(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
5 April 2001 (05.04.2001)

PCT

(10) International Publication Number
WO 01/24003 A1

(51) International Patent Classification[7]: G06F 9/46

(21) International Application Number: PCT/US00/26789

(22) International Filing Date:
28 September 2000 (28.09.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/156,499 28 September 1999 (28.09.1999) US
60/184,945 25 February 2000 (25.02.2000) US

(71) Applicant *(for all designated States except US)*: **DATALEX USA WEST, INC** [US/US]; 1200 N.W. Naito Parkway, Suite 200, Portland, OR 97209 (US).

(71) Applicants and
(72) Inventors: **TILDEN, Mark, D.** [US/US]; 17979 N.W. Gilbert Lane, Portland, OR 97229 (US). **HOPKINS, Scott, D.** [US/US]; 20867 S.W. Eggert Way, Aloha, OR 97007 (US). **STONIER, Brett, J.** [US/US]; 1263 N.E. 17th Avenue, Hillsboro, OR 97124 (US).

(74) Agent: **STOLOWITZ, Micah, D.**; Stoel Rives LLP, 900 S.W. Fifth Avenue, Suite 2600, Portland, OR 97204-1268 (US).

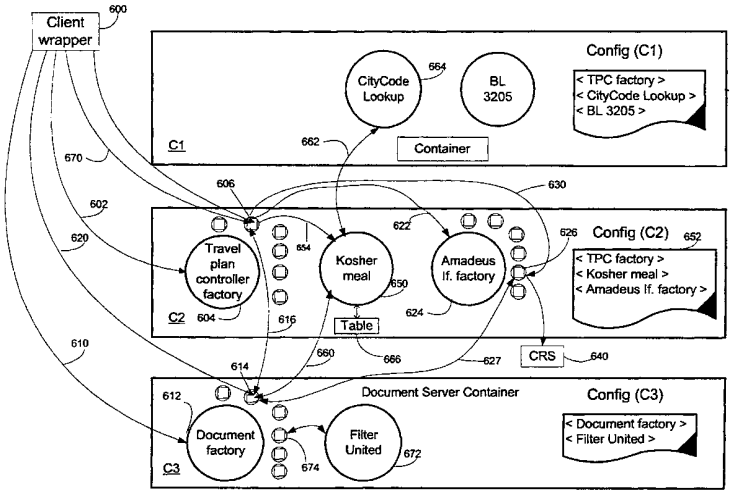
(81) Designated States *(national)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, BZ, CA, CH, CN, CR, CU, CZ, DE, DK, DM, DZ, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) Designated States *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A SOFTWARE COMPONENT-CONTAINER FRAMEWORK FOR DYNAMIC DEPLOYMENT OF BUSINESS LOGIC COMPONENTS IN A DISTRIBUTED OBJECT ENVIRONMENT



(57) Abstract: This application pertains to multi-tiered, distributed computing systems and, more specifically, is directed to a component container middleware framework for dynamic deployment of business logic software components on a CORBA distributed object infrastructure. Each server container includes a configuration file which is read upon initializing the container. The container process loads software components as specified in the configuration file, and the configuration file includes information on names and locations of components creators, and defines relationships among components. The configuration file, expressed in XML syntax, makes it easy to deploy or update business logic components without exposing system source code, while maintaining a robust, scalable distributed system for demanding applications like supporting travel web sites.

WO 01/24003 A1

A SOFTWARE COMPONENT-CONTAINER FRAMEWORK FOR DYNAMIC DEPLOYMENT OF BUSINESS LOGIC COMPONENTS IN A DISTRIBUTED OBJECT ENVIRONMENT

Related Applications

This application is a continuation of and claims priority from U.S. Provisional Applications No. 60/156,499 filed September 28, 1999, and No. 60/184,945 filed February 25, 2000.

Technical Field

This application pertains to multi-tiered, distributed computing systems and, more specifically, is directed to a component container middleware framework for dynamic deployment of business logic software components on a CORBA distributed object infrastructure.

Background of the Invention

On-line travel is one of the largest single e-commerce categories today. First-generation travel sites, those erected roughly in 1994 to 1996, were full of images and copy describing travel destinations and packages. These were essentially on-line brochures or "brochureware" and lacked interactivity. Sites were hard to maintain and were populated with static content.

On-line booking was introduced in approximately 1996 in second-generation travel sites. Many airline sites and virtual travel agents introduced booking services for air, hotels and rental cars. For the first time, individuals had access to information previously available only through travel agents. A revenue stream developed, but profitability remained elusive due to the cost of accessing reservation systems such as Apollo and SABRE, high-maintenance costs, commission caps, etc. There remained a high ratio of "lookers" to "bookers."

The third-generation travel sites, those emerging today, provide a high degree of personalization as distinguished from their predecessors. Modern travel sites present dynamic content based upon information volunteered by millions of users. Pages are presented dynamically upon information stored in the member's profile. Such profiles can include home airport, seating preferences, usual destinations, membership award status, etc.

Supporting all of this interactivity, and the loads imposed by thousands of simultaneous users, present challenges to the computer programers and architects that design such systems. The present invention is directed to a "middleware" framework, *i.e.*, the software that lies between a Web site and a computer reservation system such as SABRE. Such a system must be robust, scalable and reliable.

More specifically, such a system should provide to its users, *i.e.*, travel-site system integrators, the ability to rapidly modify their own business logic software and deploy new business logic objects quickly and easily. If one can isolate modules of functionality and develop a simple and flexible means for introducing those modules of functionality into a booking system, system integrators will be able to provide more compelling and effective interface to customers while lowering their development and maintenance costs.

CORBA—the common object request broker architecture—defined by the object management group (OMG), is a protocol definition for communicating among heterogeneous systems. CORBA allows systems using different hardware, operating systems and programming languages to communicate. CORBA allows the interactions between those systems to be defined in a platform-independent way,

giving the system a set of well-defined interfaces. Thus, CORBA provides an infrastructure for distributed computing.

The present invention is implemented on a CORBA platform. Conventionally, a CORBA server, *i.e.,* a single, executable program, contains all the various components that may be required by its users or clients. This executable usually is hard-coded and compiled. A number of different compiled executables are provided to implement the system. Accordingly, to update that software or introduce a new component, code has to be modified, affected servers recompiled, and then the new executables distributed across the appropriate machines. This is a time-consuming and expensive process. It is also an inflexible solution that is difficult to extend. What is needed is an open solution that allows an integrator to introduce a new component into a distributed system quickly and easily, without compromising scalability and performance of that system.

Summary of the Invention

One general aspect of the present invention pertains to multi-tiered, distributed computing systems and, more specifically, is directed to a component container middleware framework for dynamic deployment of business logic software components on a CORBA distributed object infrastructure. The invention is described by way of example and not limitation by presenting salient features of such a system that implements a travel planning and booking system. Such a system can be used to allow consumers, using a web browser and connected via the internet, to investigate travel options, including airline flights (as well as hotel rooms, rental cars, etc.) and actually book their own reservations, i.e. purchase tickets "on line."

The booking engine is built on top of a unique CORBA framework. This "middleware" framework is the basis for the travel system's flexibility, scalability, and monitoring capability. These same services are available to programmers implementing new business objects for the travel system or even separate applications that will co-exist with the existing system. Thus, in practical application, the present invention allows system integrators to easily deploy new business rules or logic into a robust,

scalable, distributed system not only without rebuilding that system, but indeed without even having access to the existing system source code.

The present framework is based on a component-container model. The framework includes a standard "container" that can contain "components." The container preferably is a generic CORBA server that can load one or more components, based on a "configuration file." The container automatically reads the configuration file when it starts up and it loads the components listed in the configuration file.

Components are implemented by deriving from a framework base class. The programmer implements the desired functionality in the component objects and loads the resulting objects into the container by adding them to the container's configuration file as further explained later. The components are implemented as shared libraries, so the container can load them at run time without recompiling or linking the generic server container. Figure 2 illustrates how the container and components fit together; this will be explained in greater detail below.

The generic server container provides a rich set of services to components it contains. These services relieve much of the burden of creating components by handling the details required for load balancing, monitoring, configuration, logging, and threading. Illustrative services that the generic server and component base classes can provide include the following:

- Reading XML configuration files
- Loading components based on the configuration file
- Establishing connections between components listed in the configuration files
- Setting properties of the components based on the configuration file
- Three different threading models for components
- Publishing and consuming events
- Logging
- Administrative functions to allow monitoring components and the container
- Gathering and publishing statistics for components
- Component pools—part of the framework, but not generic server or component-base classes

· Dynamic properties for trading

In addition, the new framework can implement higher level services that are built on top of the generic server and component model. These higher level services can include, for example, a trader service that is specifically tuned for load balancing, and a document server for storing, querying, and persisting XML documents for state management. In this way the new framework allows a programmer to focus on creating the business logic of components without worrying about the myriad of details required to build a robust and scalable system. The generic server container implements many of these services for any components it contains-nothing special is required in the components as long as they inherit from the framework base classes.

The present invention thus can be described as a configurable middleware framework comprising a client layer interface; a container process for supporting one or more business objects or components; and a resource interface such as an interface to an existing computer reservation system. The container process has access to a configuration file, and can read XML, for configuring the container process at run time to include components specified in the configuration file. These can include internal components for loading into the logical container, and well as references to external components. The configuration file is implemented in XML, while the components themselves are not restricted in terms of implementation other than a CORBA compliant interface.

The generic server also allows components to extract information from the configuration files through interfaces in the component base class. This allows a programmer to easily store parameters or other information in the XML configuration file and modify those parameters without changing any code. The process of reading and parsing the XML is handled entirely by the generic server container and component base class, so the programmer doesn't have to worry about file formats or parsing issues. The present invention thus enables an integrator to add new business logic -- and deploy it over a distributed system -- without taking everything down; without modifying and recompiling source code; and without having to copy recompiled

source code to each and every server. In fact, an integrator or other authorized user can add or modify functionality without exposing source code.

Additional objects and advantages of this invention will be apparent from the following detailed description of preferred embodiments thereof which proceeds with reference to the accompanying drawings.

