Title	Inheritance of Agent Permissions at Runtime
Product	Integranova Modeler
Last update	22/06/2011

1 Purpose/Target

This document describes an enhancement to the runtime behaviour of agent permissions as well as improvements in tool support for the definition of such permissions at modelling time.

These enhancements and improvements are found Integranova Modeler and Integranova Transformation Engines for both the Presentation Layer and the Business Logic.

2 Description

So far, agent permissions are defined by creating one or more interfaces, each between two classes: one playing the role of *agent class* and the other playing the role of *server class*. Then one or more Views can be defined, a View being a set of interfaces and upon transformation of a model into the Presentation Layer (e.g. the GUI) of an application, one of such Views is selected.

At runtime, a user logs on to the application as an instance of a given class (providing said class plays the role of agent class in at least one of the interfaces in the View for which the GUI of the application has been generated) and its effective permissions are those determined by the interfaces in which said class plays the role of agent.

Until now, if said user is an instance of a class with ancestors (e.g. superclasses) then the interfaces in which any of said ancestors play the role of agent class are also taken into account in order to determine the effective permissions of the connected user.

Example: suppose the model in Figure1 in which classes "Employee" and "Administrator" both play the role of agents in interfaces with class "Invoice" playing the role of server class.

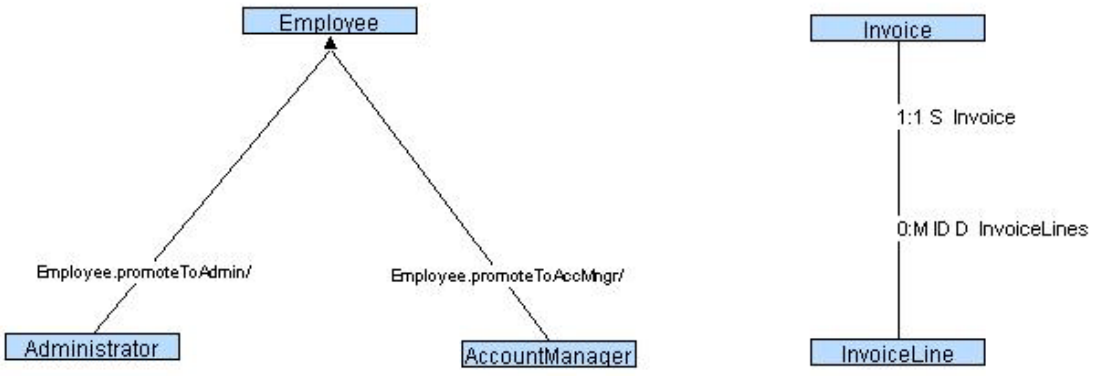

Figure 1 Sample Class Diagram

If a user logs on to the application as "Administrator", the effective permissions of said user would be those defined for class "Administrator" and those defined for class "Employee". If a user logs on to the application as "Employee", then the effective permissions would be only those defined for class "Employee".

The current situation does not take into account whether or not the instance a user logs on as an instance of is specialized (i.e.: If the user logs on as a "Employee" which is also an "Administrator" then the current behaviour fails to take into account what said user is allowed to do as an "Administrator" just because the user logged on as a plain "Employee") thus "forcing" users to log on differently depending on what they intend to do. This situation worsens as the inheritance tree for agent classes grows (i.e.: "Employee" specializes into "Administrator" and into "AccountManager" so that the same "Employee" can be, at the same time, also an "Administrator" and an "AccountManager").

Another drawback of the current situation is that, at modelling time, analysts do not explicitly see what permissions are inherited (i.e.: when defining the interface of "Administrator" with "Invoice" the analyst may not be aware of what permissions are already granted to "Administrator" as an specialization of "Employee" or, in other words, what permissions granted to "Employee" are inherited by "Administrator").

Even worse, because the inheritance of permissions is not explicitly shown by the modelling tool, analysts may define more restrictive interfaces (i.e. with less permissions) than the inherited ones.

