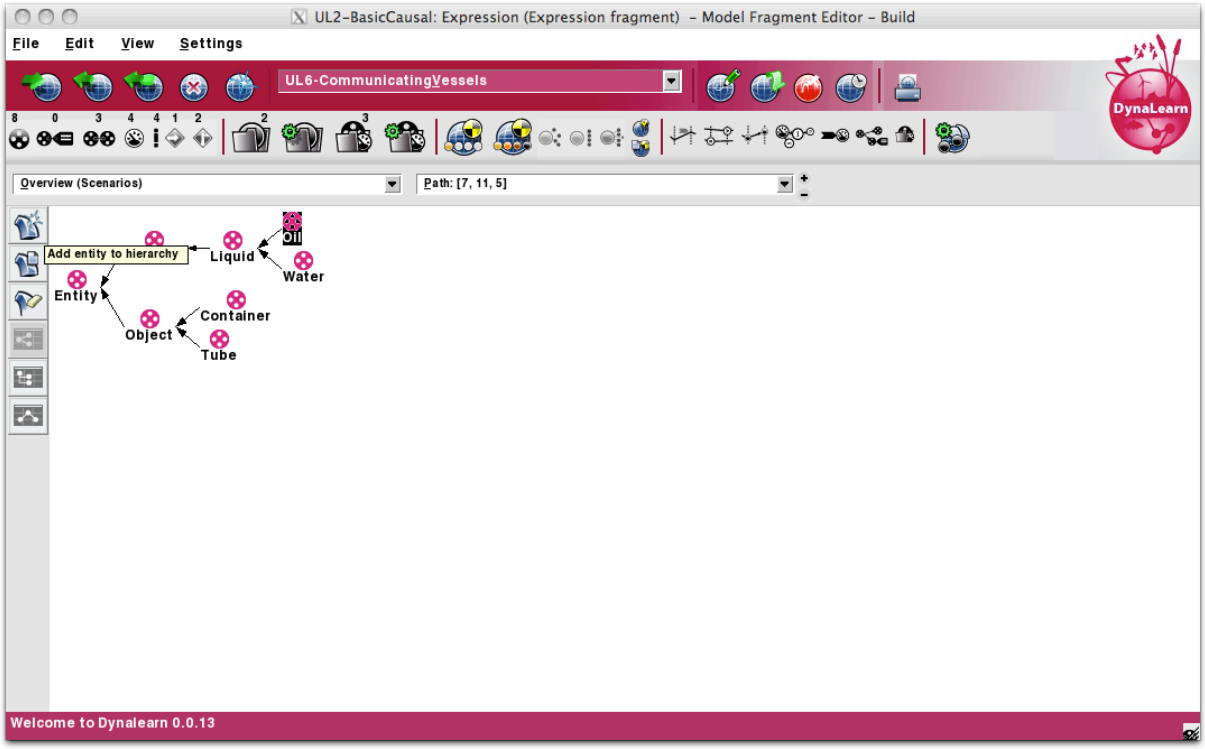


Figure 12.1: A tooltip for a button in DynaLearn.

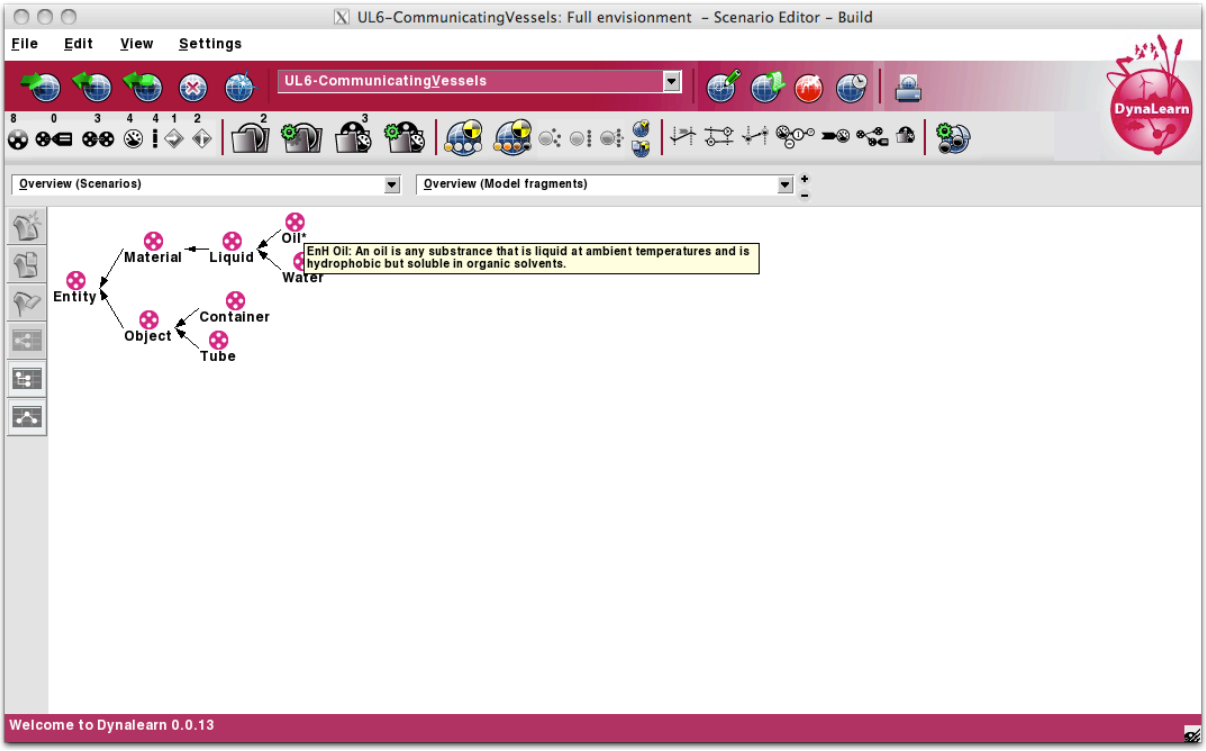


Figure 12.2: A tooltip for a model ingredient in DynaLearn.

12.4. Upward compatibility for Garp3 models at use level 6

During the implementation of DynaLearn we took particular care that Garp3 models can be loaded as DynaLearn models on use level 6. As such, no modelling effort has been lost. Furthermore, users of Garp3 can migrate to the new improved DynaLearn software.

12.5. EPS export

The diagrams created in the DynaLearn software can be exported as EPS images. These vector images can be used in presentations, reports and publications.

12.6. Switching use levels

In the near future, we envision the possibility of migrating a model to a higher use level. The concept map in use level 1 could be converted into entities and configurations on use level 2. From use level 2 to use level 3, quantity spaces are added to quantities. From use level 3 to use level 4, the – and + relationships can be converted into proportionalities, and influences. From use level 4 to use level 5, the possibility to create conditional model fragments is added. Migrating from use level 5 to use level 6 is quite complex, as use level 6 uses scenarios and model fragments, whereas use levels 2 through 5 use (conditional) expressions. A conversion algorithm has to be written to make these migrations possible.

We do not plan to add migrating models to lower use levels, since there seems no educational reason to have a student working on a higher use level use a lower use level with the same model. Technically it is also more difficult, since fewer ingredients may be used on lower use levels. Unused ingredients will have to be either converted or removed.

12.7. Screen cloning

In the Garp3 software, each workspace was shown in a separate window. Although this caused the usability problem of having too many windows open (which is resolved by the integrated user interface in DynaLearn), it did allow easy comparison of different diagrams or viewing multiple diagrams at the same time. Typical use-cases are looking at the state graph and the value history simultaneously, or looking at the differences between two dependency views of two states.

In DynaLearn, due to the integrated user interface, it has become more difficult to look at multiple diagrams simultaneously. As such, we added the possibility of showing a diagram in a separate window. This way, diagrams can still be easily compared.

12.8. Tabs in the simulate environment

To allow the learner to see which of the workspaces in the simulate environment are currently open, we plan to add tabs to the simulate workspace. This allows the learner to easily switch between different screens in the simulate environment. This is useful because particular workspaces are visualised for each state, such as the dependency view or the model fragments in state view. Quickly switching between different workspaces improves the usability of DynaLearn.

13. Implementation details

This Section describes some of the implementation details of how specific functionality has been implemented in DynaLearn. Section 13.1 describes how Change Requestors work. Section 13.2 describes how the application content is visualised. Section 13.3 describes how expressions and conditional expressions have been represented. Section 13.4 describes how expressions and conditional expressions are exported to the simulation engine representation.

13.1. Change Requestors

The DynaLearn ILE allows complex models to be built. Because of their inherent complexity, model ingredients can be made to depend on one another. For instance, after adding a relationship between two entities, neither one of the entities can be removed without affecting (1) the existence of the relationship, and (2) the behaviour of the entity that remains.

The interdependency of model ingredients poses a technical difficulty: how do we propagate changes made to a modelling ingredient? Because (1) there are many different interdependencies, and (2) those interdependencies can be nested arbitrarily deep, a thorough programming solution had to be invented to make the propagation of changes possible.

Also, some changes may have such intricate interdependencies that it should be disallowed to push certain changes through. For instance, the user is normally free to change the values of a quantity space. But once an inequality relation between two point values (belonging to the quantity spaces of different quantities) has been added, it is no longer allowed to change those values. The reason for this is that the previously instigated inequality may no longer hold for the changed value. Examples like these, which are numerous throughout the DynaLearn ILE, necessitate the incorporation of a method that checks whether a given change is allowed to be called or not. Moreover, in those cases in which a change is not allowed to be pushed though for a specific reason, that reason should be communicated to the user.

In order to encompass the above consideration, the change requestor was made (see also Figure 13.1). Change requestors are used for checking whether changes to a model are valid, for processing the changes, and for updating the workspaces that are influenced by the change. The change requestor objects can encapsulate complex changes. In order to be able to do this, the change requestor object needs the following attributes:

- An identifier for each `type` of change, in order to keep the many different changes apart.
- Each change is performed directly upon a single `object`.
- There may be (a lot of) additional objects that are affected indirectly by the change, i.e. the `route`.
- The different types of change requestors have different numbers and types of `arguments`. So these need to be stored as well.

- Since not all changes are allowed to be applied to all objects, specific `feedback` as to why this is so must be assembled.
- For each type, there are different methods for checking, applying, and handling changes. Therefore, the specific methods for these three operations are stored locally as `checkMethod`, `applyMethod`, and `appliedMethod`.
- Since the removal of a single modelling ingredient might have repercussions for other ingredients as well, the `autoFreeList` of objects that are to be freed after all changes have been applied is kept track of.

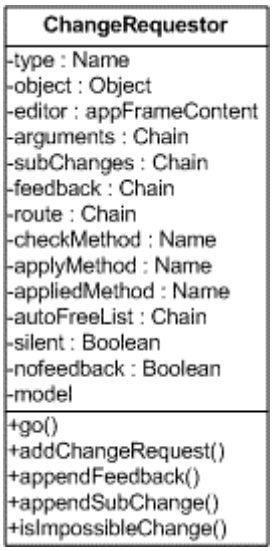


Figure 13.1: Class description for the change requestor

The algorithmic heart of the change requestor consists of the *go* method, and all subservient methods that are called by it (see Figure 13.2).

The `go` method orchestrates the `routeRequest` method that verifies whether the change can be pushed through and the `apply` method that applies the changes (if possible). The method's sole parameter is the content route to which the change will be applied. The content route consists of an arbitrary number of objects and/or chains containing objects. The `go` method is itself called by a model object's `changeRequest` method, where the content route is assembled and then passed through to the change requestor object.

The `routeRequest` method checks whether the changes can be applied to the objects in the content route. The method that is used for this verification purpose is stored as attribute `checkMethod`. Every modelling ingredient has its own check method (these are defined in the view editor object). The feedback from these check methods is assembled inside the `feedback` attribute of the change requestor object. The reason for assembling the various feedback messages, is that some of those individual feedbacks may block the changes, whereas the feedbacks taken in isolation would not. Whether the collation of feedbacks makes the changes impossible to perform, is decided over inside the `apply` method treated below.

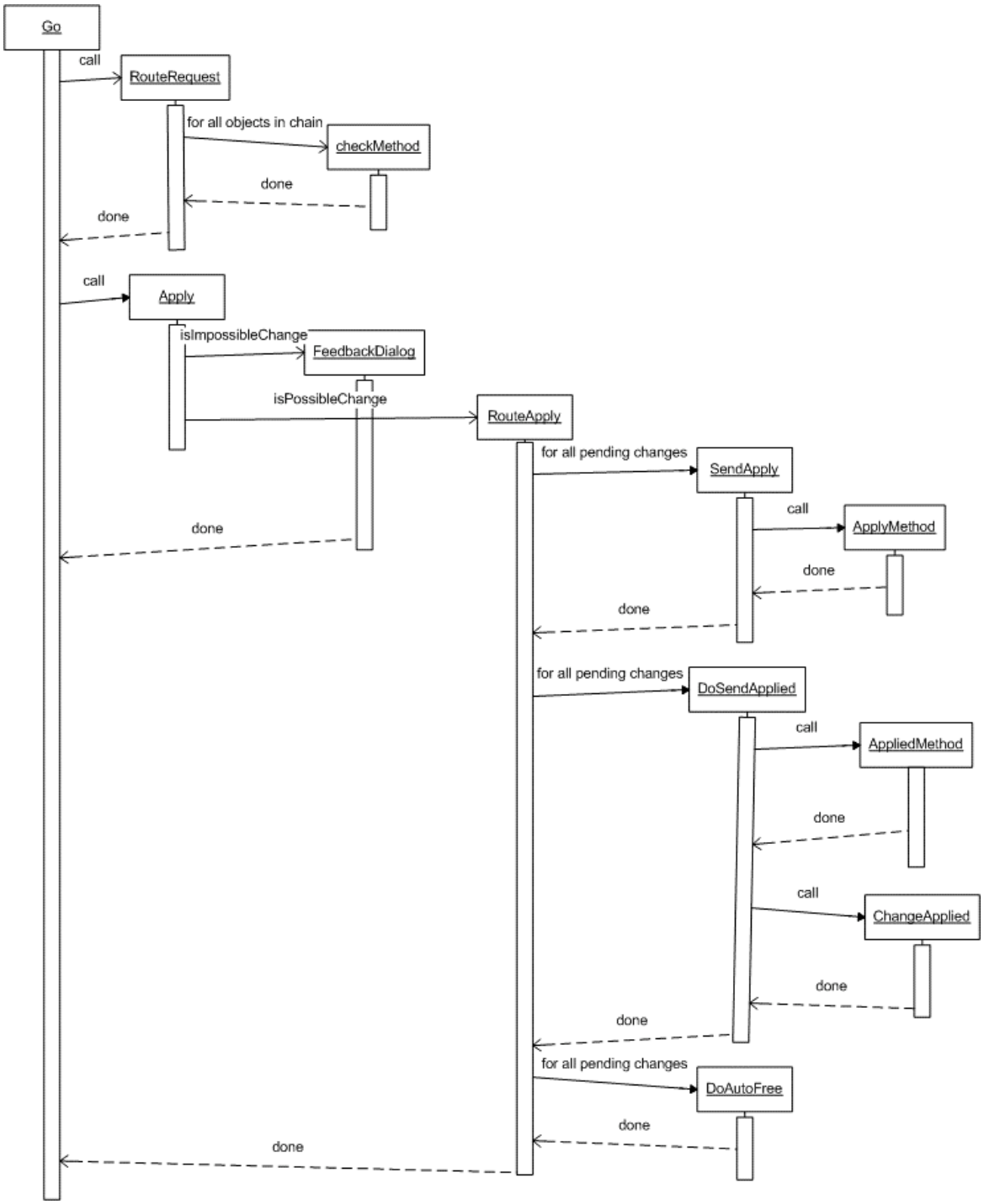


Figure 13.2: Process diagram for the change requestor.

After the `routeRequest` method succeeds, the `apply` method is called. First, this method uses `isImpossibleChange` in order to verify whether the changes can be applied, given the feedback that was assembled by the various check methods executed by the `routeRequest` method. If the changes are indeed deemed impossible with respect to the content route, then feedback is given as to why this is the case. If the changes are not blocked with respect to the content route, then the `routeAplly` method is called.

The `routeApply` method does three things:

1. It applies the changes to the objects that are within the content route. This is done by method `sendAplly`, which is run for each individual change.

2. `routeApply` calls method `doSendApplied`, again for every individual change. This performs any operation that may be required after all of the changes have been made (by `sendApply`).

3. All the changes are freed by method `doAutoFree`.

Method `sendApply` performs the change to a specific object. It calls the method that is locally stored as attribute `applyMethod` in order to perform this change.

Method `doSendApplied` performs two tasks:

1. It performs the locally stored attribute method `appliedMethod` onto all active contents (see section 13.2). This is necessary since there may be multiple views in which any of the changed modelling ingredients in displayed. All views containing the changed ingredient must therefore be updated after an update has been performed. `appliedMethod` makes sure these updates are performed.

2. The locally stored attribute method `changeApplied` is performed on all elements within the content route.

13.2. Application content

The DynaLearn ILE has a uniform application window in which the various views and editors for modelling and simulation are displayed. Whenever a new view or editor is chosen, not only the view content changes, but also the menu commands, the buttons on the left hand side of the application window, and the keyboard commands. Because so many elements of the interface are tied together with the notion of content switching, it becomes natural to think of interface views as modular objects.

But there is another reason for treating of user interface content switches as modular objects. It should be possible to clone any of the views within the DynaLearn ILE at any given time. This means that the user can click a button in order to transfer the currently displayed content to a separate window. But not only the content is transmitted; the menus, left hand side buttons, and keyboard commands should be transferred as well.

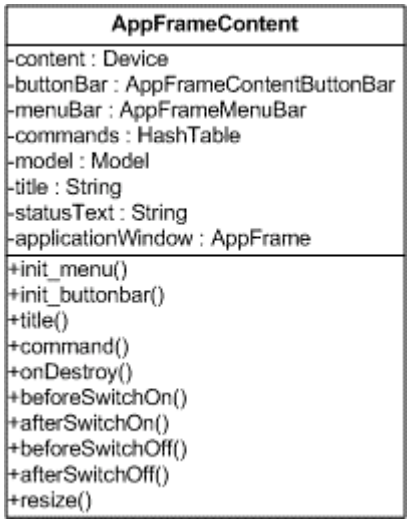


Figure 13.3: Class description of `AppFrameContent`.

For this purpose, the notion of application content (represented in class `AppFrameContent`) was created (see also Figure 13.3). Application content consists of the following elements:

- Attribute `client` contains the content that encompasses the current view. This is where the model ingredients are displayed.
- The menu for this content, `menuBar`.
- The left hand side button bar for this content, `buttonBar`.
- The `commands` that make the menu bar, button bar, and keyboard shortcuts all interact with the same content displayed on the `client` device.
- The `title` text that should be displayed within the title bar of the window in which the content will be displayed.

The menu bar and button bar are initialised by instantiating the methods `init_menu` and `init_buttonbar` respectively.

In order to provide the ability to make the switching between contents seamless, various standard methods are provided for preparing the switch to a new content, as well as methods for cleaning stuff up after an old content has been switched from. The former are `beforeSwitchOn` and `afterSwitchOn`. The latter are `beforeSwitchOff` and `afterSwitchOff`. In order to incorporate additional functionality for specific contents, these methods should be instantiated by the relevant instantiating classes (and a call to the super class is needed to incorporate the general mechanism as defined in `appFrameContent`).

Another important general method that applies to every piece of content in the DynaLearn workbench is `resize`. Because content is always embedded, either within the application window or within a cloned window, resize operations will never reach the content object directly. Instead, resize operations are put through to the content object via this method. The parameters are the delta in width and height (in that order) that resulted from the resize operation.

