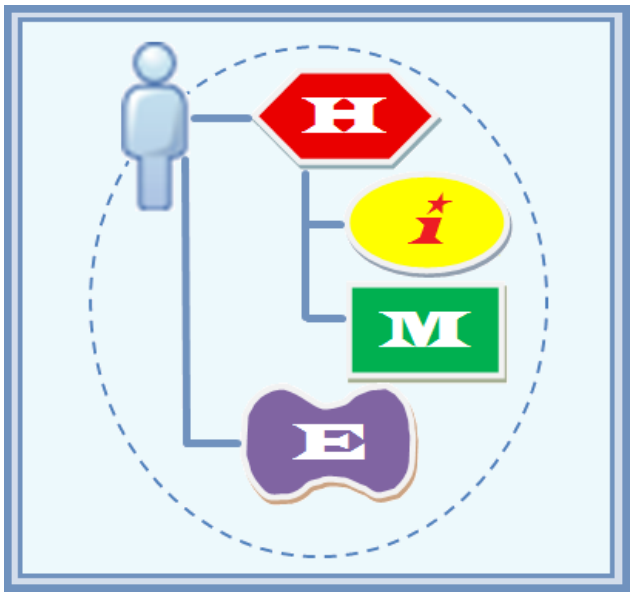


# Hierarchical *i** Modeling Editor

**User's Guide (v. 2.0)**

# Table of Content

# Table of Figures

H*i*ME is an *i** editor that includes specific features for dealing with inheritance operations when inheritance appears in the models.

The intention of this document is provide a guide for using the modeling features of H*i*ME for the creation of i* models. It has been structured in the following sections:

- General View of H*i*ME
- Managing and Viewing models
- Working with models
- Working with inheritance

# 1 General View of H*i*ME

The main window is divided in two parts: the model navigator (left side) and the statistics (right side). On the model navigator view, the model is represented like the folder tree used by the operating systems. On the model statistics, there is some information about the complexity of the model.

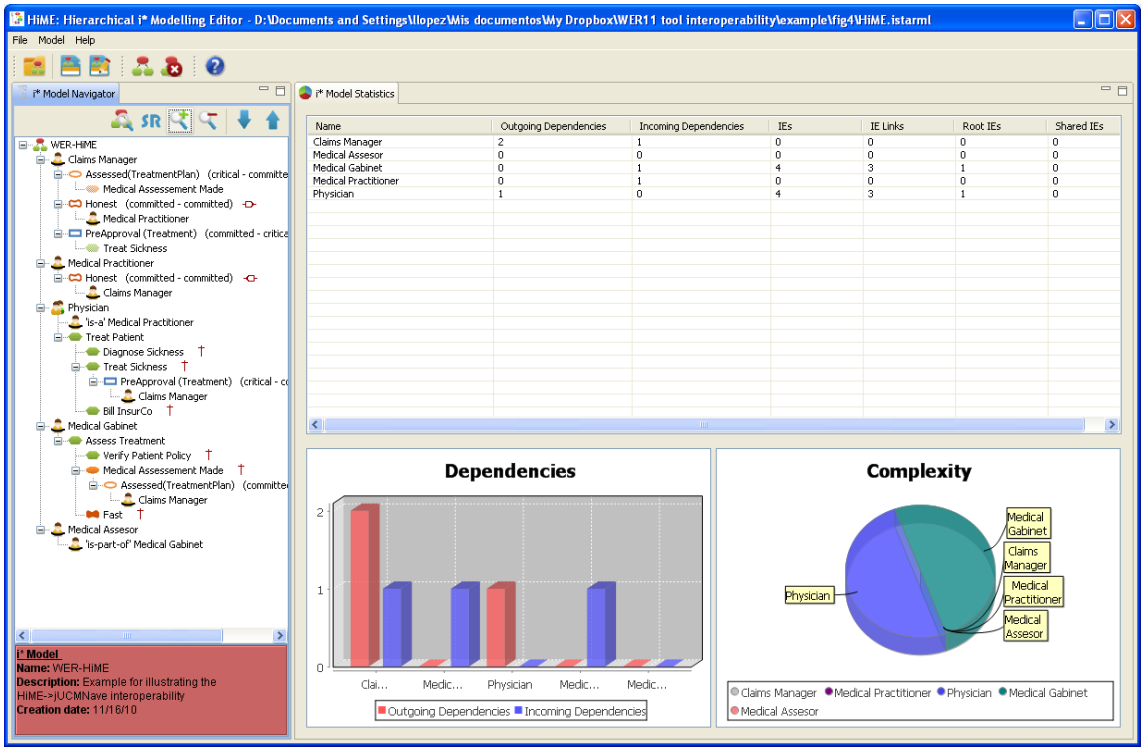


**Fig. 1. H*i*ME Main Window**

## 1.1 Model Navigator View

This view is for viewing and managing the model. This view shows the model as a folder tree where the icons gives some information about what kind of element is represented.

It has its own toolbar to manage the view and at the bottom there is a special red area where some information is shown about the selected element on the model.

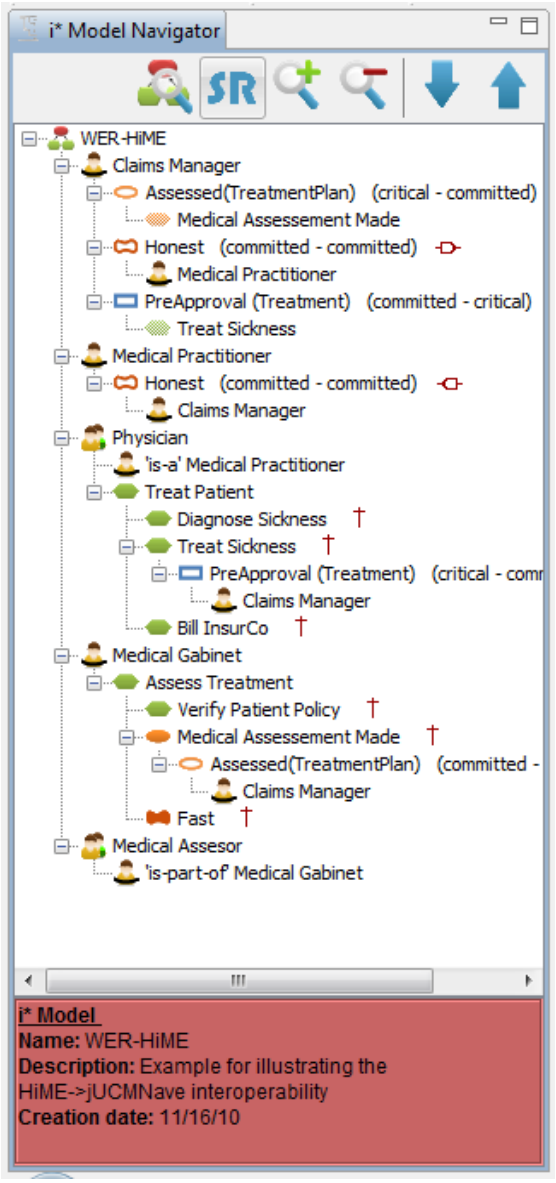


**Fig. 2. Model Navigator View**

This view has associated a toolbar that provides some functionality for the management of the view.

