1 Designing Organisation in ALIVE

In this tutorial we show how to make an organisation model (OM) for a simplified information display scenario (based on the TMT use-case discussed in D7.2).

We start by creating a new organisation model, either by using the 'New' option in the 'File' menu (you will have to select the 'OperA OM Model' wizard in the list, which is in the 'OperettA' folder), using the right click menu as shown in the lefthand-side of figure 4, or via the dashboard.

To create an organisation model via the dashboard, make sure that you first select your project in the 'ALIVE Navigator' view on the left of the ALIVEclipse window. This ensures that the dashboard is displaying the status of that project (The active project is also shown in the top-right corner of the dashboard). Clicking the 'Create' link in the Organisation Model box will bring up the 'OperA OM Model' wizard, shown below.

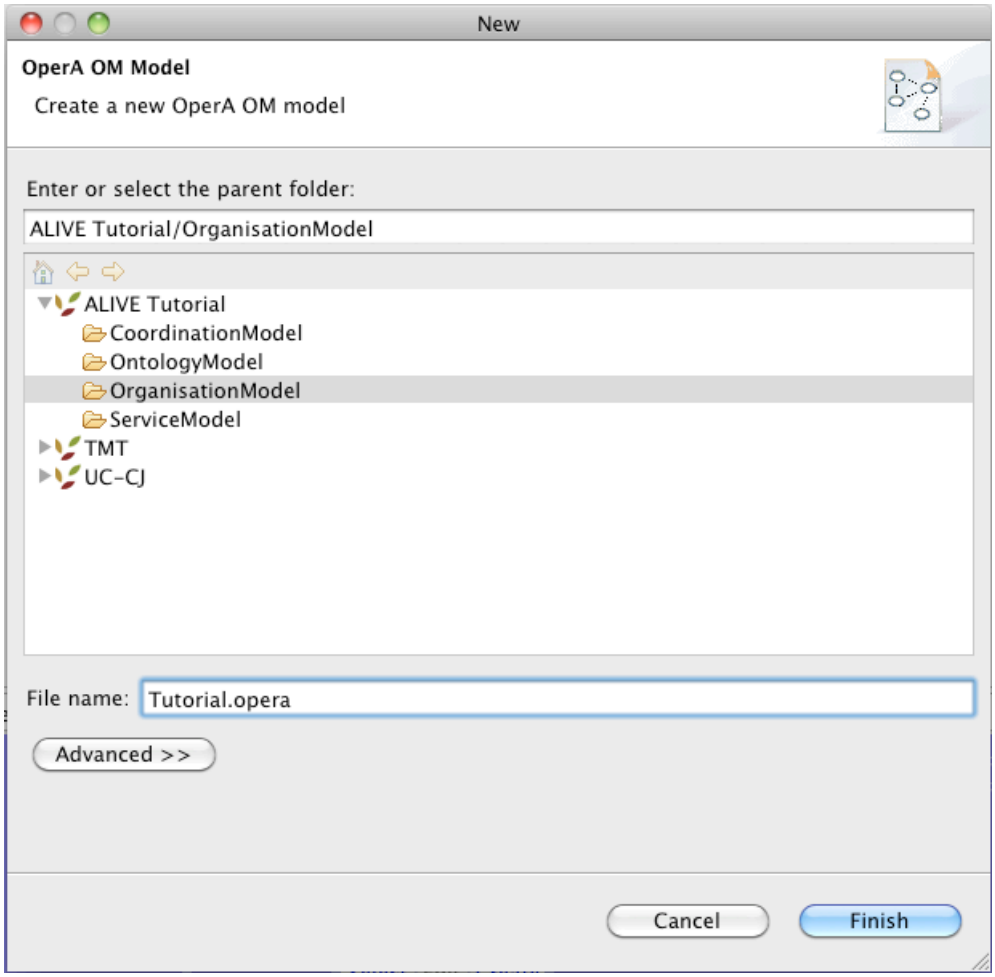
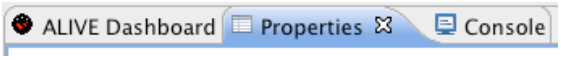


Figure 1: OperA OM Wizard.

Select the correct location for the organisation model in the top part (if not done so already). Organisation Models typically go into the 'OrganisationModel' folder of an ALIVE project. Give the OperA OM Model a representative name in the 'File Name' field at the bottom. Make sure that the extension is '.opera' or the wizard will not let you finish the creation process.

After selecting 'Finish' ALIVEclipse will automatically open the new OperA OM in the editor area (the middle-right-top part of the window). For creating/designing organisation models we will not

need the dashboard, but will need the 'Properties' view. You can switch the view from the dashboard to the properties view by selecting the correct tab in the lower left portion of the window.



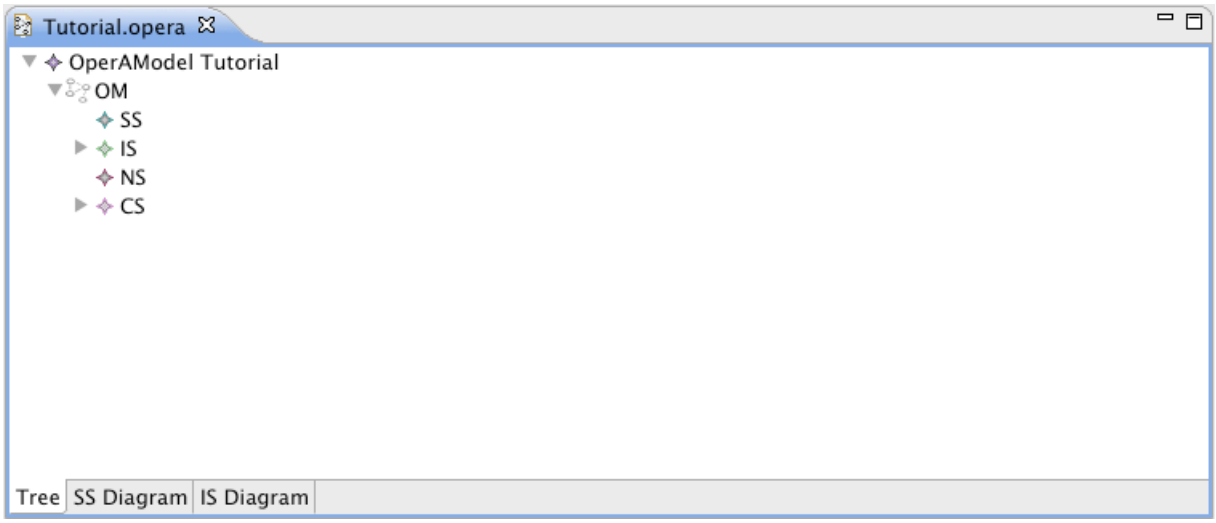
If the lower view has no tab for the properties, you can open the properties in two different ways. Either right click any element in the editor area and select 'Show Properties View', or open the 'Window' menu, select the 'Show View' option, select the 'Other…' option, and find the Properties view in the General folder.

Although not mandatory for ALIVE systems, it is a good thing to name your Organisation Model. Click the 'OperAModel' element in the editor area of ALIVEclipse and change the name of your Organisation Model by changing the value in the 'Name' field shown in the Properties view.

On creation of an organisation model, some elements (which are mandatory for every OM) are already created for you. Clicking the small arrow next to the 'OperAModel' element will expand the tree to show these various elements already in your organisation model.

The elements already in an OperA OM file are:

- OperAModel: container for the other model elements, allows for setting the name of the organisation
- OM: stands for the Organisation Model
- SS: stands for Social Structure; this element is used for setting the social aspects of the organisation. Creation of the SS is detail in section 1.1 .
- IS: stands for Interaction Structure; this element is used for specifying the interactions needed for achieving the organisational objectives, and any restrictions that apply to (the order of) these interactions. Specification of the IS is detailed in section 1.2 .
- NS: stands for Normative Structure; this element contains all the normative aspects of the organisation (e.g., norms). Details on the creation of norms can be found in section 1.4 .
- CS: stands for Communicative Structure; used for specifying the communicative elements (such as formulas and ontologies) required by the organisation. Details about the CS can be found in section 1.3 .



