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

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
1 September 2011 (01.09.2011)

PCT

(10) International Publication Number
WO 2011/104367 A2

(51) International Patent Classification:
G06F 9/44 (2006.01)

(21) International Application Number:
PCT/EP2011/052855

(22) International Filing Date:
25 February 2011 (25.02.2011)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
61/308,208 25 February 2010 (25.02.2010) US
1009606.3 8 June 2010 (08.06.2010) GB

(71) Applicant *(for all designated States except US)*: **SITA INFORMATION NETWORKING COMPUTING IRELAND LIMITED** [IE/IE]; Building One, Letterkenny Office Park, Windyhall, Letterkenny, Ireland Co. Donegal (IE).

(72) Inventors; and

(75) Inventors/Applicants *(for US only)*: **FINDLAY, Denise** [GB/GB]; 1 London Gate, Blyth Road, Hayes, Middlesex UB3 1BW (GB). **ATTAR, Michael Joseph** [US/US]; 55 Orville Drive Bohemia, Long Island, Islip, New York (US). **EULER, Eric William** [US/US]; 55 Orville Drive Bohemia, Long Island, Islip, New York (US). **SERRATORE, Cory Allan** [CA/CA]; 777 Walkers Line, Burlington, Ontario, Toronto L7N 2G1 (CA). **ELIAS, Lissy** [US/US]; 3100 Cumberland Boulevard, Atlanta, Georgia 30339 (US). **VALENTE, Leonardo Granado** [US/US]; 55 Orville Drive Bohemia, Long Island, Islip, New York (US). **LESTYAN, Gabor Janos** [CA/CA]; 777 Walkers Line, Burlington, Ontario, Toronto L7N 2G1 (CA). **FLENLEY, John Martin** [GB/GB]; 1 London Gate, Blyth Road, Hayes, Middlesex UB3 1BW (GB).

(74) Agent: **LLOYD, Patrick Alexander Desmond**; Reddie & Grose, 16 Theobalds Road, London, Greater London WC1X 8PL (GB).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AO, AT, AU, AZ, BA, BB, BG, BH, BR, BW, BY, BZ, CA, CH, CL, CN, CO, CR, CU, CZ, DE, DK, DM, DO, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, GT, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LY, MA, MD, ME, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PE, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, ST, SV, SY, TH, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LR, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG,

[Continued on next page]

(54) Title: SOFTWARE APPLICATION DEVELOPMENT TOOL

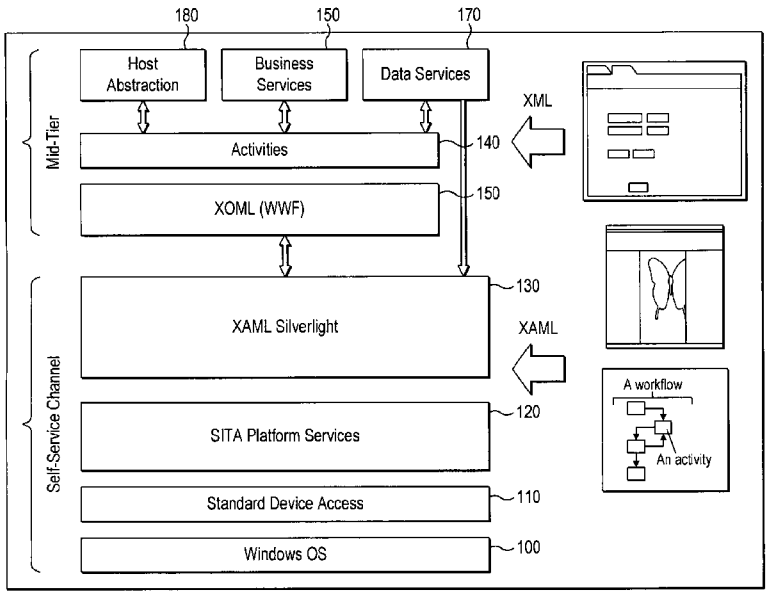

FIG. 1

(57) Abstract: A software development tool for use with external systems and services uses a common code base and defines all data and messages using XML Schema System components are defined which include a device abstraction layer which handles interactions between the application and devices. A host abstraction layer handles interactions between a host system and the application. A graphical tool models the work flow of the application and includes screens and services defined by Schema. The application is assembled using the graphical tool, declarative XML rules and customisations of system components without the user having to generate any coding.