**Table 1. Model Navigator View Toolbar**

| Button | Functionality |
|---|---|
|  | Searching model elements in the navigator view |
|  | Model Navigator allows different views depending if we want to see:<br>• all the model (SR).<br>• only SD elements (actors and dependencies) .<br>• SD with only outgoing dependencies.<br>• SD with only incoming dependencies. |
|  | Expands the selected element childs. |
|  | Collapse the childs for the selected element on the model. |
|  | Moves the selected element one position up in the same level. Only actors and intentional elements can be moved. |
|  | Moves the selected element once position down in the same level. |

For each element, depending on its type (actor, goal, dependency …) there is a contextual menu for creating the new elements, for example in:

- the model element, the user can add actors.
- an actor element, the user can created intentional elements or dependencies.
- a softgoal element, the user can create contribution links and dependencies as is shown in Fig. 3 when the softgoal "*Atención oportuna*" is selected.

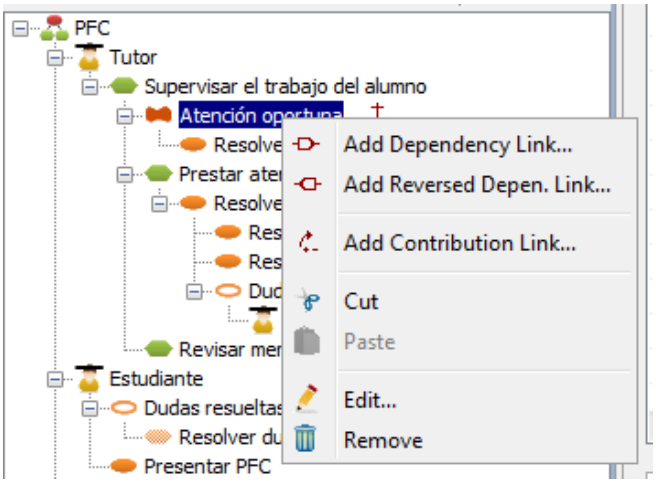


**Fig. 3. Contextual Menu for Softgoals**

## 1.2 Statistics View

This view provides some quantitative information related to the model for each actor. On the list on the top there is information for each actor about how many dependencies (outgoing and incoming), intentional elements (IE), IE links, IEs that are root (they are not the source for any IE link) and which IEs are in more than one decomposition.

Graphics at the bottom show information about the actors' vulnerability and complexity. The vulnerability is measured by the number of outgoing and incoming dependencies. On the other hand the complexity is measured by the elements inside the actor's boundary (intentional elements and intentional element links).

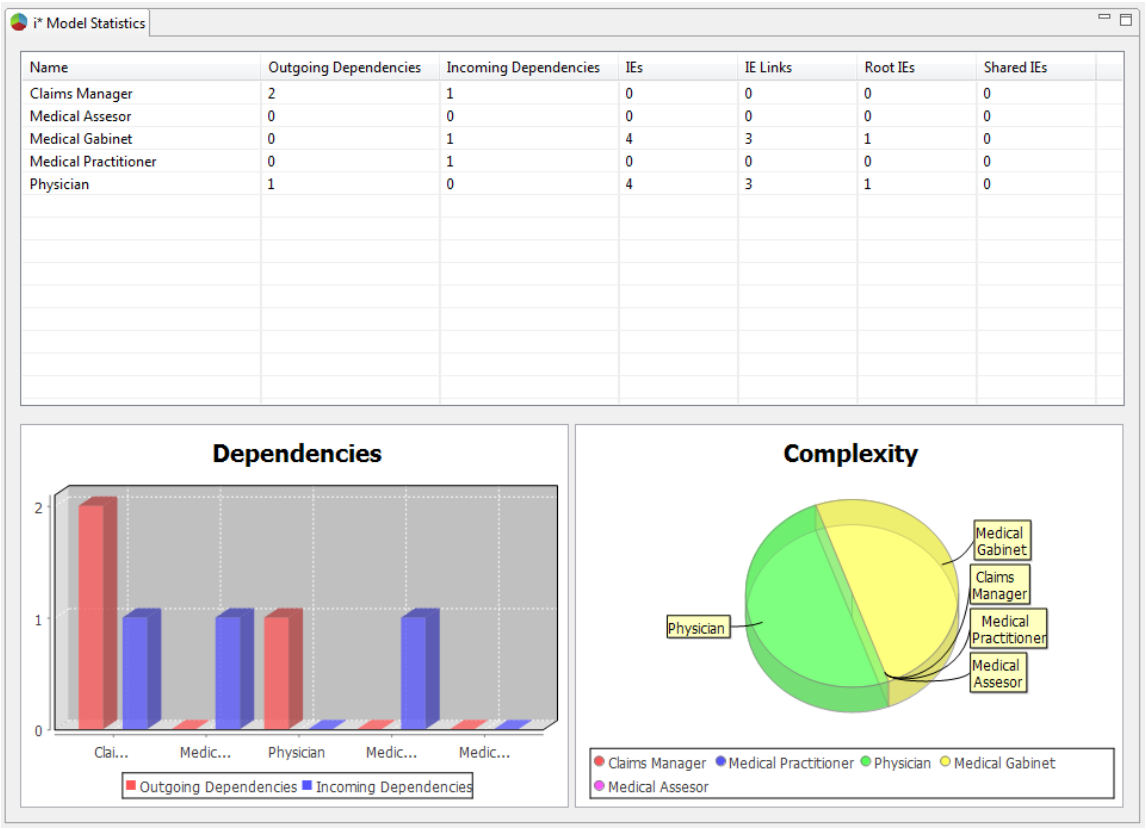


**Fig. 4. Statistics View**

## 1.3 Menu Options

There are three menus: File, Model and Help. For each menu, the options are describe in the following tables. Some of these options are also directly accessible by the toolbar; in the description the button from the toolbar will be included in the description.

**Table 2. Menu Options**

| Menu | Option | Toobar Button | Functionality |
|---|---|---|---|
| **File** | Menu File | | |
| | Open |  | Opens a dialog for selecting the file where the model is stored. The model is loaded in the editor. |
| | Save… |  | Saves the current model in the loaded file. If there is no file, the user will be asked for one |
| | Save As… |  | Opens a dialog for selecting the name of the file where the current model is going to be saved. |
| | Exit | | Closes the application |
| **Model** | New… |  | Creates an empty model. |
| | Clear | | Deletes all the current model elements. |
| | Close |  | Closes the current model. If there is unsaved changed, it will ask for saving confirmation. |
| | Add Actor… | | Adds an actor to the current model. |
| **Help** | Help Contents |  | Show the help files. |
| | About | | Show information about H*i*ME. |

# 2 Managing and Viewing models

## 2.1 Managing *i** Models

This tool manages models and store this information in text files, H*i*ME uses iStarML format for storing models on files.