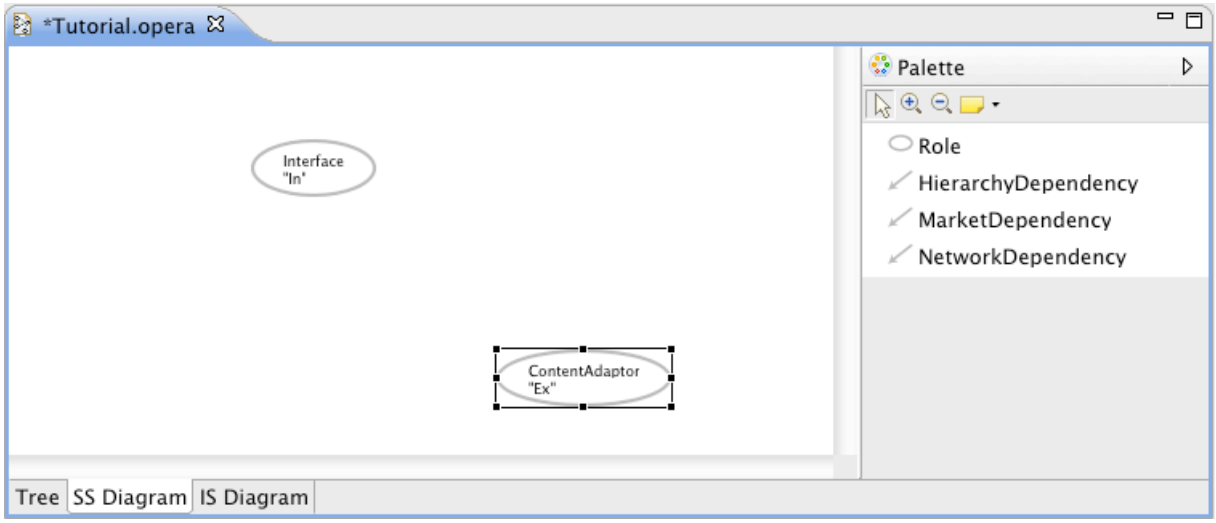
1.1 Social Structure

The social structure of an organisation describes the roles and the relations between those roles in the organisation. It defines the breakdown of 'responsibilities' towards the achievement of the organisational objective in terms of important parties (roles) and how these roles should interact to achieve the objective (dependencies).

Creating Roles

The easiest way to create a social structure is via the diagram editor. Click the tab with 'SS Diagram' on it to show the editor. You will get a blank screen with a Palette on the right side of it. This palette contains the concepts of the social structure, namely the roles and the different types dependencies between the roles. Click on 'Role' in the palette and move your mouse over to a location on the diagram. By clicking in the diagram you create a role, and are requested to type a name for the Role. We name it 'Interface'. As soon as you press 'Return' after typing the name, the properties of that Role will be updated. Note that not only the "Name" field is filled, but that also the "Concept" field now has a value. Concepts (in the ontology) are automatically created while you are designing your organisation. Concepts can also be used for naming several aspects of your organisation, like roles. We get back to using the ontology and naming elements via an imported ontology later on.

Let's create a second role named 'ContentAdaptor'.



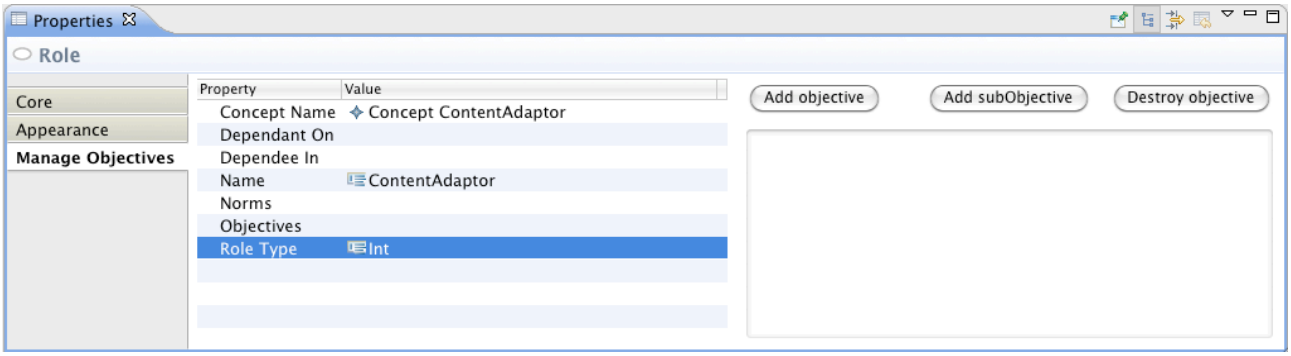
Creating Objectives

We now have an organisation with two internal roles[1]. The roles need to have a purpose for being in the organisation; that is, they need to have a description of why they are important in the organisation. This is done via objectives. Objectives describe the states that should be reached by a role. Through the use of dependencies (discussed below) and objectives, the social structure thus gives a description of how the roles are to participate/interact to achieve an overall 'goal'.

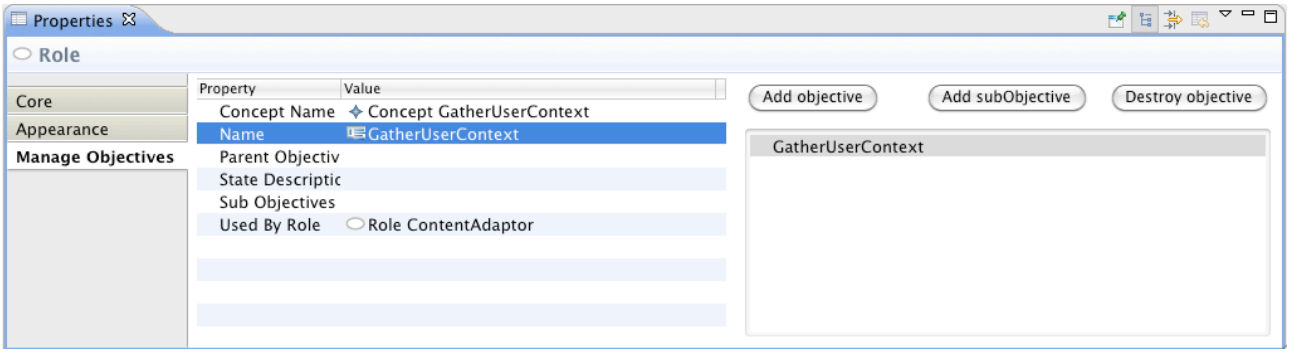
[1] Roles are by default created as 'internal', which means that they will be enacted in the implementation by agents that we create/generate ourselves. Roles can be set to 'external' to indicate that an agent 'outside' your control (that is, created by a third-party) can enact that role, by setting the 'Role Type' property to 'Ext'. To show the difference between internal and external roles, the role icons are labelled with "In" or "Ex" respectively.

In our small scenario, the ContentAdaptor is responsible for achieving various objectives, like assessing the context in which the user is accessing the information display, gathering the information to be provided to the user, gathering the user preferences, personalising the content that is being provided, etc. Let us model the first one here.

Click the ContentAdaptor role figure to select it. Open the 'Manage Objectives' tab in the left-hand-side of the Properties view and click on 'Add objective'.



Select the new objective in the list on the right (it will be called "undefined") and type a name in the "Name" field on the left-hand-side. Name it "GatherUserContext".



If you forgot to select the role before adding the objective, it will not be allocated to the proper role (the 'Used By Role' field will be empty) and will have to be allocated manually. Let's try this as well.

Make sure you have no role selected in the diagram (you can achieve this by clicking anywhere outside the roles in the diagram). Go to the 'Manage Objectives' tab in the Properties view and click 'Add objective'. You can verify that your new "undefined" objective is not attached to a role by checking the 'Used By Role' field (it should be empty now). Let's name this objective "ShowContent".

