

It is provided for `open_allowed` and `update_allowed` so that they can redirect the decision to the version selection object.

19. `vso_open`: This method verifies that the version selection object allows subobjects it controls to be viewed by the user.

20. `vso_update`: This method verifies that the version selection object allows subobjects it controls to be updated by the user.

The `find_object` method (defined on `EKNAA201`) is used to find objects for cross object verifications. Given the source logical key and source sequence, and the target logical key, which may be in a different complex object structure or perspective, it determines the "best" version of the object matching the target key to get, and retrieves it. A class reference to the type of object desired is also passed in, so that find methods can be sent to it. In `EKNAA201`, this method uses the passed sequence number and the new logical key, and invokes the `find_applicable` method. In `EKNAA300`, it is specialized to make decisions based on the perspective differences indicated by the old key and new key, and the type of decision, which is determined by the `status_type` method.

The following decision patterns are supported:

1. DECISION "A": The passed sequence number is globally valid for all complex objects in the application. The data should be selected using the new key, the passed sequence, and `find_applicable`.

2. DECISION "B": The passed sequence number is only valid within the scope of the complex object identified by the old key. However, the name of the associated version, identified by the version selection object, makes sense across other complex objects. If there is a version selection object with matching name on the complex identified by the new key, use its sequence number, and the `find_applicable` method, to select data for the new key. Otherwise, find the version selection object which is the best match by status (and possible effectivity) and use `find_applicable`.

3. DECISION "C": The passed sequence number is only valid within the scope of the complex object identified by the old key. However, the name of the associated version, identified by the version selection object, makes sense across other complex objects. If there is a version selection object with matching name on the complex identified by the new key, use its sequence number, and the `find_inserted_by` method, to select data for the new key. If the data is not directly affected by the version selection object, or there is no version selection object, find the version selection object which is the best match by status, (and possibly effectivity) and use `find_applicable`.

4. DECISION "D": The passed sequence number is only valid within the scope of the complex object identified by the old key. As an additional complication, the new key identifies a different perspective. The name of the associated version, identified by the version selection object, makes sense across other complex objects in the perspective associated to the old key (but not necessarily in the perspective indicated by the new key). If there is a version selection object with matching name on the complex object identified by the new key, but in the perspective identified by the old key, use its sequence number, and the `find_inserted_by` method, to select data for the new key. This also requires that the new key be modified to select data in the perspective indicated by the old key. If the data is not directly affected by the version selection object, or there is no version selection object, find the version selection object which is the best match by status (and possibly effectivity), for the

complex object identified by new key in its perspective, and use `find_applicable`.

5. DECISION "E": The passed sequence number is only valid within the scope of the complex object identified by the old key. The version name is not of any significance across complex objects. Find the version selection object which is the best match by status (and possibly effectivity), for the complex object identified by new key in its perspective, and use `find_applicable`.

The presentation of version controlled objects is a specialization of the presentation of complex objects. The initial selection panel of lists of complex objects will also include version selection for those objects. This may be done by including user version selection criteria on the selection popup which precedes the list panel, or by having the list panel show several versions of the same complex object. FIG. 18 illustrates an example using the purchase order complex object. A list of purchase orders would be selected from a main menu panel. The selection popup restricts the purchase orders that appear in the list. In this case, several versions of the same complex object appear in the list. Note that information about the version selection, in this case a version selection object for version 2, is included in the ruler list passed to the initial parts panel. This version information becomes part of the focal point data for all related panels, and the version selection object is included in all subsequent ruler lists.

Complex object support provides generic support for using stream and find methods to open list and data entry panels from the parts panel. These methods are further refined here to make use of the version control information contained in the ruler list. The `ruler_seq` method on the version control metaclass is used to get the sequence number associated to the version. This is then used to find the objects applicable at that version (either one or a stream).

The change method in `EKNAA202` allows an informal version to be established based on data in a formalized version. The `update_allowed` method in `EKNAA202`, `EKNAA201` and `EKNAA300` is used to block updates using a formal version. The `promote` method in `EKNAA202` is used to promote a version from informal to formal status. The promote verifications can be used as a part of the approval process. The `undo` method in `EKNAA202` can be used to remove the affect of a version which is not approved.

The use of insert and extract sequence numbers to select data to be presented at a particular version means the same data may be applicable at many different versions. The change method in `EKNAA202` will create a new version of a simple object when that version actually changes the data in the simple object. Other simple objects belonging to the same complex object will not be affected. It may happen that an object is not directly affected by a version, but its subobjects are. The selection of objects for complex objects actions such as delete, promote and undo is adjusted for this, through the use of the "delete_at" and "affected_at" streams.

In the drawings and specification, there have been disclosed typical preferred embodiments of the invention and, although specific terms are employed, they are used in a generic and descriptive sense only and not for purposes of limitation, the scope of the invention being set forth in the following claims.

That which is claimed:

1. A system for controlling versions of selected objects in an object oriented computing system on a computing platform, each object including an object frame containing data attributes and at least one object method for performing