EXPOSING PROCESS FLOWS AND CHOREOGRAPHY CONTROLLERS AS WEB SERVICES

COPYRIGHT NOTICE

[0001] A portion of the disclosure of this patent document contains screen images that are subject to copyright protection. The copyright owner has no objection to the facsimile reproduction by anyone of the patent document or the patent disclosure as it appears in the Patent and Trademark Office patent file or records, but otherwise reserves all copyright rights whatsoever.

BACKGROUND OF THE INVENTION

[0002] The present invention relates to computer-based devices and methods supporting document exchange choreographies. More particularly, aspects of the present invention relate to devices and methods that facilitate evolution of systems by various combinations of choreography versioning, service versioning and document versioning. It provides for choreography management using a choreography agent and presents choreography-enabled interfaces to non choreography enabled applications. Additional aspects of the present invention include a graphical design tool and transparent aliasing of a host service as multiple context setting franchised services. Particular aspects of the present invention are described in the claims, specification and drawings.

[0003] Business-to-business (B2B) and application-to-application (A2A) electronic commerce are replacing former protocols for electronic data interchange (EDI). As businesses strive to improve their efficiency with B2B and A2A systems, a number of incompatible platforms and competing standards have emerged. Among compatible standards, gaps remain to be filled. For instance, the industry has defined what a simple web service is. Standards related to simple Web service include UDDI, WSDL, XSDL and SOAP. However, these standards do not fully meet the security, reliability, manageability, and choreography requirements for practical B2B and A2A electronic commerce. In this context, choreography of message exchanges among services includes setting a pattern for the choreographed exchange and tracking messages that are part of the exchange with an identifier, such as a conversation ID. Choreography in particular presents competing platforms with numerous options and

configuration issues. Collaborative web services and their choreographies are expected to evolve as non-web businesses do. There is no any comprehensive or unified device or method that dynamically resolves issues arising from evolution.

[0004] One way of implementing B2B and A2A electronic commerce is use of web services. Web services are a fundamentally new way of exposing functionality in a modular way over the web. At the present stage of web service evolution, much attention is focused on document exchange and RPC web services. An RPC web service supports an RPC programming paradigm where the invoker invokes a web service by invoking something that looks like a procedure call with in/out parameters.

[0005] Current implementations of web services typically conform to three standards. UDDI is a registry standard for discovering web services based on search criteria and downloading its interface. WSDL is a interface standard that defines the messaging interface of a web service. SOAP is a envelope protocol standard with defined bindings to HTTP(S) transports that are used to carry the payloads over the wire. Web services typically rely on the XML standards like XSDL, Xlink, Xbase, and Xpointer and general web standards like HTTP(S) and URI.

[0006] Most of the web services today are simple web services. Simple web services use synchronous unreliable HTTP(S) transports and have no support for security except for transport level security, or agreed to security information in the payload understood privately by the communicating parties. Simple web services contain operations. Invoking a web service means invoking an operation of the web service. An operation might be designed to accept a one-way message or to support a request/response paradigm where the invoker blocks for the response and the invoked service immediately responds. Simple web services may use the RPC programming paradigm or a document based programming paradigm.

[0007] An emerging challenge in standard-compliant web services is to define high performance web services to improve the business performance of the enterprise. Such web services will support invocation with reliability (exactly once), asynchronous transports like SMTP, JMS, privacy (encryption), integrity (signing), authentication and authorization, and routing.

[0008] Another emerging challenge will be to define collaborative web services. These web services might hold long duration interactions with other web services and might invoke non collaborative web services without a choreographed, conversational context. Such collaborative web services will require support for correlating related messages in a conversation and support for a return address.

[0009] There are a number of industry initiatives to extend standards applicable to B2B and A2A electronic commerce. Choreography efforts include ebXML/BPSS from OASIS, WSFL from IBM, and XLANG from Microsoft. Conversation efforts include ebXML/TRP from OASIS and Microsoft's WS-routing. Additional efforts include BPEL4WS (Business Process Execution Language for Web Services), from IBM, Microsoft and BEA Systems, and Web Service Choreography Interface, from Sun Microsystems, WSCL (Web Services Conversation Language), and BPML (Business Process Modeling Language). So many choreography efforts are under way that some industry leaders in September 2002 asked the W3C standards body's Web Services Architecture Working Group to set a standard that will resolve incompatibilities. For reliability, there are proposals from Microsoft, ebXML/TRP from OASIS, and HTTPR from IBM. W3C is addressing standardization in all of these areas. Key industry players have formed a rival consortium called WSI. However, they have not addressed the service evolution and choreography issues.

[0010] Accordingly, an opportunity arises to develop methods and devices that support evolving document exchange choreographies.

SUMMARY OF THE INVENTION

[0011] The present invention relates to computer-based devices and methods supporting document exchange choreographies. More particularly, aspects of the present invention relate to devices and methods that facilitate evolution of systems by various combinations of choreography versioning, service versioning and document versioning. It provides for choreography management using a choreography agent and presents choreography-enabled interfaces to non choreography enabled applications. Additional aspects of the present invention include a graphical design tool and transparent aliasing of a host service as multiple context setting franchised services. Particular aspects of the present invention are described in the claims, specification and drawings.

BRIEF DESCRIPTION OF THE DRAWINGS

[0012] Figure 1 illustrates the capability to dynamically determine document versions and translations, service versions and choreography versions.

[0013] Figure 2 illustrates how messages exchange regarding service versions.

[0014] Figure 3 illustrates reference to one or more registries is used to discover information for negotiations.

[0015] Figure 4 illustrates the four types of activities.

[0016] Figure 5 illustrates interactions between activities and an internal process.

[0017] Figure 6 illustrates an internal process flow executing in a process engine used to construct a choreography agent.

[0018] Figure 7 illustrates message correlation.

[0019] Figures 8 and 9 help distinguish between a regular, non-franchised service and a franchised service.

[0020] Figures 10-17 are screen shots of one embodiment that practices aspects of the present invention. Figure 10 is a manage host packages screen.