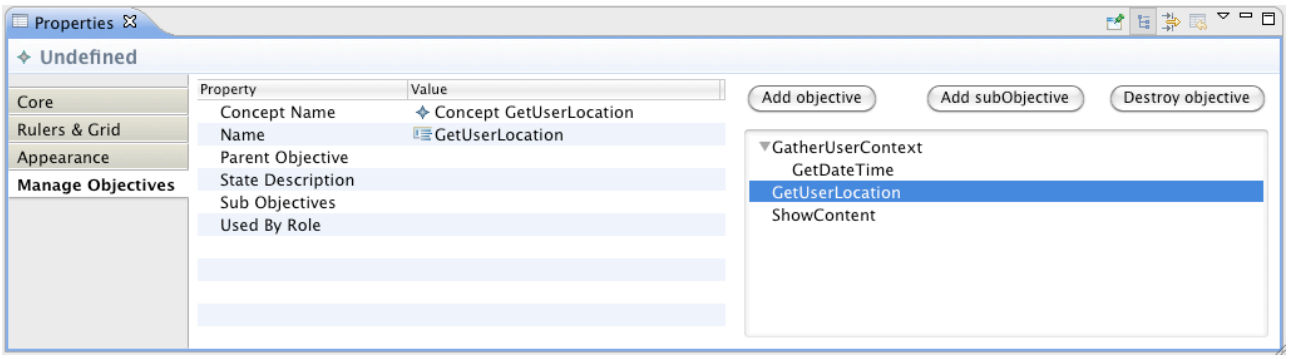
This objective is to be attached to the role "Interface", to indicate that the interface is responsible for showing the content (derived by the rest of the system) to the user. Select the "Interface" role by clicking on its role icon in the diagram. Click the "Objectives" field in the properties, and click on the button with "…" on it on the far left of that field. This shows the Objectives Selector. The Objectives on the left are all the objectives in the organisation, the list on the right are the objectives attached to this role. Make sure that the "ShowContent" objective appears in the right list by either double-clicking it in the left list, or selecting it in the left list and clicking "Add". Click "Ok" to

confirm. The "Objectives" field of the "Interface" role now shows that it has an objective "ShowContent".

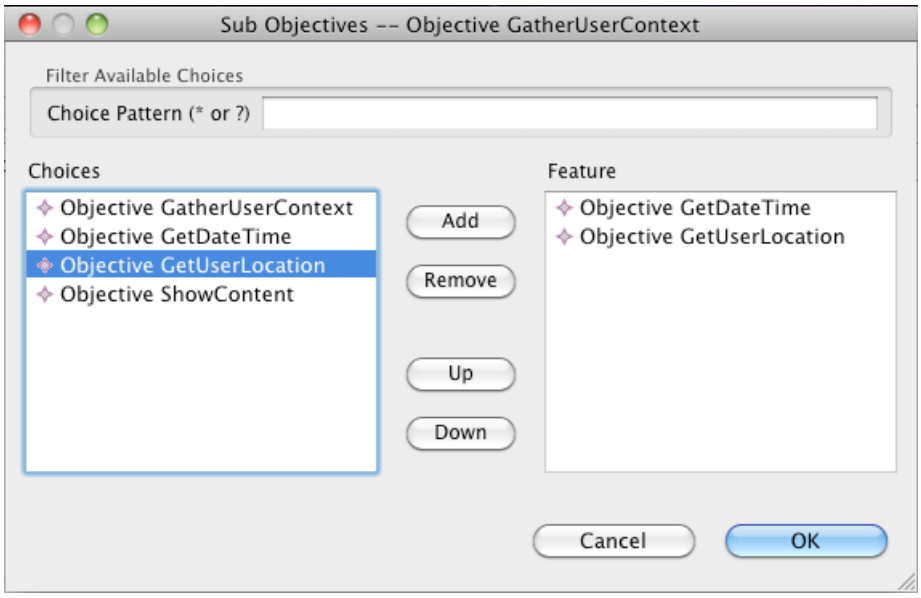
Next we add some sub-objectives. Sub-objectives are child-elements of an objective to indicate that that objective can be split into several smaller pieces. These smaller pieces can be delegate to others (see dependencies below) or an ordering can be set on their achievement. The "GatherUserContext" has two sub-objectives, namely "GetDateTime" and "GetUserLocation". We will use two separate ways of adding these to the "GatherUserContext" objective.

First, we add it directly through the "Manage Objectives" interface. Select the "GatherUserContext" objective in the list, and click the 'Add subObjective' button. An "undefined" objective is being added underneath the "GatherUserContext" objective in the list. Select the "undefined" objective and give it the name "GetDateTime".

Sometime you will want to assign an already created objective as an sub-objective of another objective. We will do this now with the second sub-objective of "GatherUserContext". First, we create a new objective called "GetUserLocation" by clicking the 'Add objective' button (make sure no role is selected, or the new objective gets assigned to that role).



Next we will add this objective as a sub-objective of "GatherUserContext". Select "GatherUserContext" in the list, select the 'Sub Objectives' property field and click the button with "…" on the far right of that field. We get a selection window similar to the one we saw earlier. Select the "GetUserLocation" in the left list, and add it to the right list before closing the window by clicking 'Ok'.



As an objective is a description of the state that is supposed to be achieved by a role, it requires a (logical) description of what that state actually means. The name we gave to the objective gives us some information about what that objective actually means, but to make sure that the system will 'know' as well, we need to attach formal semantics. This achieved by filling the 'State Description' property of the objective. Every objective should have a state description in order to be used by the ALIVE system.

Attaching a state description to an objective is easy, you select one from the drop-down list in the 'State Description' field. However, there are no state descriptions in our model yet. The creation of state descriptions is detailed in a section below.

Creating Dependencies

Our simple organisation specifies no interaction between the two roles present. The roles have their own objectives, but the interactions between the roles have not been specified. Typically roles depend on each other for the completion of their objectives. They need to interact to achieve the complex objectives assigned to them. These interaction needs are specified by means of dependencies.

There are three different types of dependencies.

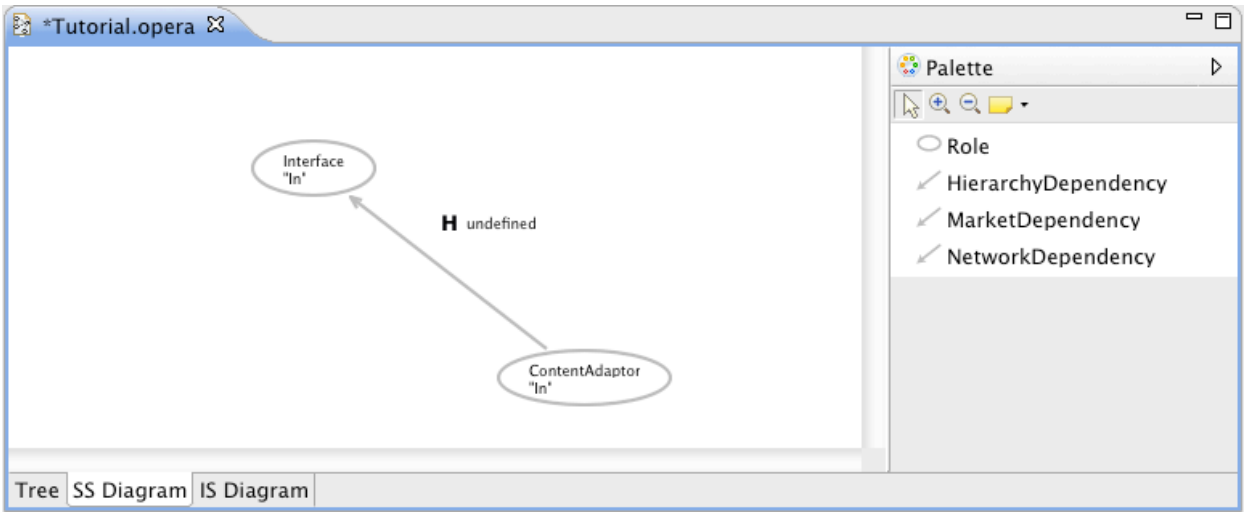
- Hierarchical dependencies: these work like delegations; the dependee role is obliged to achieve the objective being assigned to it via this dependency;
- Market dependencies: these act like bidding or auction relations, a 'call for proposals' is done towards the dependees, which each propose a 'bid' for achieving that objective, and the dependant can then choose whom it will allow to fulfil the objective;
- Network dependencies: in this case the roles related are considered 'equals' with respect of the objective, and have to coordinate among them to achieve it.