Brief Description of the Drawings

Fig. 1 is a high-level block diagram of a multi-tiered, distributed computing system to implement a robust, flexible, scalable, high-performance system for connecting end-users, via the internet, to a back-end resource such as a computerized reservation system.

Fig. 2 is a conceptual diagram of a generic container for loading software components based on a configuration file according to the present invention.

Fig. 3 is a conceptual diagram illustrating a configuration file specifying connections between components.

Figs. 4A and 4B together form an example of a configuration file implemented in XML for a city code lookup process.

Figs. 5A-5C illustrate software processes, each of which implements a generic container having a corresponding configuration file.

Fig.6 is a simplified diagram illustrating operation of a multi-tiered, distributed computing system to support a travel planning and reservation web site.

Fig. 7A illustrates a portion of an XML configuration file corresponding to the container process of Figure 5B.

Figs. 7B-7C comprise a pseudo-code listing of an implementation of a Kosher meals component of the type deployed in the container of figure 6.

Figs. 8A-8B illustrate by example a flight schedule request expressed in XML.

Figs. 9A-9B illustrate by example a flight schedule response expressed in XML.

Figs. 10A-10B illustrate a city code lookup process (container) configuration file.

Fig.11 is an illustrative portion of a travel plan controller process (container) configuration file.

Figs. 12A-12C illustrate XML data, a DOM hierarchical model of the XML data and a corresponding node list, respectively.

Figs. 13A-13B sets forth an interface definition for document server object.

Detailed Description of Preferred Embodiment

Figure 1 illustrates a three-tiered e-commerce architecture with CORBA providing the distribution mechanism. In the top tier, an HTTP server 102 is coupled to the Internet 104 to provide access to multiple end-users 106 running standard Web browsers, such as IE or Navigator. Behind the Web server 102 is an application server, such as BroadVision application server 108. The application server interacts with the business logic and framework "middleware" through a client wrapper layer 110 provided by the middleware vendor. The client wrapper layer can be implemented, for example, using javascript and provides methods appropriate to the application, in this case air travel, hotel reservations and car rental. The client wrapper does not implement business logic. Business logic is implemented at the framework (middleware) level and at the user interface level via java scripts.

The second, intermediate tier implements the business logic and middleware framework of the present invention. This tier includes an XML document store 112, framework objects 114, business objects 116 and various interfaces for interfacing with computer reservation systems and/or inventory sources. By way of illustration, figure 1 shows a database access layer 120 for connection to a customer profile database 122 and it further shows a CRS (Computer Reservation System) interface 126 for interfacing to a computer reservation system 130. This drawing is highly simplified; for example, various of the framework objects 114 and business objects 116 may include factories for creating multiple instances of corresponding objects, and these various objects will be distributed in most applications across multiple servers. As noted above, a central theme of the present invention is to implement a middleware framework that greatly simplifies a deployment of business

objects 116 without exposing the system source code. The business logic that drives the site include such things as ticketing rules or restrictions on a particular itinerary. Business logic components can implement policies for pricing, discounts, seat assignments, etc. In one example, described later, a kosher meals object confirms the availability of kosher meals when requested. The CRS interface 126 in a practical implementation is likely to manage on the order of 1,000 simultaneous connections with CRS mainframes such as the SABRE System.

Figure 2 illustrates a generic server container according to the present invention as mentioned in the summary above. Components are implemented by deriving from a framework base class. The programmer (or travel site integrater) implements desired functionality in the component objects and loads the resulting objects into the container by adding them to the container's configuration file 202 in Figure 2. The components are implemented as shared libraries, so the container 204 can load them at run time without recompiling or linking the generic server container. The configuration files, expressed in XML syntax, are the basis for specifying which components the generic server should load, and how they should be connected together.

When the generic server starts up, one of the first things it does is look for a configuration file based on a name passed on the command line for the server. There is a separate configuration file for each server process in the preferred embodiment, as well as a shared configuration file that the separation configuration files can include. This makes it easy to put common elements, such as certain shared object references in the shared file where every server can use the information.

In addition, the configuration file can specify connections between components that implements specific framework-defined interfaces. The connections can include components that are located within the same server process (internal components), components located in other generic server processes, and even objects located in other processes not implemented within the framework (external components). Any CORBA object whose object reference can be entered in the configuration file in string form can be connected—even if the object is written in another language supported by

CORBA. Figure 3 illustrates a connection between two components as specified in a configuration file. In this case, component 1 requires a validator service, and a connection is provided in the configuration file to component 2, which implements the validator interface, and happens to be located in another server, identified by the component 2 object reference listed in the configuration file.

Threading Policies

The generic server also supports three different threading models for handling requests for the objects it contains.

1. Single Threaded. This is the simplest model. Each request coming into the generic server for any object contained in that server process is handled serially. Incoming requests are stored in a queue and each request executes to completion before the next request is handled. Notice that this model applies to all the components contained in a server process, so only one request for any component in the server process is handled at a time.

2. Thread per Request. The generic server container also supports a model where each incoming request is handled by a separate thread. Each new request causes a thread to be created to handle that request, and the thread executes until the request is completed. Then the thread terminates. A parameter specified in the XML configuration file can set the maximum number of threads to be created. If this maximum is reached, new requests are blocked until at least one previous request finishes and its thread terminates.

3. Pool of Threads. In this model, the generic server starts a number of threads specified in the configuration file at startup. Incoming requests are assigned a thread from the pool of available threads. If no threads are available when a request comes in, the request waits in the queue until a thread becomes available. The threading model used is specified by the configuration file. However, it can also be changed at run time. The generic server also checks the thread-safe attribute of each component it loads and forces the single threaded model if any of the components it is loading are not thread-safe. In addition, the generic server container handles thread maintenance issues, such as handling dead or hung threads, and reporting statistics on thread usage.

Figures 4A and 4B illustrate a configuration file, in this case, for a server identified as the city code lookup server. This provides a city code lookup capability which is made available to the application server through the client wrapper layer of Figure 1. So, for example, if an end-user of the travel Web site enters a city name which is not unique (Portland, Oregon/Portland, Maine) or uses an airport abbreviation (PDX) or misspells the city name, the application server can call on the city code lookup to obtain one or more valid alternatives. Referring to Figure 4A, the first portion of this file lists attributes of the container, such as time out, thread policy and maximum number of concurrent threads ("MaxThreads").

<InternalComponents>

Internal software components are listed in this section, in this case, beginning with the component name "CityCodeLookup". For each internal component, the configuration file indicates a creator name and location, relationships (in this example there are none) and attributes. A presently preferred embodiment, a component can have three types of attributes: simple, structure and sequence. In this example, all of the listed attributes are of the simple type.

The CityCodeLookup configuration file continues on Figure 4B. Here, the next component is listed beginning with the component name "CityCodeStrictLookup." Once again, a creator name was provided and a location of the creator, *i.e.*, a library. There are no relationships in this example, and again various attributes are listed. Accordingly, when the CityCodeLookup server or container process initializes, it will read this configuration file, establishing the general attributes as noted, and then it will load (create) instances of the CityCodeLookup and CityCodeStrictLookup components. The simple-type attributes, and their corresponding values, provide the means for passing perimeters to these components. This enables the integrated to change the perimeters for a given business logic component simply by editing the XML configuration file.

Also, the configuration file defines the order of calling components simply by the order in which they appear in the configuration file. This allows new business logic (a new component) to be inserted at the correct point in an existing process

simply by inserting it at the corresponding point in the component list in the configuration file. As further explained later, the configuration file can define relationships or "connections" between components; for example, defining which components use what other component's services. By "sequencing" a new component into the configuration file list at the correct location, in defining relations to other components, a new logic component can be fit into the overall system without changing any existing source code. Put another way, the present component container architecture presents a higher level of abstraction beyond, but still compatible with the CORBA infrastructure.

Figure 5A presents a simple example of a container process C1 and corresponding configuration file "Config (C1)". Container 500 includes a travel plan controller component 502, a kosher meal component 504 and any other business logic component "XX" 506. Figure 5B illustrates a second container process 510 that includes the components 502 and 504 previously mentioned, and introduces a new component, Amadeus Interface Factory 512. The interface factory creates interface objects as necessary, illustrated by interface objects 514. The Interface Factory component is added to the container by listing it in the corresponding configuration file "Config. (C2)". These configuration files are simplified for purposes of illustration and in practical application would contain more details such as those of Figure 4A-4B. Figure 5C illustrates another container process C3, in this example a document server container. This container includes two components, namely a document server component 520 and a Filter United component 516, as listed in the corresponding configuration file "Config. (C3)". The document server creates individual document objects as necessary, for example, objects (or "documents") 522 and 524.

In the travel site application, a document will be created for each end-user "session" handled by the Web site application server. The application server maintains session state information, and calls on the document server (the document factory) to generate a new document, and conversely, it can notify the document container to discard a given document when the corresponding session is concluded.

The individual document objects store user/session information, such as a flight itinerary, in an XML format as further described later.

Figure 6 illustrates operation of the software system described above. Referring now to Figure 6, at the beginning of a client session on the travel planning Web site, the application server (see figure 1), using the client wrapper interface 600, makes a call 602 to a travel plan controller factory component 604 shown in container C2. This is essentially a request for the factory to create a travel plan controller instance 606. This object 606 will implement the travel planning process for this particular session. The client wrapper 600 also is used to make a call 610 to a document factory 612, in this case, located in a document server container C3. The document factory 612 creates document objects, in this example, document object 614, for storing information associated with this current session. The corresponding travel plan controller 606 communicates with the document 614 via path 616.

Assume that the end-user in the current session requests information about flights between given locations on a specified date, etc. This flight request information is written to the session document 614 via call 620 in an XML syntax. The travel plan controller (TPC) 606 makes a call 622 to the Amadeus IF Factory 624 to request an interface to the CRS. The factory 624 creates (or assigns from a pool) an interface object 626 as requested; makes it available to the TPC. The IF 626 reads the flight request information 627 from the document 614 using a query language explained later, and creates a flight schedule request in suitable form to present the flight schedule request to the computer reservation system 640. The IF receives a flight schedule response and writes that information into the document 614, again formatted into an XML Syntax. The client wrapper 600 can read the updated document via another call, and provide this information to the application server which, in turn, provides the flight information to the Web server (See Fig. 1). Note that the client wrapper layer provides an interface to the document for the application server to access XML data.