In order to prevent these drawbacks, several enhancements have been introduced both at modelling time (i.e.: in Integranova Modeler) and at runtime (i.e.: in Integranova Transformation Engines), that apply to the definition of interfaces (modification of existing feature), the definition of views and classes allowed for logging on to the system (new feature) as well as the behaviour of the application at runtime with respect to effective permissions (modification of existing feature).

2.1 Defining Interfaces

When defining an interface (between a class playing the role of *agent* and a class playing the role of *server*) analysts define permissions from two perspectives: Vertical Visibility and Horizontal Visibility.

We refer to Vertical Visibility as the permissions defined on services (i.e. which services of the server class can the agent class execute), on attributes (i.e. which attributes of the server class can the agent class query) and on roles (i.e. which roles of the server class can the agent class traverse).

We refer to Horizontal Visibility as the constraint that defines which instances of the server class can be queried by the agent class (i.e.: the agent class will only see the instances of the server class that fulfil the Horizontal Visibility constraint).

With respect to Vertical Visibility, Integranova Modeler will show the permissions inherited by the agent class: those defined in interfaces with the same server class and any ancestor of the agent class.

Example: suppose the model depicted in Figure 1 above and the definition of interfaces depicted in Figures 2 and 3 below. In this case, when defining the interface of agent class "Administrator" with server class "Invoice"

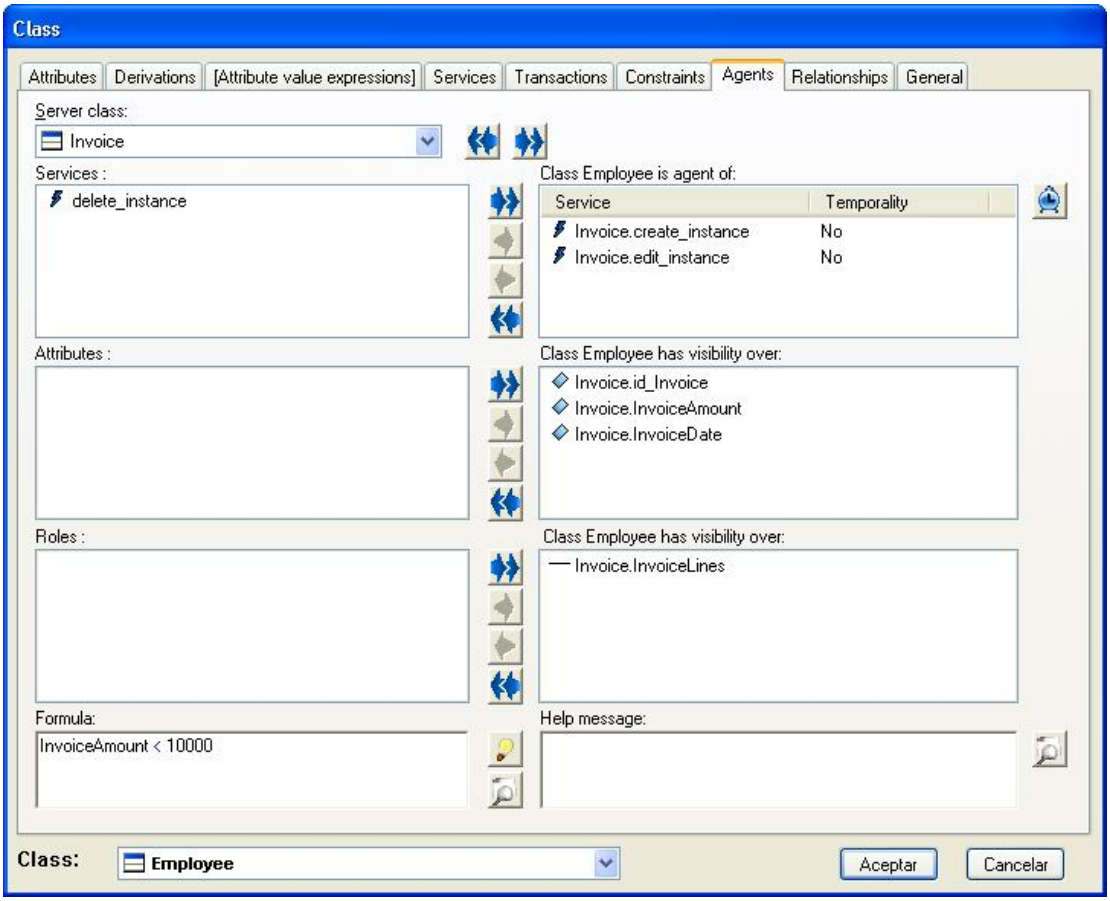

Figure 2 Interface of agent class "Employee" with server class "Invoice"