### 2.1.1 Creating Models

Models are created on the tool using option New of menu Model or the toolbar button  .

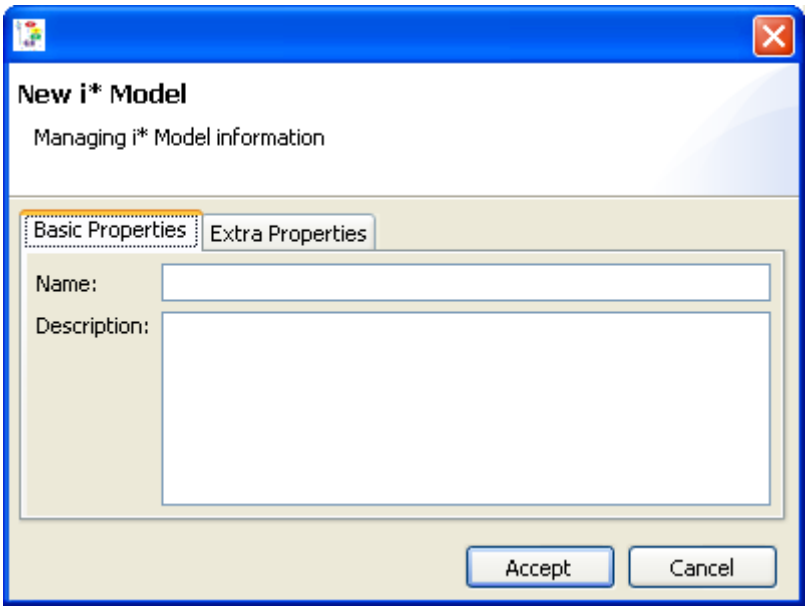


**Fig. 5. Creating Models Dialog**

Besides the basic information, models can have some customizable information. This information is managed in the Extra Properties tab. The information about how manage this extra information is detailed in Section 3.1.

### 2.1.2 Saving and Loading Models

Models are stored as text files, therefore the user will be asked for the file name to save the model on the tool or to load one.

For storing the model there is two options: *Save* or *Save As* options from menu *File* depending if the user is saving the current model in a file with the same name or a new one. These options can be also directly accessed from the toolbar buttons  and  respectively. Note that the first time that the model is stored; a name is asked although the option used has been saved.

When a model is loaded, if there is any problem a list of errors and warnings appear giving some detailed information about the errors. This list can be exported as a text file.

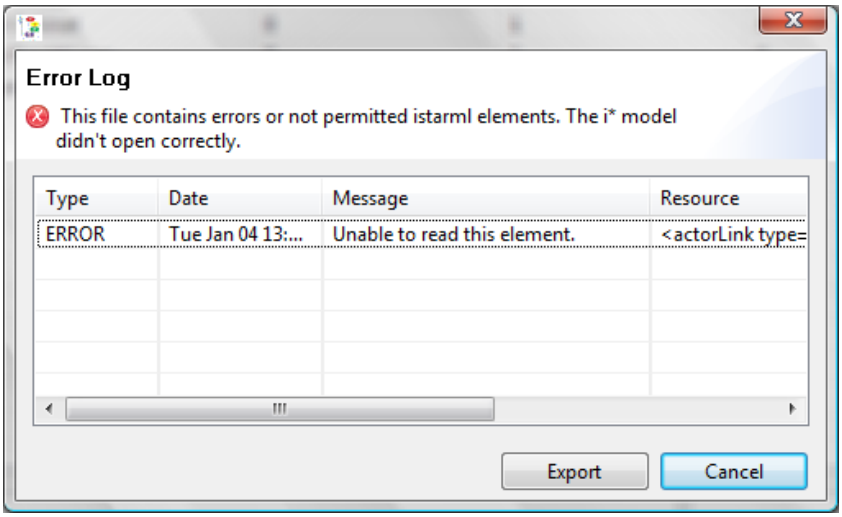


**Fig. 6. Errors Loading a Model Dialog**

Fig. 7 shows an example of file using iStarML format. Detailed information about iStarML format can be found in the iStarML web site[1].

```
<?xml version="1.0"?>
<istaml version="1.0">
  <diagram name="WER-HiME" creation_date="11/16/10" description="Example for illustrating the HiME->jUCMNave interoperability">
    <actor id="AA09" name="Claims Manager" type="role" creation_date="11/16/10"/>
    <actor id="AA10" name="Medical Practitioner" type="role" creation_date="11/16/10"/>
    <actor id="AA02" name="Physician" type="role" creation_date="11/16/10">
      <actorLink type="is_a" aref="AA10" creation_date="11/24/10"/>
      <boundary>
        <ielement name="Treat Patient" type="task" creation_date="11/16/10">
          <ielementLink type="decomposition" iref="AA01" aref="AA02" creation_date="11/16/10"/>
          <ielementLink type="decomposition" iref="AA03" aref="AA02" creation_date="11/16/10"/>
          <ielementLink type="decomposition" iref="AA04" aref="AA02" creation_date="11/16/10"/>
        </ielement>
        <ielement id="AA01" name="Diagnose Sickness" type="task" creation_date="11/16/10"/>
        <ielement id="AA03" name="Treat Sickness" type="task" creation_date="11/16/10"/>
        <ielement id="AA04" name="Bill InsurCo" type="task" creation_date="11/16/10"/>
      </boundary>
    </actor>
    <actor id="AA06" name="Medical Gabinet" type="role" creation_date="11/16/10">
      <boundary>
        <ielement name="Assess Treatment" type="task" creation_date="11/16/10">
          <ielementLink type="decomposition" iref="AA05" aref="AA06" creation_date="11/16/10"/>
          <ielementLink type="decomposition" iref="AA07" aref="AA06" creation_date="11/16/10"/>
          <ielementLink type="decomposition" iref="AA08" aref="AA06" creation_date="11/16/10"/>
        </ielement>
        <ielement id="AA05" name="Verify Patient Policy" type="task" creation_date="11/16/10"/>
        <ielement id="AA07" name="Medical Assessement Made" type="goal" creation_date="11/16/10"/>
        <ielement id="AA08" name="Fast" type="softgoal" creation_date="11/16/10"/>
      </boundary>
    </actor>
    <ielement name="Assessed(TreatmentPlan)" type="goal" creation_date="11/16/10">
      <dependency>
        <depender aref="AA09" value="critical"/>
        <dependee iref="AA07" aref="AA06" value="committed"/>
      </dependency>
    </ielement>
  </diagram>
</istaml>
```

**Fig. 7. iStarML file**

[1] http://www.essi.upc.edu/~gessi/iStarML/index.html

## 2.2 Viewing Models

*i** models are represented in the application in a tree-view form. All element links which are not decomposition or means-end are represented by the duplication of the target node, for example a dependum is duplicated in the depender and the dependee actors.

The operations that can be done on the model are grouped depending on the node where they can be initiated:

This node represents the whole diagram. The only child kind this node admits is the actor. Therefore, it is not possible to draw i* elements which are not related to an actor. The allowed operations are adding actor, edit model information and deleting all model content.