Now that we have described the structure and functionality of the `appFrameContent` object storing application content, we now turn to the way in which the application window brings about switches between various content objects.

In the class representing the application window, i.e. `appFrame`, there is a method `addContent`. It takes a content object (of type `appFrameContent`) as its parameter, and adds this to the internal content stack.

Because of cloning functionality, there may be an arbitrary number of contents that are all open at the same time, i.e. that are all part of the content stack.[8] The local attribute `activeContent` always points to the content object in the stack that is currently displayed (so that inner representation and outer presentation are always in sync).

In addition to adding content to the application, it is possible to remove content from it. This is done by the `removeContent` method. This method provided no content-specific handling mechanisms though. This means that all operations that need to be performed when content is removed from the stack should have been dealt with in that content's `beforeSwitchOff` and `afterSwitchOff` methods see above).

After content has been added to the stack, it needs to be displayed within the application window. This is done by the `switchToContent` method. It takes as argument a content that is already in the application content stack. This method first erases the currently displayed content from application window (not necessary erasing the old content object from the application content stack though), and then populates the new content's left hand side button bar, menu bar, client content, and title bar text (in that order).

13.3. Representation of (conditional) expressions

Figure 13.4 shows the class hierarchy starting from the model fragment class. The Static, Process and Agent fragments (in use level 6) are model fragments in which specific types of ingredients can be used, which were already available in Garp3. The learner can create complete hierarchies of model fragments below these model fragments. The Scenario class is used describes a start state within the system, and was also already available in Garp3.

New in DynaLearn is the concept of Expression, which merges aspects of scenarios and model fragments. That is, everything is a consequence in the sense that everything in an expression always applies, as is the case in scenarios. However, in an expression causal dependencies and correspondences can also be used, which normally only possible in other subclasses of model fragment. When performing a simulation, the expression is divided into a model fragment and a scenario part (Section 13.4). Similar to the scenario class, the expression class reuses code from the model fragment class.

[8] Because there can be an arbitrary number of views on the same model, the results of any changes that are made to the model must be propagated throughout the various views. This is done by incorporating the application stack in the handling of the change requestor. See section 13.2.

In use level 5 it becomes possible to specify conditional knowledge in addition to the expression. This knowledge is specified in conditional expressions. These conditional expressions incorporate the expression, but allow new conditions and consequences to be added. For example to indicate that the boiling process is only active when the water temperature is bigger or equal to the boiling point.

In the implementation, the conditional expressions are subclasses of the Agent Fragment (although this fact is hidden from the learner). This allows learners to use the complete spectrum of ingredients in them. The Expression is imported in these Conditional Expressions in the same way a model fragment is incorporated into another model fragment.

When a model fragment in incorporated the colour coding makes the imported model fragment content black (and the model fragment reference red). Since we prefer the colour coding for inherited model fragments, a Hidden Expression is introduced between the Agent Fragment and the Conditional Expressions (see Figure 13.4). The Hidden Expression incorporated the Expression, and the Conditional Expressions become subclasses of this Hidden Expression. This ensures the colour coding is correct.

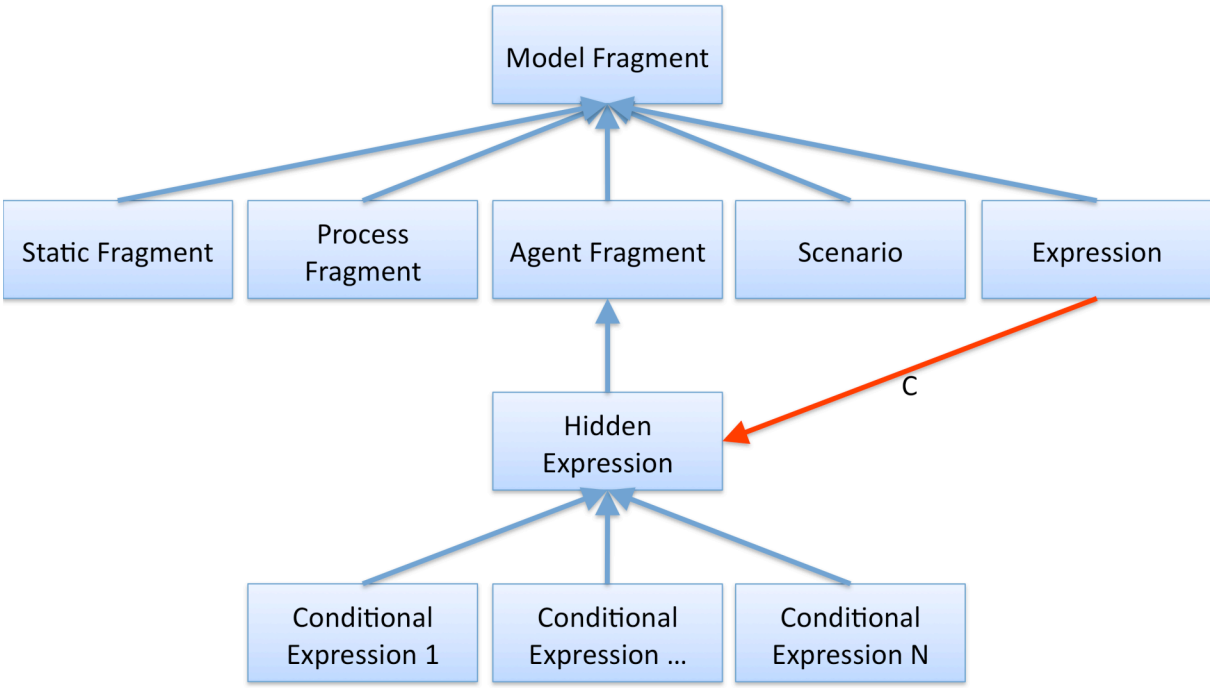


Figure 13.4: The class hierarchy of model fragments, expressions and scenarios.

13.4. Export to reasoning engine

Since an Expression (Figure 13.4) is incorporates both aspects of a scenario and of a model fragment, the reasoning engine in DynaLearn cannot run simulations with them without modification. In Garp3 there would always be a scenario (a particular start situation of the simulation) and a set of model fragments (general knowledge that applies in certain situations). Garp3 checks which model fragments applies to the scenario, introduces the knowledge in those model fragments to the scenario, and uses this knowledge to predict the next states of behaviour.

In DynaLearn (in use level 2 - 5) there is only an expression (and conditional expressions in use level 5). As such, to be able to use the reasoning engine from Garp3, the expression has to be divided into a scenario and a model fragment part. Since the reasoning engine uses its own internal representation, we have decided to update the export functionality that writes the model to the reasoning engine representation.

13.4.1. Expression export to model fragment and scenario part

The following lists enumerate which ingredients are exported to the scenario part of the expression and which parts are exported to the model fragment part. The ingredients are colour-coded. Blue indicates that the ingredients are exported as consequences, while red means that they are exported as conditions. By making the structure (entities, agents, assumptions, configurations and attributes) conditional in the model fragment and consequential in the scenario we make sure that the model fragment part always fires on the scenario part.

Scenario:

- Entities
- Agents
- Assumptions
- Configurations
- Attributes
- Quantities
- Value Assignments
- Inequalities

Correspondences and causal dependencies are not exported to the scenario.

Model fragment:

- Entities
- Agents
- Assumptions
- Configurations
- Attributes
- Quantities
- Causal dependencies
- Correspondences

13.4.2. Altered export functions in model fragment class

Some of the functions that export a model to the reasoning engine representation had to be adapted to deal with expressions.

- The `exportRelations` function exports correspondences, inequalities, causal dependencies and operators (as a result of exporting inequalities). The function was adapted to export the inequalities to the scenario part of the expression, and the correspondences and causal dependencies to the model fragment part of the expression.
- The `exportSystemElements` function exports entities, agents, assumptions, configurations and attributes. This function was adapted to export all of these ingredients as conditions to the model fragment part of the expression, and also export all of these ingredients as consequences to the scenario part of the expression. This makes sure that the model fragment part will always match the scenario part of the expression.
- The `exportValues` function exports the value assignments. This function is only called to export the value assignments to the scenario.
- The `exportParameters` function exports the quantities. This function is used to export the quantities as consequences to both the scenario and the model fragment part of the expression.
- The `exportSystemStructures` function exports model fragments inherited or imported into a model fragment. This function is not called for expression, since to model fragments can be imported in it, nor does it have a parent from which content is inherited.

Value assignments and inequalities are not exported to the model fragment part.

13.4.3. Exogenous behaviour constant for derivative value assignments

Due to the constrained vocabulary on the lower use levels in DynaLearn (particularly on use level 3), it is not possible to derive the derivatives of quantities using the causal relationships. On use level 2 and use level 3 influences (the cause of change within a model) cannot be used, but only + and – relationships (which are proportionalities in terms of the reasoning engine). As such, using the default reasoning as done in Garp3 the derivative values set in the expression are forgotten in the successor states of the first state (as they would have to be recalculated using causal dependencies which are not available yet).

To resolve this issue, the value assignments in expressions (i.e. on use levels 2 through 5) are considered constant. The exogenous behaviour constant (as can be used in scenarios in Garp3) is added to each derivative value assignment in the expression. This assures that the derivatives maintain their value during the complete simulation. Notice, that the other exogenous behaviours available in Garp3 can only be used in DynaLearn at use level 6.

14. Conclusion

The DynaLearn Interactive Learning Environment (ILE) for constructing *conceptual* knowledge has been successfully completed. The main contributions of this software include:

- Multiple use levels
- Integrated interface ('Single' workspace)
- Support for multiple simulations
- Saved state and path selections, and saved simulation support
- Upward compatible for Garp3 models (open in use level 6)
- Entity on/off preference (in use level 1 - 5)
- Alphabetic order to simplify selection of:
    - Model fragments
    - Scenarios

The use levels allow teachers and learners to work on different levels of complexity, as well as to focus on particular representational features in order to highlight and investigate specific qualitative aspects of systems behaviour.

Having all the interactive windows integrated in a single workspace ('single screen') significantly enhances the usability of the overall workbench.

Being able to run multiple simulations within a single model (albeit for different scenarios) and save these simulation results in the model is convenient for users as it opens new possibilities for teachers and learners to share their work within the community.

The software is available via the DynaLearn website (http://www.DynaLearn.eu).

15. Discussion

Further improvements of the DynaLearn software depend on the availability of resources and requirements put forward by the user community, particularly within the DynaLearn project. Additional features could include:

- Integration of the OWL import/export functionality
- Copy/paste functionality for the new and improved data structures
- Switching between use levels and support 'upgrading' of an expression to the next use level
- Display the contents from different views side-by-side (screen cloning)
- Provide tabs to store views on simulation results and be able to quickly switch between these

In addition to implementing further details, it is expected that users of the software will provide feedback on software bugs, if any. Following this, the stable release of the software, and its current list of features, is expected within a few months after the release of the beta version. Once the stable version of the DynaLearn software is available, Garp3 (http://www.Garp3.org) users may consider moving to the DynaLearn software (http://www.DynaLearn.eu) due to its enhanced usability and additional features.

16. References

[1] Bredeweg, B. and Salles, P. Mediating conceptual knowledge using qualitative reasoning. 2009. In: Jørgensen, S.V., Chon, T-S., Recknagel, F.A. (Eds.), Handbook of Ecological Modelling and Informatics. Wit Press, Southampton, UK, pp. 351–398.

[2] Bredeweg, B., Linnebank, F., Bouwer, A. and Liem, J. 2009. Garp3 — Workbench for qualitative modelling and simulation. *Ecological Informatics*, 4(5-6), pp. 263-28.

[3] Bredeweg, B. (ed.), André, E., Bee, N., Bühling, R., Gómez-Pérez, J.M., Häring, M., Liem, J., Linnebank, F., Thanh Tu Nguyen, B., Trna, M. and Wißner, M. 2009. Technical design and architecture, DynaLearn, EC FP7, STREP Project no. 231526, Deliverable D2.1.

[5] Forbus, K.D, Carney, K., Harris, R. and Sherin, B.L. 2001. A qualitative modeling environment for middle-school students: A progress report. The 15th International Workshop on Qualitative Reasoning, San Antonio, Texas, 17-18 May

[6] Leelawong, K. and Biswas, G. 2008. Designing Learning by Teaching Agents: The Betty's Brain System. *Artificial Intelligence in Education*, 18(3), pp. 181-208.

[7] Novak, J.D. and Gowin, D.B. 1984. Learning How to Learn. Cambridge University Press, New York, New York.J.D.

