

[0440] Document workflow requires data storage for documents and the recording of a documents progress through a series of steps. The document workflow must additionally create a process entity model that is written to the database each time a step is started or completed. The computer executes some steps. People who perform some task and enter additional data into the computer complete other steps. Frequently, one or more steps involve approval of a document in order to progress to the next step.

[0441] Document Workflow essentially uses the scheduling service of a component integration tool to scan for records in a database table periodically. Any record entering into an automated step is retrieved through object-relational mapping and given as a parameter to the step, which is an operation implementation.

[0442] If the step is not automatic, the document workflow generally must provide a way to enter data into an object model that is then stored in the database before marking a step complete. The object structure and mapping to a database has already been discussed. Providing a mechanism for user input can be accomplished through web services by using the serializer concept to generate an HTML form directly from the object description, then setting the attributes of the object from the values entered on this form. Alternatively, a client-server program could create form elements based on the model descriptors and set the values directly on the object from those form elements. Once entered from a form, the object data and the process entity for the step can be recorded.

[0443] FIGS. 5-34 is a multi-part diagram illustrating how a component integration engine of the present invention operates. In FIGS. 5-34, the "user" is an end user.

[0444] After a developer defines the software models that are part of each part of each application, the system administrator can configure the parts of the Component Integration Engine of the present invention of FIG. 5. The parts of the Component Integration Engine are based on standards to allow the use of existing and new components developed by third parties or by the programmers who program the component integration engine. In many cases, the components of the Component Integration Engine, such as: databases, email servers, etc. are already owned by the company for whom the system administrator works or the components are provided as part of the Component Integration Engine to the system administrator. The Component Integration Engine is open to allow new components if existing components are not a good fit.

[0445] As illustrated in FIG. 5, a developer defines the descriptors of a metamodel repository of a component integration engine of the present invention. A system administrator configures authentication instances, access point instances, shared service instances and persistence engines that use authentication components, user access components, service components and persistence components, respectively. A 3<sup>rd</sup> party company and a programmer are both able to write to authentication components, user access components, service components and persistence components.

[0446] In FIG. 6, with the Component Integration Engine configured and the application models available, a developer may now create flowchart assemblies. The flowcharts

describe various responses to events that may occur. Some events may originate from a user and other events may originate from automatic triggers. The flowchart may create or dispose of model instances, invoke methods, generate signals or perform other necessary work. The flow chart assembly may also call upon shared service or component assemblies as necessary. The user access point provides a mechanism for organizing these flow charts in a way that the user may execute them to perform some useful activity. This organization of flowcharts, services, models, and instances creates a software application.

[0447] As shown in FIG. 6, a developer creates flowcharts in the flow chart assembly(s) of the Component Integration Engine. A model instance of the flowchart assembly(s) uses component assemblies. A method instance of the flowchart assembly(s) also uses component assemblies. A constraint instance of the flowchart assembly(s) uses authentication instances. A method instance of the flowchart assembly(s) uses shared service instances. A method instance of the flowchart assembly(s) uses persistent engines.

[0448] In FIG. 7, a user of a software application is unaware of the Component Integration Engine. The user accesses the software application through a user accesspoint, such as: a client-server program, a socket, a web page, an email message, etc. The user access point determines the correct flowchart assembly to invoke. The parts of this assembly are free to create new instances of any of the existing implementations, invoke operations, or use any of the shared services or resources. The results from the flowchart assembly are returned to the user access point and the display to the user is updated appropriately.

[0449] As shown in FIG. 7, a user sends an application signal by clicking a button, entering data, etc. to user access point instances of the Component Integration Engine. The user access point instances calls the flowchart assembly(s) and the flow chart assemblies return results to the user access point instances.

[0450] FIG. 8 is a static diagram showing the relationships between classifiers such as: interfaces, datatypes, models, etc. of the Component Integration Engine. One or more static diagrams may be converted to descriptors to be stored in the metamodel repository of the meta-implementation layer of the Component Integration Engine. To convert from UML to the metamodel repository, the metamodel repository contains descriptions of the UMS metamodel elements.

[0451] In the following figure descriptions, the use of <1>, <2>, <n> following an element name in a figure indicates that zero-to-many instances of that named element may exist in the containing element. In some instances only the <1> and <n> or just the <n> appears; the same meaning is intended in these cases, but for reasons of space or clarity the <1> and <2> are omitted.

[0452] In the figures described below, "handler" refers to a component that understands a certain format of input and converts that input into the appropriate object structure as its output. In the figures described below, many handlers produce descriptors and source from an input object that has the same first part of the name as the descriptor but no following tag. These are "descriptions" originating from an external source and may vary in format. In one embodiment of the