WO 2011/104367 A2

ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AL, AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI, FR, GB, GR, HR, HU, IE, IS, IT, LT, LU, LV, MC, MK, MT, NL, NO, PL, PT, RO, RS, SE, SI, SK, SM, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report (Rule 48.2(g))*

SOFTWARE APPLICATION DEVELOPMENT TOOL

FIELD OF THE INVENTION

This invention relates to the development of software applications systems for use with systems that communicate with a variety of different external systems.

BACKGROUND TO THE INVENTION

In the airline industry there are many common functions that must be performed by all airlines such as baggage handling, and check-in. Some of these functions, such a check-in are gradually being moved to a self-service channel where the action is performed by the passenger or customer rather than the airline. In the case of check-in, the function may be performed on-line, at designated self-service kiosks at an airport, or through the traditional airline check-in desk where the passenger will be checked-in by an airline representative. It is anticipated that check-in via a mobile phone or other pda will become available in the near future.

Thus there is a plurality of channels though which a check-in service may be delivered. The basic function is the same for all airports and airlines although every airline and every airport has its own business requirements and rules and may require enhanced functionality such as self-bag tagging, and revenue generating opportunities such as ancillary revenue and targeted marketing.

Over the years individual code bases have been developed by each airline and each airport to handle check-in such that there is now an almost one to one relationship between applications and code bases. This is highly inefficient and requires support and maintenance and long certification cycles. It is also difficult to enhance and develop existing systems as the airlines must revert back to the original developers to write changes to the original code bases. It presents particular problems for airlines as they are required to use different code streams for different modes of check-in. It also creates difficulties for airports who have to work with multiple airlines, each of which may have its own set of code bases. It further creates difficulties for suppliers who have to provide support and maintenance for a large variety of incompatible code bases.

This problem is not confined to check-in but is also present in other areas where different systems have evolved to handle the same tasks, for example, ticketing and baggage handling.

Although this is a particular problem in the airline industry, it also exists in other fields where a number of parallel implementations of solutions have evolved over years. For example, in different modes of travel and in the entertainment and hospitality industry, particularly in event ticketing and hotel check-in as well as areas such as the financial industry where devices such as ATM machines are often running on a variety of different code bases. The financial self-service industry faces similar issues as the airline industry. There has been a proliferation of channels, and a range of customers from large multi-national to small regional banks, each of whom seek to differentiate themselves via the self service channel while looking for revenue generating opportunities on that channel. The financial industry also faces issues of legacy back-end systems and industry specific devices and device interfaces.

SUMMARY OF THE INVENTION

The invention aims to address these problems to enable systems to be integrated, developed and upgraded in a much more simple and cost effective manner without the need for the original developer to write new code.

According to the invention there is provided a method of developing a software application in which a plurality of devices communicate with external systems and services, the method comprising: providing a common code base; describing data and messages used by the application using a declarative data description language; defining a library of system components including a device abstraction layer for interactions between devices and the application and a host abstraction layer for interactions between a host and the application; providing a graphical (GUI) tool to model the workflow of the application, the workflow including screens and services described declaratively by a declarative data description language; and assembling the application using the graphical tool, declarative rules and customisations of system components selected from the library.

The invention also provides a software application development tool for assembling applications in which a plurality of devices communicate with external systems and services, comprising: data and messages used by the application described using a declarative data description language; A library of system components including a device abstraction layer for interactions between devices and the application and a host abstraction layer for interactions between a host and the application; a graphical (GUI) tool to model the workflow of the application, the workflow including screens and services described declaratively by a declarative data description language; and an assembler for

assembling the application using the graphical tool, declarative rules and customisations of the system components selected from the library.