This node represents an actor of the *i** diagram. You can create several i* SR elements which are direct children of the actor. This means that an actor can have more than just one root SR element. The children the actor node admits can be either SR Elements or dependencies with other actors to get or provide a dependum. The allowed operations are creating main intentional elements, links and dependencies with other actors, editing its fields and removing it from the model.

This node represents a goal, a kind of SR Elements which represents a condition to be achieved. This can be described through goal decomposition. The allowed operations are adding a dependency link to other actor and a mean-end link with a new intentional element (goal decomposition), editing its fields and removing it from the model.

This node represents a softgoals, similar to (hard) goals except that the criteria for the goal's satisfaction are not clear-cut, it is judged to be sufficiently satisfied from the point of view of the actor. The means to satisfy such goals are described via contribution links from other elements. The allowed operations are adding dependency link to other actor and contribution links from other intentional elements in the same actor, editing its fields and removing it from the model. When a softgoal has contributions, they are represented adding the intentional elements as children as is shown in the image below. When a child node is selected, it is possible to go to the original intentional element.


This node represents a task, a kind of SR Elements which represents a way to reach a condition or a procedure that is done. A description of the specifics of the task may be described by decomposing the task into further sub-elements. The allowed operations are adding dependency link to other actor and a task-decomposition link with a new intentional element, editing its fields and removing it from the model.

This node represents a resource, an entity of the model. The actor desires the provision of some entity, physical or informational. According to i* syntax, this element cannot have any kind of decomposition. The allowed operations are adding dependency link to other actor, editing its fields and removing it from the model.

Dependency nodes are represented with intentional elements icon empty, depending on its type. A dependency can be from and to an actor or on an intentional element. In both situations, the linked actor appears as a child in the dependum.


When both strengths has the default value they are not shown (left figure), otherwise the strengths are shown (right figure).

# 3 Working with models

## 3.1 Creating Extra Properties

Some model elements have the option to create some customizable information. These elements are:

- Models
- Actors
- Intentional Elements
- Dependums in dependencies

For all these elements, the procedure is the same. Extra properties are shown as a separate tab in the creation/edition elements dialog as Fig. 8 shows.

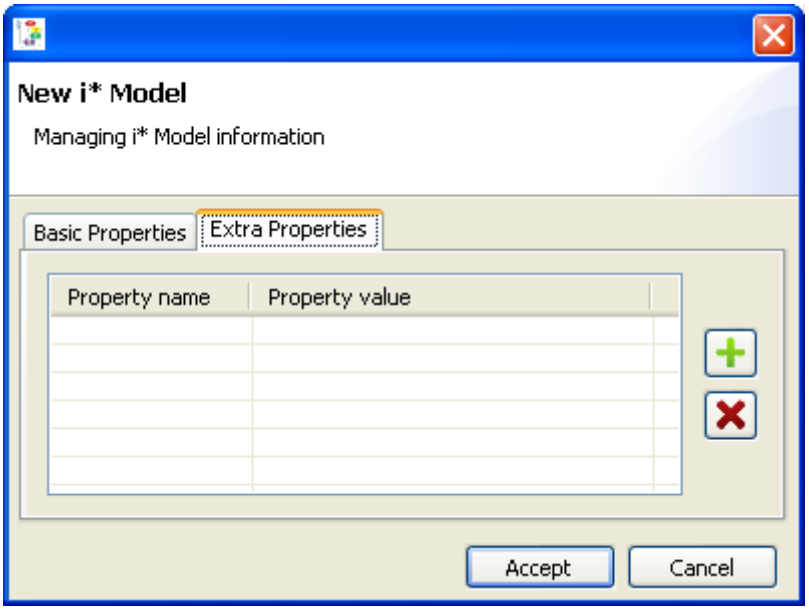


**Fig. 8. Extra Properties Tab**

This is the dialog to create a model, besides to the basic properties tab there is one for fostering the extra properties. Properties are pairs of name value that the user can define. Properties can be created or deleted using + and ✖ buttons respectively.

Once the property is created, both the name and the value can be modified directly on the grid. The following dialog appears when a new property is going to be created.

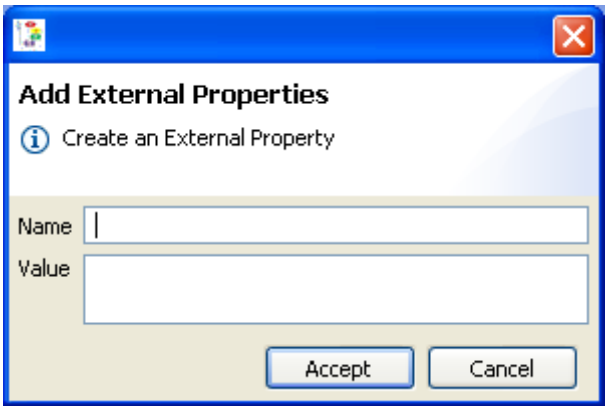


**Fig. 9. Creating Extra Property Dialog**

## 3.2 Creating an actor

This option is accessible in the popup menu over the model and it is named *Add actor...*. This is the only option to create model elements that is also accessible in the option with the same name in menu *Model*. Fig. 9 shows the dialog that appears to fill some information about the actor. Notice that actors also have extra properties (see Section 3.1).

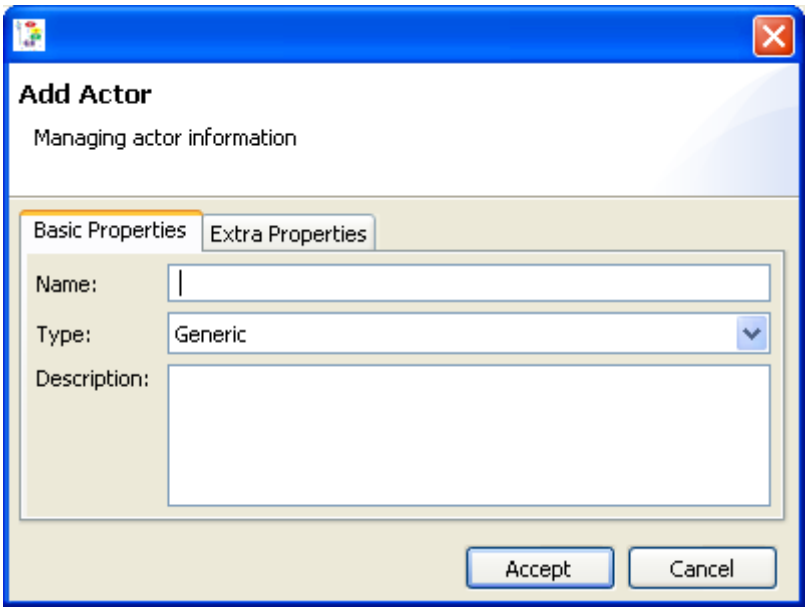


**Fig. 10. Creating Actor Dialog**

## 3.3 Creating a link between actors

There are different menu options to create links between actors. These options depend on the types of the actor that the user wants to connect.