[0021] Figure 11 depicts parts of registering a new host service.

[0022] Figure 12 depicts a screen used to edit document exchange activities.

[0023] Figure 13 depicts a screen used verify or modify signing and encryption policies.

[0024] Figure 14 depicts a screen used to register a host service package.

[0025] Figure 15 depicts a screen used to select a new package to franchise.

[0026] Figure 16 depicts a screen used to select the franchise options.

[0027] Figure 17 depicts a screen used to edit service details.

DETAILED DESCRIPTION

[0028] The following detailed description is made with reference to the figures. Preferred embodiments are described to illustrate the present invention, not to limit its scope, which is defined by the claims. Those of ordinary skill in the art will recognize a variety of equivalent variations on the description that follows.

[0029] Communities and networks of communities are one environment in which computer-assisted devices and methods implementing document exchange choreographies are useful. Among these communities, a community maintains a local registry that includes information such as users, companies, services and connectors that are part of the community. The community can be a marketplace, an enterprise or a sub enterprise. Communities can belong to one or more community networks. Typically, communities and networks have some common business interest. Interoperation is between member communities in one or more community networks. A network of communities makes available a global registry of communities. The global registry permits lookup of the community and determination of one or more routes to that

community, or to external connectors through which the electronic commerce documents bound for the community may be routed. Documents routed from one community to another may be routed directly between external connectors for the two communities or indirectly through one or more intermediary communities. Business and security rules for transactions involving the communities also can be defined and maintained in community registries. Connector is a general term for applications that communicate with other applications. Connectors may communicate on a peer-to-peer (P2P) basis or on a directed basis through other connectors that function as hubs, gateways, external ports, central connectors, etc. Connectors that communicate P2P are able to communicate with other connectors that use the same transport/envelope protocols. Connectors that communicate P2P optionally may enlist the assistance of other hub connectors that perform translation services, when trying to communicate with a connector that does not use the same transport/envelope protocol. Connectors that communicate on a directed basis communicate through hub connectors according to routing rules. Routing rules among connectors can be mapped in a directed graph, supporting one or more hub and spoke topologies for one or more transport/envelope protocols. A hub and spoke topology directs communications along spokes to hubs, in one or more tiers. This facilitates centralized services such as billing, business intelligence collection, tracking, auditing, accounting, or others. Multiple hub and spoke organizations may overlay the same connectors to support different transport/envelope protocols and technologies. For instance, a stronger hub and spoke organization may be required to use Sonic as a transport technology than to use HTTP or HTTPS. Optionally, communication routes may depend on whether the source and destination are part of the same community. Within a sub-community (which may include the whole community), centralized functions may be unneeded and P2P communications permitted among connectors that otherwise are directed to communicate with parent connectors when communicating with destinations in other sub-communities.

[0030] Connectors may be labeled simple connectors (sometimes simply called connectors), hubs (sometimes called gateways or routers) or central connectors. Alternatively, they may be described functionally. Simple connectors are directed to communicate via hub connectors, except when they are permitted to communicate P2P among connectors in the same sub-community. So-called hubs are used by connectors that are explicitly directed or linked to them. Hubs may serve more than one function and, accordingly, may appear more than once in a route from a source to a destination. Hubs forward electronic commerce documents or messages. Hubs also may translate among transport protocols that support a common envelope protocol.

For instance, a hub may translate envelope protocols and also implement a different transport protocol upon transmission than upon receipt. A central connector is a special case of a hub, which can be used by connectors that are not explicitly directed or linked to them. A central connector is useful, for instance, to carry out translation functions when traversing connectors from a source according to routing rules does not lead to any hub that supports the transport/envelope protocol used by the destination.

[0031] In the standards context, WSDL is a standard-based language used to define so-called service definitions (which WSDL calls porttypes), service instances (services), and connectors (ports). A service definition is like a class definition. WSDL names the service type, defines a set of operations for the service definition, which are essentially methods of the service, and defines the input/output messages of the operations. Operation types specified by WSDL include notification (one outward bound message), one-way (one inward bound message), solicit/response (synchronous, with one outward message followed by one inward message) and request/response (synchronous, with one inward message followed by one outward message). WSDL does not deal with choreography of interactions between services.

[0032] Aspects of the present invention go well beyond WSDL to address choreographed operations in a service. Collaborative operations are invoked by collaborative interactions, where an operation in one service instance invokes an operation in another service instance. The collaborative interactions between two service instances are part of a single conversation, where a conversation is a bi-directional flow of messages and is a choreography instance. A message in a conversation typically includes a conversation ID, the choreography that applies, and identification of or a return address to the initiating service/operations.

[0033] Figure 1 illustrates the capability, combining several aspects of the present invention, to dynamically determine document versions and translations 101, service versions 102 and choreography versions 103, for a particular message exchange choreographed as a conversation. These dynamic determinations are described in some of the related applications and below.

[0034] A message in WSDL has a name, and a set of parts. Each part has a name, a MIME content type, and a schema definition for the part if XML.

[0035] The present invention adds the concept of a document family, instead of an explicit scheme definition for an XML part, which supports dynamic determination of document versions. A document family is a set of document versions whose scheme defines the document structure. This allows two interacting services to support different versions of the document. Version transformation logic is provided to transform to the source document to the required

target document schema. This allows evolution of document versions without upgrading all services in the network that interact with each other using that document. This is so-called document version interoperation, which is further described in the Registry Driven Interoperability and Exchange of Documents application, identified above.