We next assume that the end-user selects her itinerary from among the flights offered. The selected flight information is written to the session document 614

through the client wrapper as described. Before the session is concluded, various business logic components may be called to modify and/or validate travel plan information as appropriate. These other components can be called by the travel plan controller for this session (606), and they will interact with the data stored in the corresponding document 614. Numerous components can be implemented to provide a variety of business logic, as mentioned above, such as pricing, seating assignments, special fares on discounts, mileage club, accommodations and even food requests. We illustrate this last feature by way of a kosher meal business logic component 650. The kosher meal component is shown deployed in container C2 and therefore appears in the corresponding configuration file 652.

The kosher meal component is to determine whether or not kosher meals are available on a flight requested by the end-user. For example, if a kosher meal is requested, the travel plan controller makes a call 654 to the kosher meal component 650. Component 650's job is to determine whether kosher meals are available on the flight segments requested. Two requirements must be satisfied: first, the departure city must be equipped to provide kosher meals; and second, a minimum number of days of advance notice is necessary to provide kosher meals. The number of days of advance notice required will be passed to the kosher meal component, via the configuration file C2, at run time as further explained later. In response to the call 654, the kosher meal component 650 makes a query 660 to the document 614. To determine, for each flight segment, two pieces of information; namely, the departure date and the departure city code. The kosher meal component 650 then makes a call 662 to a CityCodeLookup component 664 to determine whether the departure city indicated is capable of providing kosher meals. We assume this information is available to the CityCodeLookup, as it would have information about virtually all the commercial airports in the world. If kosher meals are available at the indicated departure city, and if the prior notice requirement is met, in terms of the number of days from ticketing (or the current date) to the departure date, the kosher meal component 650 will modify the document 614 to indicate that kosher meal is available on the corresponding flight segment. This process can be repeated for each flight

segment on the itinerary. The kosher meal availability could be implemented with a Lookup table 666, but utilizing the city code lookup component would be preferred so that airport information is collected in one place. Travel plan controller 606 can provide this updated information to the client wrapper 670. An illustrative implementation of a kosher meal component is provided in Figures 7B-7C. This component uses the query language described below to access data stored in the session document object.

Of course, many other user sessions can be processed at the same time. The application server, via the client wrapper, will request travel plan controllers as necessary and corresponding documents from the document factory. In the figure BL-205 merely identifies another business logic component. Another component, illustrated in container C3, is a filter component called "Filter United" 672. The component framework includes a generic operation called filter. This is used to modify a document that the system is working on. It essentially examines the document and deletes certain information. In this case, Filter United has the task of removing from the document United Airlines flights that appear in the flight schedule response data. To add any new filter, the programmer can simply write one, derived from the base filter class, and add it into the list of filters in the configuration file. For illustration, we show the Filter United updating a document 674.

Figure 7A illustrates the C2 configuration file in greater detail. The configuration file, shown in XML syntax, begins with the server name and attributes of the server. Next, it lists the three internal components, travel plan control, kosher meals, Amadeus IF Factory, as mentioned before. For each component name, the configuration name includes a creator name and a library location of that creator. It also includes attributes and relationships to other components, as appropriate. For the component KosherMeals, a creator name and library location of the creator are listed. Just one attribute is shown, of a simple type, called "PriorNotice" with a parameter value of 5, indicating five days advance notice required for kosher meals. A container process reads the configuration file and provides this attribute information to the KosherMeals component as a parameter at run time. In this way, the advance notice

requirement can easily be changed by simply editing the XML configuration file. The file also shows that component KosherMeals has a relationship in that it uses the city code lookup component.

Figure 8 is an example of a flight-schedule request. The various elements of the flight schedule request are summarized in the following table.

Table 1. Flight Schedule Request Method Element List:

Element : FlightSchedule
Description: Contains all the information to perform a flight availability and flight schedule request and display the results.
Attributes : None
SubElements : Request – Below, 1 or more, required
Element : Request
Description: Contains all the information to perform a flight availability request.
Attributes : ID – unique identifier
SubElements : RequestedFlightSegment – Below, 1 or more, required
ClassOfService – Below, exactly 1, required
PassengerType: Below, 1 or more, required
ValidCarriers: Below, 0 or 1, optional
Element : RequestedFlightSegment
Description: Contains all the information for one flight query.
Attributes : ID – unique identifier
SubElements : Departure – Below, exactly 1, required
Arrival – Below, exactly 1, required
IncludeConnectingCities – Below, 0 or 1, optional
Element : Departure
Description: description of the departure point
Attributes : Date – departure date
Time – departure time
SubElements : City – TravelPlanXML, exactly 1, required
Element : Arrival
Description: arrival city
Attributes : None
SubElements : City – TravelPlanXML, exactly 1, required
Element : IncludeConnectingCities
Description : List of required connection cities, which the flight choices must include as stopover cities
Attributes : None
SubElements : City – TravelPlanXML, exactly 1, required
Element : ClassOfService
Description: cabin request for all segments
Attributes : Code – cabin code for requested cabin
SubElements : None

Element : ValidCarriers
Description: limiting list of carriers which are acceptable for the returned flights
Attributes : None
SubElements : Carrier – TravelPlanXML, exactly 1, required
Element : PassengerType
Description: Number of passengers to request availability for – the sum of all passenger type totals will be used in the request
Attributes : Number – number of passengers in this type
SubElements : None
Element : NumChoices
Description: maximum number of flight choices to return
Attributes : Value – number of passengers in this type
SubElements : None
Element : FlightNumber
Description: This element contains the flight number the user wants to get schedule or availability information on.
Attributes : Value – The number of the flight.
SubElements : None

The example of Figure 8 shows virtually all of the elements that might be included in a flight availability search, although not all of them are required, as indicated in the table. All the information in the flight schedule request and in the flight schedule response are stored in the corresponding session document, as mentioned above. Figures 9A-9B illustrate a flight schedule response. The response begins with the corresponding flight schedule request for identification. The response has an identification number, and includes two choices (flights), choice identification numbers 1 and 2.

Figures 10A-10B form a listing of the city code lookup server configuration file. The file includes general attributes (applicable to the entire container) and internal components. Each of the internal components "CityCodeLookup" and "CityCodeStrictLookup" has multiple attributes. Again, these attributes are passed to the component as parameters at run time. Figures 11A-11B illustrate a portion of a travel plan controller container configuration file. The configuration file includes a travel plan controller factory of the type discussed above.

Document Component

The Document component is a component that internally contains an XML document and whose external interface provides clients with the ability to set and retrieve certain portions of the document at a given time. The XML document preferably is stored internally using a C++ DOM implementation. Most document users, however, will not need to deal with either XML or DOM directly, because most reading and writing of Document information can be done using the Node and NodeAttribute structures, using either a node id or an XQL query to identify a node or a group of nodes.

As noted, session data is stored in the document objects in XML format. Figure 12A illustrates the hierarchical association of tags in XML format. Figure 12B illustrates the DOM model in which the tags have a tree structure, and each node has a corresponding node id number. The present invention implements, in its query language, a node list concept. This is a list of tags and corresponding node id's. A **node id** is an internal unique identifier that the Document process uses to identify individual nodes. Users of Document should not try to determine a node id on their own, but should only use values that have been obtained from the Document instance they are working on through an operation like get_nodes_by_query. If a call to get_nodes_by_query is successful, then the nodes returned can be relied upon to have correct node ids, their values or attributes can be modified, they can be passed back to the document via the update_nodes_by_node_id, and the changes will be made in the proper spots in the document. However, if one of these node ids were to be extracted, the document saved to a file, and a new document created from that same file, the node id would no longer be valid.

Query Language

A query is a string that conforms to a defined query language. In a presently preferred embodiment of the invention, a query language suitable for accessing the content of a document is a subset of XQL (XML Query Language), which is an extension of XSL (Extensible Stylesheet Language). More information on these

languages is publicly available. Following is a table containing examples of the query commands our Document component supports and what they do:

/	Identifies the root node.
./	Identifies the current context node.
../	Identifies the parent node of the current context node.
Code	Selects all of the Code nodes that are children of the current context node.
./Code	Selects all of the Code nodes that are children of the current context node.
/Code	Selects all of the Code nodes that are children of the root node.
City/Code	Finds all of the City node children of the current context, and selects all of their Code node children.
/City/Code	Finds all of the City node children of the root node, and selects all of their Code node children.
//	Performs a recursive descent starting at (but not including) the root node.
.//	Performs a recursive descent starting at (but not including) the current context node.
//Code	Selects all of the Code nodes anywhere in the document.
.//Code	Selects all of the Code nodes anywhere below the current context node.
City/*	Finds all of the City node children of the current context, and selects all of their children nodes.
*/City	Select all City grandchildren of the current context.
/City//	Finds all City nodes that are children of the root node, and performs a recursive descent starting at (but not including) each one.
City//Code	Finds all City nodes that are children of the current context node, and selects all of their Code descendants.
//City/Code	Finds all City nodes within the document, and selects all of their Code node children.
City/@identifier	Finds all City node children of the current context, and selects only the identifier attribute from each.
City/@*	Finds all City node children of the current context, and selects only the attributes (excluding the node's name and value) from each.
City[@identifier]	Selects all City nodes that are children of the current context node and have an identifier attribute.
City[Code]	Selects all City nodes that are children of the current context node and have at least one Code child node themselves.
City[@identifier="MyCity"]	Selects all City nodes that are children of the current context and whose identifier attribute value is "MyCity".

The present invention departs from standard XQL in that the symbols Document supports includes addition of the ! operator as a modifier to the // behavior.