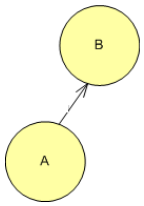


**Fig. 11. Link between Actors**

Taking into account that an actor A (source) is linked to actor B (target) as shown in Fig. 11.. The creations of link options are grouped depending if the option appears on the target or in the source. There is also a constraint that only allows some combination between types of actors. Links that the option appear on the target:

- **is-a**: both actors have the same type.
- **is-part-of**: both actors have the same type.

Links that the option appear on the source:

- **instance-of**: an agent is an instance-of an agent.
- **plays**: an agent plays a role.
- **occupies**: an agent occupies a position.
- **covers**: a position covers a role.

In all cases when the options is chosen, there are two ways to continue creating the link: creating a new "other" actor or using an existing one. If we are creating a new "other" actor, this "other" will be the source of the target of the link depending on the creation has been started from the target or source actor respectively. For example, if we are creating an is-a link, this option appears on the actor taking the role of target. Therefore, the "other" actor is the source. In the following example, the actor link is-a is created from the target ***Travel Agency*** and the result is shown in the target ***Family Travel Agency***.

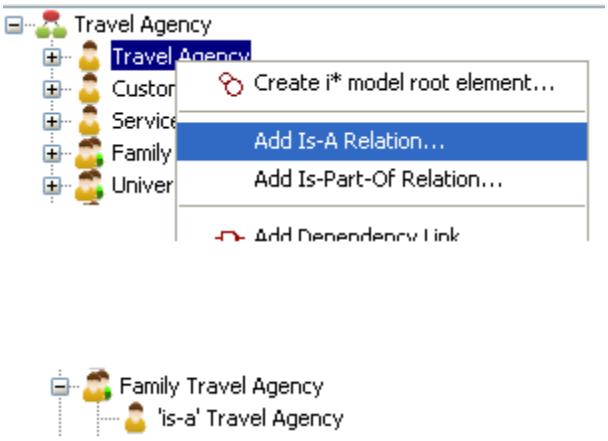


**Fig. 12. Creating is-a Link between Actors**

The link has been created using the dialog shown in Fig. 12

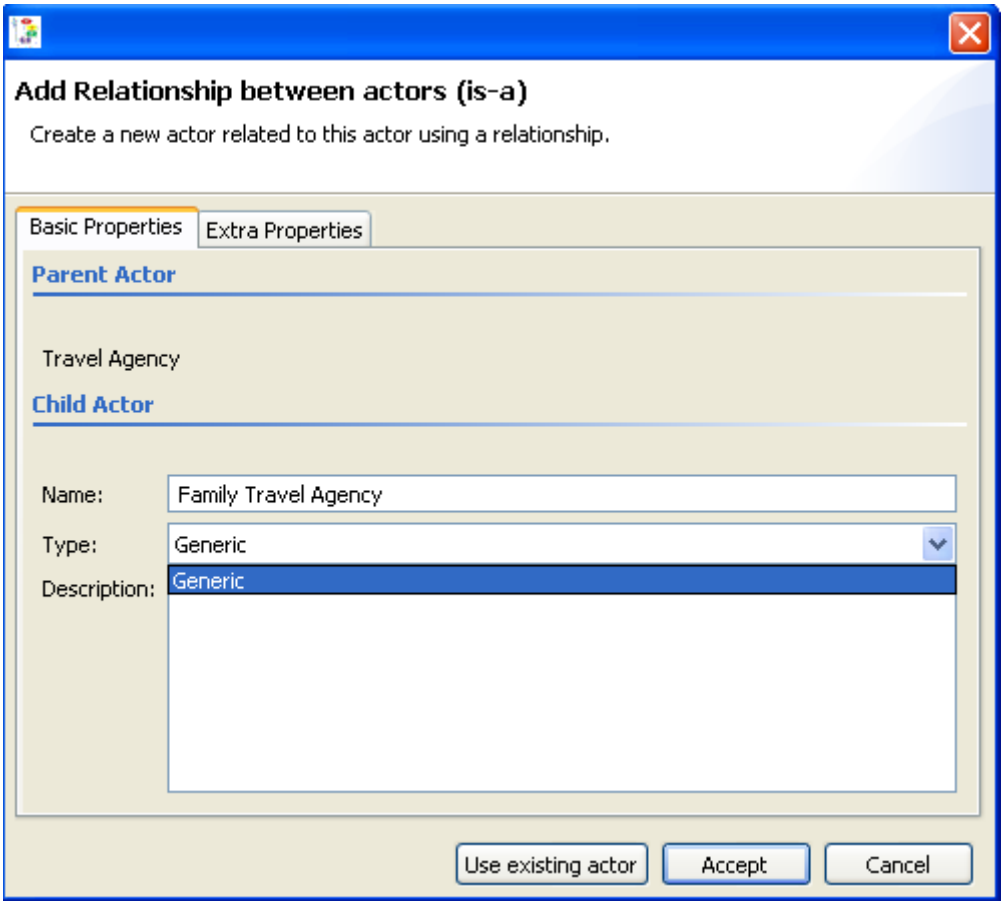


**Fig. 13. Creating Actor Link Dialog**

The value that can be selected on the field type depends on the type of the element from the link is created.

There is also the possibility of creating the link between two existing actors. In this case, in the same dialog the button ***Use existing actor*** allows the user choosing an existing actor from the model that has a compatible type.

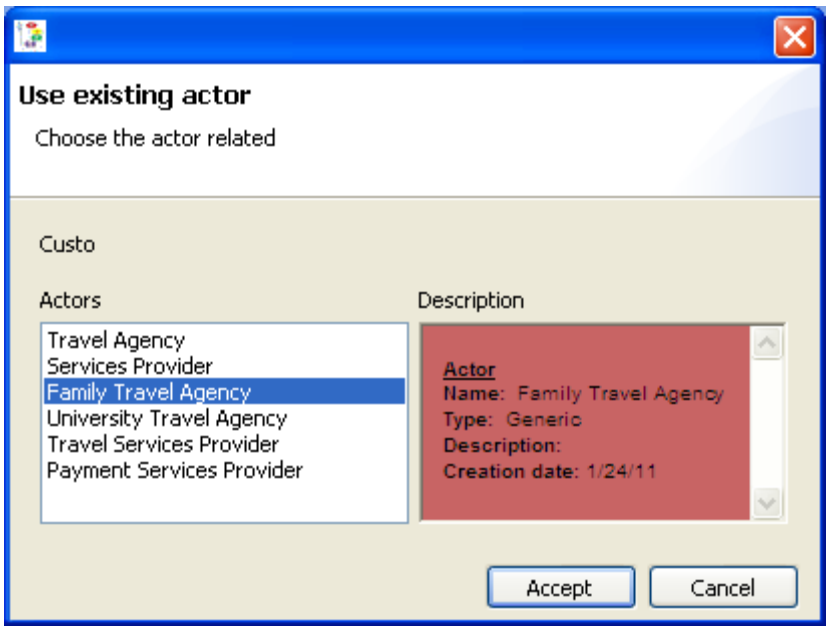


**Fig. 14. Creating Actor Link Using an Existing Actor Dialog**

## 3.4 Creating dependencies between actors

A dependency can be created at actor level or from an intentional element. For this reason, the option is accessible in the popup menu over the actors and over the intentional elements.