According to the interface definition depicted in Figure 2, an "Employee" is allowed to query all attributes of "Invoice" ("id_Invoice", "InvoiceAmount" and "InvoiceDate") and traverse all its roles ("InvoiceLine") and is allowed to execute all the services "create_instance" and "edit_instance" but is not allowed to execute "delete_instance".

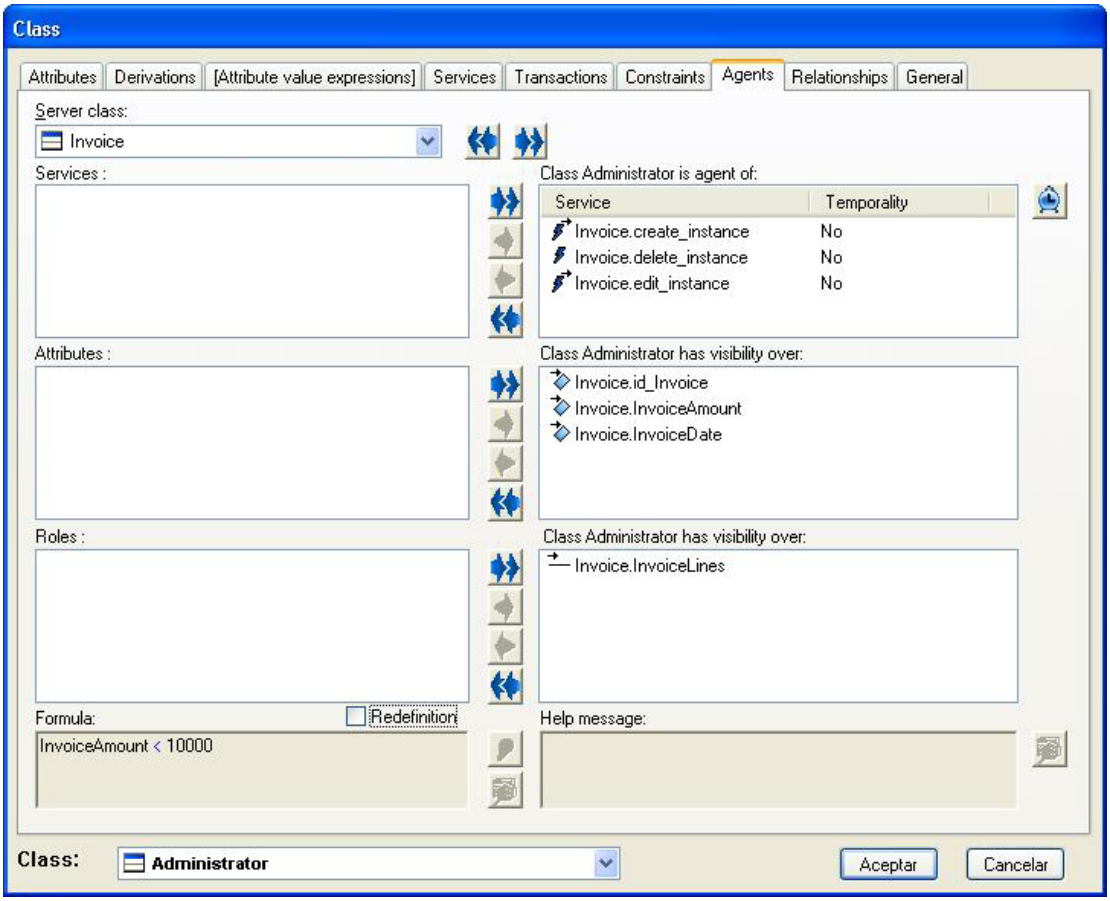

Figure 3 Interface of agent class "Administrator" with server class "Invoice"