[0036] The present invention introduces the concept of a service definition version, which also can be dynamically determined. A new version can add new optional parts or delete existing optional parts to an existing in/out parameter or document and can add new operations to an earlier version of a service. Service instances using the earlier version can correctly address with the upgraded, new version of the service instance because the version number is not part of the service definition name in the message address. (An address typically includes is a party/service definition name/operation.) In general, service definitions are templates for service instances (sometimes simply called services) that follow the definitions. Defining service template is like defining a type, while defining a service instance is like defining a variable of that type. A service definition has a name and version and defines the list of activities in the service along with one or more activity interfaces. The interface does not explicitly identify the documents supported, but at least identities the document family supported. A document family is a set of documents with the same business function. For example "purchase order" is a document family and xCBL 3.5 purchase order and Rosettanet 1.0 purchase order are members of that family. Service definitions may be established by a community administrator, may be adopted from a vendor such as Commerce One, or, in the future, may be defined by standards bodies. Service instances complete the interface definition by specifying the actual versions documents used. The instance also specifies the policies of the service, and the CP offering the service. It also specifies the connector it is hosted in. Messages are sent and received by service instances. Preferably, activities in a service instance are hosted in a single connector.

[0037] Choreography definition languages generally define a flow of messages between roles. WSDL defines service interfaces of the roles. But the choreography definition languages have not merged with WSDL. Missing is a clear binding between a choreography and a service. One aspect of the present invention includes mapping a choreography role to a service instance and a choreography message to an in/out parameter of a service operation. This is more fully discussed in the Dynamic Interface between BPSS Conversation Management and Local Business Management application, identified above.

[0038] The present invention allows a service definition to define a set of choreographies (identified by name, for example) that the service supports and the role in that choreography that

can be dynamically determined. A service instance will indicate the subset of choreographies that it supports and may indicate an order of preferences among supported choreographies. At runtime, an intersection of supported choreographies among participating service instances is computed and a particular version of a choreography is selected for use in the message exchange. The preferences of one or more services can be factored into the decision of which choreography version to use. Rules such as initiator wins, respondent wins, latest version wins, earliest version wins or a weighting of preferences might be used in the factoring. Alternatively, an agreement between the two services can be predefined that specifies the choreography version to use. All of the choreography participants use the same choreography for all subsequent messages in the choreographed conversation. This allows choreographies to evolve without requiring all services in the whole network to be upgraded. This is so-called choreography version interoperation. This interoperation is achieved independent of the language used to describe the choreography, but does depend on the services understanding the meaning of a choreography from its name.

[0039] Two methods of selecting a choreography version or discovering a service version are illustrated in figures 2 and 3. In figure 2, messages are used to exchange information about versions between a first service 201 and a second service 202. This approach may be extended to more than two services with a first message sent to all of the other services or with an additional first message being sent from the second or subsequent services to the other services. In choreography, for instance, the first message 211 in differing variations and versions of a choreography are substantially the same. Messages may differ slightly, due to document version considerations. Other minor variations in the first message may be allowed, while the first message remains compatible with all choreography versions. After the choreography used is selected in the first message, all subsequent messages in either direction between the services are part of this choreography. After a choreography version is selected, subsequent messages 212-224 conform to the selected choreography. In figure 3, reference to one or more registries is used to discover information about versions supported by a first service and a second service. The service local to the discovery process may store version information in a first registry 331 that is accessible to a method 321. The remote service would store version information in second registry 332 that is accessible to a method 323. In some cases, the remote service's information would be in the first registry 331 and accessible to a method 322. For instance, if the first and second services were in the same community and relied on the same registry. Or, if the first and second services had had recent contact and the first service cached the second

service's information in the first registry. Information about versions supported by both services would be considered by version selection logic 311.

[0040] Thus, aspects of the present invention can be combined in several ways to dynamically determine some or all of document versions, service versions and choreography versions. Implemented in accordance with the present invention, a web service can be substantially updated and retain interoperability with other services that have not been similarly updated.

[0041] Further aspects of the present invention enable complex choreographies and business processes to be defined and executed between distributed collaborative web services across the Internet or another network, in so-called peer-to-peer interactions or collaborative interactions. It also supports the standard client/server interactions (or non collaborative interactions) that are common in legacy web services implementations. In peer-to-peer interactions, choreography support for correlates related messages exchanged among services and includes return addresses in messages of the conversation.

[0042] A collaborative service might have a virtual conversation with a non collaborative web service or other application interface. Basically the two services use client/server interactions with each other and know out of band the return address. The services correlate messages using application logic that examines the message payloads.

[0043] Collaborative services have zero or more associated contexts. New contexts are created by one or more trigger operations. Alternatively a context may be initiated spontaneously (from a messaging standpoint). Subsequent non-trigger operation initiations are tied to this context. A process flow instance is the context referred to. One operation kicks off the process instance and all subsequent messages relate to this instance. This context helps in having a bi-directional conversation with an initiator and bi-directional conversation with other services driven by the context. For non-collaborative services, each operation is complete in itself, the initiator could be any program and need not be a service, and it is independent of the invocation of every other operation in the service.

[0044] More generally, there are two types of message interactions between parties. A peer-to-peer message is typically associated with a choreography. In a peer-to-peer interaction, a few additional pieces of information are provided. A return address is provided, typically identifying the sending CP, service definition, and activity definition. Message correlation data is provided, such as a conversation ID and choreography ID and refers to ID. The return address identifies where the invoked service should respond. To further assist this service, the applicable

choreography used for this interaction may be returned to both services on each message in the conversation. The conversation ID has the same value for every message in the conversation between two services. This will allow the services to easily correlate related messages. The choreography ID is selected on the first message and stays the same for the conversation. The refers to ID refers to the message ID of the message to which this is the response. A single message might have multiple responses.

[0045] The second type of message interaction is a client/server interaction (also called non-collaborative interaction.) In a client/server interaction, there typically is no sending service/activity. The request and the optional single response typically completes the interchange and subsequent client/server interactions are unrelated to this one.

[0046] One special aspect of client/server is transformation support for the response in a request/response invocation. In a peer-to-peer interaction the sending service says in its interface what document it expects in the response. However for client/server, this is not the case, and the requestor has to say in the request the documents it expects in the response. If the requestor does not specify the expected document for a part, the document as generated by the responder is sent without any transformation.

[0047] A service may include a plurality of document exchange activity interfaces. A document exchange activity interface may be defined both as part of a service definition and (customized) when the service instance is registered. An activity is described by its interface and its policies. . UI services, for instance, may register their activities in the registry for privilege determination, and other access control information such as single sign on web agents, and resource paths.