XQL defines the // operator to be a recursive descent beginning at a specific node. Therefore, .// would indicate a recursive descent starting at, but not including, the current context node. This is significant in our use of XQL because a common XML query will be to find certain nodes and select them and all their descendants. For example, you might want to select all City nodes and their descendants anywhere in the document. Following XQL by the letter, you could write the query //City, which would select all City nodes anywhere in the document, but would not select their descendants. You could also write the query //City//, which would find all City nodes anywhere in the document and select their descendants, but not the City nodes themselves. Therefore, you'd have to do at least two queries to select the City nodes and their descendants, a potentially expensive process if being done across a network, and certainly one that will return a messy result. Our solution was to allow a ! operator, directly following a //, to modify the // behavior to include the starting node. This would allow the following queries:

/City//!	Finds all City nodes that are children of the root node, and performs a recursive descent starting at (and including) each one.
City//Code//!	Finds all City nodes that are children of the current context node, finds all of their Code descendants, and performs a recursive descent starting at (and including) each one.
.//!Code	Selects all of the Code nodes if the current context node or anywhere below the current context node (a very tiny difference.)
City//!Code	Finds all City nodes that are children of the current context node, and selects all of their Code descendants. In this case, the ! is useless, because it would modify the // behavior to check each City node to see if it is a Code node, which obviously would never be true.
//!City/Code	Finds all City nodes within the document, and selects all of their Code node children. Again, here the ! is useless, because it would modify the // behavior to see if the root node is a City node, which would

never be true because the root node is the document and is not an XML node.

The **current context** is an important concept when using a query. If a query begins with a '/', it begins at the root, which is the base of the document tree. However, if it doesn't begin with a '/', it begins at the current context. The current context is analogous to a pointer or cursor in other technologies, in that it refers to a specific place in the document. When a document is loaded from a file or otherwise created, the current context is set to the root. From here the current context can be moved to any node in the document. Once the current context is set, it remains until changed explicitly or until the document is destroyed. This allows queries to be applied only to a specific portion of the document, if desired.

Some operations that operate by query, such as set_current_context_by_query and add_nodes_at_query, can only perform their function if the query specifies a single node in the document. In other operations, such as get_nodes_by_query, queries can specify and select multiple nodes. A simple example of use of these types of queries is illustrated in the Kosher Meals example of Figures 7B-7C.

It will be obvious to those having skill in the art that many changes may be made to the details of the above-described embodiment of this invention without departing from the underlying principles thereof. The scope of the present invention should, therefore, be determined only by the following claims.

Claims

We Claim:

1. A multi-tiered, distributed object framework comprising:

(a) a client wrapper layer for interaction with a client process;

(b) a middleware layer including a container process for hosting one or more business logic objects; and

(c) a resource interface layer for connection to and interfacing with a computer reservation system;

(d) the container process having access to at least one configuration file for configuring the container process at run time by loading into the container one or more components specified in the configuration file.

2. A framework according to claim 1 wherein the configuration file includes identification of at least one external component.

3. A framework according to claim 1 wherein the configuration file includes at least one attribute of the container process.

4. A framework according to claim 3 wherein the container attribute comprises selection of a predetermined threading policy.

5. A framework according to claim 1 wherein the configuration file includes, for the specified component, identification of at least one attribute for passing to the component at run time as a parameter.

6. A middleware framework according to claim 5 wherein the component attribute comprises a selected parameter and a value of the selected parameter, for passing the value of the selected parameter to the said internal component by editing the value in the corresponding configuration file.

7. A framework according to claim 1 wherein the configuration file is formed in XML syntax.

8. A framework according to claim 1 wherein the configuration file is implemented in a text file.

9. A framework according to claim 1 wherein the configuration file is encapsulated as a software object.

10. A framework according to claim 1 wherein the configuration file includes for the specified component, identification of at least one relationship of that component to another component.

11. A document server container process comprising:

a document factory component for creating document objects;

each document object encapsulating data stored internally in an XML syntax;

each document object further implementing an interface for accessing the encapsulated data; and

the document server container process having access to a configuration file associated with the document server container for configuring the container process at run time by loading into the container one or more software components specified in the configuration file, including the document factory component.

12. A document server container process according to claim 11 wherein the document object implements a hierarchical tree structure of the encapsulated data, and assigns to each node of the XML data, a corresponding unique internal node identifier (id); the node id corresponding to a specific node within a hierarchical tree model.

13. A document server container process according to claim 12 wherein the document interface maintains an indication of a current context, for executing query language commands on the encapsulated data relative to the current context.

14. A document server container process according to claim 12 wherein the document object implements a node list, the node list including a list of tags of the XML data and a corresponding node id for each tag.

15. A document server container process according to claim 12 wherein the document interface implements a predetermined query language for accessing the encapsulated data; the query language including commands for accessing the data based on node identifiers.

16. A document server container process according to claim 15 wherein the query language implements commands for obtaining query results including node identifiers.

17. A method of storing and maintaining dynamic session data in a distributed object environment, comprising the steps of:

providing a document factory for creating a document object for each session;

in each document object, storing selected session data in an XML syntax;

implementing a hierarchical internal tree structure associating the data stored in the document object;

assigning to each tag of the stored data a unique internal node identifier corresponding to the tag location on the tree structure; and

in each document object, implementing an interface for accessing the session data stored in the object based on the assigned node identifiers, thereby encapsulating the actual XML syntax data.

18. A method of storing and maintaining dynamic session data according to claim 17 wherein the document object implements a predetermined query language that includes query commands that return selected node identifiers.

19. A method of storing and maintaining dynamic session data according to claim 18 wherein the document object implements a predetermined query language that includes query commands that update the document object at node identifiers specified in a query command.

20. A method according to claim 18 and further comprising maintaining an indication of a current context in the document, for executing query language commands on the encapsulated data relative to the current context.

FIG. 1

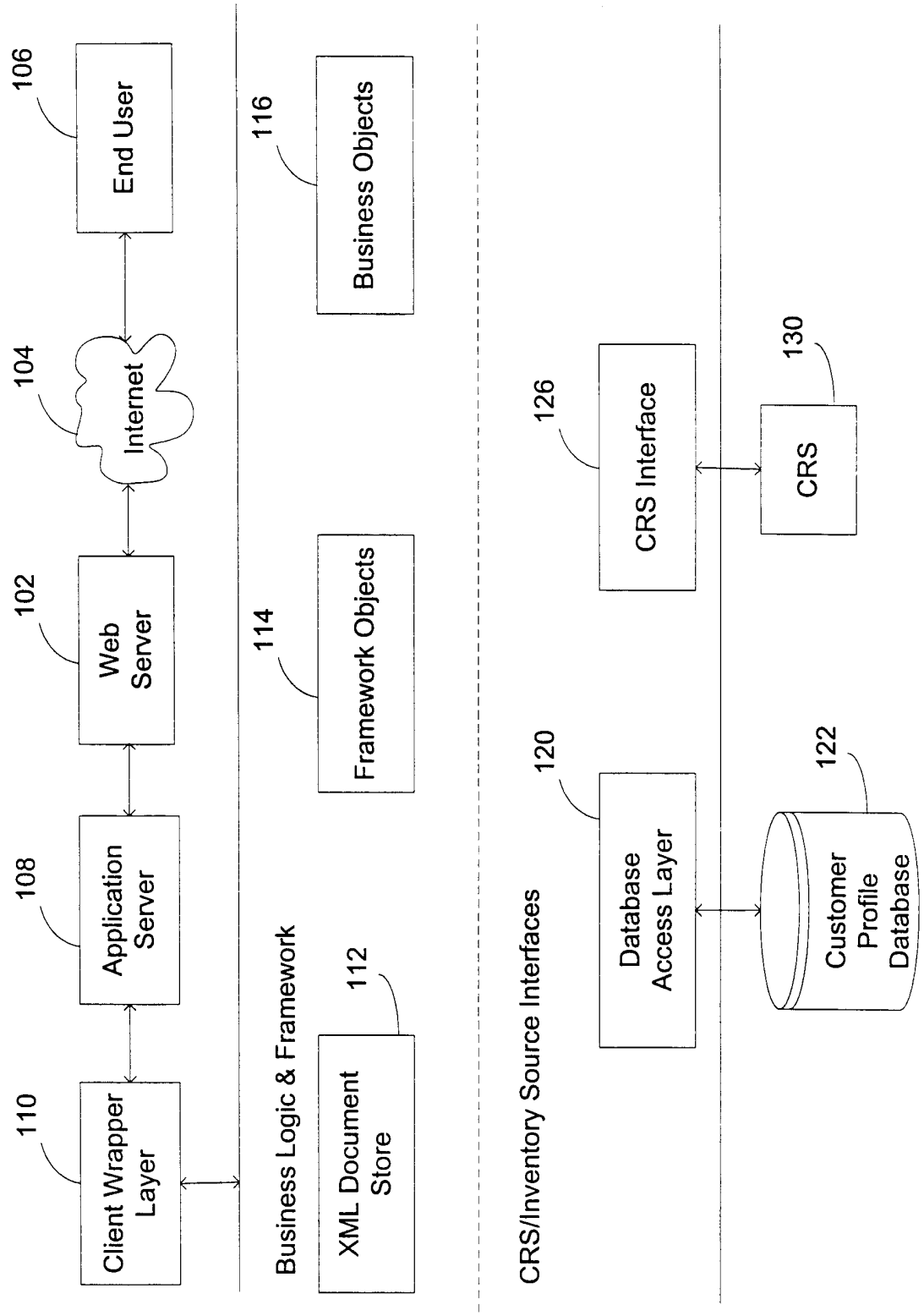

106
End User
104
Internet
102
Web Server
108
Application Server
110
Client Wrapper Layer
116
Business Objects
114
Framework Objects
112
XML Document Store
Business Logic & Framework
CRS/Inventory Source Interfaces
126
CRS Interface
130
CRS
120
Database Access Layer
122
Customer Profile Database