17. Appendix A – Interface design (intermediate from D2.1 to D3.1)

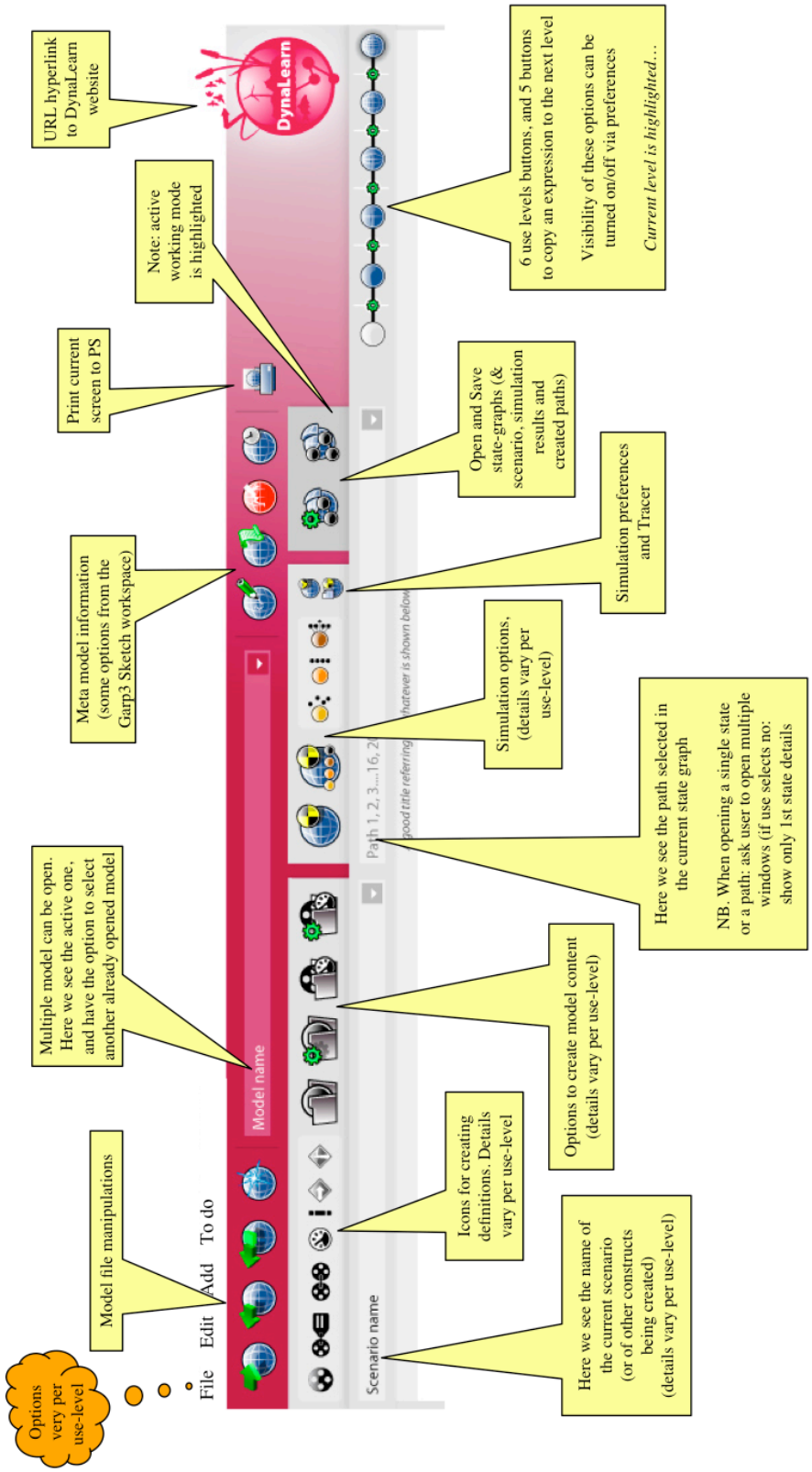


Figure A.1: Design main interface

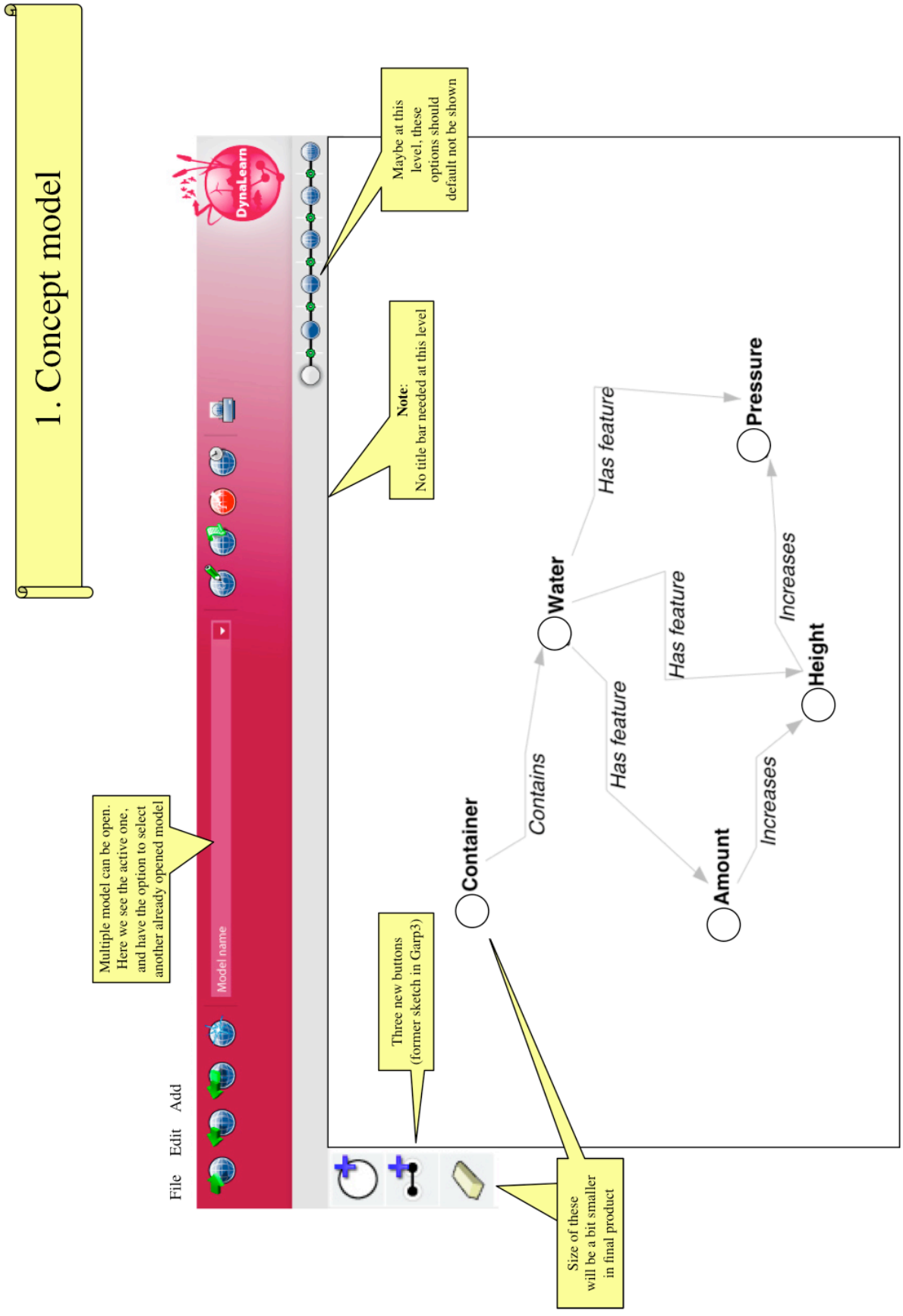


Figure A.2: Use level 1 - Build

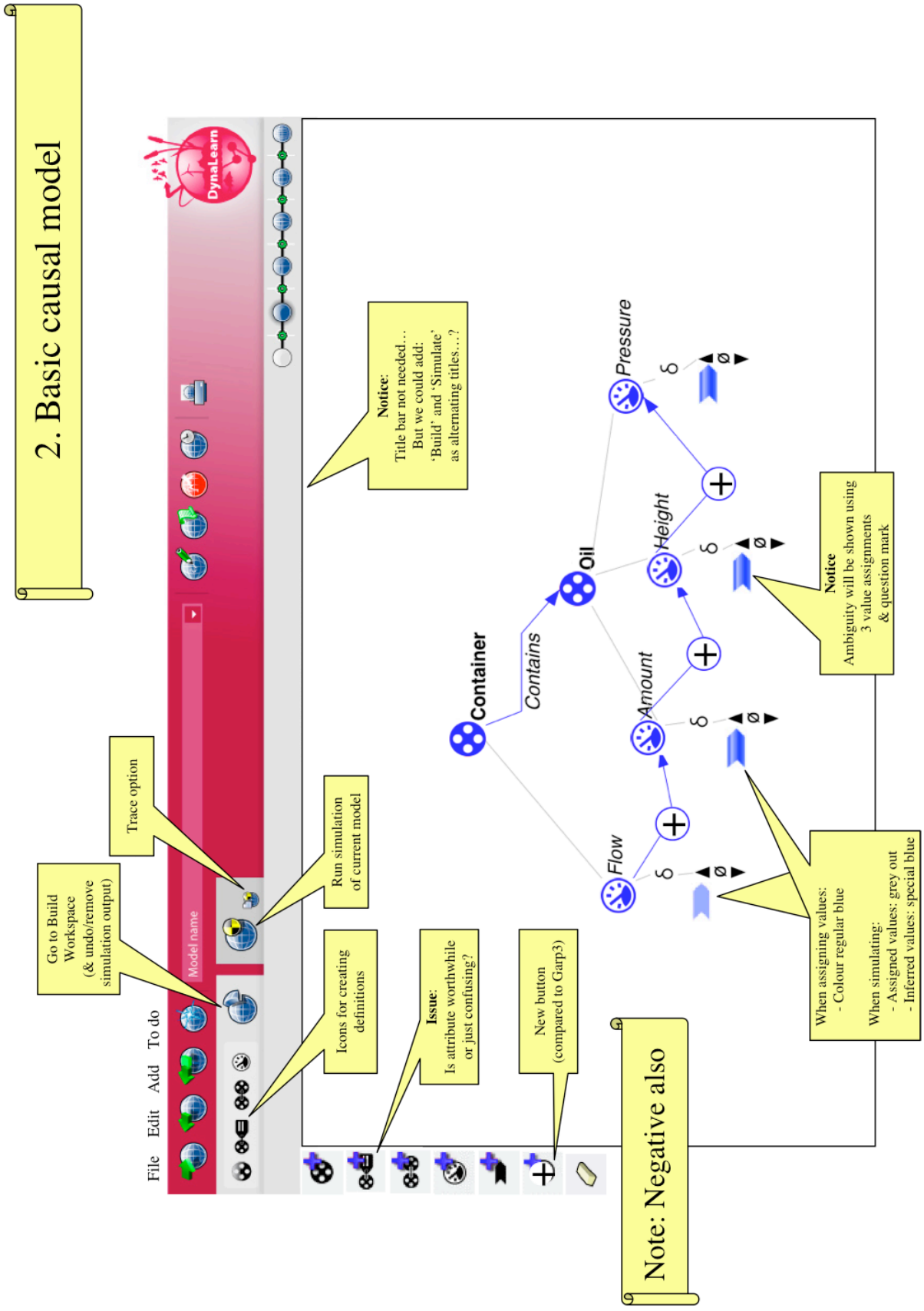


Figure A.3: Use level 2 - Build

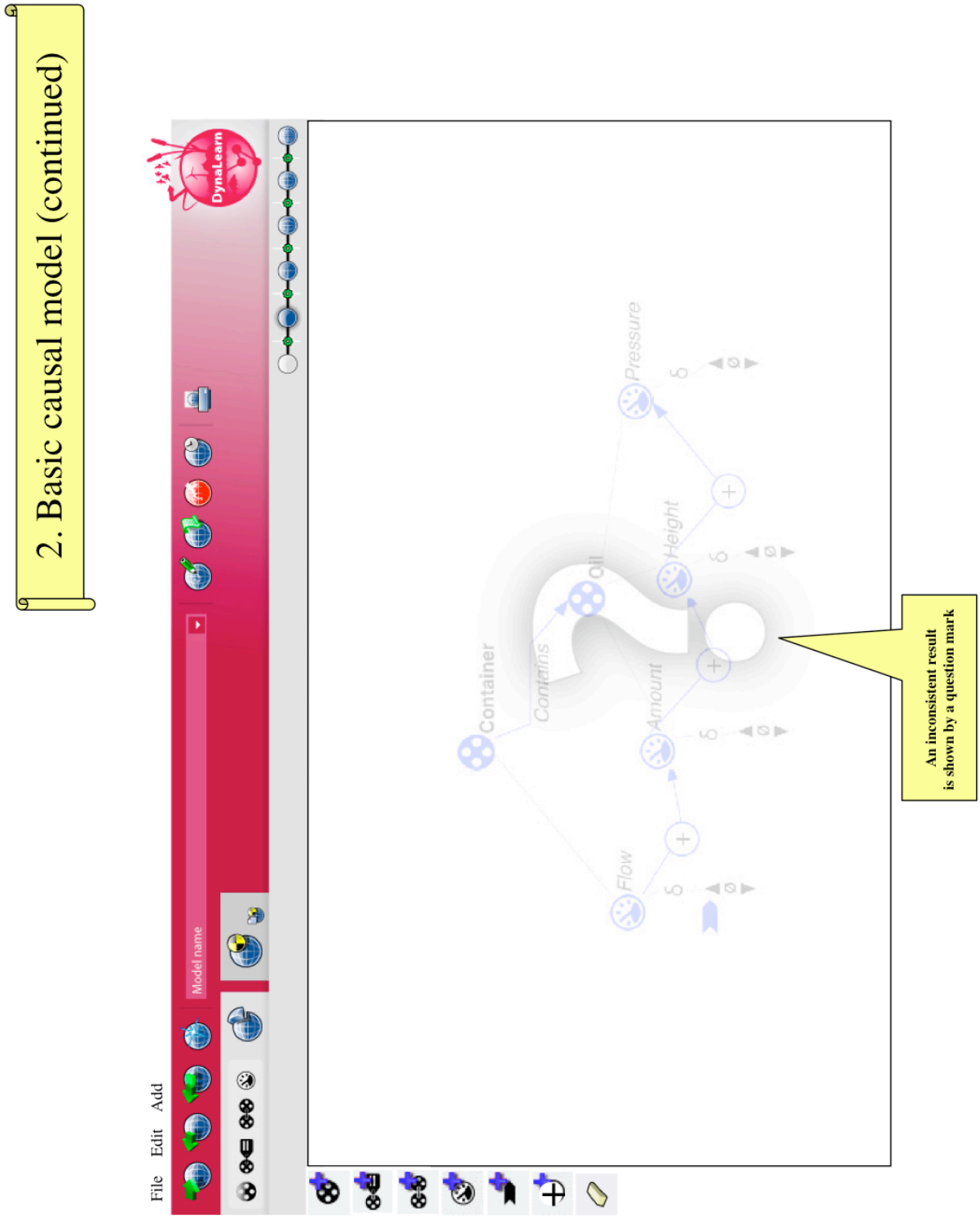


Figure A.4: Use level 2 - Simulate showing inconsistent result

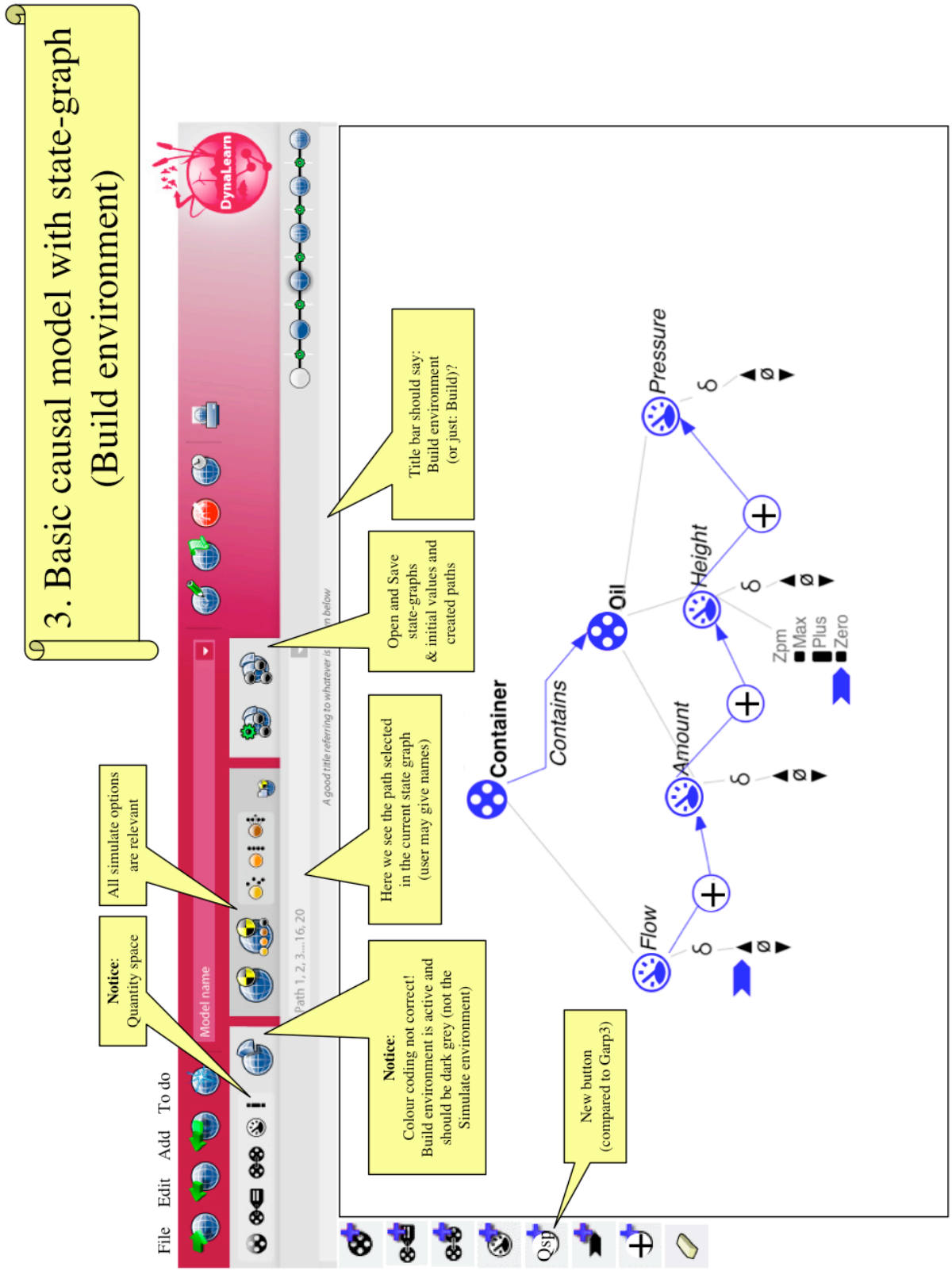


Figure A.5: Use level 3 - Build

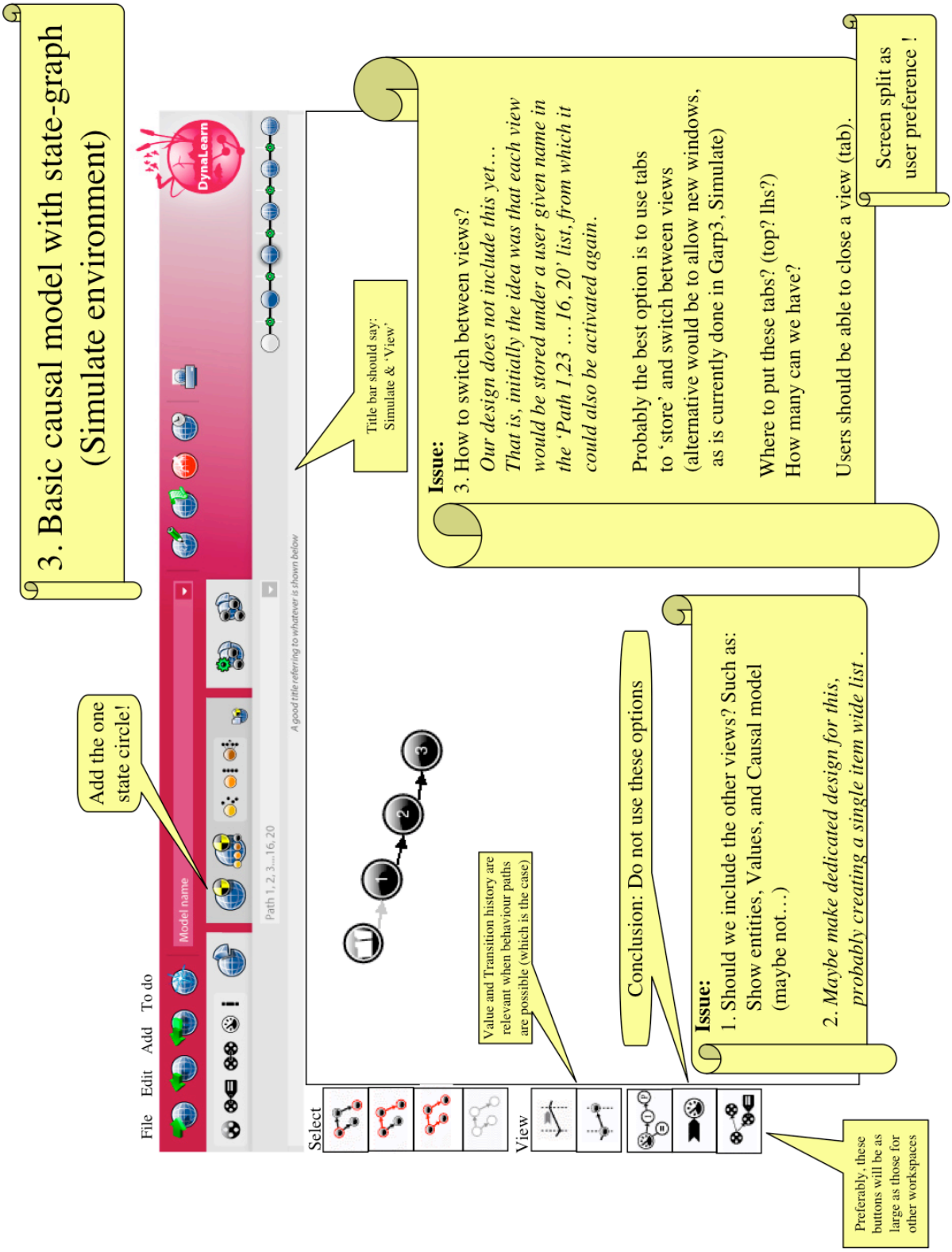


Figure A.6: Use level 3 - Simulate showing State graph

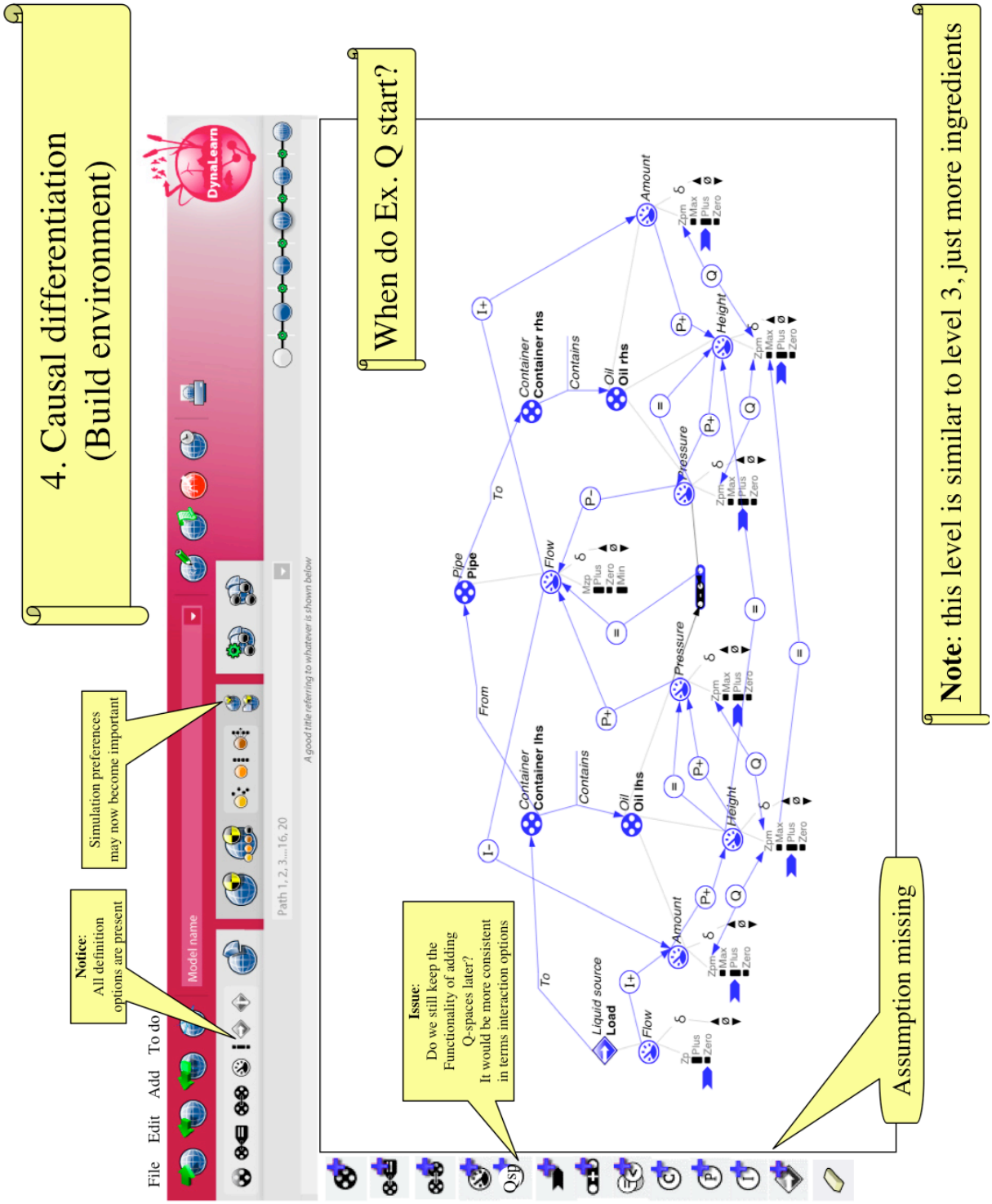