[0048] An activity can be of four types, illustrated in figure 4. A request/response 411: The activity receives a request and responds. A solicit / response 412: This is the inverse interface of a request/response. A solicit response activity may invoke a request/response activity in another service instance. One-way 414: This is an activity that receives an asynchronous message. Notification 413: This is the inverse interface of a one-way. A notification activity invokes a one-way activity in another service instance. These definitions are consistent with WSDL.

[0049] The input or output to an activity is called a message. A message consists of multiple parts. Parts are described in the following way. The part name is a unique name for the part that is reused for every message from/to the activity for that part. The part type could be an XML part, or a part that is described by a MIME type. If the part does not have a MIME type defined, then the MIME type for the part can be supplied at run time by the invoker. The part location

specifies whether the part is in the body, or an attachment. Alternatively, it may be in a message header or external to the message and the message only refers to it. A part may be required or optional. A root indicator: There should be one root part in a message. This typically is the main theme of a message and the typically is an XML part. The root part indicates the starting point of traversal to find the other parts. It may include references to other parts, but not necessarily all of them can be discovered by traversing from the root part. A message also may include a document ID list. Any part (XML or non XML) can belong to a document family. Associated with members of a document family (called document Ids) are descriptor information, optionally a schema, and optionally transformation maps of various types from/to this member. Document family metadata is registered independently of services.

[0050] Within a choreographed service, a process flow can be used to coordinate activities on opposite sides of the interface. The concept of a process flow, which is defined using design tools and the definition executes in a process engine, has been used in other contexts. Commerce One, for instance, has a process flow solution called CPM. A process defines a flow with potential branching points and parallel threads of execution. A process instance has associated with it a context and is initiated by some event and is long lasting. This context goes through a series of state transitions triggered by other events and the process itself can initiate events as part of the flow. The events may include UI interactions, document exchange events (sending or receiving messages with document payloads), a change in the context state or associated database state, or invocation of business logic.

[0051] Figure 5 illustrates interactions between activities and an internal process. The activities 411 through 414 exchange messages with the process flow from 521 through 525. The process flow steps pass messages among themselves. In another instance, the process flow could include branching and parallel execution of processes.

[0052] An internal process flow executing in a process engine also can be used to construct a choreography agent that centrally manages choreographies, as illustrated in figure 6. Choreographies can be centrally managed by making all messages flow through a choreography agent 610 that, itself, appears as a collaborative web service and utilizes an internal process flow 612. Adapted to a choreography agent, a message from service 630 to service 640 is changed to be a message from service 630 to the intermediate service 610, which forwards it to service 640. An asynchronous response from service 640 back to service 630 is changed to be a message from service 640 to the intermediate service 610, which forwards it to service 630. The choreography agent 601 includes interfaces 611, 613, 614 through which messages are passed.

An internal process flow 612 can be used to track the status of one or more choreographies. Some or all of the internal process flow steps may store information in a transaction rollback log, to be used in case the choreographed message exchange terminates without completion. Interfaces can be added to support querying the status of one or more choreography instances that are being coordinated through the choreography agent. Similarly, activity statistics can be compiled during choreography coordination and reported through an interface. This centralization allows the status of the choreography to be queried easily for tracking. Error handling and error compensation actions can be more easily implemented. When the intermediate service is implemented as a process flow, the choreography is likely to be globally correct, because the internal process flow conforms to the process definition. Global checking of the choreography for correctness is facilitated.

[0053] A process flow executing in a process engine gets messages that trigger events and sends messages as part of processing an event. These messages can be described using a standard web services definition language like WSDL. These descriptions can be easily discovered with standard registries like UDDI. Therefore, these messages can be grouped together into web services. The messaging system can route messages to the activities of this service to the connector executing the flow. Some operations can have the semantic of launching a new process instance. Messages to these operations will result in a new process instance. For other inbound messages, a dispatcher in the connector can link the message to an existing process instance by deducing the instance based on payload content, the state of the process instance (if it is at a step where it is waiting for a particular message to a particular operation) or message correlation data in the message envelope (useful when the inbound message is a asynchronous response to a previous outbound message).

[0054] Process flows that execute in a distributed fashion in multiple process engines 610, 620, 630, 640 passing their internal state between themselves can be treated as a set of collaborative services interacting with each other.

[0055] A further aspect of the present invention is optionally requiring subscription to a service prior to being invoked. During service subscription, for a collaborative service, one of the possible choreographies can be selected and agreed for use by both the consumer and provider. Other aspects of the service also can be agreed at subscription time, such as access control.

[0056] Service oriented access control has three key concepts that work together to facilitate manageability and enforcement of access to services and activities/operations in a service:

service visibility, service subscription, and service privileges or authorizations. Service visibility allows the service provider to specify who can discover the service, and allows the discovery mechanism to restrict the discovery based on the visibility. The service visibility rules may include: visible to anyone, visible to administrator/operator, visible to specific parties in community, visible to specific roles in community, visible to any community in the network, or visible to specific communities in the network. Visibility also applies to collaboration parties (sometimes called trading parties or partners). When a party is registered in the community, the party is by default visible in the community. Additionally, the party can be made visible to any community in the network, or to specific communities in the network. Service and party information also can be pushed into public UDDI registries based on the visibility rules specified.

[0057] For each service activity, set of privileges also can be established at subscription time (or by default.) For each service, the application can also specify service roles that effectively are privilege groups. For a privilege, additional details can be specified, such as 'whether the privilege supports scope of specific organization units' and 'whether the privilege can only be used as part of a service role', etc. A Service can also specify if it requires some sort of subscription prior to someone using it. For example, a supplier may specify in an order management service, that buyers can send orders to this service only after subscribing. The subscription process allows the service provider to receive information regarding the subscribing party, as well as the capabilities of the subscribing service, if required. Based on this information, the service provider may choose to approve the subscription or reject it.
Once a party administrator subscribes to a service, they can create user roles within their own organization by grouping the service roles and/or privileges within the subscribed services and assigning these user roles to user accounts.