FIG. 2

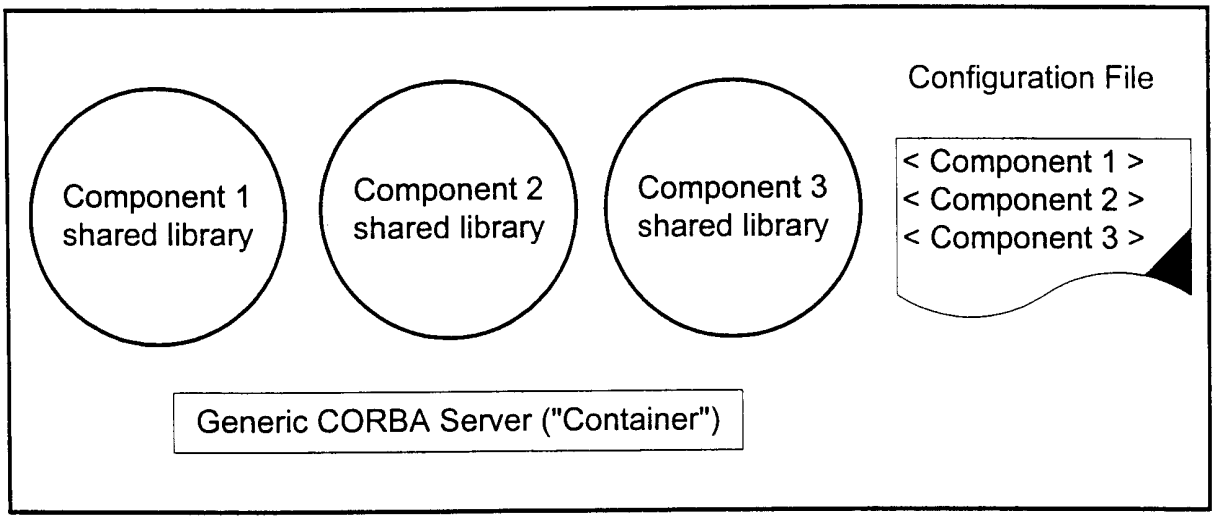

Configuration File
< Component 1 >
< Component 2 >
< Component 3 >
Component 1 shared library
Component 2 shared library
Component 3 shared library
Generic CORBA Server ("Container")

FIG. 3

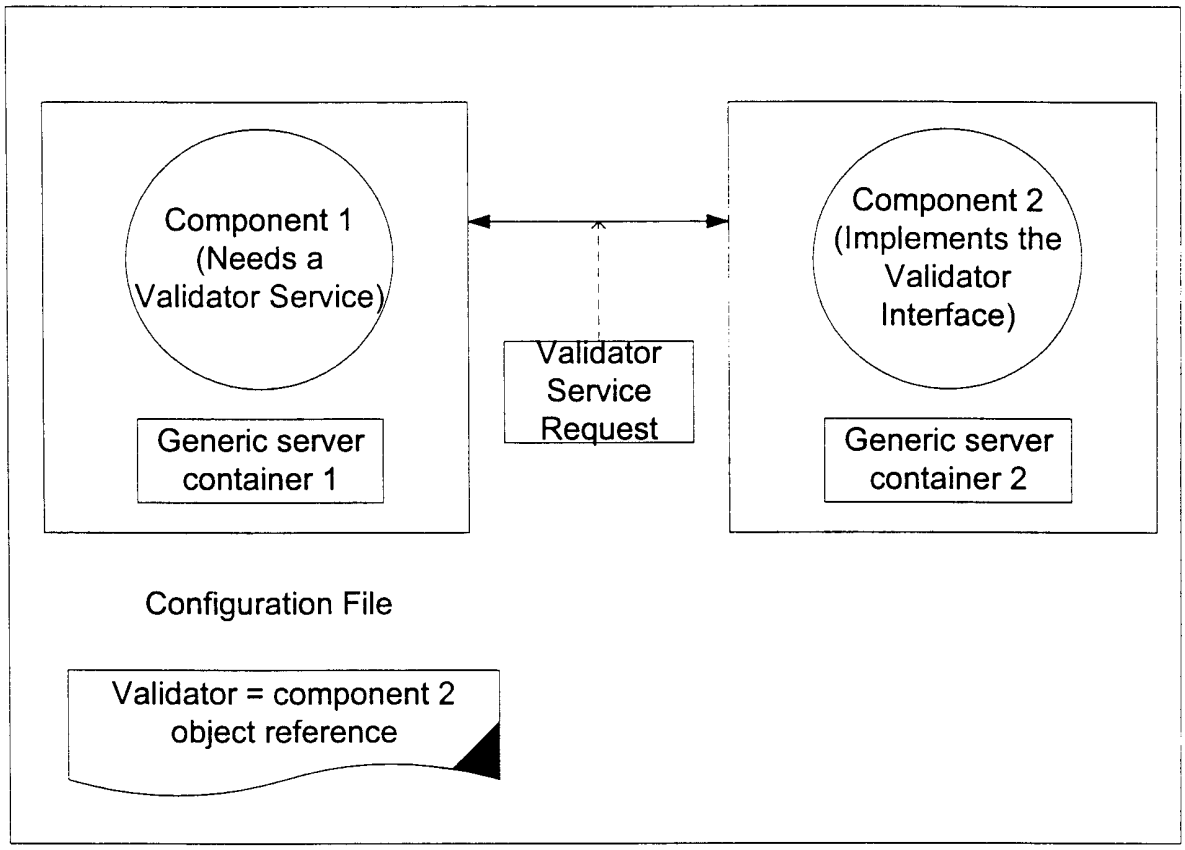

Component 1
(Needs a
Validator Service)
Generic server
container 1
Validator
Service
Request
Component 2
(Implements the
Validator
Interface)
Generic server
container 2
Configuration File
Validator = component 2
object reference

FIG. 4A

```
<Server Name="Bk/CityCodeLookupSvr">
 <Attributes>
   <Simple Name="Timeout">0</Simple>
   <Simple Name="ThreadPolicy">thread-pool</Simple>
   <Simple Name="MaxThreads">20</Simple>
   <Simple Name="Description">This server contains components related to Car related
Code Lookups.</Simple>
   <Simple
Name="IncludeFile">/net/godzilla/usr2/BookSmart/v2.0/run/CORBA/TraderInclude.xml</Si
mple>
  </Attributes>
       <InternalComponents>
   <Component Name="CityCodeLookup">
    <Creator Name="ConverterCreator"
      Library="libSnSFrameworkConverterCreators.so"/>
    <Relationships>
    </Relationships>
                    <Attributes>
                            <Simple Name="ServiceType">CityCodeLookup</Simple>
                            <Simple Name="SubstringMatch">True</Simple>
                            <Simple Name="TagName">City</Simple>
                            <Simple Name="CodeAttribute">Code</Simple>
                            <Simple Name="NameAttribute">Value</Simple>
                            <Simple Name="Library">libora12d0146.so</Simple>
                            <Simple Name="Database">WG80</Simple>
                            <Simple Name="UserName">BROADVISION</Simple>
                            <Simple Name="Password">BROADVISION</Simple>
                            <Simple Name="Table">BK_CITY_MAST</Simple>
                            <Simple Name="CodeColumn">CITY_CODE</Simple>
                            <Simple Name="NameColumn">CITY_NAME</Simple>
                            <Simple
Name="CodeIsPrimaryColumn">ISPRIMARY_FLAG</Simple>
                            <Sequence Name="Attributes" Type="Simple">
                                    <Simple
Name="AttributeColumn">COUNTRY_CODE</Simple>
                                    <Simple
Name="AttributeColumn">CURRENCY_CODE</Simple>
                                    <Simple
                            </Sequence>
                    </Attributes>
            </Component>
```

FIG. 4B

```
<Component Name="CityCodeStrictLookup">
  <Creator Name="ConverterCreator"
    Library="libSnSFrameworkConverterCreators.so"/>
  <Relationships>
  </Relationships>
                    <Attributes>
                        <SimpleName="ServiceType">CityCodeStrictLookup</Simple>
      <Simple Name="VerboseErrors">True</Simple>
                        <Simple Name="SubstringMatch">True</Simple>
                        <Simple Name="TagName">City</Simple>
                        <Simple Name="CodeAttribute">Code</Simple>
                        <Simple Name="NameAttribute">Value</Simple>
                        <Simple Name="Library">libora12d0146.so</Simple>
                        <Simple Name="Database">WG80</Simple>
                        <Simple Name="UserName">BROADVISION</Simple>
                        <Simple Name="Password">BROADVISION</Simple>
                        <Simple Name="Table">BK_CITY_MAST</Simple>
                        <Simple Name="CodeColumn">CITY_CODE</Simple>
                        <Simple Name="NameColumn">CITY_NAME</Simple>
                        <Simple
Name="CodeIsPrimaryColumn">ISPRIMARY_FLAG</Simple>
                        <Sequence Name="Attributes" Type="Simple">
                            <Simple
Name="AttributeColumn">COUNTRY_CODE</Simple>
                            <Simple
Name="AttributeColumn">CURRENCY_CODE</Simple>
                            <Simple
Name="AttributeColumn">ISVALIDHOTELRENTAL_FLAG</Simple>
                            <Simple Name="AttributeColumn">DAL</Simple>
                            <Simple
Name="AttributeColumn">ISETCARRIERCHECK</Simple>
                        </Sequence>
                    </Attributes>
                </Component>

  </InternalComponents>
</Server>
```