Dependencies are always specified towards a specific objective. This objective is delegated to (hierarchical), petitioned for (market) or coordinated with (network) the other role attached to the dependency. The former two dependency types have a direction[2]; network dependencies are

[2] The role initiating the dependency (or superior) is called Dependant (i.e., that role depends on the other role to fulfil

considered bi-directional.

A dependency between roles is easily created in the social diagram editor via the dependency tools on the palette. All three dependencies are represented as a button. Let's create a hierarchical dependency between our two roles. Click the 'HierarchyDependency" button on the palette. Click the "ContentAdaptor" role in the diagram and drag towards the "Interface" role to create the dependency.



Next we have to attach an objective to the dependency to indicate why "ContentAdaptor" depends on "Interface". Click the arrow we just created, select the "Object Of Dependency" property field, and click the button with "…" on the far right of that field. The list on the left gives an overview of all the objectives attached to "ContentAdaptor". Note that it includes all of the sub-objectives of its objectives as well. Select the objective(s) that are attached to this dependency in the left list and add them to the list on the right. In our case, these are "GetUserLocation" and "GetDateTime". Click 'Ok' to confirm. The label in the diagram will update after saving or reloading your model.

the objective). The undergoing role (or subordinate) is called Dependee (i.e., the role that is being depended on to fulfil the objective).

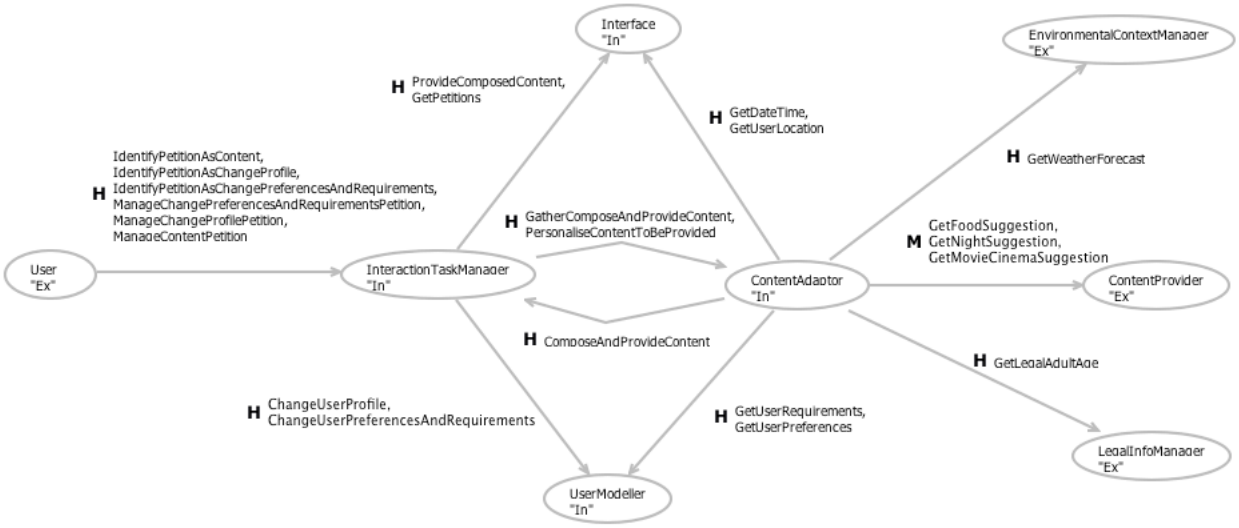


Figure 2: A complex Social Structure.

1.1.2 Best Practices (do)

Concerning the general practice of designing an organisation:

1. Start the design process by identifying the (functional) requirements of the domain. This means an analysis of the global functionalities and objectives of the society.
2. Next identify the stakeholders. The analysis of their objectives identifies the operational roles in the society. These two first steps set the basis of the Social Structure.
3. Set social norms, define normative expectations: The analysis of the requirements and characteristics of the domain is required for the specification of the normative characteristcs of the society. These are represented in the norms in the normative structure (see section 1.4 for details on creating norms).
4. Refine behaviour: Using *means-end* and *contribution* analysis, a match can be made between what roles *should* provide and what roles *can* provide. This aspect contributes to the refinement of role objectives and rights.
5. Create interaction scripts: Using the results from steps 3 and 4, one can now specify the patterns of interaction for the organisation, resulting in the interaction structure (see the next section for details on creating interaction structures).

General concerns when creating the social structure:

- When designing objectives and sub-objectives, remember that over-specifying the domain leaves little ‘freedom’ to the Coordination Level in terms of planning. That is, the Organisation Model objectives can be specified in extensive detail, specifying the smallest steps needed to be done to complete that objective (by using sub-objectives). In general, the more detail you put in the refinement of the objectives *at the organisation level*, the less freedom there is to plan around in different situations.
- The organisation level should use an ‘organisation point of view’, thus describing (i.e., at such a level of abstraction) only that which is important to the organisation. When creating roles and objectives, it is best to ask oneself whether the organisation should be concerned about those, or whether they are an agent issue. As a general example, becoming rich is a

typical agent goal (thus out of scope of the organisation), while efficiency typically is an organisational objective (possibly imposed on the agents participating).

- Sub-objectives can be used to decompose objectives into smaller parts which can be out-sourced (or delegated) to other parties in the organisation. In essence, the entire process of creating the organisation is decomposing the **organisational objectives** (i.e., those objectives in which the organisation *as a whole* is interested in achieving) into smaller bits that can be delegated/out-sources to different participants in the organisation.
- Remember, if (sub-)objectives are delegated/out-sourced to others, the responsibility of completion (or checking that it is completed) remains with the role that delegates the (sub-) objective. Typically, there will be some sort of interaction (in the interaction structure) related to both the delegation/out-sourcing and to the checking/reporting events needed.

1.2 Interaction Structure

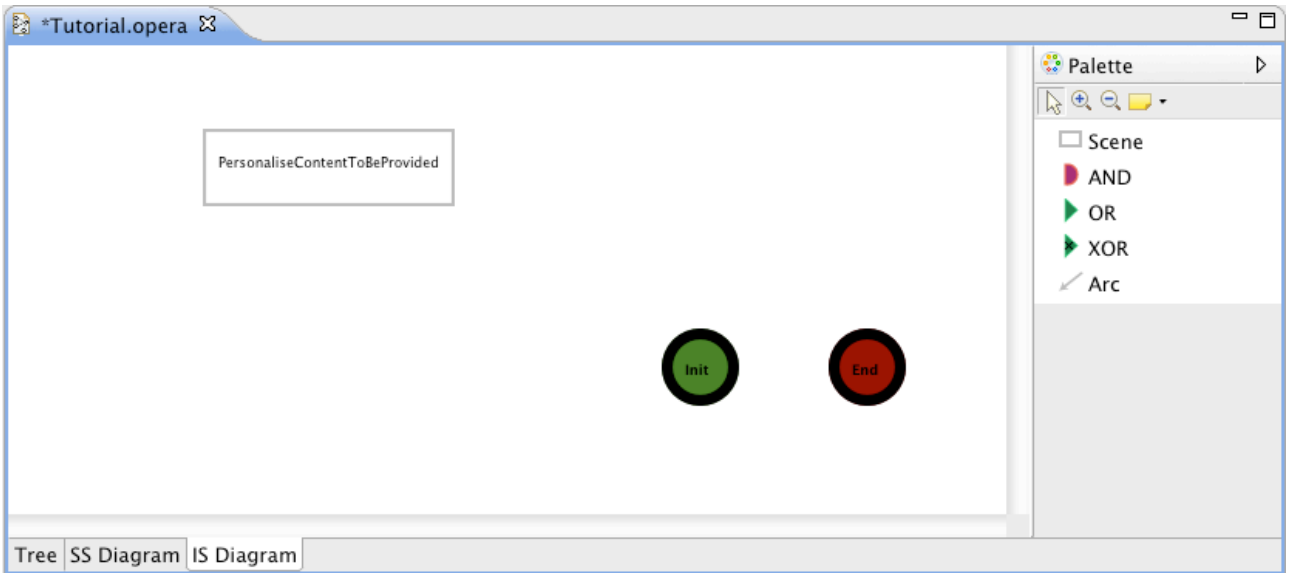
The interaction structure defines how roles should interact with each other to achieve the specified objectives. The interaction structure consists of scenes and transitions (connectors between scenes). Scenes are small pieces of the overall interaction in the organisation that can be seen independent of other scenes. They create a meaningful subset of the interaction, such that patterns of scenes (in some ordering) create the intended interaction patterns for the organisation as a whole. The transitions are used to define the allowed paths through the interaction structure, indicating the means of synchronisation and ordering between the different scenes in the organisation.