[0058] Another major challenge for support of choreographed interactions is back office systems. Use of a service composition agent can address the problems associated with back office system integration. Back office systems typically do not understand SOAP extensions that support choreographed conversations, including extensions to support for message correlation and return address specification. Therefore, it is useful to have a composition agent expose a collaborative web services interface that is mapped bi-directionally to the simple web services interface of the back office system, as shown in figure 7. The composition agent correlates related messages by executing rules against the payload and deduces the return addresses of the applications from the peer-to-peer invocation message from the application. Back office systems

expose their functionality differently, so there typically is a different set of service definitions for each back office system. A composition agent simplifies keeping track of multiple evolving back office systems by presenting them at a single interface. The composition agent selects the correct back office service and action. Back office systems support a variety of XML document definitions, when they have been augmented with EAI interfaces. Examples include xCBL, IDOC, OAGI, etc. Aspects of the present invention allow these document defined using this variety of definition languages to be treated as members of single document family. Translation logic can be invoked to convert messages back and forth between document family members. EAI interfaces to back office systems typically do not support rigorous security, reliable messaging, etc. The composition agent can support security, reliability, etc. between itself and inquiring services. The composition agent can use less robust and secure solutions can be used in the last hop between the back office system and the last native connector, or special security and reliability measures can be adopted.

[0059] The composition agent service can support logical routing (target address selection by payload examination and/or registry lookup) for conversations with the back office system, including conversations initiated by the back office system. Message correlation is illustrated in figure 7. A non choreography enabled system 710, such as a back end system, communicates with an enabling process or composition agent 720. The communications are configured, in part, by specifying fields within messages that the composition agent will receive 721, 722, 723. Message correlation logic 724 can be used to correlate messages with a new or ongoing choreography 725. This choreography can be modeled, as described above, as an internal process. The correlation logic 724 can be positioned in the message flow to process each message from the back end system as it is received. Or it can be a resource available to process steps in the process flow 725. The positioning of the correlation logic 724 is less important than its actions: inspecting messages generated by the back office application interfaces and using the specification of fields to correlate back office messages with a choreography instance. The correlation logic can be extended to conversations initiated by the back office system. When the correlation logic determines that the source of the message always initiates conversations or determines that there is no existing conversation that correlates with specified fields in the message, the correlation logic can signal that a new conversation instance needs to be created. The correlation logic further can use the specified fields to determine what choreography should be invoked. For instance, a back office system may initiate supplier orders. The supplier name field can be used to identify the proper CP and the service that the CP uses to receive new orders.

A choreography associated with that service can be invoked. The composition agent 720 is exposed to choreography enabled services 730 through standard interfaces 726, 727. One or many services may interact with the composition agent through its service interfaces. The composition agent and the correlation logic track the status of the choreographed conversation and correlate messages exchanged with the back office system 710 to the appropriate conversation ID used in messages exchanged with other services 730.

[0060] Another aspect of the present invention is exposing a single hosted service through a plurality of aliases that are franchised by the host to service franchisees. Unlike franchisees in brick and mortar businesses, these franchisees are not expected to advertise the identity of the host. In a sense, these service franchisees are more like private labelers of OEM goods. The franchised services can be registered to interact with other services or they can be routes to one or more back office systems that the host is designed to support.

[0061] Figures 8 and 9 help distinguish between a regular, non-franchised service and a franchised service. In figure 8, two services 811, 812 appear in a registry of services 810. In the non-franchised scenario, these registry entries provide logical address that correspond to two different instances 831 and 832 of the same service logic. These instances of service logic are like two licensed copies of server infrastructure software running on two different web servers. In figure 9, three franchised services 911, 912, 913 appear in a registry of services 910. These franchised services have different logical addresses. All of them are logically connected to a host service 921, which is not exposed through the registry in the same way as the franchised services. Messages sent to the different logical addresses are in different contexts, corresponding to the franchisees of the three franchised services. With the context set by the logical relationship between the franchised services 911, 912, 913 and the host 921, the franchiser who has franchised the host 921 can use one instance of the host logic 931, instead of running instances of the logic 831, 832 for each registered service.

[0062] Consider a use case in which a marketplace operator hosts an order management application (a collection of services). A set of suppliers then franchise this hosted services package to manage all orders for them. In this case, messages to a franchisee are addressed to the <franchisee CP>, <service definition>, <activity definition>. Both UI and document exchange services are franchised. In practical terms, one embodiment of implementing franchised services involves the following steps. 1. Create a host instance of a service. 2. Create a host package by collecting a set of host services together. The hosting entity is the owner of the package. Also the package is tied to a connector where the actual implementation lives. 3.

Associate to the host package a set of services that are subscribed by the franchisee that allow the franchisee to process the received documents. This allows one or more franchisees to franchise the host package. Steps 1 and 2 should, in some environments, be done by the community administrator, while step 3 can be done by a service provider.

[0063] A franchise collaborative service can optionally delegate authentication in the registry to the hosting services CP. One reason for this would be that the franchisee does not want to use the franchisee's credential because the franchisee does not want to identify itself completely. In this case, the franchisee will delegate authentication to the hosting CP for all messages from this service on behalf of this franchisee.

[0064] The franchised service is like a 'pass-through' service that sets a context and allows all information to simply flow to the host service and get processed by the host service. The franchise service will have a reference to the host service, and generally will inherit the implementation capabilities and policies from the host, with some overrides allowed based on business need. Preferably, there is no implementation tied directly to the franchise service. If customization is required, it is preferred to offer multiple hosted service variations. Alternatively, multiple choreographies might be supported by a hosted service and a subset of the choreographies supported for a particular franchised service. Messages to a franchisee are addressed to the <franchisee CP>, <service definition>, <activity definition>, as if the franchisee were operating a standalone service. Service consumers can interchangeably subscribe to and use regular or franchise services. Service consumers interacting with franchisees do not see any difference franchised and non franchised services.