FIG. 5A

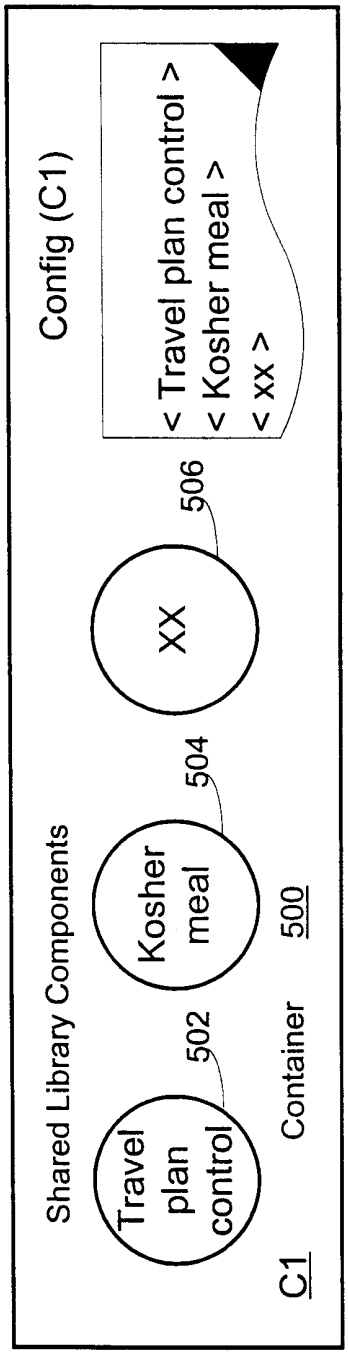


FIG. 5B

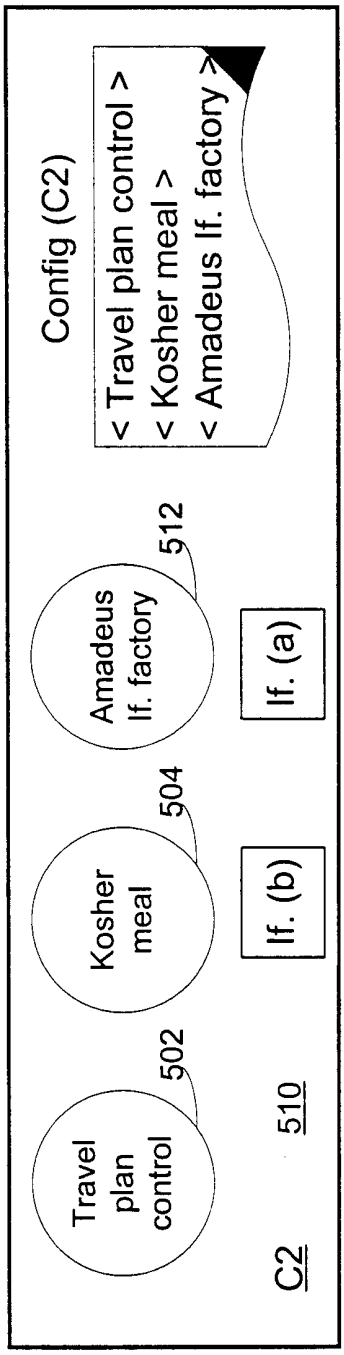


FIG. 5C

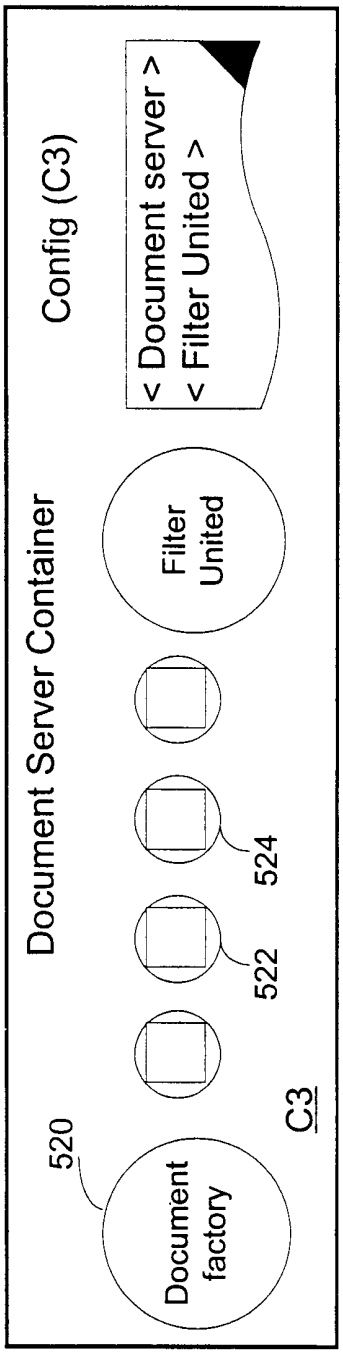
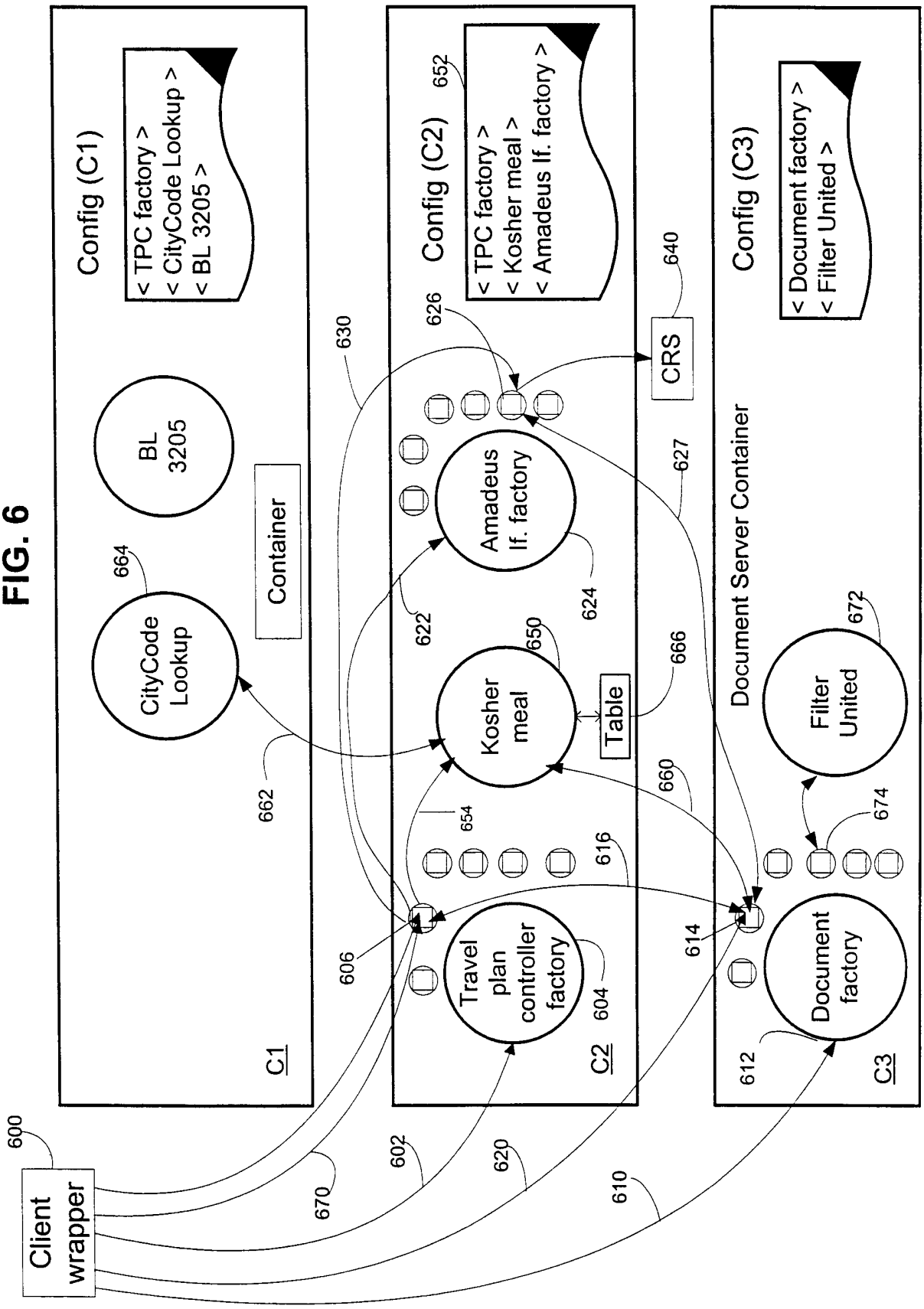


FIG. 6


Client wrapper
600
C1
Config (C1)
< TPC factory >
< CityCode Lookup >
< BL 3205 >
Container
BL 3205
CityCode Lookup
664
662
C2
Config (C2)
652
< TPC factory >
< Kosher meal >
< Amadeus If. factory >
Amadeus If. factory
624
622
626
630
CRS
640
627
Kosher meal
650
Table
666
660
654
616
606
Travel plan controller factory
604
614
C3
Config (C3)
< Document factory >
< Filter United >
Document Server Container
Filter United
672
674
Document factory
612
602
620
610
670

8/21

FIG. 7A

```
>Server Name="ContainerC2Svr">
<Attributes>
        <Simple Name="ThreadPolicy">thread-pool</Simple>
        <Simple Name="MaxThreads">450</Simple>
        <Simple Name="Description">This server contains components related to doing
travel plan operations.</Simple>
</Attributes>
        <Internal Components>
                <Component Name="TravelPlanControl">
                <Creator Name="ControllerCreator">
                        Library="libMDSframeworkControllerCreators.so"/>
                        <Attributes>
                                <Simple Name="UserName">Micah</Simple>
                        </Attributes>
                        <Relationships>
                                <Uses Name="someFactory"
Component="ThatFactory"   Type="Component"  PersistConnection="False"/>
                                <Uses Name="PassengerTypeCodeLookup"
Component="PassengerTypeCodeLookup"   Type="Converter" PersistConnection="False"/>
                        </Relationships>
                        </Component>
                 <Component Name="KosherMeals">
                 <Creator Name="KosherCreator">
                        Library="libMDSframeworkInFlightCreators.so"/>
                        Attributes
                                <Simple Name="PriorNotice">5</Simple>  // 5 days notice
                        </Attributes>
                        <Relationships>
                                <Uses Name="CityCodeLookup" ...
                        </Relationships>
                 <Component Name="AmadeusIF">
                 <Creator Name="IFFactoryCreator">
                        Library="libMDSframework IFFactoryCreators.so"/>
                        Attributes
                                ---
                        </Attributes>
                </Component>
          <Internal Components>
```