Creating Scenes, Transitions and Arcs

Click on the 'IS Diagram' tab to view the interaction structure graphical editor. Similar to the editor for the Social Structure, a palette with the concepts used in this diagram is placed on the right-hand side of the screen. Unlike the Social Diagram Editor, this editor does not start empty but with two objects already placed, namely the initial scene and the end scene[3]. These scenes represent the start and end, respectively, of the interaction process of the organisation.

Creating a new scene is done simply by clicking the 'Scene' button on the palette and clicking anywhere in the diagram to create a scene. After creation you are requested to name the scene.

[3] Due to an EMF/GMF bug, on some platforms the Initial and End scenes are not shown on opening the Interaction Structure editor for the first time. The scenes will automatically appear when placing your first scene, however.



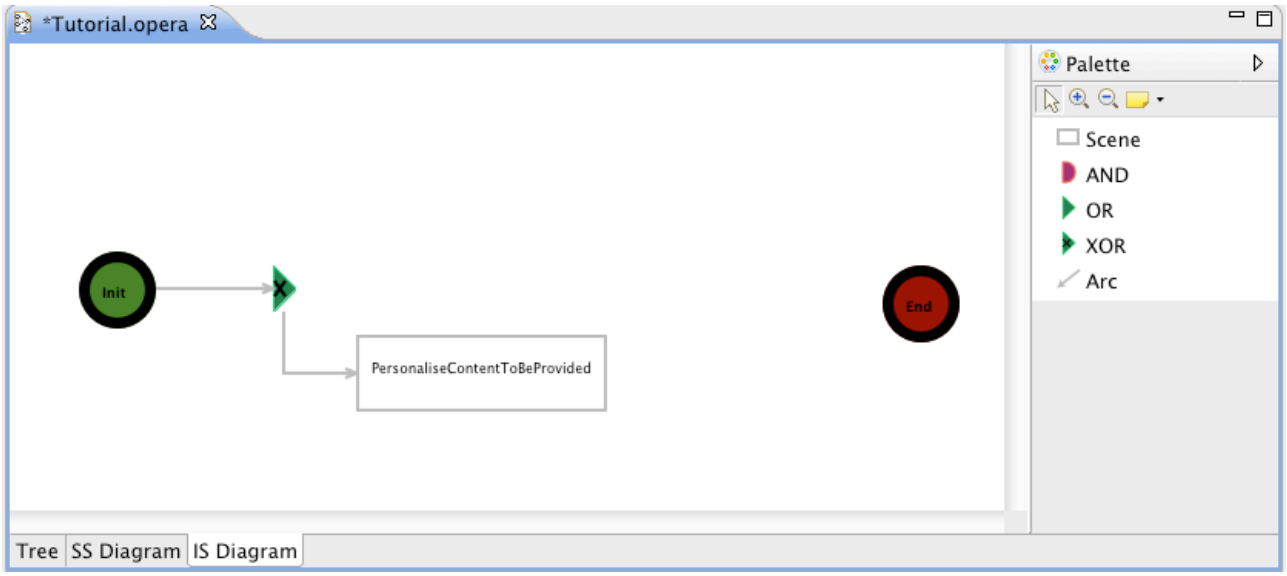
The scene we just created will contain the interactions to prepare for personalisation of content. The gathering of the content, and the actual providing of the content will be handled in a later scene. Next to a name, we can give the scene a description (this is not mandatory, however). Click the "Description" property field after selecting the scene, and type a description of what this scene is meant to represent.

The scenes of the organisation will need to be ordered to indicate the 'flow' of the organisation. This ordering is done by means of transitions. Transitions dictate how the progression from scene to scene takes place. Transitions can, for instance, be used to indicate that there are different paths through the organisation, which cannot be enacted at the same time, or that certain scenes have to be finished in parallel.

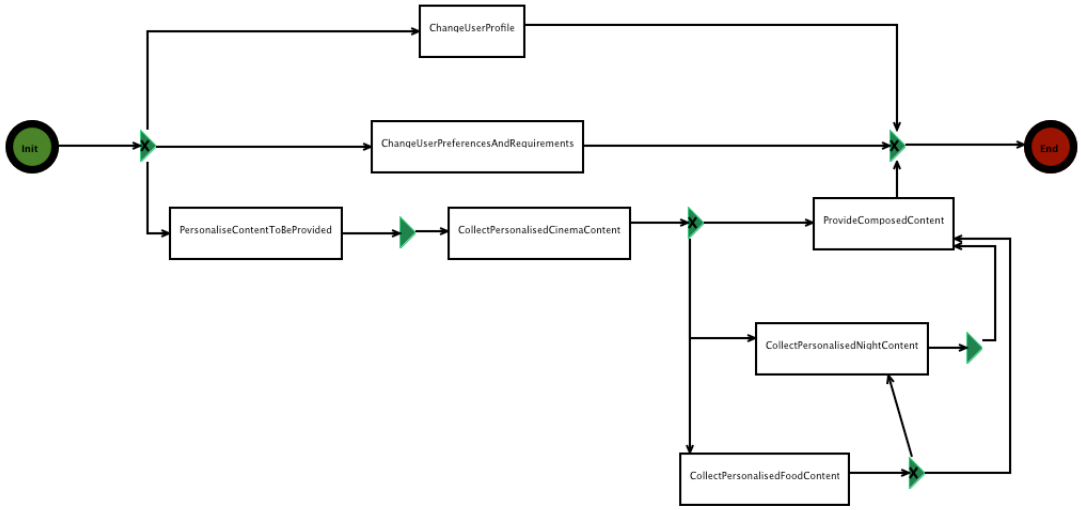
There are three different types of transitions:

- AND: this transition indicates a parallel synchronisation between scenes. If there are multiple scenes leading into an AND transition, each of these should be finished before the scenes that follow this transition may be started. If multiple scenes follow an AND transition, agents/roles that visit this transition are obliged to visit all following scenes as well.
- XOR: this transition indicates an exclusive choice synchronisation. If there are multiple scenes leading into a XOR transition, one and only one needs to be finished before allowing following scenes to be started. If multiple scenes follow a XOR, only one of those scenes may be started when agents/roles visit the transition.
- OR: this transition indicates a free choice synchronisation. No synchronisation is required between incoming or outgoing scenes.

Creating a transition is simple. You select the desired transition type in the palette and click in the diagram at the desired location to place a new transition. Scenes can then be linked into a 'flow' by selecting the 'Arc' tool from the palette and click-and-drag lines between the scenes and transitions. Note that the arcs can only connect a scene to a transition or a transition to a scene. Connections from scene to scene or from transition to transition are not possible.



Using these tools one can quickly build more complex interaction structures like the one shown below.