[0065] A host service package is a package that consists of all the services implemented generally by a hosted application. A single hosted application can publish multiple host packages. In most cases, a hosted application will publish a single host package. Services bundled with a host service package may be grouped into two types. One type of host services is 're-offered' as franchised services on behalf of the franchisee. Another type includes portal (or UI) services used by the franchisee to process information received through the franchised services, but not re-offered or exposed beyond the franchisee. For example, an order management package may have a standards based 'receive orders' service and also have other UI services such as 'view orders', 'order reports', 'approve orders' etc. When a franchisee signs up to franchise the order management package, he/she needs to be able to not only receive orders through the standards based document exchange service, but also view them and approve them

etc. A Host Package has a package connector that will be the default connector for each service in the package to send or receive messages.

[0066] A business process between the Franchisee and the Host provider is enabled to allow the franchisee to franchise the host package online, with an online approval process, where required. When a service provider franchises the host package, a franchised service is automatically created and registered as a pass-through to each host service in the package. Further, the franchisee is also automatically subscribed (and hence authorized to use) the other subscription services in the package, when the franchise is registered.

[0067] A franchise service can optionally delegate authentication to the host service, as previously mentioned. In one embodiment, the practical steps involved in delegating authentication on the host side may include: 1. Create a host instance of a service (typically only document exchange services). 2. Create the regular services that will be used by the franchisee as part of the package. 3. Create a host package & specify the collection of host services that will be enabled as provided by the franchisee. 4. Associate a connector for the host package that will send/receive messages for the host services. 5. Associate the set of services that are subscribed by the franchisee that allow the franchisee to process the received documents. 6. Host provider can specify on the package whether approval is required when a franchise is requested. 7. Host provider can specify if delegated authentication is allowed on this package. On the franchisee side: 1. Franchisee request franchise of a host package. This triggers an approval process if required by the Host Provider. 2. Host Provider receives notification to approve the request, and either approves or rejects the request. 3. If Host Provider approves the request, system creates a franchise package, subscriptions to the subscription services in the package, and franchise services for the franchise-able services in the package.

[0068] To facilitate understanding of franchised services, a supplier facing order management system is considered. For this discussion, this system is called SOMS (Supplier Order Management System). SOMS provides an order management system that can be used by suppliers to receive orders from buyers. In one embodiment, the following steps may be involved in designing and setting up SOMS. The designer divides the services to be published into groups including services to be franchised by the franchisee (e.g., receive orders, check order status etc.), services to be used (or consumed) by the franchisee (e.g., view orders, approve/reject orders, order statistics/reports etc.), and services that application provides for A2A (application-to-application) integration such as integration with other applications or back office systems. (e.g., integration with user management systems, public registries, catalog systems, etc.)

The designer determines whether all franchisees will have the same set of services or if they may potentially need different subsets. Based on this need, identify one host service package or multiple host service packages, or, alternatively, multiple choreographies of the same service. The host service package should contain the 'to-be-franchised' services and 'for-use-by-franchisee' services. The other services used for integration are typically registered as regular services with the providing cp as the application or the host provider. In a community environment, the host provider is typically the community operator, but it may also be any of the collaboration parties in the community. It should be kept in mind during design that the 'to-be-franchised' services will actually be receiving and sending messages on behalf of multiple service providers. Hence the application business logic needs to use the addressee in the message as a context for processing the messages. A host service implementation should look at an incoming message to figure out the recipient CP and trigger/map to the appropriate business process. For UI triggered business processes, the host service should take the franchisee CP as an input to the process context, and use this in the delegation policies within the process.

[0069] After design and coding, the sample SOMS service will need to be registered, at least in some embodiments. Registration of services to be franchised, services to be used by franchisees and the host service package itself is illustrated by several setup screens in the figures. One part of registration is to register the host services to be franchised. In one embodiment, a registry manager UI is used to create a new service definition for the service (if it does not exist already). Then verify that a connector is available for your application. This may be a default connector available to a franchiser or a separate connector registered for to the franchiser's application. Also verify that any required document identifiers have been loaded into the registry. Then, create the host services as follows. Figure 10 is a manage host packages screen. Click on 'Register New' 1001. Figure 11 depicts parts of registering a new host service. Select a service definition 1101. Pick the operator party as the 'Providing Party" 1102 or use a different party as appropriate. Specify the service information 1103, leaving the 'Application Instance Name' and 'Application Service Identifier' blank. Optionally, specify any service classifications 1104 that should be associated with this service. Select the choreographies that are supported by this service 1105, and the order of preference. Set the visibility 1106 to default setting, which is 'Visible to All Parties in Community'. Select the supported document exchange activities 1107 and edit the activities using a screen such as figure 12. Create privileges 1201. Leave the defaults for the 'Messaging Policies' 1202 unless your application needs special delivery receipt handling. For input 1210 and/or output 1220 messages, set signing

and encryption policies 1213, 1223 to 'required'. Edit required parts 1214, 1224 and specify an ordered list of the supported document identifiers. Add optional message parts, which may be published in the registry. Using a screen such as figure 13, verify or modify the policies as follows: Set asynchronous delivery receipt 1301 to 'Yes'. Set 'Authentication Mode' 1302 to 'Document Single Sign-on'. Set 'Authentication Mechanism' 1303 to 'Username/password'. Then, optionally, specify any service roles 1304. Click on 'Submit/Save' 1305 to register the service.

[0070] Another part of registration is to register regular services to be used by the franchisee (as opposed to hosted services.) This is done in the manner normal to web services development.

[0071] The host service package also is registered, using screens such as figure 14. At the 'Register a New Host Package' screen, specify a package name 1401, and description 1402.

Specify an application name 1403. Pick the operator party as the 'Host Service Provider' 1404 or use a different party as appropriate. Specify the service information (keep the Application Instance Name and Application Service Identifier 'blank')Specify if approval is required for franchising this package 1405. If 'Yes', an approval process will be triggered when a CP franchises this package. Specify the application connector 1406. Specify if the franchisee is allowed to delegate authentication to the host service provider 1407. Select the services to be franchised by the franchisee, out of the list of host services offered by the host service provider 1408. Select the list of services to be subscribed by the franchisee, out of the list of regular services offered by the host service provider 1409. Click on 'Save' 1410 to register the host service package.