Figure A.7: Use level 4 - Build

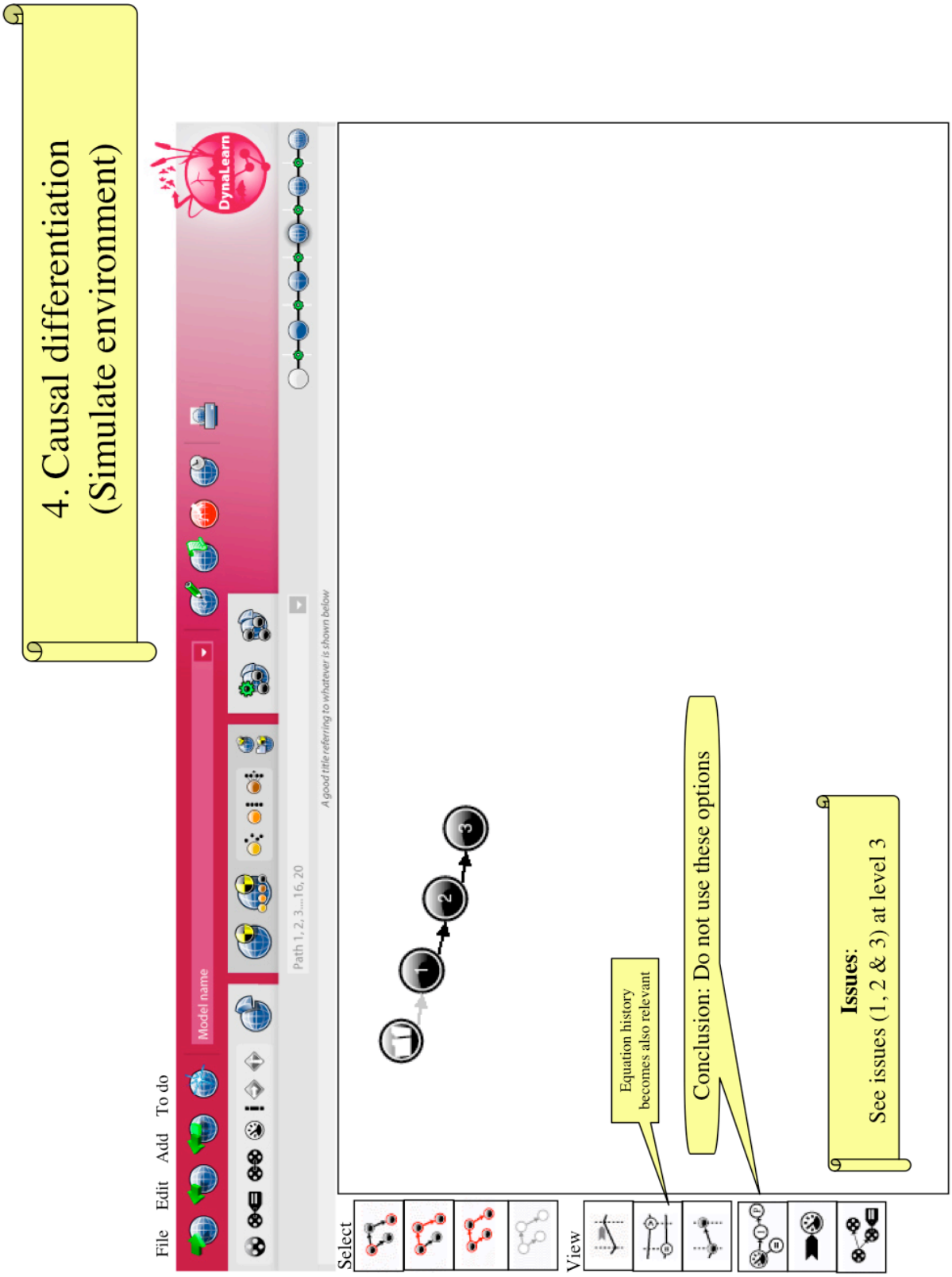


Figure A.8: Use level 4 - Simulate showing State graph

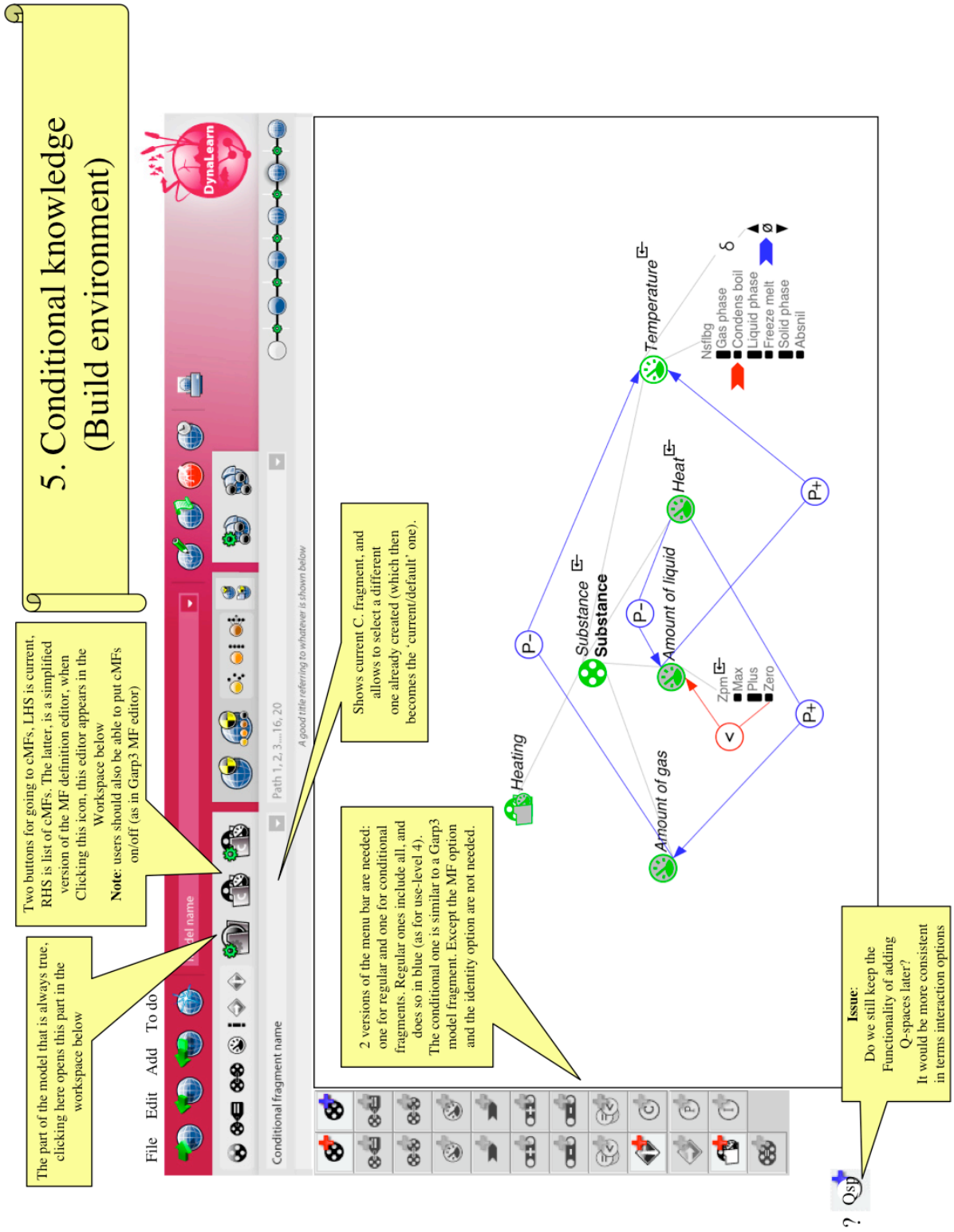


Figure A.9: Use level 5 - Build showing Conditional expression

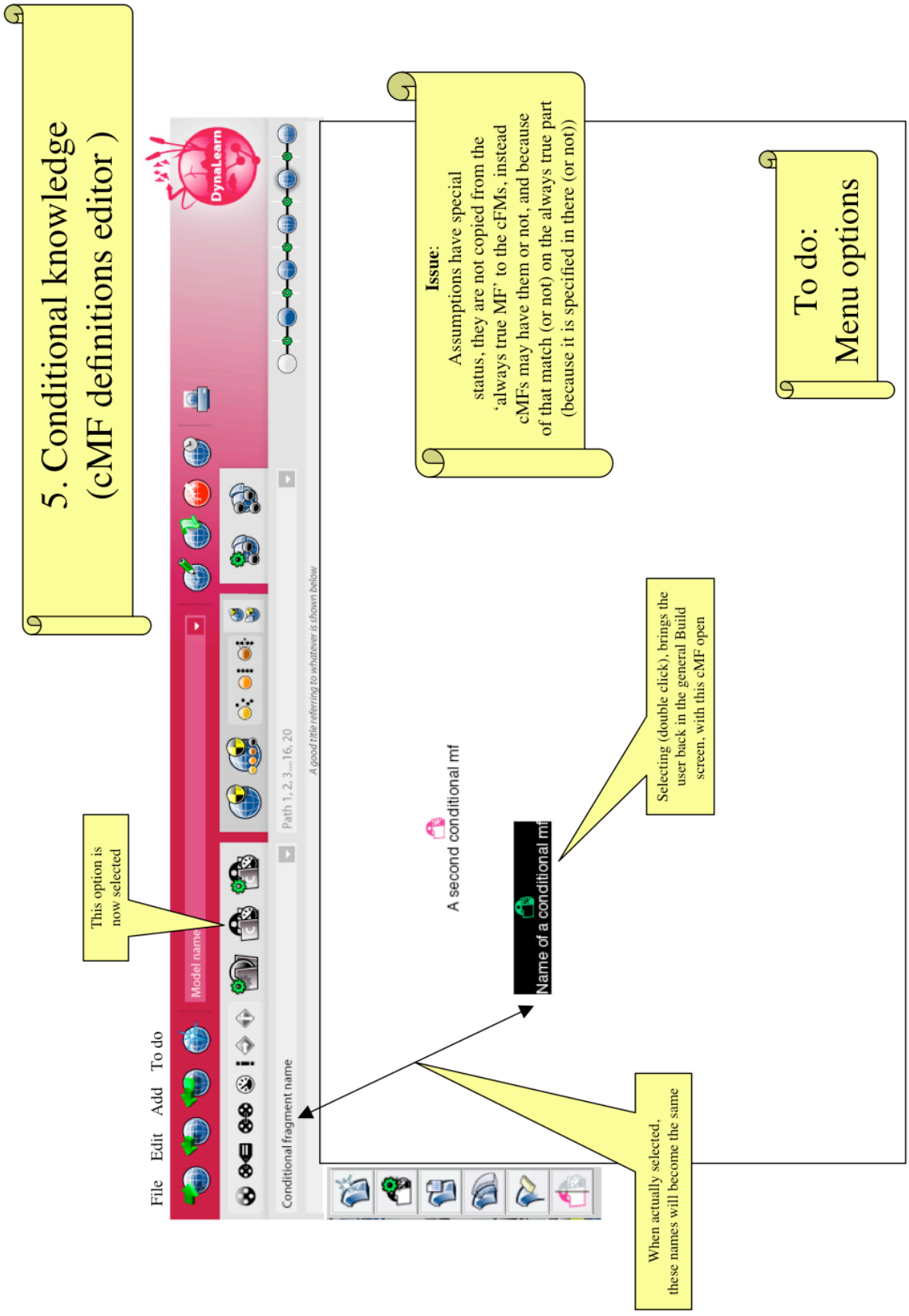


Figure A.10: Use level 5 - Simulate showing list of Condition expressions

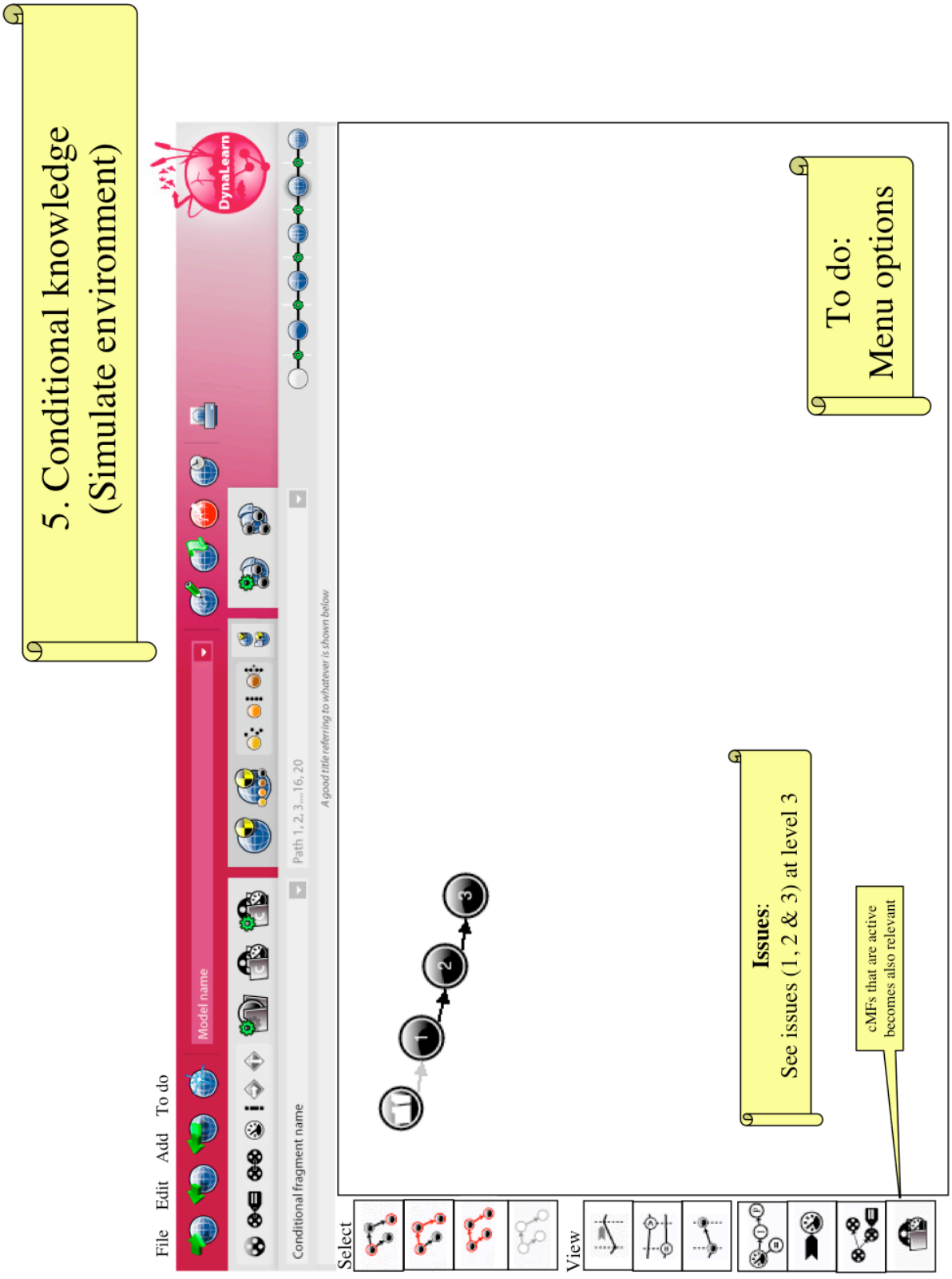


Figure A.11: Use level 5 - Simulate showing State graph

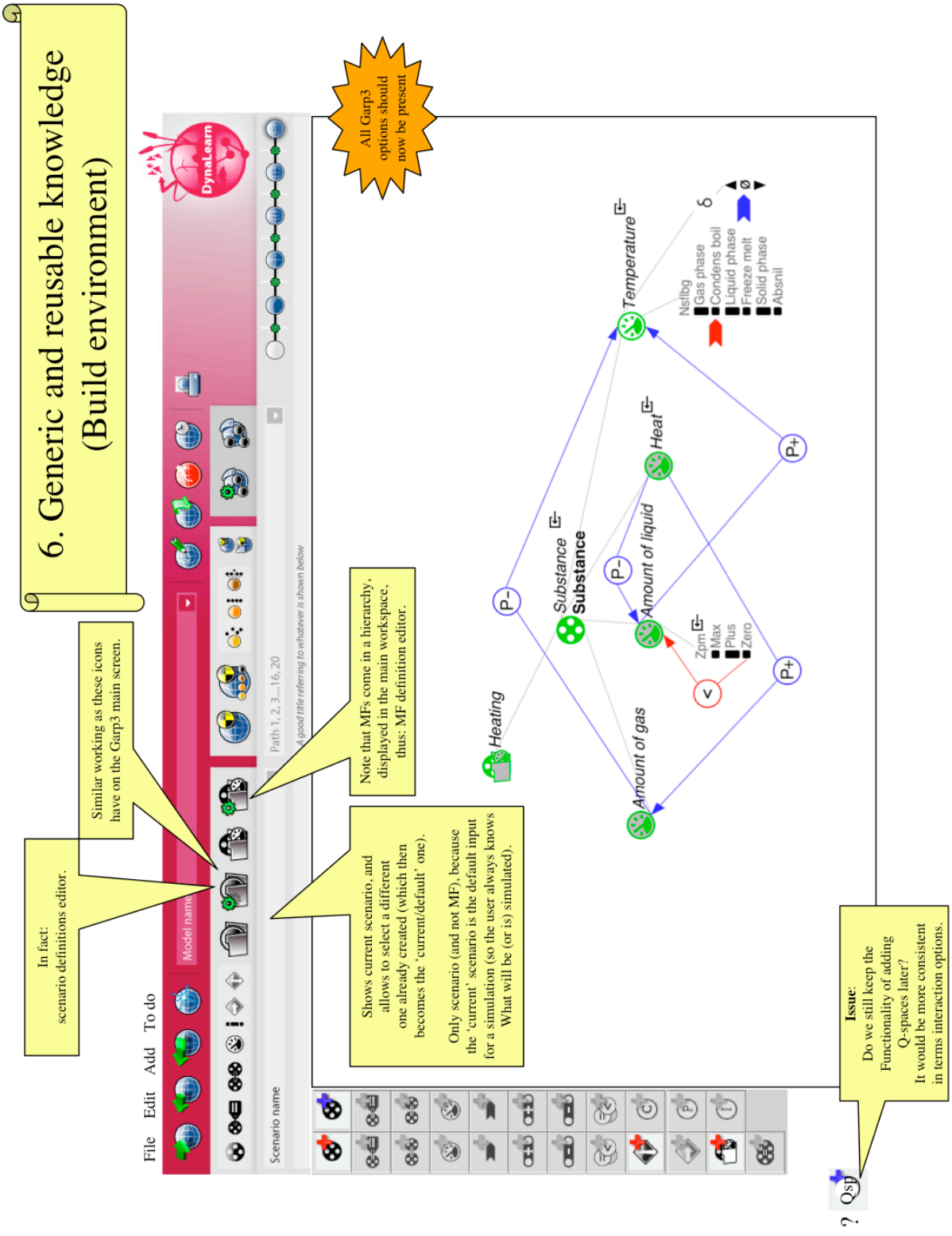


Figure A.12: Use level 6 - Build showing Model fragment (in fact a subtype of MF Heating)