In accordance with the interface definition depicted in Figure 3, an "Administrator" is allowed to query all attributes of "Invoice" and traverse all its roles. These permissions are inherited from the interface of its ancestor ("Employee") with "Invoice". Notice the icons include a little arrow to denote the permission is inherited (for attributes and for roles). An "Administrator" is allowed to execute services "create_instance" and "edit_instance" because these permissions are inherited (again notice the change in the icon to denote this circumstance:) and an "Administrator" is also allowed to execute service "delete_instance". In this case, the permission has been explicitly added (i.e. by the analyst) to the interface of "Administrator" with "Invoice", whereas the permissions to execute the other two services are implicitly added (i.e.: by Integranova Modeler) to the interface of "Administrator" with "Invoice" as a result of "Administrator" being a specialization of "Employee" which defines these permissions in its interface with "Invoice".

With respect to Horizontal Visibility, Integranova Modeler will show the Horizontal Visibility constraint inherited from the immediate ancestor of the server class (if any) and it will also allow analysts to redefine such constraint.

Example: Figure 3 above depicts the definition of the interface of agent class "Administrator" with server class "Invoice". Notice that the textbox under "Formula" (where the Horizontal Visibility constraint is specified) is greyed out and has the same formula defined in the interface of agent class "Employee" with server class "Invoice" as depicted in Figure 2.

In case the analyst wants to redefine the Horizontal Visibility constraint for class "Administrator" instead of having it inherit the constraint defined in its "Employee" ancestor, all she has to do is check the "Redefinition" checkbox and enter the new constraint formula. If the formula is left blank (as depicted in Figure 4 below) the agent class will be allowed to query all instances of the server class.