Embodiments of the invention have the advantage that a developer may use the graphical tool to drag and drop devices and functionality which will automatically generate the code for the device or function from a library. This assists the developer in assembling rather than coding the application so reducing the development time and reducing the skill level required by the developer.

The use of a declarative data description language such as XML schemas for the data and messages enables backwards compatibility to be maintained and the data model and / or messages to be extended simply by updating the schemas.

The data and messages may be retrieved from a library or be generated by the developer or a combination of both.

Preferably, high level interfaces are provided to external devices which can be accessed by elements drop and dropped on to pages and having properties set on them.

These pages are preferably specified declaratively and more preferably using an XML language such as XAML.

The host abstraction layer enables the system to be used with a wide range of external systems. In one preferred embodiment, the system is a check-in terminal, for example for an airline. The host abstraction layer enables the system to communicate with a range of different systems used by airlines to hold flight data. Examples include Amadeus and EDS. The host abstraction layer translates messages and protocols from the external system such that they become system non-specific.

The system components may be complex, generic or specific. Generic components are reusable for a domain and may be parametised to increase re-usability.

Although particularly suited to travel systems, particularly check-in systems, embodiments of the invention may be used for many other purposes, for example in the provision of ticketing and admission to venues, such as sporting or entertainment venues. It may also be used in the financial services industry, for example in the control of ATM (Automated Teller Machines) to address problems caused by the large number of code bases used by different banks for their own ATMs.

BRIEF DESCRIPTION OF THE DRAWINGS

Embodiments of the invention will be now be described, by way of example only, and with reference to the accompanying drawings, in which:

Figure 1 is a schematic block diagram of a software system embodying the invention;

Figure 2 shows an example graphical user interface for a workflow designer;

Figure 3 shows the various activities associated with a page and the links to other activities and pages; and

Figure 4 shows how data values may be accessed by a business service page.

In the following description of a preferred embodiment, a system is described which assembles applications from re-usable components without any use of programming language, or coding, by the assembling party. Applications are assembled using a graphical tool which will be described with declarative rules and customizations. The applications built may be adapted or customized declaratively without the need for any coding. This is possible as the applications are built from the same code-base and core product. All data and messages are described declaratively, preferably using XML or similar technologies which allow expansion of the data and messages over time without the need for additional written code.

A preferred embodiment of the invention uses a graphical tool and a schema, preferably an XML or similar schema. The graphical tool models the flow of the application. Screens and services are described using schema which are used by the graphical tool to guide the developer.

XML schemas are used for all data and messages to enable the data model and/or messages to be extended by the user by a simple update of the schema. This may be achieved in a backwards compatible way. As an alternative to XML schemas any declarative data description language may be used. The following description will refer only to XML for simplicity.

The following description is of a self-service check-in kiosk application but this is exemplary and other applications of the invention will be discussed.

Referring to figure 1 a schematic representation of a kiosk client is shown. The functionality is shown as a number of layers some of which are in the self-service or kiosk channel and some of which are resident at a mid-tier level. It is to be understood that this is exemplary only and that some of the software that is resident at the mid-tier level, for example the XOML layer to be described, could reside on the self-service channel, and vice-versa. A kiosk client is one that handles self-service check-in at an airport. It is to be understood that this description is illustrative only and that within the check-in environment other clients such as on-line web based clients or mobile communications clients would have different platform services, different or no device access layer and a different XAML Silverlight layer.

Airline check-in kiosks use a common standard called CUSS (Common Use Self-Service) which is a standard administered by IATA and seeks to allow multiple airlines to share kiosks to facilitate check-in while maintaining individual airline branding and functionality. Another standard known as CUPPS (Common Use Passenger Processing System) has also been proposed and might be adopted and it is therefore important that the clients can run on either of these platforms or any other platform proposed. Moreover the client should be able to run on both platforms for the duration of any migration period. The client therefore includes a device abstraction layer to protect against platform changes and to allow for migration.