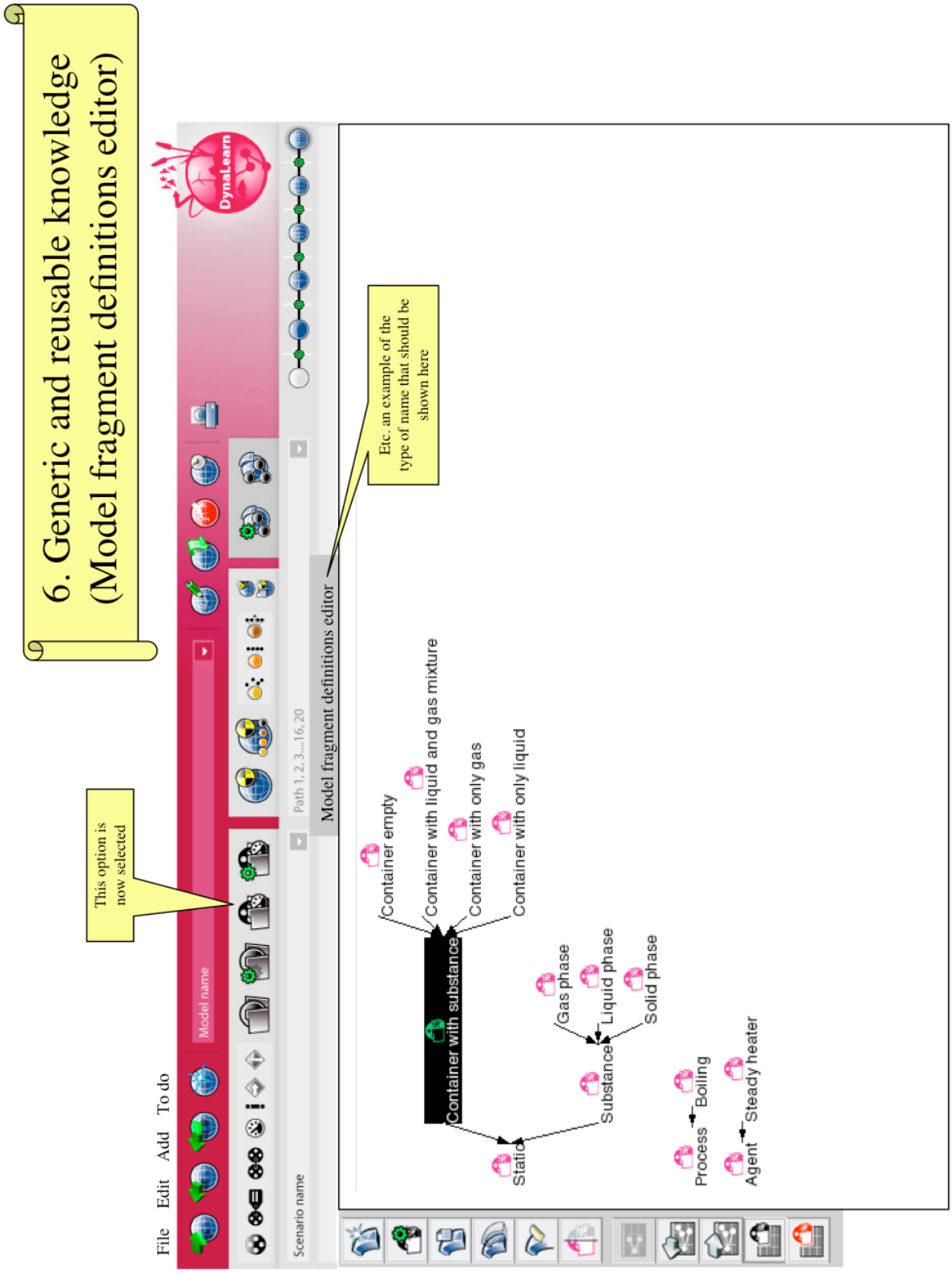


Figure A.13: Use level 6 - Build showing list of Model fragments

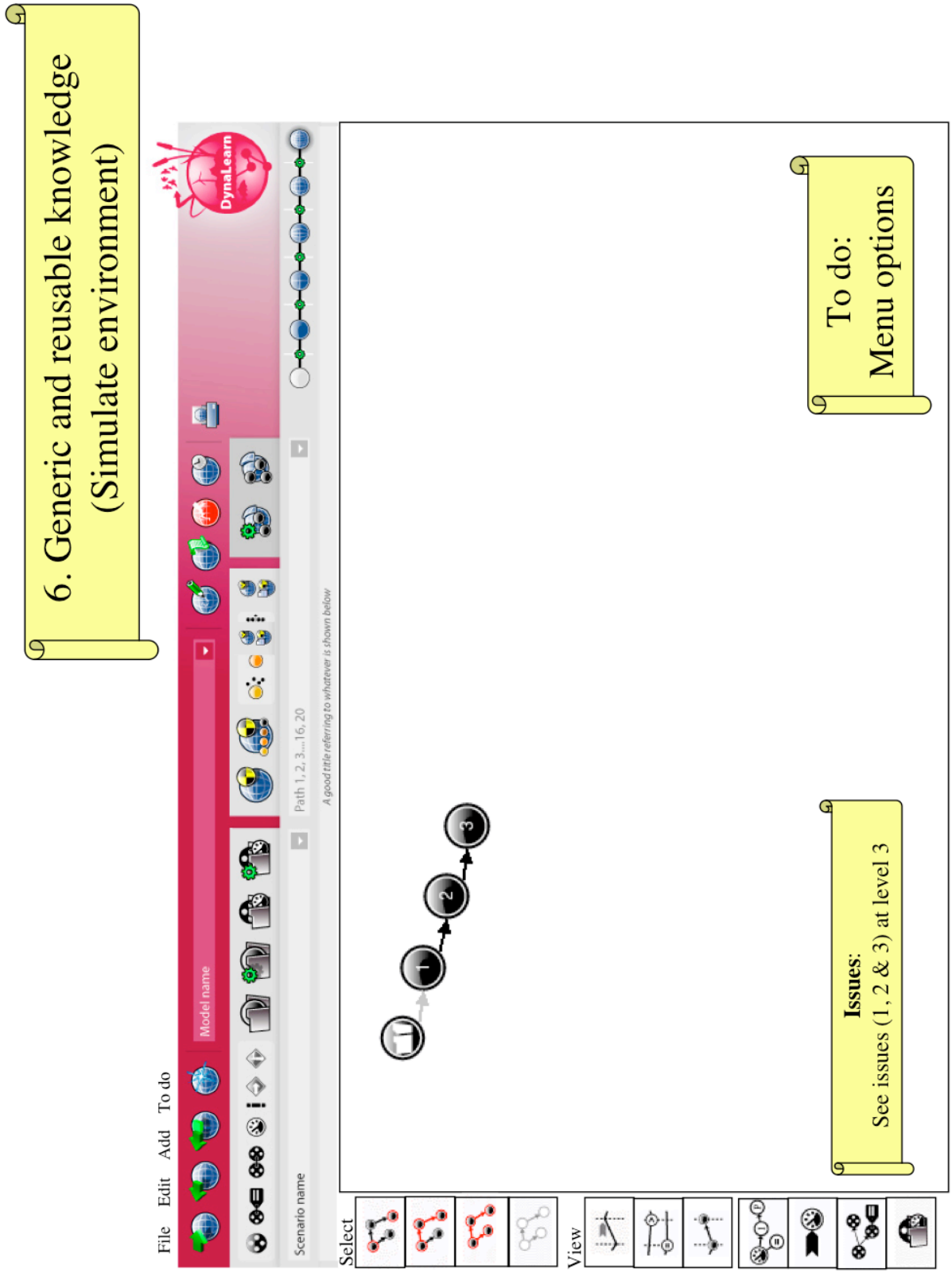


Figure A.14: Use level 6 - Simulate showing State graph

18. Appendix B – From Garp3 to DynaLearn workspaces

Below a detailed account is given of how Garp3 functionality is migrated, removed, merged or hidden at each of use levels in the DynaLearn ILE.

18.1. Main activities

- Use level 6
  - Build workspaces
    - Showing all the buttons and ingredients
    - Providing user interaction
  - Simulate workspaces:
    - Showing all the buttons and ingredients
    - Providing user interaction
  - Miscellaneous
    - Greyed icons / background colour button bar (top main screen)
    - Lists (Scenario / Model fragments, Paths + Saved simulations)
    - Pull down menus
    - Use levels as preference
    - Tooltips text
    - Meta-data
    - Trace window
    - Tab
    - Clone
- Use level 5
  - Integrating ‘definition’ with ‘adding’ dialogues (level 2, 3 and 4 use the same)
  - Filter
    - Entity / Agent / Assumption
    - Always true model fragment
    - Conditional model fragments
- Use level 4 (subset of level 5)
  - Filter
    - Only always true model fragment
    - Subset of level 5 simulation results options
- Use level 3 (subset of level 4)
  - Allowing quantity spaces to be added after a quantity is created
  - Filter
    - Only always true model fragment (subset of level 4)
    - Selection of level 5 ingredients
- Use level 2 (subset of level 3, no quantity spaces)
  - Filter
    - No quantity spaces
    - Simulation results (derivative values) in Build workspace
      - Derivatives
      - Show Ambiguous / Inconsistent simulation results
- Use level 1
  - Filter
    - Only Entities and configurations
    - Different look in terms of icons used

18.2. Architecture

- appframe
- generic
  - control
  - helpers
  - commands
- build
  - definition
  - elements
  - workspace
- simulate
  - definition
  - elements
  - workspace
- sketch
  - definition
  - elements
  - workspace
- meta
- icons

18.3. General

- New and adjusted icons
  - Adjusted icons have been created (We will include them in the current version of the software).
  - New icons are now available for: delete model (close model), plus/min for paths, simulate single state (We will include them in icons/appframe so that they can be used).
- Greyed icons are now available for all main screen icons and can be included in the new software. Greying works analogous to the main screen in Garp3. Candidates for greyed options are (main screen):
  - General (active after some ingredient has been created)
    - Save current model to file
    - Save current model to new file
    - Delete current model
    - Save diagram to EPS file (we may ignore this, and print an empty page)
  - Use level 6
    - Edit last changed scenario (active a scenario has been created)
    - Edit last changed model fragment (active after a MF has been created)
    - All 5 simulate buttons (active after a scenario has been created)
    - Open state-graphs, initial values, and created paths (active when a simulation is available)
    - Save state-graphs, initial values, and created paths (active after at least one simulation was saved)
  - Use level 5 (*note:* tooltip text will have to change, see text on tooltips)
    - Edit last changed model fragment (active after a MF has been created)

        - All 5 simulate buttons (active after a scenario has been created)
        - Open state-graphs, initial values, and created paths (active when a simulation is available)
        - Save state-graphs, initial values, and created paths (active after at least one simulation was saved)
    - Use level 4 and 3 (*note:* tooltip text will have to change, see text on tooltips)
        - All 5 simulate buttons (active after a scenario has been created)
        - Open state-graphs, initial values, and created paths (active when a simulation is available)
        - Save state-graphs, initial values, and created paths (active after at least one simulation was saved)
    - Use level 2 (*note:* tooltip text will have to change, see text on tooltips)
        - Simulate current scenario (active after a derivate value assignment has been created?)
    - Use level 1
        - No buttons available
- 'Save current model to file' *versus* 'Save to model' *versus* 'Save changes'. Issues:
    - Does save to model still occur, or is it always referred to as 'Save changes'? If yes, rename to 'Save changes'
    - Can we simply these notions, by somehow circumventing 'Save changes'
    - Note: there is also 'Save model to disk' (in the Build context). This should be renamed to 'Save current model to file'.
- Tooltips (main screen)
    - General
        - *Reminder*: Check if all are present and correct
        - Print to postscript *should be:* Save diagram to EPS file
        - DynaLearn logo *should be:* Open DynaLearn website
    - Use level 6 (Build/Simulate icons)
        - Remains as in Garp3 (seems currently correct)
    - Use level 5 (Build/Simulate icons)
        - Edit last changed scenario *should be:* Edit general model
        - Open model fragments editor *should be:* Open condition model fragments editor
        - Edit last changed model fragment *should be:* Edit last changed conditional model fragment
        - Simulate current scenario *should be:* Simulate first step
    - Use level 4 and 3 (Build/Simulate icons)
        - Simulate current scenario *should be:* Simulate first step
    - Use level 2 (Build/Simulate icons)
        - Simulate current scenario *should be:* Simulate
    - Use level 1 (Build/Simulate icons)
        - No Build/Simulate buttons available, hence to tooltip texts
- Resizing main window?
    - Should/can resizing main window be limited, such that greying (red to white) stays correct? Should we?
- DynaLearn logo (main screen)
    - Should Open DynaLearn website
- Use level icons (main screen)
    - Hidden or Present depending on user preference
    - The preference can be set in Settings (Pull down menu) using a small dialogue.
- Simulation preferences and Open trace window
    - Keep functionality as in Garp3

    - Relocate buttons (already done)
    - Trace window becomes integrated in new workspace
    - Simulation preferences stays separate interactive dialogue
- Dependencies
    - It is expected that adding dependencies to a workspace (In/equalities, Correspondences, Proportionalities and Influences) keeps working as in Garp3, and that this functionality requires no specific adaptation for the DynaLearn context. However, at the lower use levels some rewording within these dialogues will be required (e.g. Proportionality → Effects, use level 3).
- Sketch (main screen)
    - Only 'meta model' options remain (already shown in main screen DynaLearn)
- Background colour top button bar main screen (Build/Definition, Build, Simulate, and Saved states):
    - Should switch between dark (selected) and light (not selected) grey depending on selection being active
- Extra new requirement: Editable text in all workspaces (at least in the main workspace)
    - Similar to standard file-name changing etc. being able to edit a text field in workspaces by double clicking on the text field and type new text (e.g. being able to change the names of model fragments in the Model fragments definition editor without having to open the properties dialogue first).

18.4. Use level 6

How to move Garp3 screens / functionality to DynaLearn?

- Save/Open state-graphs, initial values, and created paths
    - Saves for an existing simulation:
        - The state-graph
        - Created behaviour paths with this state-graph
        - Scenario with initial values (this is of particular importance for the use levels for which the notion of a scenario is implicit for the user).
    - Opens for a previously saved simulation, including
        - The state-graph
        - Created behaviour paths
        - Scenario Initial values
    - The interactive dialogues can remain as the where in Garp3. When a saved simulation is opened, the software opens in Simulation mode, showing the state graph (as currently is done in Garp3)
    - OWL icon (Open help page) should be removed

18.4.1. Use level 6 – Build

- Build: Entity/Agent/Assumption hierarchy editor
    - Open: Click on icon in main screen
    - Close: by selecting another option in the main screen. In principle any active option is possible.
    - LHS button bar: should move LHS button bar main screen

        - *Issue*: the order of items should be unified (new, properties, erase, …, 'window content organisers')
    - Menu options: see elsewhere in this document
    - Dialogues to Add/Delete etc. ingredients: remain as in Garp3
    - Editor contents (hierarchy of E/A/A ingredients): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Build: Attribute/Configuration/Quantity/Quantity Space definitions editor
    - Open: Click on icon in main screen
    - Close: by selecting another option in the main screen. In principle any active option is possible.
        - The Close button can be removed, but when leaving the editor with unsaved changes: present save/cancel/undo options to the user and act accordingly (unsaved changes check). Taken what is present in Garp3, it becomes:
            - Save changes to model: Save changes and go to newly selected screen
            - Cancel changes: Move to newly selected screen without saving changes
            - Edit changes: Do not go to newly selected screen
        - Note possible bug: the configuration definitions editor does not seem to have an unsaved changes check.
    - LHS button bar: not applicable
        - *Issue:* some of the in-screen buttons may be moved to the LHS button, to be decided later.
    - Menu options: not applicable
    - Dialogues to Add/Delete etc. ingredients within the editor: remain as in Garp3
    - Editor contents: Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Build: Scenario definitions editor
    - Open: Click on icon in main screen
    - Close: by selecting another option in the main screen. In principle any active option is possible.
    - RHS button bar: should move to the LHS button bar main screen.
        - Some options disappear:
            - Simulate selected scenario (because already in main screen)
            - Edit selected scenario (because already in main screen top level options, and it can also be done by double clicking on the name of the wanted scenario)
        - For the buttons, small icons should be used (e.g. available from the Model fragments definition editor)
    - Menu options: not applicable
    - Dialogues to Add/Delete etc. ingredients within the editor: remain as in Garp3
    - Editor contents (List of scenario names): Listed in the workspace of the main screen
        - Future option: show the scenarios by icons
    - OWL icon (Open help page) should be removed

- Build: Model fragment definitions editor

- Open: Click on icon in main screen
- Close: by selecting another option in the main screen. In principle any active option is possible.
- LHS button bar: should move to the LHS button bar main screen.
  - Some options disappear:
    - Edit selected model fragment (because already in main screen top level options, and it can also be done by double clicking on the icon of the wanted MF)
- Menu options: see elsewhere in this document
- Dialogues to Add/Delete etc. ingredients within the editor: remain as in Garp3
- Editor contents (hierarchy of Model fragments): Listed in the workspace of the main screen
- OWL icon (Open help page) should be removed

- Build: Scenario editor
  - Open: Click on icon in main screen
  - Close: by selecting another option in the main screen. In principle any active option is possible.
  - LHS button bar: should move to the LHS button bar main screen.
    - Add the Delete icon
  - Menu options: see elsewhere in this document
  - Dialogues to Add/Delete etc. ingredients within the editor: remain as in Garp3
  - Editor contents (Scenario ingredients): Listed in the workspace of the main screen
  - OWL icon (Open help page) should be removed

- Build: Model fragment editor
  - Open: Click on icon in main screen
  - Close: by selecting another option in the main screen. In principle any active option is possible.
  - LHS button bar: should move to the LHS button bar main screen.
    - Add the Delete icon
  - Menu options: see elsewhere in this document
  - Dialogues to Add/Delete etc. ingredients within the editor: remain as in Garp3
  - Editor contents (Model fragment ingredients): Listed in the workspace of the main screen
  - OWL icon (Open help page) should be removed

18.4.2. Use level 6 – Simulate

- Simulate: 'State-graph'
  - Open: Click on icon in main screen (either Simulate current scenario or Full simulation)
    - If simulation exists: open state graph view for that simulation
      - *Note*: a simulation gets removed, when an edit action is carried out, equal to how in Garp3 simulation results (and accompanying views) are removed when an edit action is carried out.
    - If no simulation exists:

- Simulate current scenario: Performs a one step simulation with the last edited scenario (as in Garp3)
- Full simulation: Performs a full simulation with the last edited scenario (as in Garp3)

▪ Tab: a tab is created in the main screen for this window, so that it can be re-opened later on (note that also other options exist for opening, see 'Open' above in this section, and the general Clone option).