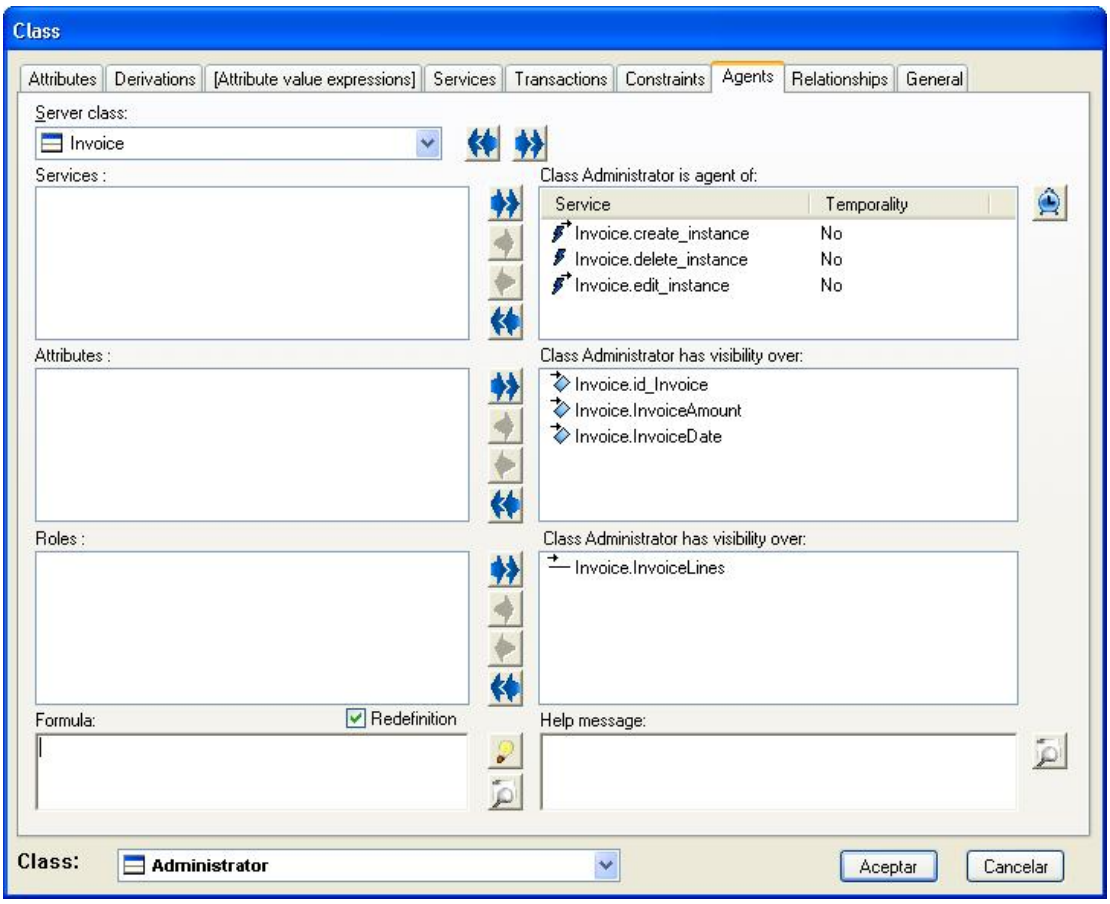

Figure 4 Interface of agent class "Administrator" with server class "Invoice" with redefinition of the Horizontal Visibility constraint

2.2 Definition of Views and selection of classes that can log on to the system

Views are defined as a set of interfaces. This task is performed in the same way. The difference now is that Integranova Modeler completes the definition of the View by adding inherited interfaces.

This means that if, when defining a View, the analyst explicitly adds an interface of a given agent class "A" with a given server class "S", then Integranova Modeler will automatically add to that View all interfaces with server class "S" in which the agent class is any ancestor of "A".

Example: if the analyst adds to the View the interface of agent class "Administrator" with server class "Invoice", then Integranova Modeler completes the definition of the View by adding the interface of agent class "Employee" (ancestor of "Administrator") with server class "Invoice".

An enhancement to the definition of Views is the ability to specify which classes can be

used to log on to the system, so the analyst explicitly picks which of the classes playing the role of agent in any of the interfaces of the View can be used by a user to log on to the system.

Example: according to the interfaces included in the View, classes "Employee", "Administrator" and "AccountManager" play the role of agent class so a user can log on to the system either as an "Employee", an "Administrator" or an "AccountManager". The analyst can then decide that users will only log on to the system as employees, as depicted in the lower section of Figure 5 below.