It is possible to create a dependency where the selected actor is the depender using the option ***Add Dependency link*** and where the actor is de dependee using the option ***Add Reversed Dependency Link***.

When the dependency is at actor level, the SR element is not selected.

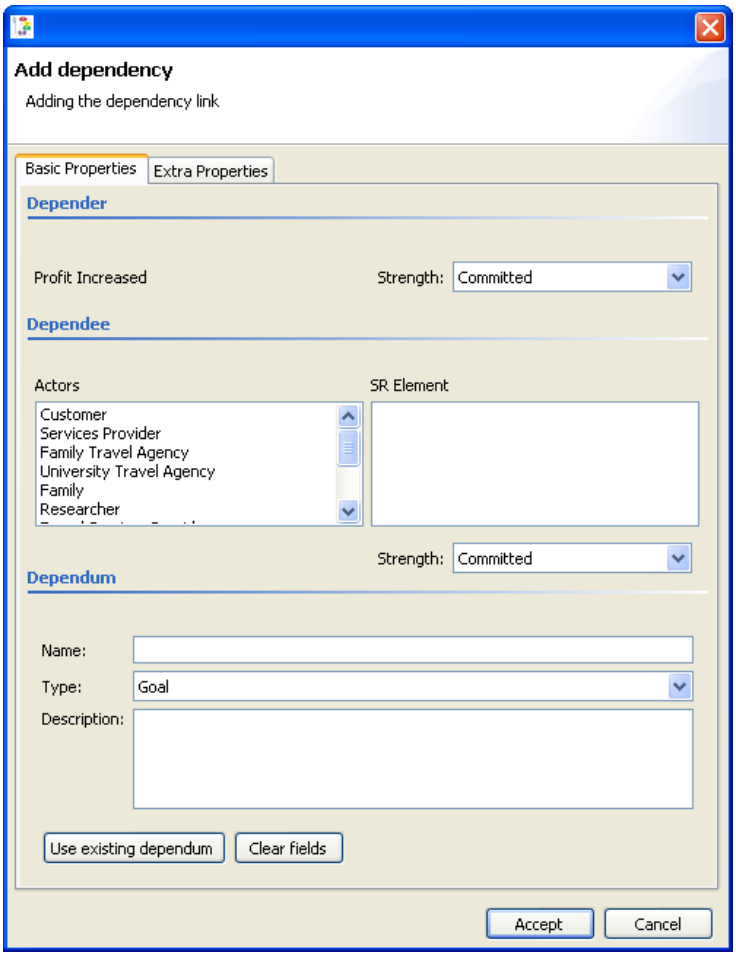


**Fig. 15. Creating Dependency Dialog**

## 3.5 Creating actor's intentional elements

The intentional elements can be represented as a tree of intentional elements. Therefore, there are an option to create a root and options to create the intermediate and leafs intentional elements.

The option to create a root intentional elements is accessible in actor nodes and it is named ***Create i* model root element***.

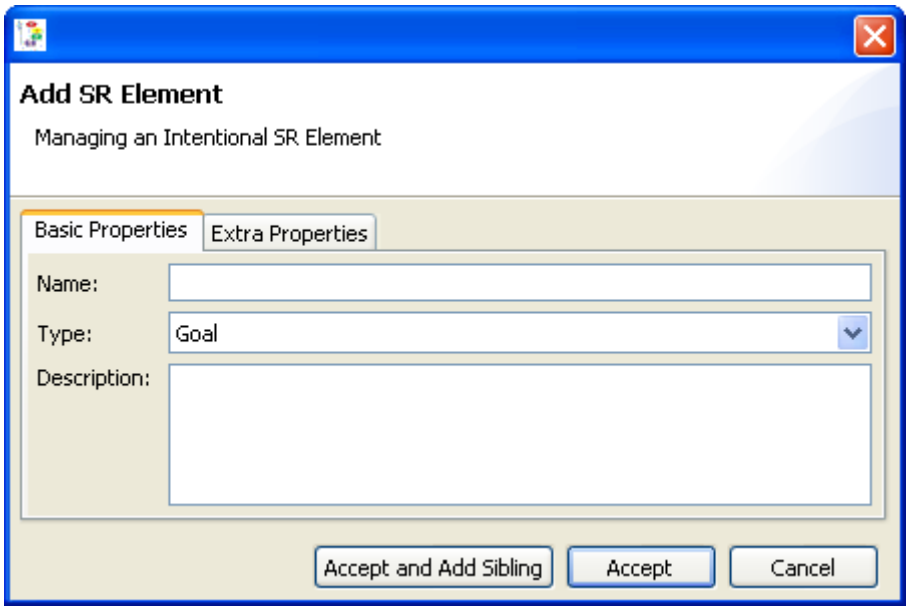


**Fig. 16. Creating Intentional Element**

Button ***Accept and Add Sibling*** can be used to create another intentional element after the element that is just created.

The option for creating intermediate and leafs is accessible in intentional elements. There are three options depending on the selected intentional element:

- ***Add Means End Link*** when the intentional element is a goal.
- ***Add Decomposition Link*** when the intentional element is a task.
- ***Add Softgoal Link*** when the intentional element is a softgoal.

Creating elements using Means-end and decomposition link use the same dialog to create the new root intentional element. The difference is the type of link created with the new intentional element.

Creating a contribution link use a different dialog because in this case the user is only creating the link between two existing intentional elements.

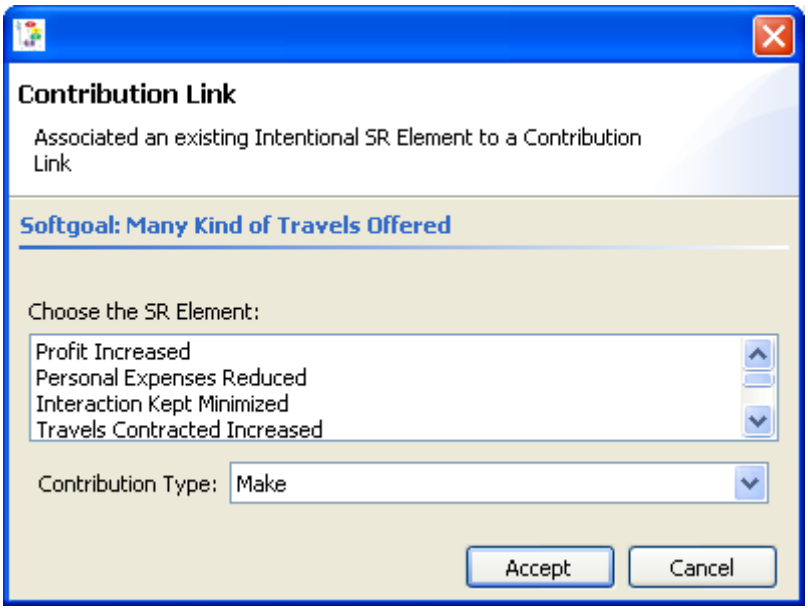


**Fig. 17. Creating a Contribution Link**

# 4 Working with Inheritance

When actor A is linked to actor B using is-a link, actor A is the subactor and actor B is the superactor. H*i*ME provides three different specialization operations that can be done over the superactor intentional elements to introduce some differences on the subactor intentional elements. These operations are:

- Refining an intentional element
- Extending an intentional element
- Redefining an intentional element

These operations are accessible in the is-a link element on the subactor.