- Close:
  - By either:
    - By selecting another option in the main screen, or
    - By closing tab, or
    - By selecting an option from the LHS button bar.
  - In principle any active option is possible.
  - *Note*: simulation gets removed when an edit action is carried out.
- LHS button bar: should move to the LHS button bar main screen.
  - Buttons with adjusted size have been created for this
- Menu options: see elsewhere in this document
- 'Dialogues' to inspect listed ingredients
  - Select:
    - Select individual states (as in Garp3)
    - Select a path (as in garp3)
    - Select all states (as in garp3)
    - Deselect all states (as in garp3)
  - View:
    - Show entities, configurations and attributes (see elsewhere)
    - Show quantity values (see elsewhere)
    - List model fragments (see elsewhere)
    - Show dependencies (see elsewhere)
    - Transition history (see elsewhere)
    - Equation history (see elsewhere)
    - Value history (see elsewhere)
  - Run (all can be removed, because already in main screen)
    - Open trace window (remove, because already in main screen)
    - Simulation preferences (remove, because already in main screen)
    - Open trace window (remove, because already in main screen)
    - 5 simulation options (remove, because already in main screen)
- Screen contents (state graph): Listed in the workspace of the main screen
- OWL icon (Open help page) should be removed
- Special attention should be given to the (see workspace bottom):
  - Selected states
  - Selected path
  - *Note*: Discussed elsewhere in this document

- Simulate (view): Show dependencies
  - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
    - The view opens for each selected state
      - Tab: Each view becomes a tab in the main screen
      - The last tab opened is actually shown
      - *Extra feature*: if number of selected states > 1 Then ask user if the view should be opened for all states (choice: Yes or Cancel)
  - Close:

        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - LHS button bar (show/hide ingredients): should move to the LHS button bar main screen.
        - Notice that the button bar of 'Simulate: State-graph' should also be shown (most LHS).
    - Menu options: none
    - Buttons below in the screen (status bar?):
        - Change layout entities (move to LHS button bar, new graphics needed?)
        - Change layout quantities (move to LHS button bar, new graphics needed?)
        - Zoom in (move to LHS button bar, new graphics needed?)
        - Zoom out (move to LHS button bar, new graphics needed?)
        - Save diagram to EPS file (remove, already in main screen)
        - Close this window (remove, superfluous)
    - Screen contents (dependencies): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): Value history
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens once for all selected states
        - Tab: a tab is created in the main screen for this window, so that it can be re-opened later on
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - The value history view has a set of 'options':
            - List of quantities
            - Sort by quantity
            - Sort by entity
            - Select all
            - Select none
            - Draw value history
            - Clear screen
        - Can these stay in the position as they currently are?
        - Notice that the button bar of 'Simulate: State-graph' should also be shown (most LHS).
    - Buttons below in the screen (status bar?):
        - Save diagram to EPS file (remove, already in main screen)
        - Close this window (remove, superfluous)
    - Screen contents (values): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): Equation history (very similar to Value history)
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens once for all selected states

        - Tab: a tab is created in the main screen for this window, so that it can be re-opened later on
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - The equation history view has a set of 'options':
        - List of in/equalities
        - Display equations
        - Derivative equations
        - Select all
        - Select none
        - Draw equation history
        - Clear screen
        - Can these stay in the position as they currently are?
        - Notice that the button bar of 'Simulate: State-graph' should also be shown (most LHS).
    - Buttons below in the screen (status bar?):
        - Save diagram to EPS file (remove, already in main screen)
        - Close this window (remove, superfluous)
    - Screen contents (equations): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): Transition history
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens once for all selected states
        - Tab: a tab is created in the main screen for this window, so that it can be re-opened later on
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - The transition history view has two 'options' (bottom of window):
        - Zoom in on details (move to LHS button bar, new graphics needed?)
        - Close this window (remove, superfluous)
        - Notice that the button bar of 'Simulate: State-graph' should also be shown (most LHS).
        - Clicking on the ingredients in the window shows a new window with more details. Let's keep this function as it is. Notice that, closing the transition history should also close (remove) those detailed windows (as it currently happens in Garp3)
    - Screen contents (transitions): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): Show entities, configurations and attributes
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens for each selected state
            - Tab: Each view becomes a tab in the main screen

            - The last tab opened is actually shown
            - *Extra feature*: if number of selected states > 1 Then ask user if the view should be opened for all states (choice: Yes or Cancel)
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - Menu options: none
    - Buttons below in the screen (status bar?):
        - Change layout entities (move to LHS button bar, new graphics needed?)
        - Zoom in (move to LHS button bar, new graphics needed?)
        - Zoom out (move to LHS button bar, new graphics needed?)
        - Save diagram to EPS file (remove, already in main screen)
        - Close this window (remove, superfluous)
    - Screen contents (structure details): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): Show quantity values
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens for each selected state
            - Tab: Each view becomes a tab in the main screen
            - The last tab opened is actually shown
            - *Extra feature*: if number of selected states > 1 Then ask user if the view should be opened for all states (choice: Yes or Cancel)
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or
        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - Menu options: none
    - Buttons below in the screen (status bar?):
        - Edit selected quantity (move to LHS button bar, new graphics needed?)
        - Close this window (remove, superfluous)
        - Clicking on the ingredients in the window opens the quantity definitions editor with the focus on the selected quantity. Can we keep this in place? And when clicking the ingredients move to this editor in the main screen?
    - Screen contents (quantities etc.): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

- Simulate (view): List model fragments
    - Open: Click on icon in LHS button bar of 'Simulate: State-graph'
        - The view opens for each selected state
            - Tab: Each view becomes a tab in the main screen
            - The last tab opened is actually shown
            - *Extra feature*: if number of selected states > 1 Then ask user if the view should be opened for all states (choice: Yes or Cancel)
    - Close:
        - By selecting another option in the main screen, or
        - By closing tab, or

        - By selecting another icon in LHS button bar of 'Simulate: State-graph'
        - In principle any active option is possible.
    - Menu options: none
    - Buttons below in the screen (status bar?):
        - Show model fragment in legacy mode (move to LHS button bar, new graphics needed?)
        - Show model fragment in context (move to LHS button bar, new graphics needed?)
        - Edit selected model fragment (move to LHS button bar, new graphics needed?)
            - Clicking on the ingredients in the window opens one of the 3 views mentioned above, for the selected model fragment. Can we keep this in place? And when clicking the ingredients move to this view/editor in the main screen? See 3 blocks below
        - Close this window (remove, superfluous)
    - Screen contents (quantities etc.): Listed in the workspace of the main screen
    - OWL icon (Open help page) should be removed

    - Simulate (view): List model fragments: *Show model fragment in legacy mode*
        - Open: happens when selected in List model fragments
            - Tab: no tab is created for this view in the main screen
        - Close:
            - By selecting another option in the main screen, or
            - By selecting another icon in LHS button bar of 'Simulate: State-graph'
            - In principle any active option is possible.
        - Menu options: none
        - Buttons below in the screen (status bar?):
            - Close this window (remove, superfluous)
        - Screen contents (MF details.): Listed in the workspace of the main screen
        - OWL icon (Open help page) should be removed

    - Simulate (view): List model fragments: *Show model fragment in context*
        - Open: happens when selected in List model fragments
            - Tab: no tab is created for this view in the main screen
            - *Note*: this view is a kind of dependency view
        - Close:
            - By selecting another option in the main screen, or
            - By selecting another icon in LHS button bar of 'Simulate: State-graph'
            - In principle any active option is possible.
        - Menu options: none
        - Buttons below in the screen (status bar?):
            - Change layout entities (move to LHS button bar, new graphics needed?)
            - Change layout quantities (move to LHS button bar, new graphics needed?)
            - Zoom in (move to LHS button bar, new graphics needed?)
            - Zoom out (move to LHS button bar, new graphics needed?)
            - Save diagram to EPS file (remove, already in main screen)
            - Close this window (remove, superfluous)
        - Screen contents (MF details.): Listed in the workspace of the main screen

        - OWL icon (Open help page) should be removed

    - Simulate (view): List model fragments: *Edit selected model fragment*
        - Open: happens when selected in List model fragments
            - Opens regular MF editor for selected model fragment
                - See *Build: Model fragment editor*

18.4.3. Use level 6 – Specials

- Pull down list for Scenarios names or Model fragment names (LHS top main screen)
    - Main goal of this list is to provide an easy access to either the list of scenarios or list of model fragments in alphabetic order.
    - Top item in the list
        - Overview (Scenarios) *OR* Overview (Model fragments)
    - This list switches focus following the editor activated (last):
        - If user selects: Open scenarios editor
            - Then show: List of scenarios, with Overview (Scenarios) selected
        - If user selects: Edit last changed scenario
            - Then show: List of scenarios, with current scenario selected
        - If user selects: Open model fragments editor
            - Then show: List of model fragments, with Overview (Model fragments) selected
        - If user selects: Edit last changed model fragment
            - Then show: List of model fragments, with current model fragment selected
    - When an item from either list is selected, the accompanying editor is opened in the main screen (with the selected details from the list, thus: either a specific model fragment or a specific scenario).
    - Selecting Overview:
        - If Overview (Scenarios) is selected from the list
            - Then the Scenarios editor opens in the main screen
        - If Overview (Model fragment) is selected from the list
            - Then the Model fragments editor opens in the main screen

- Pull down list for paths (Middle top main screen)
    - Main goal is to save references to sets of selected states (possibly paths), which can later be reused to provide a view on the accompanying state graph.
    - Given a (partial) simulation (state graph):
        - Each selection of states can be saved (as part of the model, in fact as part of simulation results)
        - Default, the name is the numbers of the currently selected states, but the user may adjust this and give a name to the selection when the selection is saved.
        - Adding or deleting a selection is done with the + and – button, on the RHS of the list.

- The saved 'selected sets of states' can be stored together with a simulation result (sometimes referred to as 'saved states'). Such a cluster can later be reopened again, and the simulation and accompanying paths reused.
  - Main screen button:
    - Save state-graphs, initial values, and created paths
    - Open state-graphs, initial values, and created paths
- When are saved states and/or saved selected paths deleted?
  - IF saved selected paths exists AND saved states do not exist
    THEN remove saved selected paths as soon as
    - A new simulation is generated
    - An edit activity has been carried out
    - But before allowing these steps to occur, inform the user about the planned deletion and ask the user whether to proceed (options: Proceed OR Cancel)
  - IF saved selected paths exists AND saved states exist
    THEN remove saved selected paths AND saved states as soon as
    - An edit activity has been carried out
    - But before allowing these steps to occur, inform the user about the planned deletion and ask the user whether to proceed (options: Proceed OR Cancel)
  - *Note*: Upon changing the 'model', all previously stored state-graphs should be deleted (but ask user to confirm. If user says no, the model cannot be changed!). How to implement this feature?
    - Locate with each editor?
    - Activate option upon saving model to file (users can then select to save to a new file (Save as), and their older results will stay intact at the original file).
- How to fill the 'current list space' with state numbers?
  - In principle we want to reuse the options available in Garp3, notably: Selected states / Selected Path (bottom state-graph screen). The hope is that we can merge these ideas and handle them as one, namely by applying the following rule:
    - If a path has been found (path list is not empty) → show path (ignoring which states the user actually selected)
    - If a path has not been found (path list is empty) → shown selected states (if any)
- It seems relevant to keep the Selected states / Selected Path as they currently are in Garp3, but now listed in the main screen (bottom) when the state-graph view is active (that is there is a simulation). But is this possible in the 'status bar'?

- Tab (to obtain requirements, work out details after users have used DynaLearn software)
  - A combination of a Path (or a set of states) and a view
  - Tabs always live within a single simulation context

- Clone (to obtain requirements, work out details after users have used DynaLearn software)
  - Being able to duplicate the main screen
    - No limitation, allow as many copies as a user wants
    - It should be possible to close each of the clones individually

- Extras
  - Undo button (go X steps back in the model building task)
  - Move simulation icons to button bar in main screen
    - When state-graph screen opens: LHS button bar shows simulation icons
    - Other views within the simulation context do not show the simulation icons in the LHS button bar.

18.5. Use level 5

How to adapt DynaLearn use level 6 details to accommodate use level 5?

18.5.1. Use level 5 – General

- General on dialogues for adding ingredients: a dialogue supports a *single action* and is automatically closed after the user has been carried out the action.
- Concerns dialogues for ingredients: Entity / Agent / Assumption / Attribute / Configuration / Quantity / Quantity space
- Basic idea:
  - *Creating and Adding:* The user opens a dialogue to add an ingredient to the main screen. While in the dialogue, the user typically provides Name and Remarks (attributes and quantities require more). Upon added the ingredient (saving) the ingredient is added to the main screen (and to the dialogue internal list?), and the dialogue itself is closed.
  - *Keeping a list:* The dialogue keeps a list of the ingredients that are created by using it. More in general: there is a single list of ingredients which is acted up in two places: in the main screen and in the dialogue.
  - *Adding existing ingredient:* The user can open the dialogue and select an already existing ingredient to be added to the main screen (obeying the general Garp3 rules of course: e.g. all entities should have unique names, so these ingredient can be used only once, but the same quantity can be assigned to different entities, so these ingredients can be used multiple times, etc.)
  - *Delete:* The user can delete an ingredient from the list. The ingredient is then also deleted from the main screen and the dialogue is closed (*note*: this is different from Garp3, which requires deleting from the main screen first). The user can also delete an ingredient from the main screen directly. If this ingredient is the last one, the ingredient should also be deleted from the list.
  - *Properties:* While having the dialogue open, the properties of a selected ingredient can be edited (Name and Remarks). This should follow by the user clicking on 'Adding/Saving'. The changes are then saved and the dialogue closed. When selecting another ingredient (after changing properties of some ingredient), the user has to confirm or cancel these changes (as in Garp3). Upon confirming, the changes are saved and the dialogue closed. Upon cancelling the changes are not saved and the dialogue is closed. Thus, effectively no switching between ingredients while editing the properties of an ingredient.
  - *Cancel:* The dialogue itself can be closed without saving a performed action (or without having done an action at all) by using the cancel option.
- Typical fields/areas in the dialogues:
  - List with names of all ingredients (scrollable)

    - Area with the name of the ingredient selected in the list (editable)
    - Remarks area for free text associated to the selected ingredient (editable)
    - Note that Attribute and Quantity have 'nested lists', because these ingredient types also require to an ingredient from another list (quantity space and values, respectively).
- Opening dialogue:
    - Clicking on the associated button in the button bar in the main screen (LHS) opens the dialogue.
        - *Possible improvement*: In Garp3 dialogues open with the top list item being selected (in focus). It would be better if the dialogue would open with nothing selected (similar to the status after the new button is clicked in the dialogues in Garp3), so that the user can start typing the ingredient name directly.
    - Clicking on the associated ingredient in the main screen also opens the dialogue (in this case with the selection in focus).
- Closing dialogue:
    - A dialogue is closed after a user action is completed. These actions are: Creating new and adding, Selecting old and adding, Deleting, Changing properties (Name and/or Remark), and Cancel.
- Condition and Consequence:
    - This distinction is relevant when creating a 'conditional' expression. It plays no role when users create the 'always true' expression.
- Main screen top: buttons opening definition editors:
    - The icons for opening the definition editors should be removed. It is sufficient to have:
        - The Creating and adding dialogues (see also below) to handle single ingredients while being in a model fragment editor
        - The model fragment editors to open a specific type of model fragment.

18.5.2. Use level 5 – Dialogues for adding ingredients

How to adapt DynaLearn use level 6 details to accommodate use level 5?

- Save/Open state-graphs, initial values, and created paths (as in use level 6)

- Build: Creating and adding Entities, Agents, and Assumptions
    - Details are the same for the Entity/Agent/Assumption hierarchy editors. The text below refers to 'Entity'
    - Two dialogues are involved whose functionality should be merge into a single dialogue:
        - Entity hierarchy editor (level 6: main screen)
        - Add a new entity (level 6: dialogue)
    - Name composed dialogue: Add entity
    - Basis idea: see general text at the beginning of use level 5 text. As a reference, note that the new dialogue Add entity will look rather similar to the dialogue for Add configuration (see items below)
    - Open/Close: see general text.
    - *Issue*: Entity preference is impossible (None → Instance → Instance & Type → Instance & Type hierarchy, see D.2.1 for details). All entities have to be of a unique

'Type' otherwise problems may occur when simulating (looping because of multiple unifications based on instances having equal type)

- Buttons (from old to new):
  - 'Entity hierarchy editor' (read: top to bottom & left to right)
    - Add entity to hierarchy → Add entity
    - Copy selected entity (New, similar as in other dialogues of this kind at use level 5) (*Note*: not present in Garp3)
    - Delete entity from hierarchy → Delete selected entity (& close dialogue!)
    - Show properties (remove, superfluous).
    - 3x layout icons (remove, because not needed for list)
  - Add a new entity
    - Apply changes → Save changes (& close dialogue!)
    - Open attribute definitions editor (remove)
    - Cancel changes → Cancel (& close dialogue!)
  - *Note*: From the two 'Remarks' fields only one should stay
  - *Note*: Layout should become similar to that of the other 'add ingredients dialogues (roughly: local buttons on the right, global buttons at the bottom)
- Filter details
  - User created Entities are all stored directly under the top node in the hierarchy (these top nodes are: Entity, Agent, and Assumption).
  - The user does not see, or have access to the top nodes. The user sees only a flat list (in alphabetic order).

- Build: Creating and adding Attributes
  - Basis idea: see general text at the beginning of use level 5 text.
  - Two dialogues are involved whose functionality should be merge into a single dialogue:
    - Attribute definitions editor (level 6: main screen)
    - Add a new attribute (level 6: dialogue)
  - Name composed dialogue: Add attribute
  - Open/Close: see general text.
  - Buttons (from old to new):
    - 'Attribute definitions editor' (read: top to bottom & left to right)
      - Add attribute definition → Add attribute
      - Copy selected attribute definition → Copy selected attribute
      - Delete selected attribute definition → Delete selected attribute (& close dialogue!)
      - 4 icons for managing attribute values (stay as they are)
      - Save changes to model → Save changes (& close dialogue!)
      - Undo changes → Cancel (& close dialogue!)
      - Close (remove, superfluous)
    - Add a new attribute
      - Open attribute definitions editor (remove, superfluous)
      - Apply changes → Save changes (& close dialogue!)
      - Cancel changes → Cancel (& close dialogue!) (see above)
    - *Note*: From the two 'Remarks' fields only one should stay
  - Filter details: none

- Build: Creating and adding Configurations
  - Basis idea: see general text at the beginning of use level 5 text.
  - Two dialogues are involved whose functionality should be merge into a single dialogue:
    - Configuration definitions editor (level 6: main screen)
    - Add a new configuration (level 6: dialogue)
  - Name composed dialogue: Add configuration
  - Open/Close: see general text.
  - Buttons:
    - 'Configuration definitions editor' (read: top to bottom & left to right)
      - Add configuration definition → Add configuration
      - Copy selected configuration (New, similar as in other dialogues of this kind at use level 5) (*Note*: not present in Garp3)
      - Delete selected configuration definition → Delete selected configuration (& close dialogue!)
      - Save changes to model → Save changes (& close dialogue!)
      - Undo changes → Cancel (& close dialogue!)
      - Close (remove, superfluous)
    - Add a new configuration
      - Switch arguments (stays as it is)
      - Open configuration definitions editor (remove, superfluous)
      - Apply changes → Save changes (& close dialogue!)
      - Cancel changes → Cancel (& close dialogue!) (see above)
    - *Note*: From the two 'Remarks' fields only one should stay
  - Filter details: none

- Build: Creating and adding Quantities and Quantity Spaces
  - Three dialogues are involved whose functionality should be merge into a single dialogue:
    - Quantity definitions editor (level 6: main screen)
    - Quantity space definitions editor (level 6: main screen)
    - Add new quantity (level 6: dialogue)
  - Name composed dialogue: Add quantity
  - Basis idea: see general text at the beginning of use level 5 text.
    - Nested dialogue: From the 1st dialogue (Add quantity) users can open a 2nd dialogue (Add quantity space). This 2nd dialogue behaves in the same general way as the other dialogues, and it has to be closed, before the action at the quantity level (Add quantity) can be completed.
  - Open/Close (Add quantity): see general text (happens from main screen)
    - Open/Close (Add quantity space): see general text (happens from Add quantity dialogue)
  - Buttons:
    - 'Quantity definitions editor' (read: top to bottom & left to right)
      - Add quantity definition → Add quantity
      - Copy selected quantity definition → Copy selected quantity
      - Delete selected quantity definition → Delete selected quantity (& close dialogue!)
      - Delete selected quantity space definition → Delete selected quantity space
      - Open quantity space definitions editor → Add quantity space
      - Save changes → Save changes (& close dialogue!)