[0072] The franchisee also registers, selecting a new package to franchise from a screen such as figure 15. One or more host packages are selected by selecting radio buttons 1501 and clicking 'Select' 1502. A screen such as figure 16 becomes accessible. The franchisee's options are limited, so the set up is straightforward. In an alternative embodiment, the franchisee would have options regarding services to offer and/or services to subscribe to, especially in circumstances when the franchisee was re-offering the services. Decide whether authentication will use the franchisee's credentials or will be delegated to the host 1601. Edit one or more services to be offered 1602, satisfying any indicated requirements 1603 for editing. Select 'Submit' 1604 to submit the franchise for processing.

[0073] When service details are edited, a screen such as figure 17 become accessible. In some embodiments, most of the details presented will only be editable by the host, not the

franchisee. This can be seen by comparing figure 17 with figures 11 and 13. Some of the details presented apply to downstream users, to whom the service is re-offered, in the same manner in which the host applies them to the franchisee. The descriptions of figure 11 generally apply. In addition, service roles, which group privileges, can be established, as described above.

[0074] Some of the services that a franchisee may subscribe to are portal services. A portal service (or an UI service) is a service that is accessed through a portal. These are simply the services exposed by an application to enable users to interact with the application. This allows users to have a service-oriented view of applications. A specific user may interact with services from different application, and may/may not interact with all services from one application. In any of these cases, the experience across the applications in terms of administration, access control/privileges, single sign on, navigation etc. will be the same, and based on services paradigms.

[0075] A portal service may not have a service definition. (There are no standards yet. While the present embodiment does not require a service definition for portal services, it is envisioned that in the future there will be a service definition for portal services.) A portal service has a list of registered activities (along with the URLs, if necessary) that can all be accessed through the portal. For activities, lists of privileges that enable access to the activity are also specified. Additionally, single sign on (SSO) information such as web agents and protected resource paths are also specified for the service or for each activity. Access to reports may also enabled through portal services, by registering report groups/folders as activities.

[0076] It is useful to consider several factors when identifying the UI services to be registered for an application. Consider the target users or parties. Application interfaces are designed for use by different target users. For example, certain UI's are meant for administrators, certain UI's are meant for buyers, certain UI's are meant for Suppliers, etc. These role differentiated UI's should be separate services. Consider if certain interfaces are targeted for only certain classes of suppliers or buyers. For example, you may want to enable additional functionality for advanced suppliers only. Keep these in separate services.

[0077] If you have many services targeted to one target group (e.g., suppliers), consider creating service subscription packages to group these for user convenience. Collaboration parties subscribe to a service, and users within the party are granted specific privileges within the service. This means that privileges and activities across parties should not be mixed. For example, buyer submit RFP and seller respond to RFP privileges should be in two different services.

[0078] For the portal services, a specific display category may be associated. The display category reserves a tab on the portal for the service. Many portal services can be in one display category. A display category is associated with a service room URL specified by the application. By registering the service room URL, the application retains full control on how the services are rendered in the portal within the tab. If you would like your application to be rendered as multiple display categories, identify services based on the display category they are associated with.

[0079] Preferably, the following relationships among services, activities, privileges, and web agents are observed. 1. Service can have one or more activities. 2. An Activity can be associated with zero or more privileges. 3. Privileges belong to and are unique within a service. 4. A Privilege can be associated with one or more activities in the same service. 5. A Service can be associated with a default web agent that handles single sign on (SSO) along with a list of web resource paths that are accessible through the service. 6. Each activity in the service can either be associated with the service web agent, or overridden with a different web agent. Access to activity specific resource paths can also be specified.

[0080] It is useful to consider several factors in identifying the activities and privileges in UI services. 1. Activities & privileges in a service should be targeted within a collaboration party. This means that privileges and activities across parties should not be part of the same service. Again, buyer submit RFP and seller respond to RFP privileges should not be in the same service. 2. It is not preferred to register every link in the application as an activity in a UI service. If access control is required, these can be registered as privileges. 3. In the simple case, a portal service can register just one activity and a bunch of privileges. For example, there could be 'auction bid' service that has one activity 'bid' with a set of privileges that allow specific bidding functionality. Based on the privileges assigned to the user, the application can choose to render completely different pages, or the same page with different options enabled. If links to these activities are enabled from other applications/services, a user may want to register more granular activities, along with the URL so that these can be discovered through the registry. For example, Catalog Management may want to register 'search catalog' activity, along with 'manage catalog' activity. The 'search catalog' activity may be presented as a link on procurement screens, source screens, and contract management screens, as well as manage catalog screens. 5. If certain links in a service are hosted in a different application server or web server and need a separate web agent to protect them, these can either be grouped into a separate service, or registered as

separate activities in the same service. (E.g., product configuration activities that may run on a different web server from the rest of the catalog service activities).

[0081] While the present invention is disclosed by reference to the preferred embodiments and examples detailed above, it is understood that these examples are intended in an illustrative rather than in a limiting sense. From the preceding description, it will be apparent to those of skill in the art that a wide variety of systems and methods can be constructed from aspects and components of the present invention. Computer-assisted processing is implicated in the described embodiments. Accordingly, the present invention may be embodied in methods for computer-assisted processing, systems including logic to implement the methods, media impressed with logic to carry out the methods, data streams impressed with logic to carry out the methods, or computer-accessible processing services. It is contemplated that modifications and combinations will readily occur to those skilled in the art, which modifications and combinations will be within the spirit of the invention and the scope of the following claims.

[0082] We claim as follows:

CLAIMS

1. The method of supporting web service interoperability among a plurality of service entities, including determining at run-time:

a particular version of a message choreography to be used for exchange of messages among the entities, wherein versions of the message choreography use substantially the same first message;

service versions in use by one or more service providers among the service entities; and

document versions to be used by the service entities in the message exchanges and translations, if needed, among the document versions.