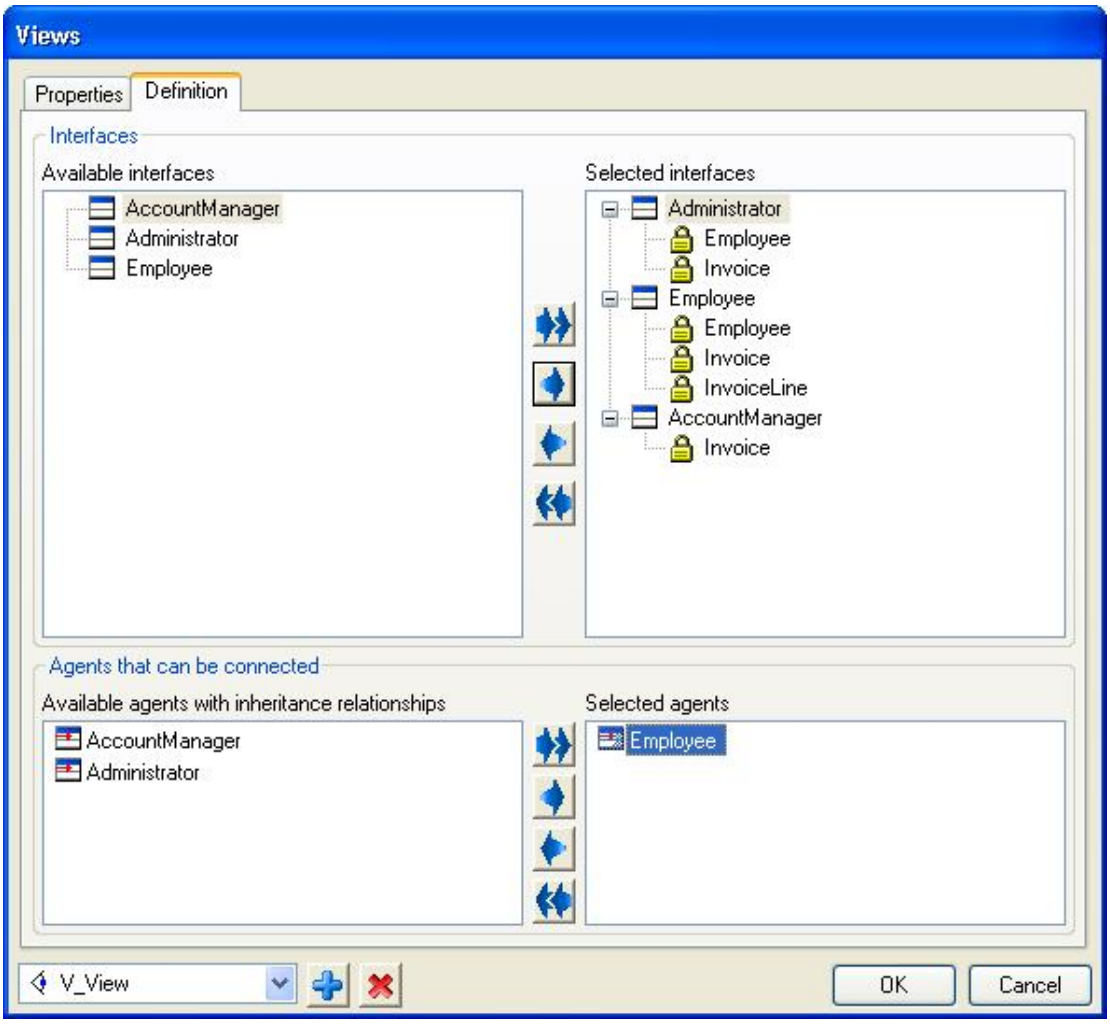

Figure 5 Interfaces in a view and classes that can log on to the system

Notice that, apart from the interface with "Invoice", there are no explicit interfaces of agent "AccountManager" with "Employee" and "InvoceLine". This means that "AccountManager" has exactly the same permissions than its immediate ancestor ("Employee") with server classes "Employee" and "InvoiceLine".

2.3 Effective permissions at runtime

At runtime, effective permissions now take into account not only what agent class the user logged on as an instance of, but also what facets are active at the time of log on.

Example: according to the definition of the View depicted in Figure 5 above, users can only log on to the system as instances of agent class "Employee". At runtime, upon logging on to the system, a given user can be just an "Employee", an "Administrator", an "AccountManager" or both. The effective permissions of said users will be determined by its active facets at the time of log on.

In case the only active facet is "Employee" then said user will only be granted the permissions defined in the interfaces of the View where "Employee" plays the role of agent class.

In case facets "Employee" and "Administrator" are active, then the effective permissions will be those defined in the interfaces of the View where "Employee" plays the role of agent class and those defined in the interfaces of the View where "Administrator" plays the role of agent class.

With respect to Horizontal Visibility constraints, the one effectively applied results from the union of the constraints defined in all active facets that are not specialized further at runtime.

Example: if, upon logging on to the system, an "Employee" is also an "Administrator" and an "AccountManager" the effective Horizontal Visibility constraint will be the union (e.g. OR) of the constraints defined in the interfaces in which "Administrator" or "AccountManager" play the role of agent class. In case of querying the population of "Invoice" the user would be allowed to see all instances satisfying the constraint defined in the interface of "Administrator" with "Invoice" (see Figure 4 above) or the constraint defined in the interface of "AccountManager" with "Invoice" (see Figure 6 below).

Notice that a blank constraint (as is the case of Figure 4) is equivalent to specifying a tautology (i.e. a formula that will always evaluate to the logical value *true*) so in this case even if there is a constraint in the interface of "AccountManager" with "Invoice", the user will be able to query all instances of "Invoice".