        - Undo changes → Cancel (& close dialogue!)
        - Close (remove, superfluous)
    - Add a new quantity
        - Open quantity definitions editor (remove, superfluous)
        - Apply changes → Save changes (& close dialogue!) (see above)
        - Cancel changes → Cancel (& close dialogue!))
    - *Note*: From the two 'Remarks' fields only one should stay
    - 'Quantity space definitions editor' (read: top to bottom & left to right)
        - Add quantity space (stays as it is)
        - Copy selected quantity (stays as it is)
        - Delete selected quantity space (stays as it is) (note also removes quantity space from the Add quantity dialogue) (& close Add quantity space dialogue!)
        - 6 icons for managing quantity spaces (stay as they are)
        - Save changes → Save changes (& close Add quantity space dialogue!)
        - Undo changes → Cancel (& close Add quantity space dialogue!)
        - Close (remove, superfluous)
- Default quantity spaces (to be added in addition to Mzp)
    - P: Plus
    - Zp: Zero Plus
    - Zpm: Zero Plus Max
    - Zsml: Zero Small Medium Large
    - Zsmlm: Zero Small Medium Large Max
    - Zlah: Zero Low Average High
    - Zlahm: Zero Low Average High Max
    - Zlch: Zero Low Critical High
    - Zlchm: Zero Low Critical High Max

18.5.3. Use level 5 – Always True and Conditional Fragment

- At use level 6, four 'view editors' exist:
    - Build: Scenario definitions editor
    - Build: Model fragment definitions editor
    - Build: Scenario editor
    - Build: Model fragment editor
    - *At use level 5 they are re-organised into three 'view editors' (discussed below):*
        - Build: Conditional model fragment definitions editor
        - Build: General model fragment editor
        - Build: Condition model fragment editor

- To accommodate the filters at use level 5, it makes sense to introduce a new model fragment type: *Expression*
    - This model fragment should in principle allow for all ingredients to be created as Conditions or Consequences (details depend on use level)
        - Should it therefore be a subtype of the model fragment type Agent, which includes all ingredients (but not all as Condition and as Consequence)?
        - Or should it be a new top-level item, next to Agent, Static and Process?

- Having this additional type is essential for discriminating this fragment from other fragments later on (e.g. for the export via OWL and repository storage), and it can also be given unique 'behaviour'.
- Note that if a new type is defined (as opposed to a subtype of Agent) this should be accommodated for in the software at some places (e.g. the engine should be augmented to also search for Expression (these adaptations are probably simple).
- Note that this new type can be used also for level 2, 3, and 4!
- The General model fragment (always true) should be of type Expression. Shall we refer to this fragment as *General*?
- Each Conditional model fragment should be a direct subtype of this 'General model fragment', such that these subtypes inherit the contents of the super type (and the conditional details can be placed in the context of the always true expression). Shall we refer to this fragment as *Conditional*?

- Build: Conditional model fragment definitions editor
  - Is in principle the same as 'Model fragment definitions editor' with the limitation that users should *only see the names* of the Conditional fragments they have created (and none of top-level model fragments nodes, such a Static, Process, Agent, and Expression), and also not the always true expression General. Hence, it seems best to use a simple list of names (using alphabetic order), and not use a graphical approach with icons to show the conditional model fragments created by the user
  - LHS button bar: all options stay in place, accept for:
    - The 5 layout buttons (*note*, if we decide to keep the icons in the workspace and not have only fragment names, then the layout buttons may be needed partially).
    - Properties button (needs use level 5 filtering)
  - Pull down menu:
    - File: as in use level 6 (accept 'Properties' needs use level 5 filtering)
    - Edit: as in use level 6
    - View: layout options disappear (details to be determined)

- Build: General model fragment editor
  - Is in principle the same as 'Model fragment editor' with the modification that *all* ingredients may be expressed, and that they are all expressed as consequences (blue).
  - If we decide to use the type General, this editor will always work with a single instance (subtype) of this type.
  - LHS button bar: has buttons to add all ingredient types as consequences (blue). However, the notion of consequences should not be shown to the user.
  - Pull down menu:
    - File: as in use level 6 (accept 'Properties' needs use level 5 filtering)
    - Edit: as in use level 6 (accept 'Properties' needs use level 5 filtering)
    - Conditions and Consequences → Rename: Ingredients (Single list of consequences, blue)
    - View: as in use level 6 (without 'Show subfragments').

- Build: Conditional model fragment editor
  - Is in principle the same as the regular 'Model fragment editor'.

    - If we decide to use the type Conditional, this editor will always work on direct subtypes of this type.
    - Note: The emphasis of Conditional fragments is on conditions of type magnitude, derivative and in/equality.
    - LHS button bar: as in the regular 'Model fragment editor'
    - However, the notion of consequences should not be shown to the user.
    - Pull down menu:
        - File: as in use level 6 (accept 'Properties' needs use level 5 filtering)
        - Edit: as in use level 6 (accept 'Properties' needs use level 5 filtering)
        - Conditions: as in use level 6
        - Consequences: as in use level 6
        - View: as in use level 6 (without 'Show subfragments').

- Colour coding
    - *General* model fragment: all ingredients: Blue
    - *Conditional* model fragment:
        - Inherited ingredients: Green (as in Garp3)
        - New conditions: Red (as in Garp3)
        - New consequences: Blue (as in Garp3)

- Export model to Simulate
    - From: *General* model fragment to:
        - *Scenario*
            - Entity
            - Configuration
            - Agent
            - Assumption
            - Quantity
            - Quantity space
            - Magnitude / Value assignment
            - Derivative / Value assignment
            - In/equality
        - *General* model fragment (subtype of *Expression*)
            - All contents, but without
                - Magnitude / Value assignment
                - Derivative / Value assignment
                - In/equality
    - From: *Conditional* model fragment(s) to:
        - *Conditional* model fragment(s) subtype of *General*
            - All contents (as defined)

18.5.4. Use level 5 – Simulate

- Simulate: 'State-graph' (as in use level 6)
- Simulate (view): Show dependencies (as in use level 6)
- Simulate (view): Value history (as in use level 6)
- Simulate (view): Equation history (as in use level 6)
- Simulate (view): Transition history (as in use level 6)

- Simulate (view): Show entities, configurations and attributes (as in use level 6, but superfluous)
- Simulate (view): Show quantity values (as in use level 6)
- Simulate (view): List model fragments (as in use level 6)
    - *This can be simplified by showing only the list of fragment names that have applied in each of the states, and hiding most of the options enumerated below. On the other hand, if the fragment types General and Conditional are imbedded correctly within the existing MF types, all these use level 6 details should also work correctly at use level 5. So what to do here depends on choices made earlier concerning the approach to this use level. Ideally, everything would simply stay as it is used at use level 6, requiring no extra work here…*
    - Simulate (view): List model fragments: *Show model fragment in legacy mode* (option: hide, make unavailable)
    - Simulate (view): List model fragments: *Show model fragment in context* (option: hide, make unavailable)
    - Simulate (view): List model fragments: *Edit selected model fragment*

18.5.5. Use level 5 – Specials

- Pull down list for Scenarios names or Model fragment names (LHS top main screen)
    - This list will only show the Conditional model fragments
    - The overall working stays as in use level 6
- Pull down list for paths (Middle top main screen) (as in use level 6)
- Tab (as in use level 6)
- Clone (as in use level 6)

18.6. Use level 4

How to adapt DynaLearn use level 5 details to accommodate use level 4?

- Save/Open state-graphs, initial values, and created paths (as in use level 5)
- Build: Creating and adding Entities, Agents, and Assumptions (as in use level 5)
- Build: Creating and adding Attributes (as in use level 5)
- Build: Creating and adding Configurations (as in use level 5)
- Build: Creating and adding Quantities and Quantity Spaces (as in use level 5)

- *Issue*: Entity preference (None → Instance → Instance & Type → Instance & Type hierarchy, see D.2.1 for details) is in principle possible at this use level, particularly: None, Instance, Instance & Type. *Note*: To be created if time permits.

- At use level 5, three 'view editors' exist:
    - Build: Conditional model fragment definitions editor
    - Build: General model fragment editor
    - Build: Condition model fragment editor
    - *At use level 4 there is only the General model fragment editor*
        - Build: General model fragment editor

- Build: General model fragment editor (as in use level 5)
    - Note: Main screen top: buttons opening definition editors:
        - The icons for opening the definition editors should be removed.
        - At this use level only the general 'Edit model' icon is needed to open the only model fragment in this model, namely *General*
        - Note that the notion of model fragment is fully hidden the user at this use level

18.6.1. Use level 4 – Export

- Export model to Simulate (as in use level 5)
    - From: *General* model fragment to:
        - *Scenario*
            - Entity
            - Configuration
            - Agent
            - Assumption
            - Quantity
            - Quantity space
            - Magnitude / Value assignment
            - Derivative / Value assignment
            - In/equality
        - *General* model fragment (subtype of *Expression*)
            - All contents, but without
                - Magnitude / Value assignment
                - Derivative / Value assignment
                - In/equality

18.6.2. Use level 4 – Simulate

- Simulate: 'State-graph' (as in use level 5)
- Simulate (view): Show dependencies (could be as in use level 5, but superfluous)
- Simulate (view): Value history (as in use level 5)
- Simulate (view): Equation history (as in use level 5)
- Simulate (view): Transition history (as in use level 5)
- Simulate (view): Show entities, configurations and attributes (could be as in use level 5, but superfluous)
- Simulate (view): Show quantity values (as in use level 5)
- Simulate (view): List model fragments (not applicable, superfluous)

18.6.3. Use level 4 – Specials

- Pull down list for Scenarios names or Model fragment names (LHS top main screen)
    - Not applicable

- Pull down list for paths (Middle top main screen) (as in use level 5)
- Tab (as in use level 5)
- Clone (as in use level 5)

18.7. Use level 3

How to adapt DynaLearn use level 4 details to accommodate use level 3?

- Save/Open state-graphs, initial values, and created paths (as in use level 4)
- Build: Creating and adding Entities, Agents, and Assumptions (as in use level 4)
- Build: Creating and adding Attributes (as in use level 4)
- Build: Creating and adding Configurations (as in use level 4)

- Build: Creating and adding Quantities and Quantity Spaces
    - Works in principle as in use level 4, accept with the following *significant modification*:
        - Quantities can be created without the user defining a quantity space
            - Filter: Such quantities are given a default quantity space (consisting of a single interval) by the dialogue (without the user noticing).
        - Quantity spaces can be adding to quantities without a user given quantity space
            - Filter: this means that the dialogue replaces the default quantity space by the user-given quantity space
        - Quantity spaces can be changed for quantities with a user given quantity space
    - *Note*: Probably for the default interval a special purpose quantity space should be defined: 'qs-default', so that it can easily be recognised and handle throughout the software.
    - *Note*: The derivative quantity space is not affected by the above, and is always added for a quantity.
    - Dialogues design:
        - Should the two dialogues be fully independent? Or do we implement special purpose handling in the operation of the joined dialogues (see use level 4 and 5). From a user point of view, independent dialogues are preferred.

- Build: General model fragment editor (as in use level 4)
    - But the scope of usable ingredients is limited (Entity, Attribute, Configuration, Quantity, Quantity space, Value assignment (also Derivative), Correspondences and Proportionalities).
        - Filter: Proportionalities are shown as + and –. This requires adapting the Add proportionality dialogue to '*Add effects'.*

18.7.1. Use level 3 – Export

- Export model to Simulate (as in use level 4, but less ingredients)
    - From: *General* model fragment to:

- *Scenario*
  - Entity
  - Configuration
  - Quantity
  - Quantity space
  - Magnitude / Value assignment
  - Derivative / Value assignment
- *General* model fragment (subtype of *Expression*)
  - All contents, but without
    - Magnitude / Value assignment
    - Derivative / Value assignment

18.7.2. Use level 3 – Simulate

- Simulate: 'State-graph' (as in use level 4)
- Simulate (view): Show dependencies (not applicable, superfluous)
- Simulate (view): Value history (as in use level 4)
- Simulate (view): Equation history (not applicable, superfluous)
- Simulate (view): Transition history (as in use level 4)
- Simulate (view): Show entities, configurations and attributes (not applicable, superfluous)
- Simulate (view): Show quantity values (as in use level 4)
- Simulate (view): List model fragments (not applicable, superfluous)

18.7.3. Use level 3 – Specials

- Pull down list for paths (Middle top main screen) (as in use level 4)
- Tab (as in use level 4)
- Clone (as in use level 4)

18.8. Use level 2

How to adapt DynaLearn use level 3 details to accommodate use level 2?

- Save/Open state-graphs, initial values, and created paths (not applicable, superfluous)
- Build: Creating and adding Entities, Agents, and Assumptions (as in use level 3)
  - *But only* Entities can be created (Agent and Assumptions are not used)
- Build: Creating and adding Attributes (as in use level 3, but maybe we don't want it?)
- Build: Creating and adding Configurations (as in use level 3)

- Build: Creating and adding Quantities and Quantity Spaces
  - Works in principle as in use level 3, accept with the following *significant modification*:
    - Quantities have *no* user defined quantity space
      - Filter: Such quantities are given a default quantity space (consisting of a single interval) by the dialogue (without the user noticing).

  - *Note*: Probably for the default interval a special purpose quantity space should be defined: 'qs-default', so that it can easily be recognised and handle throughout the software.
  - *Note*: The derivative quantity space is not affected by the above, and is always added for a quantity.
  - Dialogues design:
    - The Add quantity dialogue needs to be adapted: the quantity space details are not available.

- Build: General model fragment editor (as in use level 3)
  - But the scope of usable ingredients is limited (Entity, Attribute, Configuration, Quantity, Value assignment (*only* Derivative), and Proportionalities).
    - Filter: Proportionalities are shown as + and –. This requires adapting the Add proportionality dialogue to '*Add effects'.*

18.8.1. Use level 2 – Export

- Export model to Simulate (as in use level 3, but less ingredients)
  - From: *General* model fragment to:
    - *Scenario*
      - Entity
      - Configuration
      - Quantity
      - Quantity space
      - Derivative / Value assignment
    - *General* model fragment (subtype of *Expression*)
      - All contents, but without
        - Derivative / Value assignment

18.8.2. Use level 2 – Simulate

- Simulate: 'State-graph' (not applicable, superfluous)
- Simulate (view): Value history (not applicable, superfluous)
- Simulate (view): Transition history (not applicable, superfluous)
- Simulate (view): Show quantity values (not applicable, superfluous)

- Simulate *result should be show in the Build context* (Generic model fragment), which entails three things:
  - Derivative values
  - Inconsistency (shown by a '?')
  - Ambiguity (shown by multiple value assignments on derivatives)

18.8.3. Use level 2 – Specials

- Pull down list for paths (Middle top main screen) (not applicable, superfluous)
- Tab (not applicable, superfluous)
- Clone (as in use level 3, but not in fact superfluous)

18.9. Use level 1

How to adapt DynaLearn use level 2 details to accommodate use level 1?

- Build: Creating and adding Entities (as in use level 3)
    - *But ingredient icon is different, this requires some adaptation*
        - *Dialogues*
        - *Display in main screen*

- Build: Creating and adding Configurations (as in use level 3)

- No other Build features are available
- There is no Export to Simulate
- None of the Simulate features is available

18.10. Menu options

Approach: Some options fixed (always present), and others change depending on the context.

18.10.1. Main menu – General items

File

Open model from file (formats: hgp, owl)

Save current model to file (formats: hgp, owl, legacy)

Save current model to new file (formats: hgp, owl, legacy)

Delete current model

Start new model

---- (horizontal line)

Save diagram to EPS file

---- (horizontal line)

Quit

Issue: can we easily change the main menu options themselves also? Thus whether 'Edit' as such is sometimes shown and sometimes not? Or should the top list always be the same, and only its contents change? Below we assume that the top categories change, therefore the order is slightly different from Garp3/Build such that the 'always shown options' are listed first (accept for 'Settings' which is shown last, and yet always present). Settings may also be placed after Edit if that is easier given implementation constraints.

View

Always active in Build and Simulate context.

Edit

Only active in Build context.

Ingredient (was Element in Garp3) *OR* **Conditions & Consequences**

Only active in Build context. What is shown depends only the editor that is open

Settings

Always active in Build and Simulate context. Has 2 options from Garp3/Build and the item 'Settings' from Garp3/Simulate (currently under Display).

Display

Only active in Simulate context.

18.10.2. Menu options – Details

Garp3	DynaLearn (main screen)

Note: Below there are options that have two versions Hide and Show. Of those cases only the default is mentioned. Both options should be treated in the same way. It concerns: Hide/Show parent-child relations, Hide/Show conditional relations, and Hide/Show model ingredient tooltips.

18.10.2.1. Build: Scenario editor

File:	File:
Scenario properties !!	same (only if Scenario editor active)
Save diagram to EPS file	same (always present)
Save model to disk	rename: Save current model to file (always present)
Edit (scenario ingredients):	Edit: (only if Scenario editor active)
Delete	same (only if Scenario editor active; new in LHS bar)
Properties	same (only if Scenario editor active)
Element:	rename: Ingredient (only if Scenario editor active)
Entity	(items as in Garp3)
Attribute	
Configuration	
Quantity	
Value	
Plus	
Min	
Inequality	
Assumption	
Agent	
View:	View: (only if Scenario editor active)
Collapse	(items as in Garp3, with 2 exceptions)
Expand	
Collapse relations	
Expand relations	
Show relevant	
Full redraw (show all, default placing)	
Expand all	

Hide	
Translations	Settings: Language
Hide model ingredient tooltips	Settings: Hide model ingredient tooltips

18.10.2.2. Build: Model fragments definitions editor

File:	File:
Properties (of ingredients)	move to Edit if possible (top item, then horizontal line)
Save diagram to EPS file	same (always present)
Save model to disk	rename: Save current model to file (always present)
Edit (MF type ingredients):	Edit: (only if MF definitions editor active)
Add child	(items as in Garp3)
Edit	
Delete	(new in LHS bar)
Clone	
Copy	
Paste	
Make inactive	
View:	View: (only if MF definitions editor active)
Default view	(items as in Garp3, with 2 exceptions)
Save this view	
Open other view	
Hide parent-child relations	
Show conditional relations	
Translations	Settings: Language
Hide model ingredient tooltips	Settings: Hide model ingredient tooltips

18.10.2.3. Build: Model fragment editor

File:	File:
Model fragment properties	same (only if MF editor active)
Save diagram to EPS file	same (always present)
Save model to disk	rename: Save current model to file (always present)
Edit (MF ingredients):	Edit: (only if MF editor active)
Delete	same (only if MF editor active; new in LHS bar)
Properties	same (only if MF editor active)
Conditions:	Conditions: (only if MF editor active)
Entity	(items as in Garp3)
Attribute	
Configuration	
Quantity	
Value	
Plus	
Min	
Inequality	
Assumption	
Agent	
Model fragment	
Identity	
Consequences:	Consequences: (only if MF editor active)
Entity	(items as in Garp3)
Attribute	
Configuration	
Quantity	
Value	

Plus

Min

Inequality

Correspondence

Proportionality

Influence

View:	View: (only if MF editor active)
Show sub fragments	(items as in Garp3, with 2 exceptions)
Collapse	
Expand	
Collapse relations	
Expand relations	
Show relevant	
Full redraw (show all, default placing)	
Expand all	
Hide	
Translations	Settings: Language
Hide model ingredient tooltips	Settings: Hide model ingredient tooltips

18.10.2.4. Build: Entity / Agent / Assumption editor

File:	
Save diagram to EPS file	same: (always present)
Edit (entity/agent/assumption ingredients):	Edit: (only if E/A/A editor active)
Add child	(items as in Garp3)
Delete	
Properties	
Copy	

Paste	
View	View: (only if E/A/A editor active)
Horizontal	(items as in Garp3, with 2 exceptions)
Vertical	
List	
Collapse	
Expand all	
Translations	Settings: Language
Hide model ingredient tooltips	Settings: Hide model ingredient tooltips

18.10.2.5. Simulate: State-graph view

File:	File:
Select Scenario	leave out
Save simulation in model	leave out
Open saved simulation	leave out
Simulate all scenarios	leave out (is debug facility)
Save diagram to EPS file	same (always present)
View: (same as LHS button bar):	View: (active when in simulate context) Window
E-R structure	(items as in Garp3)
Quantity values	
Model fragments	
Dependencies	
Transition history	
Equation history	
Value history	
Current scenario	
Scenarios (‘to scenario editor’)	

Display:	Display: (active when in simulate context)
Layout	(items as in Garp3)
Layout states	
Layout terminations	
Settings	Settings: State graph settings

19. Appendix C – GIT Software version management HOWTO

Note: This manual is meant to give pointers towards using Git. The manual is provided on an "as is" basis without warranties of any kind. We have moved our work from Subversion to Git (to use our university infrastructure, particularly to have reliable backups), but our experience is still young. It would therefore be helpful to get feedback to improve this HOWTO. Send comments to J.Liem@uva.nl. The latest version of this document can be found on: http://www.science.uva.nl/~jliem/versionmanagement/

19.1. Distributed Version Management using Git

Version management [1], also called revision or (source) code control, allows users to manage the changes to documents, source code, or other files. Version management is often used in software engineering, as it allows multiple authors to work on the same files simultaneously and easily integrate their mutual changes.