The proposed CUPPS standard also permits agent devices and so components developed for a kiosk, with this standard, could be used for an agent device. An agent device is one used by the airline representative or agent to check in a passenger. In the airline industry agents perform a super-set of the self service tasks. However the host systems and the devices are the same. This solution is a cross-channel solution that also applies to agent (attended) functions. In the financial industry the same conditions apply and therefore the solution could apply also to Teller applications. An example of an application in the financial industry is described later.

In Figure 1 the kiosk includes a standard Windows (RTM) or other operating system 100 and a standard device access layer 110. The abstraction of the platform layer is handled at layer 120 labelled 'SITA Platform services'. The abstraction includes abstraction of the monitor and the application manager of the relevant standard. The platform services layer provides device access in a simple and intuitive manner via Silverlight (RTM) controls which can be dragged and propped onto a Silverlight page. The Silverlight layer is

represented in figure 1 by layer 130. Silverlight is a browser plug-in provided by Microsoft Corporation that facilitates design and development of web based applications and is cross-platform and cross-device. Silverlight uses XAML (Extensible Application Mark-up Language). This provides a very simple and intuitive tool for the application developer.

Silverlight pages are activated as a result of a message from the workflow. The Silverlight pages comprise two parts: code behind that will communicate with the workflow and may include a worker thread which can communicate with the device with the data resources, web services and which may perform some processing; and a screen. The screen is preferably written in XAML but other declaratory languages such as HTML may be supported. A suitable tool for the screen is Expression Blend provided by Microsoft Corporation and which is intended for use with the Silverlight platform.

The presentation logic is divided between run-time and design-time. At design time a tool is used to create a graphical representation of the application flow using the Activities layer 140 and the workflow. The workflow is comprised of a number of linked activities as will be described. This encapsulates the application logic and uses declarative rules to decide the flow of the applications. Specialised property editors, validity checking and tracking services may also be provided in this tool.

The tool produces XOML files (shown in layer 150) which are fed to the workflow engine or compiled into dlls (dynamic link libraries). The workflow engine and the workflows are hosted by a workflow host which resides on the mid-tier. This is not essential and the workflow host could reside on the client. The Activities in the Workflow communicate with business services 160 which are resident on the mid-tier via SOAP (Simple Object Access Protocol) which is an XML based protocol designed to allow web based applications to exchange information.

Data services 170 also form part of the mid-tier and are resident on it. In the case of a self-service kiosk, there is only exchange of data to the client from the user and the client can only read data; it cannot update it. The client can communicate with the data services via REST (Representational State Transfer) messages although other communications protocols such as Xpath or related technologies may be used. This simplifies the programming as the application merely asks for the piece of data it needs. The messages are lightweight and the mid-tier is very loosely coupled from the client. The data service, as well as providing traditional airline data may connect to external web-services to provides other data, such as travel related data including weather reports.

The host abstraction layer is indicated at 180 and gives the application access to airline systems departure control systems. This layer is important as it makes the application independent of any particular system. The host abstraction layer hides the complexity of host protocols from the rest of the system and abstracts the host system so that the same components can be deployed to different hosts without change. Changes to a host, or the addition of a new host, do not necessarily imply that there should be a new release of the host abstraction layer, other than a new connector if it is a new transport protocol. Instead, the layer acts as an interpreter and uses scripts and/or schemas to manage the different hosts and protocols.

In use, the software system will be adapted by customers to suit their particular needs. Business analysts will determine with the customers the particular functionality they require, the user interface flows and any specific customisation. User interface designers will design the screens and the branding which are displayed to the user. By providing a componentised architecture, these tasks can be carried out in parallel to reduce the development time and improve the time to market greatly.

By capturing complex, generic, behaviour inside component interfaces, customisation can be greatly facilitated. The application developer can then use these interfaces to assemble applications and customisations without requiring knowledge of low level details.

The principles of the software architecture will now be described. The architecture divides the system into three component types: Complex, Generic and Custom. This division aids in providing a system that can bring a solution rapidly to market. The Complex components absorb the complexity of the system within high level interfaces. These are usually developed using low-level system programming skills and in-depth technical domain knowledge. Examples of Complex components include the host abstraction layer and the device abstraction layer.