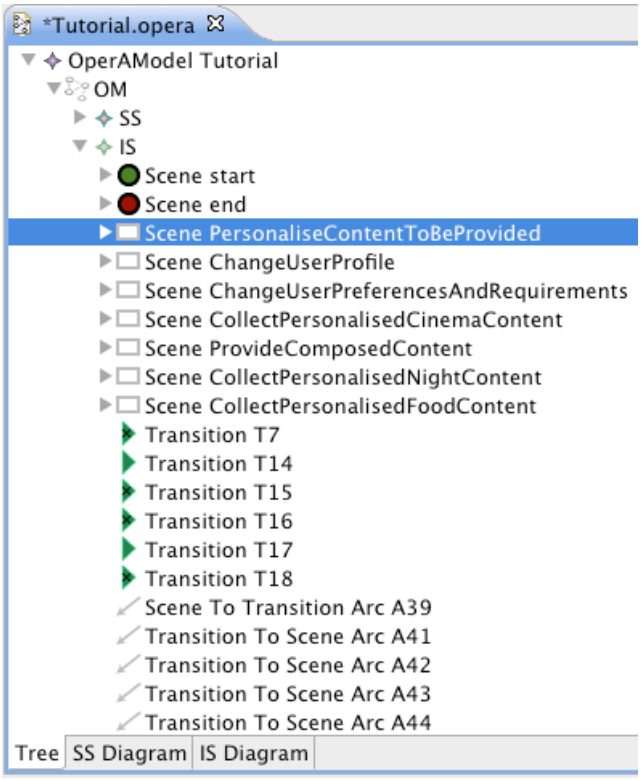
Creating Players

The scenes we just created only give an overview of the ordering between subsets of interactions. The scenes currently have no clear meaning other than perhaps the interpretation of its name. Scenes themselves contain information about what the subset of interaction it represents.

There are two important elements of scenes: the landmark pattern and players. Players are an indication of which roles are typically required to achieve the scene. The landmark pattern indicates the pattern of interaction (on a high level of abstraction) to achieve the scene. This pattern of interaction is expressed through landmarks (important states in the life-cycle of the organisation) and landmark orderings.

Let's look back at our "PersonaliseContentToBeProvided" scene and introduce players and landmarks to it. To add our earlier created roles "ContentAdaptor" and "Interface" as players in this

scene, first select the “Tree” tab on the bottom of the editing area. Open the Organisation Tree to show the scene in the interaction structure.



Right-click the scene, select “New Child” and select “Player” in the list to add a new player to the scene. Open the scene, and select the new player to give it an ID. Players are typically identified by a single character. We name this player ‘c’. To link the player to a role, click the “Role” property field and select “Role ContentAdaptor”. Also add a player ‘i’ enacting the “Interface” role to the scene.

Creating Landmarks

The landmark pattern of the scene was automatically created when we created the scene. It is still empty though, and needs to be filled. Right click the “Landmark Pattern” in the scene-tree and select “New Child” and “Landmark” to add a landmark to the scene. This landmark will indicate that the “ContentAdaptor” and “Interface” need to interact to determine the context of the user (which is based on its time and location). Open the landmark pattern-tree and click the landmark to show its details.

Landmarks are characterised as important states in the life-cycle of the organisation. The definition of which state is meant is done via the landmarks Partial State Description. We discuss how to assign a state description to a landmark later when we discuss how to create Partial State Descriptions below.

Instead of trying to figure out the meaning of the (possibly complex) state description attached to a landmark, it is simpler to just give it a meaningful name. Click the “Name” property field and type the name for the landmark. We name ours “UserContextGathered” to indicate that this landmark represents the moment when the context of the user is known to the “ContentAdaptor”.

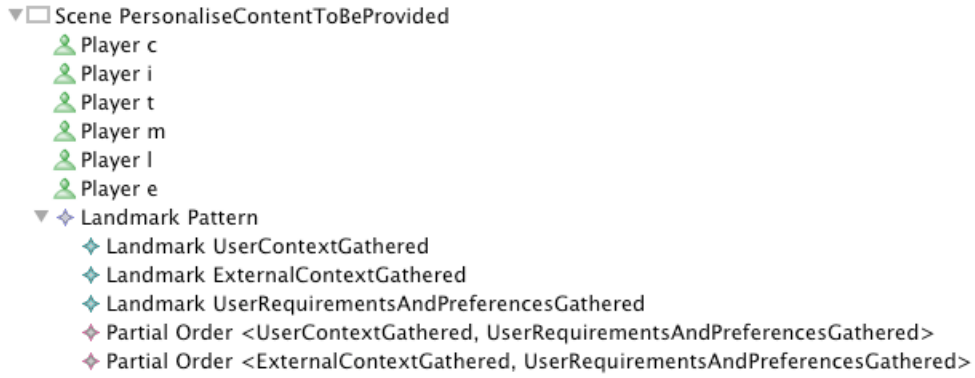
Landmarks are linked to objectives in that both specify the 'desired' states for an organisation. A landmark pattern, moreover, specifies the ordering in which these important states should be reached, thus expressing in which ordering objectives are to be achieved. To indicate which objective(s) are achieved by reaching a landmark, you have to add these to the "Entails" property field of the landmark. To set an entailed objective you click on the "…" button in the right-hand side of the "Entails" property field.

To express the ordering between landmarks, the landmark pattern allows the use of "Partial Order" elements. This ordering is partial because it only indicates which landmarks should have been reached before the next landmark may be achieved.

Creating an ordering between landmarks works like creating a landmark. Right-click "Landmark Pattern", select "New Child" and select "Partial Order". In the "from" property field you select the source landmark, in the "to" property you select the destination landmark. This then defines that the landmark in the "to" field should only occur when the landmark in the "from" field has already occurred.

Landmarks can have several incoming and outgoing orderings. Landmark patterns, however, should never be cyclic or contain cyclic orderings.

Using similar steps we can create the entire landmark pattern for this scene (using additional players):



Scene results

While the beginning of a landmark pattern is implicitly defined with the orderings, the landmark pattern requires the explicit definition of the 'end points'. These landmarks of the landmark pattern are considered to be the results of the scene; that is, achieving these landmarks completes the scene. There are two ways to define the scene results.

Select the landmark that is a result landmark, in our case it is the landmark at the end of the landmark pattern: "UserRequirementsAndPreferencesGathered". In the "Is Result" property field switch the value from "false" to "true" to set this landmark as a result landmark of the scene.

The other way is via the scene itself. Select the Scene in the tree. In the right-hand side of the "Results" property field click the button with "…". Select the landmarks that are intended as the results of this scene.

Note that doing either will update the information in the other place. That is, marking a landmark as a result by setting "Is Result" to true, will automatically add it to the "Results" field of the scene, and vice versa.

1.2.2 Best Practices (do)

Some considerations when creating an interaction structure:

- Every role in the organisation needs to be represented in the interaction structure in at least 1 place (that means, every role should have at least 1 player in a scene). If your role is not connected to a player it either means that:
    - That role is superfluous in the interaction pattern and can be removed from the organisation; or
    - The interactions linked to that role (typically its dependencies and objectives) are not (correctly) modelled in the interaction structure, or the role is not included in the scenes where these are handled.
- Every objective that has no sub-objectives should be linked to an entailing landmark in the interaction structure. These objectives represent the 'atomic'/basic interactions that take place in the organisation, and the interaction structure needs to represent where that interaction is taking place.
- If an objective has sub-objectives, it can still be linked to a landmark that entails it, but it is also possible that the objective is split over several landmarks entail different parts of the state description of the objective (e.g., like in the case where every sub-objective is entailed by a different landmark in which case the objective itself is only achieved when the last of its sub-objectives has been achieved).
- Landmark patterns do not need an explicit representation of their starting points, as this is derived from the partial orderings and the set of result landmarks of that pattern. It does help, however, to create landmarks labelled "Start" in a landmark pattern to represent them explicitly. Labelling such landmarks as "Start" or "start" ensures that the validation framework will not check whether it has a state description attached.
- Representing different paths in a landmark pattern is possible by having more than one landmark in the results set of the scene. In essence, every landmark in the results set is an exit point from the scene, at which time the scene is achieved and progression to the next scene(s) is possible.
- The state descriptions of a landmark are generally easily derivable from the objectives that the landmark entails. Conceptually, the landmark describes the state in which that objective is achieved, and the state description of the objectives it entails should, therefore, be (logically) derivable from the state description of the landmark. In simple cases, this means that the state description of the landmark equals the conjunction of the state descriptions of the objectives that are entailed by that landmark.