2. The method of claim 1, wherein the method is adapted to a plurality of choreography definition languages.

3. The method of claim 1, wherein the method is adapted to a plurality of service entity interface definition languages.

4. A method of choreographing web services, wherein a choreography includes a plurality of messages mapped to standard-compliant interfaces of a plurality of service entities, assigning a conversation identifier to the messages, and conducting messages exchanges among the service entities according to the choreography, the method including:

routing the message exchanges through a choreography agent, wherein the choreography agent:

inspects messages,

updates the status of a choreography instance corresponding to the conversation identifier based on the inspection of the messages, and

forwards the messages to their destinations.

5. The method of claim 4, wherein the method is adapted to a plurality of choreography definition languages.

6. The method of claim 4, wherein the method is adapted to a plurality of service entity interface definition languages.

7. The method of claim 4, wherein the choreography agent further responds to inquiries about the status of the choreography.

8. The method of claim 4, wherein the choreography agent further updates a transaction roll-back log.

9. The method of claim 4, wherein the messages include documents, further including:

maintaining document versions of the documents and translation logic to convert among versions; and

determining particular document versions to be used by the service entities and translations, if needed, among the particular document versions.

10. A method of choreographing web services, wherein a choreography includes a plurality of messages mapped to standard-compliant interfaces of a plurality of service entities, assigning a conversation identifier to the messages, and conducting message exchanges among the service entities according to the choreography, the method including:

maintaining choreography versions of a choreography, wherein the choreography versions use substantially similar first messages; and

responsive to a first message, determining a particular message choreography version supported by the service entities and utilizing the particular choreography version after the first message.

11. The method of claim 10, wherein determining a message choreography version includes accessing a registry that identifies choreography versions supported by the service entities.

12. The method of claim 11, wherein the registry further identifies preferences of the service entities among the supported choreography versions and determining further includes utilizing the preferences of at least one of the service entities.

13. The method of claim 10, wherein the messages include documents, further including:

maintaining document versions of the documents and translation logic to convert among versions; and

determining particular document versions to be used by the service entities and translations, if needed, among the particular document versions.

14. A method of choreographing web services, wherein a choreography includes a plurality of messages mapped to standard-compliant interfaces of a plurality of service entities, assigning

a conversation identifier to the messages, and conducting message exchanges among the service entities according to the choreography, the method including:

maintaining choreography versions of a choreography, wherein the choreography versions use substantially similar first messages;

maintaining in a registry data that identifies choreography versions supported by the service entities;

accessing the registry and determining a particular message choreography version supported by the service entities; and

utilizing the particular choreography version for the message exchanges.

15. The method of claim 14, wherein the registry further identifies preferences of the service entities among the supported choreography versions and determining further includes utilizing the preferences of at least one of the service entities.

16. The method of claim 14, wherein the messages include documents, further including:

maintaining document versions applicable to the documents and translation logic to convert among versions; and

determining particular document versions to be used by the service entities and translations, if needed, among the particular document versions.

17. A method of versioning a service entity having one or more standard-compliant interfaces, the method including:

maintaining service versions of the service, wherein the service versions add optional operations to a base service;

maintaining in a registry data that identifies the service version supported by the service entity;

accessing the registry and determining the service version supported by the service entity; and

conducting a message exchange with the service entity utilizing the supported service version.

18. A method of versioning a service entity having one or more standard-compliant interfaces, the method including:

maintaining service versions of the service, wherein the service versions add optional operations to a base service;

sending a first message to the service entity that does not depend on the service version supported by the service entity;

receiving from the service entity a response identifying the service version that it supports; and

proceeding with a message exchange with the service entity utilizing the supported service version.

19. A mechanism for correlating messages generated by a plurality non choreography-enabled interfaces with a conversation identifier corresponding to an instance of a choreography, including:

an internal process flow mapped to a plurality of application interfaces that generate messages lacking a choreography conversation identifier;

a specification of fields in messages generated by the application interfaces to be used to correlate the messages with a choreography instance; and

logic and resources that inspect messages received at the internal process flow from the application interfaces, apply the specification of fields, and correlate the messages with a particular conversation identifier corresponding to the choreography instance.

20. A mechanism for correlating messages generated by a plurality non choreography-enabled interfaces with a conversation identifier corresponding to an instance of a choreography, including:

a mapping between one or more choreography enabled interfaces and a plurality of application interfaces that generate messages that lack a choreography conversation identifier;

a specification of fields in messages generated by the application interfaces to be used to correlate the messages with a choreography instance; and

logic and resources that inspect messages received from the application interfaces, apply the specification of fields, correlate the messages with a particular conversation identifier corresponding to a choreography instance and present them at the choreography enabled interfaces.

21. A method of composing choreographed messages including a conversation identifier from non choreographed messages lacking a conversation identifier, the choreographed messages presented at a first interface, the non choreographed messages received at a second interface from a plurality of third interfaces, the method including:

specifying fields in messages generated by the third interfaces to be used to correlate non choreographed messages generated by the third interfaces with choreography instances;

maintaining status information for a particular choreography instance, as messages are exchanged between the second and third interfaces;

using the specification of fields and the status information to correlate a particular non choreographed message received at the second interface with a conversation identifier corresponding to the particular choreography instance; and

presenting at least part of the non choreographed message and the conversation identifier at the first interface.

22. The method of claim 19, further including determining at run-time:

a version of the message choreography for the brokering agent and the requesting entity to use;

a service version in use by the brokering agent; and

document versions to be used by the brokering agent and the requesting entity in the message exchanges and translations between the document versions.

23. A method of aliasing a hosted service as a plurality of franchised services, the method including:

registering one or more franchised services as web services having distinct logical addresses;

providing a hosted web service including logic to examine messages received through the franchised services to determine the context of the messages; and

configuring the franchised services to pass messages received through to the hosted web service.