FIG. 7B

```
// modify: When called, should be pointed at a FlightSchedule or FlightShopping
// Response node. The function queries all FlightSegments within the Response,
// checks their Departure Date and City information, and if it checks out,
// inserts a XML node that indicates that Kosher meals are available on that Flight Segment.

CORBA_Boolean KosherMealModifier_i::modify(
        SnSFramework_ErrorStore& error,
        SnSFramework_Document_ptr documentIn,
        SnSFramework_DocumentNodeId nodeIdIn,
        const SnSFramework_WString& sSpecialInstructions)
{
    SnSFramework_WString sSegQuery;
    c_str_to_idl_wstring("Choice/Flight/FlightSegment", sSegQuery)

    SnSFramework_NodeList_var varFlightSegments =
            documentIn->get_nodes_by_query_at_node_id(error,
            sSegQuery, nodeIdIn);

    CORBA_ULong nNumOfSegs = varFlightSegments->length();
    for (CORBA_ULong nSegIndex = 0; nSegIndex < nNumOfSegs; nSegIndex++)
    {
        // read departure date and city from segment

        SnSDate departureDate;
        SnSWString sDepartureCityCode;

        SnSFramework_WString sDateQuery;
        c_str_to_idl_wstring("Departure/@Date", sDateQuery);

        SnSFramework_NodeList_var varDepartureDate =
                documentIn->get_nodes_by_query_at_node_id(error,
                sDateQuery, (*varFlightSegments)[nSegIndex].nodeId);

        if (varDepartureDate->length() > 0)
            departureDate = (*varDepartureDate)[0].attributes[0].sValue;

        SnSFramework_WString sCityQuery;
        c_str_to_idl_wstring("Departure/City/@Code", sCityQuery);

        SnSFramework_NodeList_var varCityCode =
                documentIn->get_nodes_by_query_at_node_id(error,
```

10/21

FIG. 7C

```
                    sCityQuery, (*varFlightSegments)[nSegIndex].nodeId);

        if (varCityCode->length() > 0)
            sDepartureCityCode = (*varCityCode)[0].attributes[0].sValue;

        // check data for kosher possibility

        bool bKosherOk = true;

        if (departureDate < SnSDate::today() + m_nKosherAdvanceNoticeDays)
            bKosherOk = false;

        SnSFramework_WString sKosherOkPropertyCheck;
        c_str_to_idl_wstring("ISKOSHEROKAY", sKosherOkPropertyCheck);

        if (!m_varCityLookup->is(error, sDepartureCityCode.idl_wstring(),
                sKosherOkPropertyCheck))
            bKosherOk = false;

        // add kosher available XML if okay

        if (bKosherOk)
        {
            // create <KosherOk Value="True"/> to FlightSegment XML

            SnSFramework_NodeList kosherNode;
            kosherNode.length(1);
            c_str_to_idl_wstring("KosherOk", kosherNode.sName);

            kosherNode.attributes.length(1);
            c_str_to_idl_wstring("Value", kosherNode.attributes[0].sName);
            c_str_to_idl_wstring("True", kosherNode.attributes[0].sValue);

            documentIn->insert_nodes_at_node_id(error,
                kosherNode, (*varFlightSegments)[nSegIndex].nodeId,
                false, false);
        }
    }

    return true;
};
```

FIG. 8A

FlightScheduleRequest XML

```
<BookSmart>
  <FlightSchedule>
    <Request ID="1">
      <RequestedFlightSegment ID="RFS_1"/>
        <Departure Date="09052000" Time="1200">
          <City Code="PDX" Value="Portland, OR" />
        </Departure>
        <Arrival>
          <City Code="RNO" Value="Reno, NV" />
        </Arrival>
        <IncludeConnectingCities >
          <City Code="SEA" Value="Seattle, WA" />
        </IncludeConnectingCities>
        <ExcludeConnectingCities>
          <City Code="SFO" Value="San Francisco, CA"/>
        </ExcludeConnectingCities>
        <NonStopOrConnectingCities>
          <City Code="SEA" Value=" Seattle, WA""/>
          <City Code="SFO" Value="San Francisco, CA"/>
        </NonStopOrConnectingCities>
      </RequestedFlightSegment>
      <BookingCabin Code="Y" Value="Coach" BookingClassCode="B"/>
      <PassengerType Code="ADT" Value="Adults" Number="2"/>
      <PassengerType Code="CHD" Value="Children" Number="1"/>
      <ValidCarriers>
        <Carrier Code="AA" Value="American Airlines"/>
      </ValidCarriers>
```

FIG. 8B

```
        <InValidCarriers>
          <Carrier Code="AS" Value="Alaska Airlines"/>
        </InValidCarriers>
        <PreferredCarriers>
          <Carrier Code="AA" Value="American Airlines"/>
        </PreferredCarriers>
        <ConnectionTypesAllowed NonStop="T" Direct="T" Connecting="T"/>
        <DepartAfter Time="1200"/>
        <DepartBefore Time="1800"/>
        <NumChoices Value="10"/>
        <MaxQueries Value="10"/>
        <FareRestrictions ApplyAll="T" Penalties="T" NonRefundable="T"
         AdvancePurchaseRequired="T"/>
        <FlightNumber Value="239"/>
      </Request>
      <Request ID="2">
        <RequestedFlightSegment ID="RFS_1"/>
          <Departure Date="09102000" Time="1800">
            <City Code="RNO" Value="Reno, NV" />
          </Departure>
          <Arrival>
            <City Code="PDX" Value="Portland, OR" />
          </Arrival>
        </RequestedFlightSegment>
        <BookingCabin Code="Y" Value="Coach"/>
      </Request>
    </FlightSchedule>
</BookSmart>
```

FIG. 9A

FlightScheduleResponse XML

```
<BookSmart>
  <FlightScheduleRequest>
  ...
  </FlightScheduleRequest>
  <FlightScheduleResponse ID="1">
    <Choice ID=1>
      <Flight>
        <Origin Date="03032000" Time="1050">
          <City Code="DFW" Value="Dallas/Fort Worth"/>
        </Origin>
        <Destination Date="03032000" Time="1208">
          <City Code="LHR" Value="London Heathrow, England"/>
        </Destination>
        <ElapsedFlyingTime Value="1010"/>
      <FlightSegment ID="1" Suffix=??? FlightNumber="90"
    IsCommuter="F" IsCodeShare="F"
        FrequentFlyerMiles="445" NumberStops="1"
    IsChangeOfGuage="T"
        ElectronicTicketingQualifier="T">
          <Departure Date="03032000" Time="0600" Terminal="B">
            <City Code="DFW" Value="Dallas/Fort Worth" />
          </Departure>
          <Arrival Date="03032000" Time="2210" Terminal="3">
            <City Code="LHR" Value="London Heathrow, England"/>
          </Arrival>
          <Carrier Code="AA" Value="American Airlines"/>
          <OperatedBy>
            <Carrier Code="" Value=""/>
          </OperatedBy>
```

FIG. 9B

```
<BookingCabin Code="Y" Value="Coach"
                BookingClassCode="M"
              CabinChanged="F"
              Seats="9"/>
<Meal Code="L" Service="1" Value="Lunch"
CabinName="First"/>
<Meal Code="S" Service="1" Value="Snack"
CabinName="First"/>
<Meal Code="S" Service="2" Value="Snack"
CabinName="Coach"/>
<AircraftInformation Code="M80" Value="MDonnel MD80"/>
<LevelOfAccess Value="1A"/>
<Stop EquipementChange="T">
<Arrival Date="03032000" Time="0805" Terminal="3">
<City Code="ORD" Value="Chicago, Illinois" />
</Arrival>
<Departure Date="03032000" Time="0900" Terminal="C"/>
<TimeOnGround Value="0055"/>
<AircraftInformation Code="77" Value="Boeing 777"/>
<KosherOK Value = "True"/></Stop>

</FlightSegment>
</Flight>
</Choice>
<Choice ID="2" >
...
</Choice>
</FlightScheduleResponse>
<FlightScheduleResponse ID="2">
...
</FlightScheduleResponse></BookSmart>
```

FIGURE 10A

```
<Server Name="Bk/CityCodeLookupSvr">

  <Attributes>
    <Simple Name="Timeout">0</Simple>
    <Simple Name="ThreadPolicy">thread-pool</Simple>
    <Simple Name="MaxThreads">20</Simple>
    <Simple Name="Description">This server contains components related to Car related Code
Lookups.</Simple>
    <Simple
Name="IncludeFile">/net/godzilla/usr2/BookSmart/v2.0/run/CORBA/TraderInclude.xml</Sim
ple>
  </Attributes>

        <InternalComponents>

   <Component Name="CityCodeLookup">
    <Creator Name="ConverterCreator"
      Library="libSnSFrameworkConverterCreators.so"/>
    <Relationships>
    </Relationships>
                    <Attributes>
                        <Simple Name="ServiceType">CityCodeLookup</Simple>
                        <Simple Name="SubstringMatch">True</Simple>
                        <Simple Name="TagName">City</Simple>
                        <Simple Name="CodeAttribute">Code</Simple>
                        <Simple Name="NameAttribute">Value</Simple>
                        <Simple Name="Library">libora12d0146.so</Simple>
                        <Simple Name="Database">WG80</Simple>
                        <Simple Name="UserName">BROADVISION</Simple>
                        <Simple Name="Password">BROADVISION</Simple>
                        <Simple Name="Table">BK_CITY_MAST</Simple>
                        <Simple Name="CodeColumn">CITY_CODE</Simple>
                        <Simple Name="NameColumn">CITY_NAME</Simple>
                        <Simple
Name="CodeIsPrimaryColumn">ISPRIMARY_FLAG</Simple>
                        <Sequence Name="Attributes" Type="Simple">
                            <Simple
Name="AttributeColumn">COUNTRY_CODE</Simple>
                            <Simple
Name="AttributeColumn">CURRENCY_CODE</Simple>
                            <Simple
Name="AttributeColumn">COTERMINAL</Simple>
                            <Simple
```

16/21

FIGURE 10B

```
Name="AttributeColumn">ISVALIDPOO_FLAG</Simple>
                                        <Simple
Name="AttributeColumn">ISVALIDCARRENTAL_FLAG</Simple>
                                        <Simple
Name="AttributeColumn">ISVALIDHOTELRENTAL_FLAG</Simple>
                                        <Simple Name="AttributeColumn">DAL</Simple>
                                        <Simple
Name="AttributeColumn">ISETCARRIERCHECK</Simple>
                                    </Sequence>
                                </Attributes>
                            </Component>
```