1.3 Communicative Structure

The communicative structure of an organisation describes the elements needed in the interactions and for the formal specification of various elements. In essence, it is a collection of the ontologies and formulas used and present in the organisation. We describe each of these in detail.

Ontologies

While designing the social and interaction structures of your organisation, you have already been busy defining the ontology of your organisation. Each of the role names and objectives names have

been collected and added to the organisational ontology. To have a look at the current content of the organisation ontology, open the communicative structure "CS" and open the "Ontology Default" in the tree-editor.

Importing Ontologies

An existing ontology (e.g. one describing elements from the domain that you are modelling) can be imported into the editor to use for picking names for the organisation concepts (such as roles, objectives, predicate atoms, etc.). To import an ontology, either select "Import" from the "Ontology" menu in the menu-bar and locate the ontology you want to import, or select the ontology on the dashboard (via the "Select" button in the Domain Ontology box) and clicking the "Import" button on the arrow pointing from the Domain Ontology box to the Organisation Model box.

Naming elements based on existing concepts in the ontology is simple. For example, lets assume our ontology contains a concept "User" and we want to use that name for a role in our organisation. First let's create a new role as described in Section 1.1 , but instead of typing the name of the role in the "Name" property field, we instead select the "Concept User" from the list in the "Concept" property field. After selecting and pressing Return, the name of the role will automatically be filled with the name of the selected concept.

This is not only useful for the design when using complex existing ontology (as you make sure that you are using the correct terms in your organisation, e.g., eliminating confusions such where "Fireman", "FireBrigade", "FireDepartment" are supposed to describe the same actors), but also makes sure that the connections between the ontology and the organisation model is made in a correct way (which is important for the use in later stages of an ALIVE system).

Exporting the Organisation Ontology

The organisational ontology, that was created while designing the organisation (that is, all the concepts in the "Ontology Default") can be exported. There are two easy ways to export this ontology to an external OWL file. Either select "Export" from the "Ontology" menu in the menu-bar, or select the "Export" button on the arrow leading from the Organisation Model to the Domain Ontology in the dashboard.

In the file dialog that appears you can specify where the ontology file should be placed. Be warned that overwriting an existing owl-file will remove all of it current contents and filling it with the concepts from the organisation ontology. Changes made to an earlier exported version of the ontology will have to be made again in an external ontology editor.

Creating Partial State Descriptions

We briefly mentioned partial state descriptions before when discussing objectives and again when discussing landmarks. Partial State Descriptions (PSD) are logical formulas that are representations of (parts of) the state of affairs in terms of the properties of that state (e.g., goods are sold, thermostat is on, room is cold, etc.) that hold at a certain point in time. Partial State Descriptions are either a Predicate Atom (with *0* to *n* arguments), or complex formulas combining Partial State Descriptions with logical operators (like negation ¬, conjunction ∧, disjunction ∨, material implication →, etc.). The arguments of the Predicate Atoms are Terms, which can either be Constants, Variables or Functions (over Terms).

Now let us create a Partial State Description to describe the states of affairs referred to by the objectives ("GetDateTime", "GetUserLocation", and "GatherUserContext") to give an example of how to create PSDs and attach them to their respective organisational component. The objectives "GetDateTime" and "GetUserLocation" will be represented as simple propositional atoms (that is, a predicate atom without any arguments). Right click the "CS" element in the tree-editor and select "New Child" and choose "Atom". Click the new atom, select the "Predicate" property field and type "UserLocationGet" to give it that name (we use a slightly different name than the name of the objective to differentiate between their concepts in the ontology). Note that the "Concept" property field is again automatically filled with a (newly created) concept from the ontology. Also note the "ID" field, which every PSD has. The ID shows the entire logical formula at that point in the PSD-tree. Basically, it gives a line-based representation of the logical formula, for quick reference (for instance, to quickly see whether you are composing your logical formula correctly).

To attach this PSD to the objective in question, we have to select the objective in the list. Either navigate through the "SS" element of the tree-editor to locate it or switch to the "SS Diagram" tab and select the "Manage Objectives" tab in the properties view and locate it there. Select the objective "GetUserLocation", click on the "State Description" property field and select "UserLocationGet" from the list. This attaches the PSD to the objective, giving it formal meaning in the organisation.

Likewise, create a Predicate Atom "DateTimeGet" that needs to be attached to the "GetDateTime" objective.

As mentioned earlier, objectives with sub-objectives can have more complex state descriptions, in that the state description of such objectives can be composed of the state descriptions of its children. The state description of "GatherUserContext" is one of such state descriptions that expresses that doing either "GetUserLocation" or "GetDateTime" is enough to fulfil the objective. This is expressed by creating a disjunction between the state descriptions of the sub-objectives. To create this state description, again right click the "CS" element in the tree-editor and select "New Child" and choose "Disjunction". A disjunction requires two state descriptions as its arguments (to represent the left-hand and right-hand sides of the disjunction). To insert the "UserLocationGet" and "DateTimeGet" state descriptions as its arguments, click the "Left State Description" and "Right State Description" property fields, respectively. Select the correct state description from the pull-down list.

In similar ways, you can use the other operators available to create more complex partial state descriptions. Here is a brief overview of the operators available:

- Atom: atomic components of PSDs to represent a proposition (an atom with no arguments) or predicate (with *1* to *n* arguments).
- Negation, conjuncation, disjunction, implication: typical (classical) logical operators matching $\neg$, $\wedge$, $\vee$, and $\rightarrow$. Negations require 1 state description as argument, the others require 2.
- For All Paths, Exists Path: state descriptions to quantify over temporal paths. Require 1 path formula as argument.
- Path negation, path conjunction, path disjunction, path implication: similar to their state variants described above, but require path formulas as argument instead.
- Next, sometime, always: path formulas describing that something should hold in the next state on the path, in some state along the path, in every state along the path, respectively.

Require 1 path formula as argument.

- Until: path formula expressing that the first argument should hold up to the point where the second argument holds along the path. Requires 2 path formulas as argument.

Other components available for the specification of partial state descriptions:

- Constant: a reference to a specific element in the domain of reference that can function as an argument to a predicate atom or a function.
- Variable: an undefined (later to be instantiated) reference to elements of the domain that can function as argument to a predicate atom or functions.
- Function: a relationship or expression over constants, variables and/or functions that can be function as an argument to predicate atoms or functions.

Creating Counts-As Expressions

Creating counts-as expressions (to operationalise the abstract terms/concepts used in the organisation) is done in a similar way as creating partial state descriptions. A count-as relation requires partial state descriptions on its left- and right-hand sides, in addition to a label for the context (to which it belongs and which it helps defining).

1.3.2 Best Practices (do)

Some considerations when creating concepts/formulas in the communicative structure:

- Concepts in the organisational model should directly reflect the concepts defined in the ontology of the ALIVE system (which is a combination of a domain ontology and the organisational ontology generated).
    - Every class should be defined as a concept with type “Class”.
    - Every property should be defined as a concept with type “Property”. The domain and range of these should be set in the external ontology editor.
    - Every individual should be defined as a concept with type “Individual”.
- Formulas defined in the CS must be in accordance with the (imported) ontology.
    - Every atom should have a name or concept specified (the other is created/generated automatically).
    - If the atom’s “Concept” corresponds to a Class type, the arguments of that atom (if any) should match that Class (that is, being of that Class as well, or being an individual in that Class).
    - If the atom’s “Concept” corresponds to a Property type:
        - The atom can have only 1 or 2 arguments with some “Concept” of Class type;
        - The arguments’ “Concept” must reflect the Domain and Range defined for the equivalent property in the ontology;
        - Atoms with a “Concept” of Property type and only 1 argument are considered to be a “Datavalued Property” with Range “Boolean”, and must be modelled as such in the ontology.

