

indicates an objects category), and 6) query results. Smart containers may leverage any of these in order to best represent the logical business object. Structure in compound objects should describe important attributes, hierarchy, relationship, and version binding. In some embodiments, version binding describes which version of a document should be referenced. The binding options can be specified with the following options: 1) symbolic version label, such as “Approved”, or “Current”, etc. 2) explicit version label such as “1.0” or “1.1”, etc. 3) early binding meaning associate the document of the specified version at design time, and/or 4) late binding meaning associate the document of the specified version at runtime. In **602**, structure links between an object and other objects are specified. For example, one object is a root folder and other objects are folders or documents within the root folder or other folders. In some embodiments, **602** may be applied/defined differently for the various nodes or objects of the smart container.

[**0022**] FIG. 7 is a flow diagram illustrating an embodiment of a process for defining policies. In some embodiments, process **404** of FIG. 4 is implemented by FIG. 7. In various embodiments, a policy comprises a retention policy that is associated with a root folder or other folder, check-in rules for member documents (for example, do not allow same version on check-in), auto-naming rules for smart container and member objects (for example, object names are assigned when created, when there is a lifecycle state change, etc.), role based privilege policy, rule for when placeholder should be replaced, policy for placement of an incoming member object within a runtime structure, workflows (e.g., a workflow defines the activities and flow of a business process, for example a loan application workflow would automate the flow of a loan file through the loan approval process.), specific to smart container or member objects, lifecycles for smart container or member objects (e.g., a document typically moves through different stages in which it may have different meta-data values, security, folder location, etc., and these stages can be modeled and automated with a lifecycle—one example of a lifecycle is the typical draft, review, approve stages of a document), content storage policy, distributed content configuration policy, content caching policy, full text indexing policy, etc. In **700**, a policy is specified. In **702**, the scope and inheritance of the policy is specified. The scope of a policy determines to which objects of a smart container a policy will be applied when it is evoked at instantiation or runtime. When a policy is associated to an object which may contain children (e.g. a folder or object with relationships to ‘child’ objects), the user can specify if the given policy is to be inherited, how deep the inheritance applies, and whether or not the policy can be overridden or narrowed (further constrained) by policies associated to children objects. In **704**, the policy conflict resolution is specified. It is possible, if not likely, that a smart container or its contents has associated policies that conflict. For this reason a mechanism for conflict resolution is specified. A “static resolution” mechanism may be invoked to resolve the conflict at time of association. A “dynamic resolution” mechanism may be invoked to resolve the conflict at time of instantiation and runtime (i.e., when a given policy is applied or evoked). The appropriate mechanism depends on the given policy and association. In **706**, object(s) or structure links associated with policy are defined. In some embodiments, **706** is the first step in the flow—for example, an object class is defined;

policies are associated with the object class; and policy inheritance and conflict resolution are defined. In **708**, it is determined if the process is done. If not, then control passes to **700**, otherwise the process ends.

[**0023**] FIG. 8 is a flow diagram illustrating an embodiment of a process for defining roles. A role can be assigned a set of users and a set of entities and services that users in that role can access. In some embodiments, process **406** of FIG. 4 is implemented by FIG. 8. In **800**, a role is specified. For example, in the case of a loan processing two of the roles included are an account executive and a loan closing specialist. The account executive is responsible for processing a specific aspect of a loan application such as reviewing and validating incoming documents and recommending whether to deny the loan or send it on for further consideration. The loan closing specialist is responsible for the final review of all documents, insuring compliance with legal and regulatory requirements, funding the loan, and promoting the loan file to the “closed” state. There is a ‘Fund Loan’ workflow defined that automates the process of funding a loan. The ability to start this workflow should be available to the loan closing specialist but not to the account executive. In addition, the Fund Loan workflow should only be available when a loan file is selected and not when some other object of the smart container is selected. In **802**, object(s) or structure links associated with role are specified. Zero or more roles can be applied to an Object in a smart container in order to constrain the domain of users that can operate on instances of that Object and to define the services and entities that are accessible to each user within the context of that object. In **804**, it is determined if the process is done. If not, then control is passed to **800**, otherwise the process ends. In some embodiments, an object class is defined and then roles are associated with the object class. In some embodiments, the process flow is 1) define a role including its members and services and entities available to users of that role, and 2) associate role to object class.

[**0024**] FIG. 9 is a flow diagram illustrating an embodiment of a process for instantiating a smart container using a smart container template or other instance as a source. In some embodiments, process **302** of FIG. 3 is implemented using the process of FIG. 9. In the example shown, in **900** the source for creating an instance is specified. In various embodiments, a new smart container can be created from a saved smart container template, or from another runtime instance. In **902**, template parameter values and other instantiation-time modifications for the instance are specified. Template parameters are initial values (e.g., content, meta-data, or policies) that change from instance to instance and are required in order to create a new instance. For example, when creating a room smart container from a template, the user may need to select the initial members; when creating a loan file smart container from a template, the loan processor may need to specify an originating document, i.e. loan application; or when creating a finished goods specification smart container, the user may need to enter the name and description of the end product. A Smart Container cannot be successfully created until values have been supplied for all required parameters. In some embodiments, auto-generated forms will be used to prompt users for parameter values. In **904**, instantiation actions are executed. Instantiation may result in one or more of the following actions: object creation, replacing placeholders with actual objects, policy evaluation and execution including actions