In traditional version management systems (such as CVS and Subversion) there is a central place in which the current version of the files and their change history is stored, which called a *repository*. Users obtain the latest version from this repository, which is called a *working copy* or *checkout*. After editing the files in the working copy, the changes can be *committed* to the central repository.

Versioning systems also allow *branching* or *forking* of development. Conceptually, a branch is a copy of the files under version management that allows these files to be developed independently from other branches. As such, the initial files put in the version management can be considered a branch. Branches are typically used to develop new features separately (so that the new features/bug fixes do not conflict with the existing stable codebase). When development of such a feature is done, the branch is *merged* back to the branch from which it was forked.

The new version management tool available at the UVA is Git [2]. Git is used by several large projects such as the Linux Kernel, Perl and Google Android. Git takes a more modern approach and is a distributed versioning system. As such, each user has a personal repository (cloned from another repository). Checkouts are made from the local repository and commits are written to the local repository. Changes can either be *pushed* to, or *pulled* from, another repository[9].

Due to the distributed nature of Git, two branches with the same name in different repositories are considered two separate branches. For example, a branch *feature1* in the repository of Richard is considered a different branch than the branch *feature1* in the repository of Sam. If Richard wants the latest version of the *feature1* code, Richard has to fetch the latest changes from Sam's repository, and merge his *feature1* code with the code of Sam's *feature1* branch.

19.2. Git Documentation (Linux/Windows/MacOSX)

The Git community provides excellent documentation. There is an official Git tutorial [3] and the Git-SVN Crash Course [4] for those who already know Subversion. Furthermore, there are lectures by Randal Schwartz [5] and Linus Torvalds [6]. The command 'git help' provides access to the git manual pages.

[9] This HOWTO does not discuss 'pull'. Furthermore note that instead of 'clone', 'remote add' + 'fetch' is often used, as it allows naming the repository that is cloned.

19.3. Setting up Git

19.3.1. Setting up Git at the UVA/FNWI (Linux/Windows/MacOSX)

To use Git, the Git executable is required. Moreover, to 'push to' or 'pull from' another repository, the Git executable has to be available *on* the computer hosting that repository. To make Git available on (`mremote/sremote/owXXX`).science.uva.nl and the lvl Linux machines, the following paths should be added to your PATH variable (ask Google how to set environment variables). Add `/usr/bin/` to be able to use Git on `owXXX` and lvl Linux machines. Add `/opt/arch/git/bin/` to be able to use Git on `sremote` and `mremote`. Furthermore, in the `LD_LIBRARY_PATH` variable the directory /opt/arch/lib should be mentioned before /lib. Otherwise pushes will fail (due to an old `libz` installation).

To access a repository in your FNWI home directory via SSH use `ssh://user@mremote.science.uva.nl` (or `sremote.science.uva.nl` if you are a student), since direct SSH access to other computers in the FNWI network is forbidden from outside of the FNWI.

Linux users at the FNWI can use Git by default if they changed their PATH variable appropriately. Windows users at the FNWI can SSH (using the Secure Shell Client) to one of the Linux machines in the network. Given that the PATH variable has been specified correctly, Git can be run from that computer.

19.3.2. Setting up Git on Linux

Git is available as a package in most modern Linux distributions (usually `git-core`).

19.3.3. Setting up Git on Windows

Windows users need to install a software package to use git called `msysgit` [7]. This provides them with a Git shell in which they can type the relevant commands. There is currently a graphical layer being developed (to be used on top of `msysgit`), called `tortoisegit` [8], but the maturity of this tool unclear (at the time of writing).

Note that `msysgit` automatically replaces Unix newlines to Windows newlines (or to other way around) by default. As a result, doing a checkout action can result in modified files immediately. To change this behaviour do:

```
$git config core.autocrlf false
```

19.3.4. Setting up Git on MacOSX

MacOSX users can install Git via MacPorts [9]. After installing MacPorts, install Git using the following command in the terminal:

```
$ sudo port selfupdate; sudo port sync; sudo port install git-core
```

19.3.5. Setting up Git Continued (Linux/Windows/MacOSX)

You can set your username and email address using `git config`. This information will be stored with the changes you commit to repositories:

```
$git config --global user.name "Jochem Liem"

$git config --global user.email J.Liem@uva.nl
```

19.4. Version Management using Git (Linux/MacOSX/Windows)

As mentioned before, Git is a distributed version management system. It supports both the traditional version management workflow with a “central” repository, and a distributed way of version management (or combinations of both). Both workflows are explained below using the use case in which *Richard* (staff-member) and *Daniel* (student) are working together on a paper.

19.4.1. Infrastructure and Permissions Basics

In order for Richard and Daniel to collaborate using Git, they must make their repositories available to each other. Git allows multiple protocols, including SSH and HTTP.

- The central repository workflow cannot normally be used over HTTP, since it does not allow files to be written from a client. As such, Richard and Daniel do not both have access to a single computer over SSH, they have to use a distributed versioning model over HTTP.[10] Note that this makes the repository public to the entire internet, unless HTTP Basic Authentication [11] is used. However, this type of authentication is relatively insecure. Passwords are sent as plain text over the web.
- When using the distributed workflow with SSH, both Richard and Daniel have to have access to the same computer. Furthermore, they need to set the permissions of their repository to both read and execute (`chmod –R 750` if they are in the same group, and `chmod –R 755` if they are not in the same group). Note that this also makes the repository available to other users on the system (either everyone in the same group, or everyone on the system).
- When using the central repository workflow with SSH, both Richard and Daniel have to have access to the same computer. Furthermore, the person running the central repository has to give full permissions to either his entire group (`chmod –R 770`) or all users on the system (`chmod –R 777`). Furthermore, the Git repository should be configured so that new files are created with write permissions for either the entire group, or every user on the system:

  ```
  $git config core.sharedRepository 0775

  $git config core.sharedRepository 0777
  ```

- An option that is usually not plausible due to security issues is sharing a separate user account for development. In this setup a central repository workflow can be used, and the repository can be set to be only accessible to that specific user (`chmod –R 700`).

Note that in all SSH cases, the super directories of the repository directory should have read and execute permissions.

[10] Note that it is possible to write files over HTTP using WebDav [10]. However, this requires Apache2 with the WebDav module, and permission to edit the `httpd.conf` file (which requires root access).

19.4.2. Distributed Version Management using Git

In the distributed version management workflow, the repository should be accessible to your collaborators. Either your collaborators have access to your system via SSH, or you make the repository available via HTTP.

Richard wants to collaborate on his paper with Daniel. Since Daniel has no SSH access to his system, he will make the paper available via HTTP.

Richard moves his paper to his public_html directory, and goes there:

```
$mv /home/richard/papers/paper312 \
/home/richard/public_html/gitroot/paper312

$cd /home/richard/public_html/gitroot/paper312
```

Richard creates a repository for the paper

```
$git init
```

Richard indicates that the files should be put in the repository:

```
$git add .
```

Richard commits all changed files to the repository

```
$git commit -a -m "I put paper312 in version management."
```

Richard sends an email to Daniel indicating the URL of the repository and that he can start working on the paper.

Daniel creates a directory for the paper in his public_html directory (and goes there):

```
$mkdir /home/daniel/public_html/gitroot/paper312

$cd /home/daniel/public_html/gitroot/paper312
```

Daniel creates a repository for the paper

```
$git init
```

Daniel indicates where the repository of Richard is:

```
$git remote add richard http://www.science.uva.nl/~richard/gitroot/paper312
```

Daniel fetches the changes from Richard's repository

```
$git fetch richard
```

Daniel merges his local master branch with the remote master branch of Richard

```
$git merge richard/master
```

Daniel makes some changes and commits them to his repository

```
$git commit –a –m "I changed the abstract and the conclusions"
```

Daniel sends the URL of his repository to Richard, so he can see his changes.

Richard indicates where the repository of Daniel is

```
$git remote add daniel http://student.science.uva.nl/~daniel/
```

Richard retrieves the changes from Daniel's repository and merges his master branch with Daniel's master branch.

```
$git fetch daniel

$git merge daniel/master
```

To collaborate with another student *Sam,* Richard and Daniel both send their repository URLs to Sam. Sam sends his repository URL to Richard and Daniel. To create the latest version of the paper Richard and Sam both indicate where Sam's repository is, they need to fetch the latest version from *both collaborators*, and merge with both of their latest versions.

Note that it is possible to mix the distributed approach with the central repository approach (below). That is, each collaborator would have a "central" (bare) repository on a web server somewhere. A working repository (based on the "central" repository) is created on the computer on which the versioned files are edited. The URL of the "central" repository is given to collaborators.

19.4.3. Version Management using a Central Repository

Consider that Richard and Daniel both have SSH access. In this case they can choose to collaborate using a central repository. Central repositories are *bare* repositories, which are repositories without a checkout. This prevents conflicts (e.g. edits on the same line) in files in the central repository. Conflicts should instead be solved locally. Therefore you can only push to a central repository if you have the latest version of the files in the repository.

Daniel has his paper in `/home/daniel/papers/paper312`.

Daniel creates a repository, adds the files, and does a local commit

```
$cd /home/daniel/papers/paper312

$git init

$git add .

$git commit –a –m "First draft of paper 312"
```

Daniel creates a central repository in /home/daniel/gitroot/paper312

```
$git clone --mirror /home/daniel/papers/paper312/ \
/home/daniel/gitroot/paper312
```

Daniel makes it possible for 'other' to write to the repository

```
$chmod –R 777 /home/daniel/gitroot/paper312

$git config core.sharedRepository 0777
```

Daniel updates the information about this repository

```
$git update-server-info
```

Daniel sends the link to the repository to Richard.

Richard creates a repository to work on the paper

```
$mkdir /home/richard/papers/paper312/

$cd /home/richard/papers/paper312/

$git init
```

Richard indicates the place of the central repository

```
$git remote add central \
ssh://richard@sremote.science.uva.nl/home/daniel/gitroot/paper312
```

Richard fetches the changes from the central repository and merges them with his local master branch (which does not exist yet)

```
$git fetch central

$git merge central/master
```

Richard makes some changes to the paper and commits to the local repository

```
$git commit –a –m "Changed the abstract and conclusions"
```

Richard pushes the changes to the central repository

```
$git push central
```

Daniel wants to continue working on the paper.

Daniel indicates the place of the central repository:

```
$git remote add central /home/daniel/gitroot/paper312
```

Daniel fetches the changes from the central repository and merges his master branch with the master branch on the central repository

```
$git fetch central

$git merge central/master
```

19.5. Issues with Laptops and Windows

People with Windows computers at home probably have neither SSH access nor a HTTP server running on their computer. As such getting changes from repositories is difficult in the distributed version management workflow. Laptops have a similar issue. Changes are difficult to get from laptops due to their changing IP address (depending on the wireless network). As such, repositories do not have a stable URL.

One possible solution is to have your own "central" (bare) repository to which you push your changes. You give the link to this repository to your collaborators. They pull your changes from this "central" repository.

19.6. Migrating from Subversion

If you want to migrate your own repositories from Subversion to Git, heed the following advice. Do not try to migrate using `git-svn` alone. Compile the latest version of Git (we had some broken branches in our repository due to an outdated version of `git-svn`), and use the `svn2git` script [12] to migrate the repository.

19.7. DynaLearn/Garp3 and Git[11]

The Garp3 codebase is versioned using Git. Depending on your goals, there are alternative ways to interact with Git. The following three use cases should cover most goals:

1. Always have the latest version of Garp3 (or one of the development branches)
2. Develop Garp3 (or one of the development branches)
3. Develop Garp3 as a UVA employee (or one of the development branches)

UVA employees use our central Garp3 repository, while others use our "public" Garp3 repository. UVA employees can access the repository using their normal username and password. The "public" Garp3 repository is password protected using HTTP Basic Authentication. Git uses the curl library to connect to repositories accessed via HTTP. To get access to the public Garp3 repository a `.netrc` file (Linux/MacOSX) or `_netrc` (Windows) file has to be created in your home directory. This file specifies the username and password to connect to the password protected Garp3 repository. The contents of this file should be:

```
machine staff.science.uva.nl login developer password G3qmsw|qr

machine www.science.uva.nl login developer password G3qmsw|qr
```

Important: The UVA is bound by contractual agreements not to publicly disseminate development versions of Garp3. If you want to work on Garp3, and want to make the repository available via the web, you are required to shield the repository using HTTP Basic Authentication [1]. Make sure to notify the UVA of the URL, username and password if you want us to integrate your changes in our Garp3 development tree.

19.7.1. Latest Development Version of Garp3

The guidelines in this section are applicable when you want to have the latest version of Garp3, but do not want to develop new Garp3 functionality. To make the guidelines applicable to any user we use the HTTP protocol (since the SSH protocol requires a user on the repository machine). We assume that a `.netrc` (or `_netrc` on Windows) has been created as described above.

Create a clone of the public Garp3 repository in your home directory:

```
$cd ~

$git clone http://www.science.uva.nl/~jliem/gitroot/Garp3.git Garp3.git
```

[11] The DynaLearn branch is, for legacy reasons, incorporated under the Garp3 branch. This means that accessing the Garp3 branch means accessing the DynaLearn branch. There are no changes whatsoever.

By default the standard development branch of Garp3 is checked out.

To update to the latest version of Garp3

```
$cd Garp3.git

$git fetch origin

$git merge origin/master
```

The steps above are sufficient for most users. However, some users might want the latest version of a particular development branch in Garp3.

Show the remote branches

```
$git branch –r
```

Create a local branch based on a remote branch and make a checkout (in this example we checkout `origin/AutomaticModelBuilding` as the local branch `AutomaticModelBuilding`

```
$git checkout –b AutomaticModelBuilding origin/AutomaticModelBuilding
```

The local checkout now reflects the `AutomaticModelBuilding` branch. Therefore merging now merges with this branch instead of the master branch.

Show all the branches (and the active branch)

```
$git branch –a
```

Update the AutomaticModelBuilding branch

```
$git fetch origin

$git merge origin/AutomaticModelBuilding
```

To switch back to the master branch do

```
$git checkout master
```

Switching to the newly created AutomaticModelBuilding branch

```
$git checkout AutomaticModelBuilding
```

Note that it is impossible to push changes to the origin repository using this method (since it is HTTP server).

19.7.2. Developing Garp3

If you are a researcher or student and want to further develop Garp3 (making the changes available to us), it is appropriate to make use of the distributed nature of Git and both the HTTP (to clone and fetch changes from the Garp3 repository at the UVA) and SSH protocol (to push changes to your own central repository). Again, we assume that the `.netrc/_netrc` file has been created. UVA uses the distributed nature of Git, which does not require write access to the Garp3 repository. Developers can make their own repository available to UVA by putting it on the web. By informing UVA about the URL of the repository, UVA can pull changes back to the UVA Garp3 repository. Irrespective of whether you are a single student, or represent a group of developers, a single *bare* repository should be made mirroring the UVA public Garp3 repository.

This bare repository should be made available over the web. Note again that bare repositories do not have a checkout. This prevents conflicts in the central repository when pushing. Conflict should be resolved locally. Therefore pushing is only allowed if you have the latest version of each branch in your local repository.

Create a mirror of the Garp3 public repository

```
$cd /home/user/public_html

$mkdir gitroot

$cd gitroot

$git clone --mirror http://www.science.uva.nl/~jliem/gitroot/Garp3.git Garp3.git
```

Make sure that other developers have write access to the repository

```
$chmod –R 777 Garp3.git  (or '770' if you are in the same group)

$cd Garp3.git

$git config core.sharedRepository 0777 (or 0775 if you are in the same group)
```

Note that the last step is not necessary if you are the only one developing. Also, perhaps fewer rights are required at your university.

Update the information about the repository location

```
$git update-server-info
```

The URL of this created public repository should be made known to the Garp3 developers (if you want your changes integrated). The URL should also be made known to the developers at your university.

Each developer (or a single student) creates a clone of the newly made public repository (over SSH, since they want to push back to the public repository).

Create a local repository to develop in

```
$git clone ssh://anotheruniversity.edu/home/user/gitroot/Garp3.git Garp3.git
```

By default the master branch of the public repository is checked out. The other branches are tracked as remote branches (e.g. `origin/AutomaticModelBuilding`).

Show the all branches

```
$git branch –a
```

Checkout a remote branch as a newly created local branch

```
$git checkout –b AutomaticModelBuilding origin/AutomaticModelBuilding
```

After some development the changes should be locally committed and pushed to the public repository

```
$git commit –a –m "Changed A, B and C"
```

```
$git push origin
```

To update a few active branches based on the public repository do the following:

```
$git fetch origin

$git checkout master

$git merge origin/master

$git checkout AutomaticModelBuilding

$git merge origin/AutomaticModelBuilding
```

Note that usually there is no reason to update all branches in your repository, since you tend to develop only a few branches.

During development your public repository may get out of sync with the UVA public repository. By adding the UVA public repository as a remote repository to your local git repository, changes can be pulled from it, and pushed to your own public repository.

Add the UVA public repository to as a remote repository in your local repository (requires the `.netrc/_netrc` file):

```
$git remote add uva http://www.science.uva.nl/~jliem/gitroot/Garp3.git
```

Get changes from the UVA Garp3 repository and integrate them

```
$git fetch uva

$git checkout master

$git merge uva/master

$git checkout AutomaticModelBuilding

$git merge uva/AutomaticModelBuilding
```

Push the changes to your public repository

```
$git push origin
```

Developing Garp3 as a UVA employee

If you are a UVA employee with SSH access to mremote.science.uva.nl, you can use the Garp3 central repository.

Create a clone of the central repository and create a master branch

```
$mkdir Garp3.git

$cd Garp3.git

$git init

$git remote add central ssh://user@mremote.science.uva.nl/home/jliem/gitroot/Garp3.git

$git fetch central
```

```
$git merge central/master
```

After making changes you can commit them locally and push them to the central repository

```
$git commit –a –m "Changed A, B, and C"

$git push central
```

If you like to work on another branch create a local branch with the same name first

```
$git checkout –b AutomaticModelBuilding central/AutomaticModelBuilding
```

Again, you can do development and commit. To get changes made by others, merge your master branch with the master branch on the central repository.

```
$git fetch central

$git merge central/AutomaticModelBuildin
```

19.8. Acknowledgements

Many thanks to Jeroen Roodhart for manually installing Git on the `mremote/sremote` Solaris machines, and to Adri Bon for installing Git on all the Education/Research Linux Machines.

[1] http://en.wikipedia.org/wiki/Revision_control

[2] http://git-scm.com/

[3] http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html

[4] http://git.or.cz/course/svn.html

[5] http://www.youtube.com/watch?v=8dhZ9BXQgc4

[6] http://www.youtube.com/watch?v=4XpnKHJAok8

[7] http://code.google.com/p/msysgit/

[8] http://code.google.com/p/tortoisegit/

[9] http://www.macports.org/

[10] http://www.kernel.org/pub/software/scm/git/docs/howto/setup-git-server-over-http.txt

[11] http://maymay.net/blog/2008/08/08/how-to-use-http-basic-authentication-with-git/

[12] http://github.com/jcoglan/svn2git/tree/master

e-mail: Info@DynaLearn.eu
website: www.DynaLearn.eu