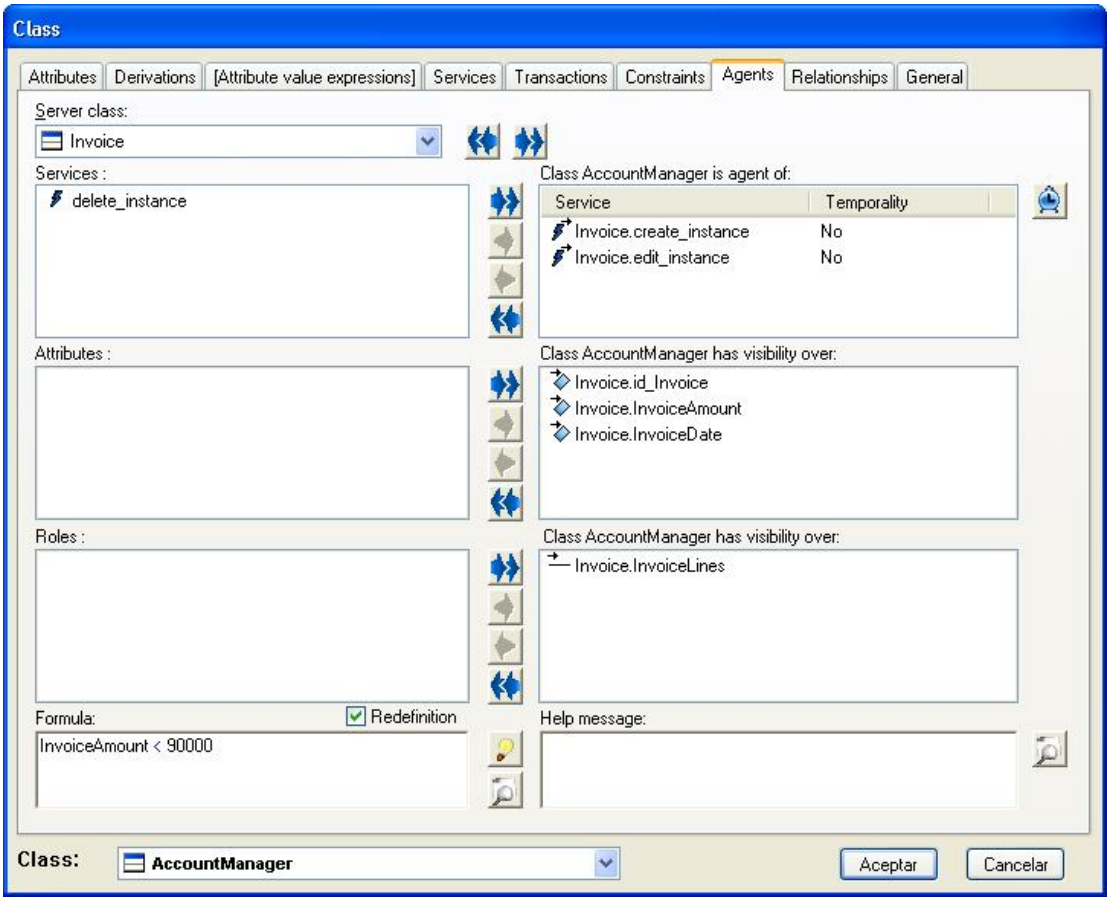

Figure 6 Interface of agent class "AccountManager" with server class "Invoice" with redefinition of the Horizontal Visibility constraint

3 Modelling considerations

With these enhancements to the runtime semantics of agent permissions, classes in the inheritance hierarchy of agent classes can be viewed as "(security) roles". Therefore, analysts can grant the topmost class in the hierarchy the basic permissions and then grant each descendant agent class the permissions specific to the (security) role represented by said agent class.

Carrier and liberator events can be used at runtime to assign/de-assign users to different roles (i.e.: having an "Employee" become "Administrator") and thus more effectively managing user privileges at runtime without the need to changing the model and re-generating the application (i.e.: if an "Employee" user becomes "Administrator", next time said user logs on she will be granted the permissions of "Administrator" too). And since an instance of a class can specialize into multiple descendants (i.e.: the same "Employee" can be both "Administrator" and "AccountManager") a single user can be assigned to multiple (security) roles.