FIG. 11A

```
Server Name="Bk/TravelPlanControllerSvr">

 <Attributes>
   <Simple Name="Timeout">0</Simple>
   <Simple Name="ThreadPolicy">thread-pool</Simple>
   <Simple Name="MaxThreads">20</Simple>
   <Simple Name="ThreadBusyThreshold">180</Simple>
   <Simple Name="Description">This server contains components related to doing Travel Plan
operations.</Simple>
   <Simple
Name="IncludeFile">/net/godzilla/usr2/BookSmart/v2.0/run/CORBA/CommonInclude.xml</Simple>
 </Attributes>

       <InternalComponents>

  <Component Name="BkTravelPlanControllerFactory">
       <Creator Name="SnSFactoryCreator"
                                    Library="libSnSFrameworkFactoryCreators.so"/>
   <ChildCreator Name="SnSControllerCreator"
     Library="libSnSFrameworkControllerCreators.so"/>

                    <Attributes>
                        <Simple Name="Policy">Factory</Simple>
                        <Simple Name="ProductsAreGateways">False</Simple>
                        <Simple Name="MaintenanceThreads">1</Simple>
                        <Simple Name="CreatorName">SnSControllerCreator</Simple>
                        <Simple Name="AvailableBuffer">2</Simple>
                        <Simple Name="ServiceType">TravelPlanControllerFactory</Simple>
                    </Attributes>

   <Children Name="TravelPlanController">
                        <Attributes>
                            <Simple Name="LogAllErrors">True</Simple>
                            <Simple Name="PublishErrorStats">True</Simple>
                            <Simple Name="PublishPerformanceStats">True</Simple>
                            <Simple Name="AdminThreadPause">300</Simple>
                            <Simple Name="TimeBetweenStatPublishing">100</Simple>
                        </Attributes>
                        <Relationships>
```

FIG. 11B

```
                    <Uses Name="AmadeusIFFactory"
Component="AmadeusIFFactory"
                        Type="Component"
PersistConnection="False"/>

                    <Uses Name="HotelController" Component="HotelController"
                        Type="Component"
PersistConnection="False"/>
                    <Uses Name="FlightShoppingServer"
Component="FlightShoppingServer"
                        Type="Component"
PersistConnection="False"/>
                    <Uses Name="CarController" Component="CarController"
                        Type="Component"
PersistConnection="False"/>

            <Uses Name="ErrorLookup" Component="ErrorLookup"
                Type="Component" PersistConnection="True"/>
        <Uses Name="ContactInfoValidator" Component="BkContactInformationValidator"
                Type="Validator" PersistConnection="True"/>
        <Uses Name="PassengerTypeCodeLookup" Component="PassengerTypeCodeLookup"
                Type="Converter" PersistConnection="False"/>
        <Uses Name="ServiceRequestCodeStrictLookup"
Component="ServiceRequestCodeStrictLookup"
                Type="Converter" PersistConnection="False"/>
        <Uses Name="SpecialEquipmentCodeStrictLookup"
Component="SpecialEquipmentCodeStrictLookup"
                Type="Converter" PersistConnection="False"/>
        <Uses Name="ContactInfoCodeLookup" Component="ContactInfoCodeLookup"
                Type="Converter" PersistConnection="False"/>
                    <Uses Name="CityCodeLookup"
Component="CityCodeLookup"
                        Type="Converter" PersistConnection="False"
Priority="1"/>
                    <Uses Name="CurrencyCodeLookup"
Component="CurrencyCodeLookup"
                        Type="Converter" PersistConnection="False"
Priority="1"/>

                    <Uses Name="ConvertAir"
```

STRING

< parent tag >
< child tag >
< grandchild tag >
< /child tag >
< /parent tag >

Fig. 12A

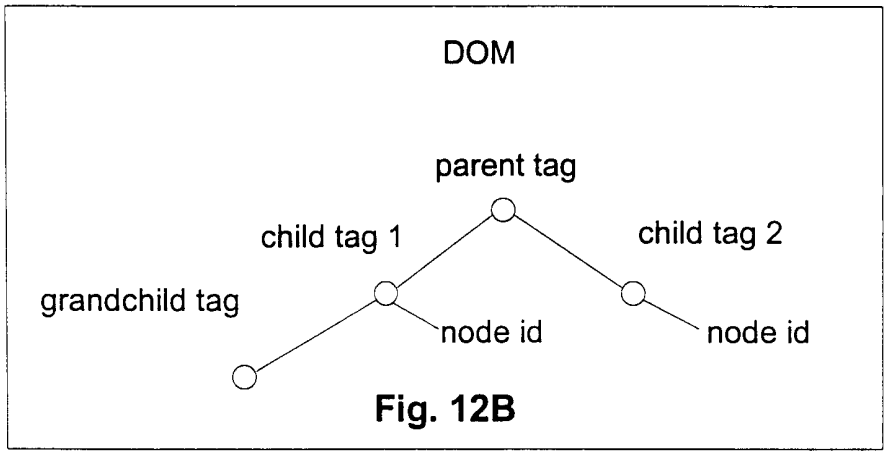


Fig. 12B

NODE LIST

Parent tag 1, node id

Child tag 1, node id

⋮

Fig. 12C

FIG. 13A
Document IDL

```
typedef WString DocumentNodeId;

enum EncodingFormat { ASCII, UTF8, UTF16 };

struct NodeAttribute
{
        WString sName;
        WString sValue;
};
typedef sequence<NodeAttribute> NodeAttributeList;

struct Node
{
        DocumentNodeId nodeId;
        WString sName;
        WString sValue;
        NodeAttributeList attributes;
        sequence<Node> children; // same type as NodeList
        WString sNodePath; // for informational purposes only
}
typedef sequence<Node> NodeList;
interface Document : Component
{
        DocumentNodeId get_current_context_node_id(inout ErrorStore error);

        boolean set_current_context_by_node_id(inout ErrorStore error, in DocumentNodeId
contextNodeId);

        boolean set_current_context_by_query(inout ErrorStore error, in WString
sContextQuery);

        NodeList get_nodes_by_query(inout ErrorStore error, in WString sSelectQuery);

        NodeList get_nodes_by_query_at_node_id(inout ErrorStore error, in WString
sSelectQuery,
                SnSFramework_DocumentNodeId startingNodeId);
```

FIG. 13B

```
boolean insert_nodes_at_node_id(inout ErrorStore error, inout NodeList nodes, in
DocumentNodeId parentNodeId,
        in boolean bAllOrNothing, in boolean bEnforceUniqueTags);

boolean insert_nodes_at_query(inout ErrorStore error, inout NodeList nodes, in
WString parentQuery
        in boolean bAllOrNothing, in boolean bEnforceUniqueTags);

boolean update_nodes_by_node_id(inout ErrorStore error, in NodeList nodes, in
boolean bAllOrNothing);

boolean delete_node_by_node_id(inout ErrorStore error, in DocumentNodeId
deleteNodeId);

boolean delete_nodes_by_query(inout ErrorStore error, in WString sDeleteQuery, in
boolean bAllOrNothing);

boolean load_from_file(inout ErrorStore error, in WString sFileName);

boolean save_to_file(inout ErrorStore error, in WString sFileName, in
EncodingFormat encoding);

WString get_nodes_by_node_id_as_xml(inout ErrorStore error, in DocumentNodeId
startingNodeId);

boolean add_nodes_at_node_id_from_xml(inout ErrorStore error, in WString sXml,
in DocumentNodeId parentNodeId);

WString transform_into_xml(inout ErrorStore error, in WString sStyleSheet,in
DocumentNodeId sourceNodeId);

boolean transform_into_document(inout ErrorStore error, in WString sStyleSheet,
        inout Document resultDocument, in DocumentNodeId sourceNodeId, in
DocumentNodeId resultNodeId);
};
```

INTERNATIONAL SEARCH REPORT

Intern ıal Application No
PCT/US 00/26789

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 G06F9/46

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)
IPC 7 G06F

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)
EPO-Internal, INSPEC, IBM-TDB

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category °	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
A	US 5 822 580 A (LEUNG WYATT) 13 October 1998 (1998-10-13) column 1, line 14 - line 45 column 5, line 50 -column 6, line 44	1-10, 17-20
A	EP 0 786 723 A (HITACHI LTD) 30 July 1997 (1997-07-30) column 1, line 1 -column 7, line 50	11-16
A	US 5 956 508 A (MCKELLEY JR CHARLES R ET AL) 21 September 1999 (1999-09-21) column 2, line 7 - line 22 column 8, line 26 - line 42	1-10, 17-20
A	US 5 515 508 A (PETTUS CHRISTOPHER E ET AL) 7 May 1996 (1996-05-07) column 4, line 45 -column 5, line 63	1-10, 17-20

☐ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

° Special categories of cited documents :

"A" document defining the general state of the art which is not considered to be of particular relevance

"E" earlier document but published on or after the international filing date

"L" document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

"O" document referring to an oral disclosure, use, exhibition or other means

"P" document published prior to the international filing date but later than the priority date claimed

"T" later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

"X" document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

"Y" document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

"&" document member of the same patent family

Date of the actual completion of the international search	Date of mailing of the international search report
12 December 2000	19/12/2000

Name and mailing address of the ISA	Authorized officer
European Patent Office, P.B. 5818 Patentlaan 2 NL – 2280 HV Rijswijk Tel. (+31–70) 340–2040, Tx. 31 651 epo nl, Fax: (+31–70) 340–3016	Brandt, J

1

INTERNATIONAL SEARCH REPORT

...ormation on patent family members

Intern ...al Application No
PCT/US 00/26789

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
US 5822580 A	13-10-1998	CA 2238973 A	24-07-1997
		DE 69701623 D	11-05-2000
		EP 0861467 A	02-09-1998
		WO 9726597 A	24-07-1997
EP 0786723 A	30-07-1997	JP 9204348 A	05-08-1997
		US 5887171 A	23-03-1999
US 5956508 A	21-09-1999	NONE	
US 5515508 A	07-05-1996	AU 7393294 A	03-07-1995
		WO 9517060 A	22-06-1995