- The "Concept" of constants must either be of "Class" or "Individual" type.

1.4 Normative Structure

The normative structure expresses organisational norms and regulations to roles, where norms define the obligations, permissions, and prohibitions of the actors in the organisation, related to the roles they play, or to a particular area of activity.

To create a norm, right click the "NS" element in the tree-editor, select "New Child" and choose "Norm". Norms are automatically given an identifier for tracking, which can be edited (but should never be empty). A norm consists of a number of partial state descriptions that describe the moment that the norm becomes active ("Activation Condition"), when the norm is no longer active ("Expiration Condition"), what should be maintained during the life-cycle of the norm ("Maintenance Condition"), and possibly the deadline before the roles should have acted upon this norm ("Deadline"). With the exception of the latter, each of these conditions needs to be filled. This is done by creating a Partial State Description as described in the previous section, and selecting that from the drop-down list that appears when you click on either of these property fields.

Every norm needs to be given a Deontic Statement, which defines the type of norm (Obligation, Permission, Prohibition), the role or player to which this norm applies, and what is obliged/permitter/forbidden by this norm. There are two types of Deontic Statement:

- Role Deontic Statement: this type of statement applies to all actors/players enacting a particular role;
- Individual Deontic Statement: this type of statement applies only to a particular player in a particular scene.

A Deontic Statement is created by right clicking the norm in the tree-editor, selecting "New Child" and then choosing the appropriate type of Deontic Statement. In the properties of the Deontic Statement you can set the modality of the norm (the type), the role or player that this norm applies to, and the partial state description that expresses *what* this norm is about.

1.4.1 Best Practices (do)

Some considerations when creating norms in the normative structure:

- Variable instantiation in the normative reasoner of ALIVE happens based on those present in the "Activation Condition". Basically, when the norm is activated, a norm instance is created with its variables substituted in such a way that the "Activation Condition" matches the current state. Any variables that are *not* present in the Activation Condition, but in other parts of the norm, are essentially considered as 'unimportant' for the evaluation of that condition/formula.

1.4.2 Norm creation example

This example models the creation of a norm to fulfil the following constraint: "*It is forbidden to suggest adult content to underage users*".

As in the TMT application the role responsible of suggesting content is the *Content Adaptor* role, the deontic statement of the norm applies to the *Content Adaptor* role.

To model the norm we start by defining the concepts on the norm, that is (adultContent, suggest and

underage) as shown in the figure below. Notice, some of the concepts are high-level concepts derived from lower level concepts via counts-as rules. For instance, a pub or a peep-show being adult content, or the *Underage* concept.

Once the concepts have been defined we go on applying them to the norm.

- Activation condition: The norm becomes active when the user is underage. Thus, underage concept comes in handy for this.
- Deadline: Does not apply to this norm
- Expiration condition: The norm stops being active when the user is not underage. Thus, underage concept comes in handy for this.
- Maintainance condition: During all the norm life-cycle (after it is active, and until it has been deactivated) it is true that, if a given content is considered to be Adult age content (that is, counts as adult age content) this content has not been suggested to the user.

Notice the norm includes some concepts derived from basic concepts (such as negations, and the implication).

Once the norm has been defined, we define the deontic statement for the norm:

- Modality: Forbidden, the norm is forbidding something
- Role: Content adaptor role, being the one responsible of the concept used in *what.*
- What: suggest(X), the concept (in this case, an action) that is being forbidden.

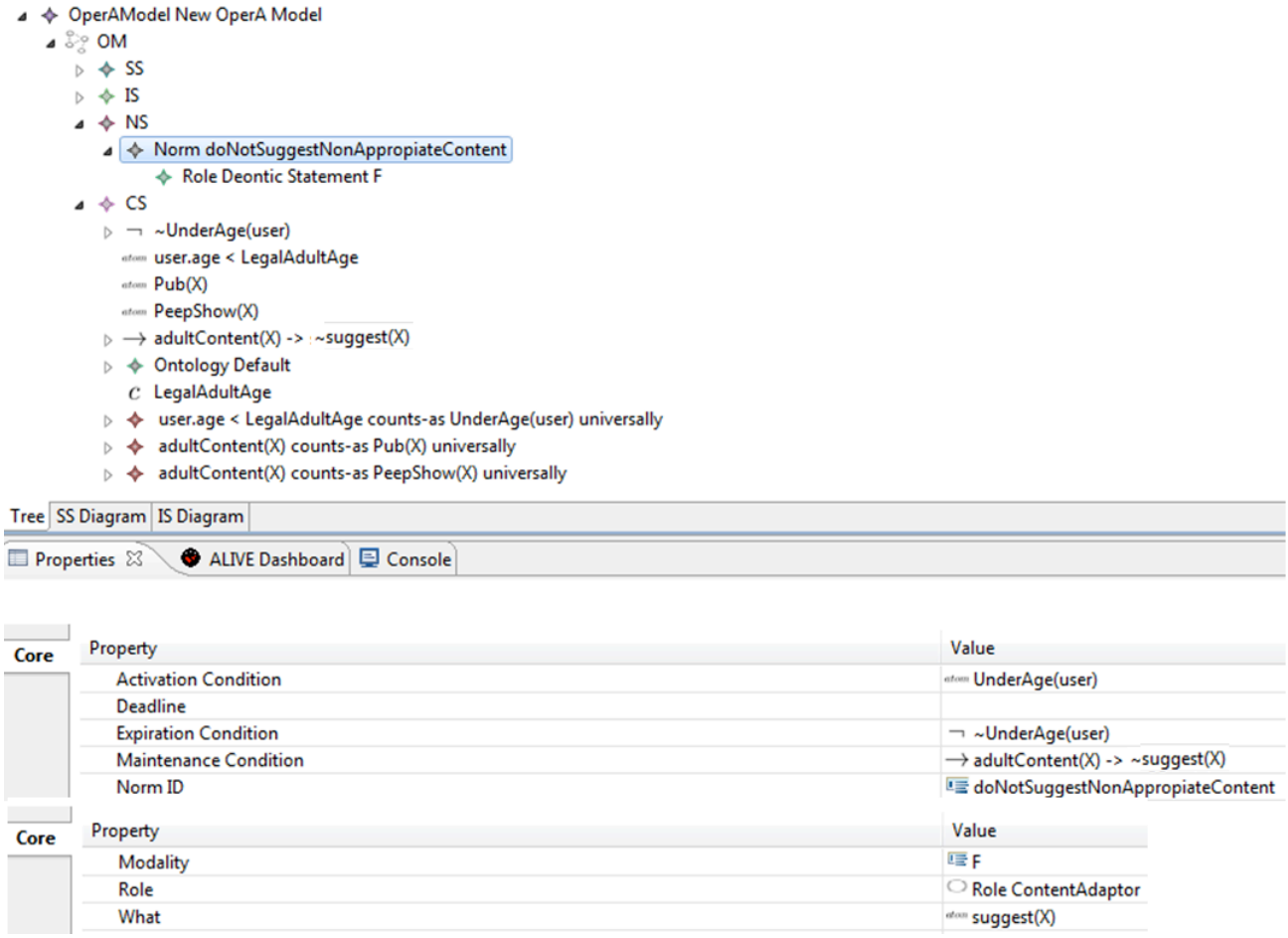


Figure 3: Norm creation example via the operetta tab based on the TMT case

1.5 Validating Organisation Models

Organisation models can be verified to check their consistency (and to check some overall design principles). While it is possible to verify an entire organisation at once, it usually works best to select smaller sub-elements of the organisation and test those independently (relations to other elements outside that element are automatically verified as well).

To verify the consistency of an element in the organisation you do the following. Select the element that you want to validate, right click and select “Validate”. The validation will now check the constraints specified for that element and gives feedback on, e.g., missing elements.

The feedback given by the validation is either Error messages or Warning messages. Error messages indicate serious mistakes in the design (such as roles without a name), while Warnings typically indicate mistakes that might be correct in some (extreme) cases.