## 4.1 Refining an intentional element

The option is accessible in the popup menu for the is-a link in the subactor and it is named ***Refine superactor element***.

Refinement is used to specialize de intentional element. In this version, refinement is only available for non-decomposed intentional elements (leafs) on the subactor. Besides the name, the type can also be changed. But, type can be only changed using the following operator: softgoal > goal > task. The type must be less or equal than the type in the superactor.

This option presents a 2-step wizard. On the first step, the intentional element to be refined is selected from the proposed list of intentional elements on the superactor that are leafs and no inheritance operation has been applied over them.

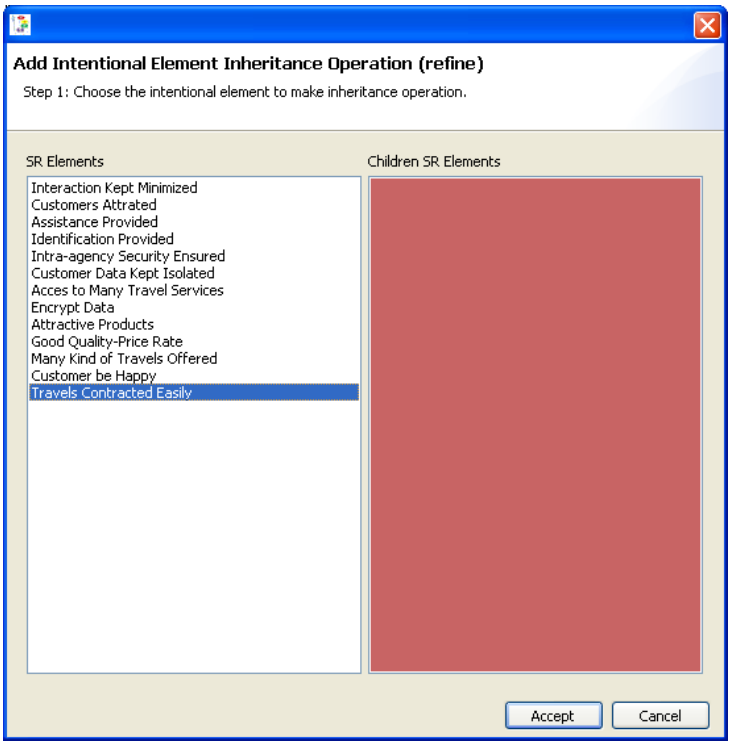


**Fig. 18. Refining Intentional Element (Step 1)**

On the second step, the user must fill the new name and change (if necessary) the type for the refined element in the subactor.

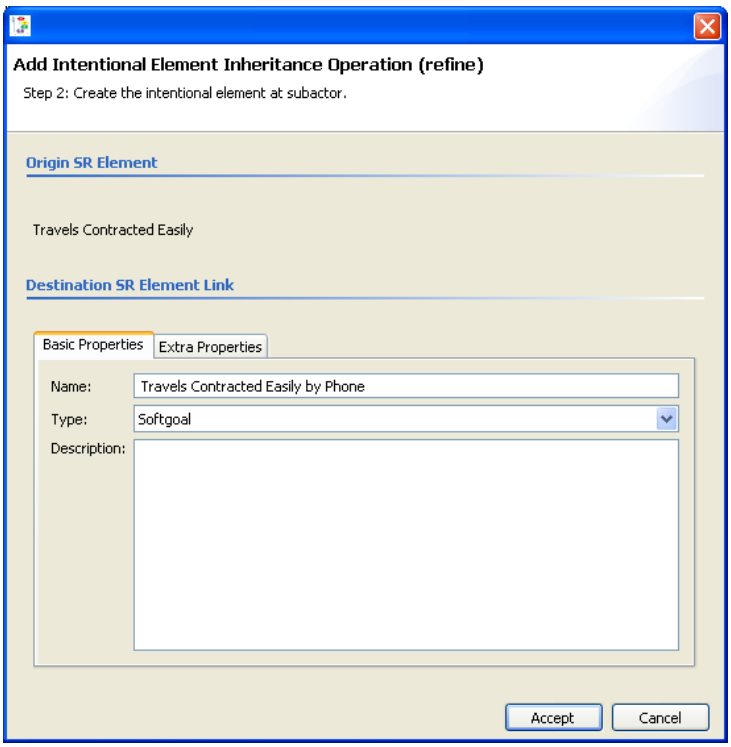


**Fig. 19. Refining Intentional Element (Step 2)**

## 4.2 Extending an intentional element

The option is accessible in the popup menu for the is-link in the subactor and it is named ***Extend superactor element***.

Extending means adding new intentional elements linked to a superactor intentional element. Link type will depend on the extended intentional element type. If a goal is extended, the link will be a means-end and if it is a task it will be a task-decomposition.

This option presents a 2-step wizard. On the first step, the intentional element to be extended is selected from the proposed list of the intentional elements on the superactor that no inheritance operation has been applied over them. In this case, when an intentional element is selected in the left list, the children that this element has in the superactor are shown in the list on the right (if it has).

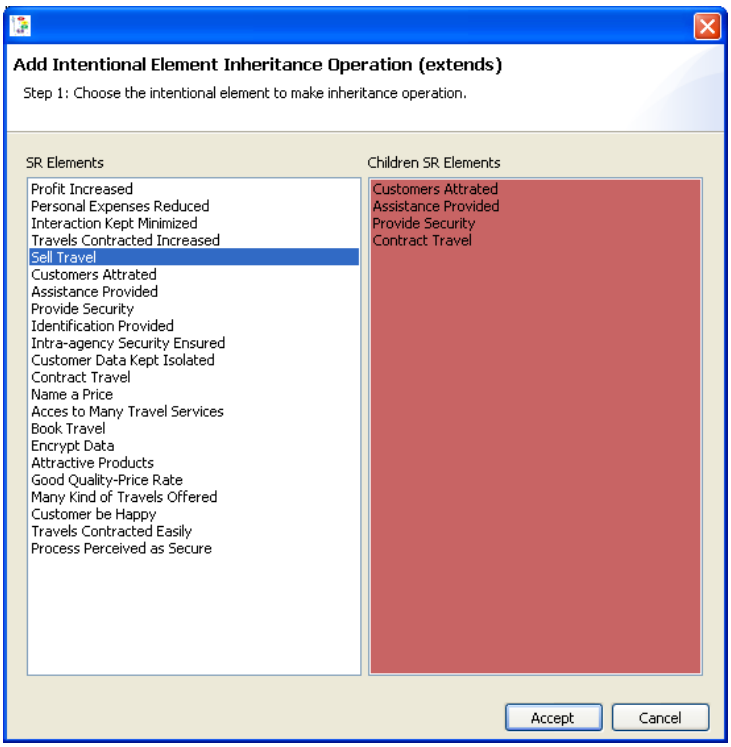


**Fig. 20. Extending Intentional Element (Step 1)**

On the second, the user will create the new intentional elements with the same dialog used for creation of intentional elements (see Section 3.5).

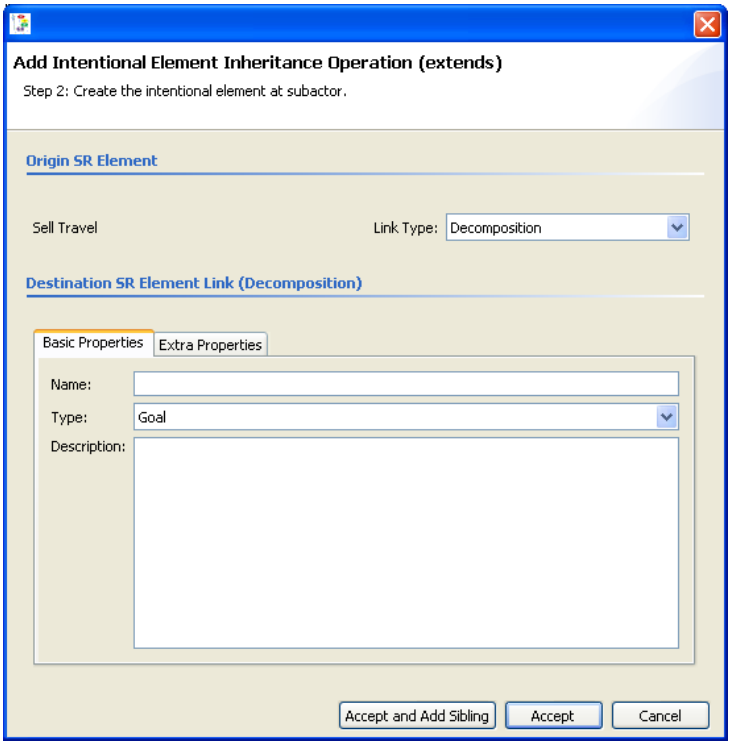


**Fig. 21. Extending Intentional Element (Step 2)**

## 4.3 Redefining an intentional element

The option is accessible in the popup menu for the is-link in the subactor and it is named ***Redefine superactor element***.

Redefining means losing the decomposition placed in the superactor and creating a new one. New intentional elements will be linked to the redefined. Link type will be depending on the redefined intentional element type. If a goal is extended, the link will be a means-end and if it is a task it will be a task-decomposition. Only decomposed intentional elements on the superactor can be extended.

On the first step, the intentional element to be redefined is selected from the proposed list of the intentional elements on the superactor that are decomposed and no inheritance operation has been applied over them. In this case, when an intentional element is selected in the left list, the children that this element has in the superactor are shown in the list on the right.

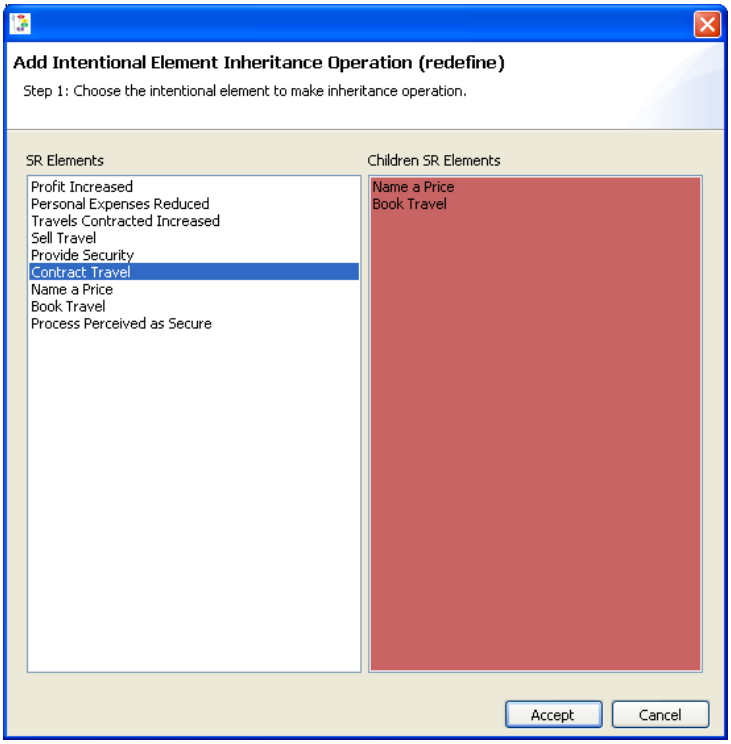


**Fig. 22. Redefining Intentional Element (Step 1)**

On the second, the user will create the new intentional elements with the same dialog used for creation of intentional elements (see Section 3.5).

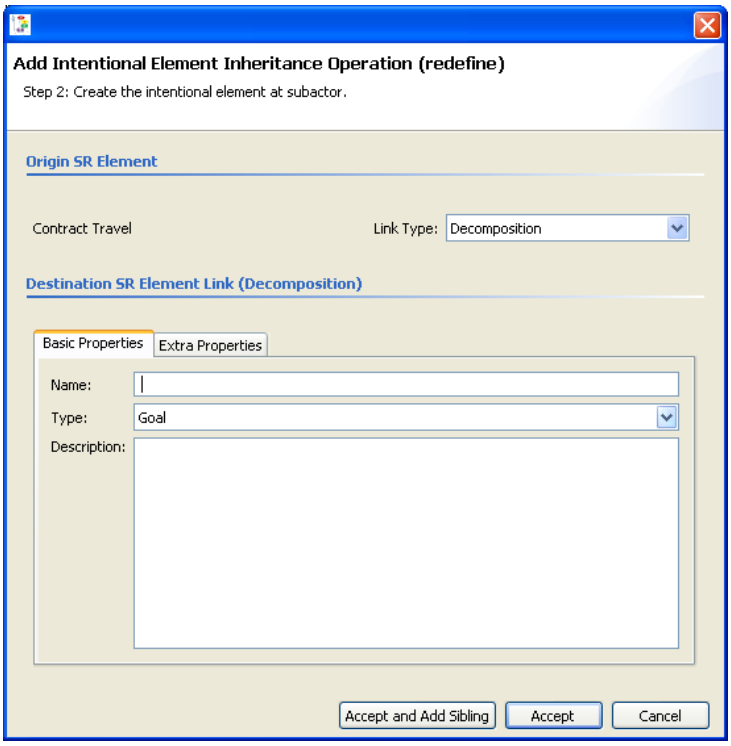


**Fig. 23. Redefining Intentional Element (Step 2)**