Generic components encapsulate common functions for a particular domain with a focus on reusability. These are components that any application is likely to want to use and which can simply be selected by the developer without the need for any coding. In our self-service kiosk example, a printer function is a good example of a generic component. It will always be required to print boarding passes and by providing a generic printer function, the developer can simply select that function to achieve the functionality. Generic components must be cohesive so that they perform a single well defined task completely. Thus in the printer example, the printer function will print the document while informing the user of the

progress of the print. It will then complete the task with a result when the print has finished. These components may be parameterised or further componentised to enable greater re-usability. By developing generic components within each domain within a framework, the high level design will be the same. Once the framework and wizards are developed, new re-usable components can be created rapidly. Developers of generic components require mid-level programming and technical skills and some business focussed domain knowledge. The generic components include screens and application services.

Custom components will be unique to each user and may be developed rapidly in response to business requirements. Custom components may be produced without creating any new coding by the developer who is provided with the appropriate tools, frameworks and reusable components. The main tasks for the business application developer are a business process model including rules and the user interface design.

A common data model is used in conjunction with the host abstraction layer. All parameters to the services are strings which are serialised and de-serialised by the application using a shared schema. The data is specified using schema and the process of specifying the data is iterative. In the example of the self-service kiosk, schemas are provided for:

Access to departure control systems;

Workflow – the schemas that the workflow uses internally;

Workflow session – the schemas for the data set that is used during a session. This is data that is gathered from the user, form the departure control system and from other web services;

Workflow configuration – the workflow configuration data which sets the Workflow properties and is passed to the workflow when the workflow is started;

SL (Silverlight) Application Data – this data is passed to the SL application when the transaction workflow is started.

One example of these schemas is:

Workflow data example: Screen Specification

Workflow session data example: Current flight

Configuration data example: Services Offered

SL application data example: Theme

Thus, all data in the system is specified by schema and where the data is accessed by objects, those objects should be generated using an automated tool (xsd.exe). The data model is not created for the system but rather it is built up from the inputs to the system either from the end-user or from web services. So the data model of the application is the sum of all the inputs to the application. Therefore the data model is automatically extended when there is a new input to the system.

The data schema specifies the data format of the session data. The values of this schema are set during runtime execution of the workflow. The values are accessed via xpath which is the path through the schema to the value that is to be set or read. The data schema encompasses all data returned from the services or supplied by the customer during execution of the Transaction workflow.

The workflow path schema specifies the format of the path through the workflow to the workflow properties. The workflow path is used to point to the properties in the workflow. The workflow contains Scenarios, each Scenario being an Activity which contains Activities. Both Scenarios and Activities can have Properties that can be navigated to using the path through the Workflow. Activities have unique identifiers within a Scenario.

Screens may preferably be developed in XAML or HTML, although any other presentation language may be used. The code behind the screen is generic that responds to user interface (UI) events from the screens and sends to the workflow. The UI events may be from a device. All screens are viewable and editable in Expression Blend.

Each screen has an XML specification. This specification is used by the designer for presentation and validation.

The schema elements are:

Screen Name (1)

Required Properties – "key=value" (0-*)

Optional Properties – "key=value" (0-*)

Range of Return Values (1-*)

Return Data – "key=value" (0-*)

Pre Conditions (0-*)

Post Conditions ()-*)

Description of behaviour.

The device is a Silverlight User Control which can be placed on the Silverlight toolbar and can be dragged and dropped onto the page. The device controls have properties that can be set at design time or at run time.

The embodiment described provides an approach to creating applications which can be assembles rather than coded. The workflow of the application is modelled using a graphical tool and screens and services are described using schema. The tool uses these schemas to guide the developer.

XML schema are used for all data and messages enabling the data model and/or the messages to be extended by the user by updating the schema in a backwards compatible manner.

High level interfaces are provided to devices which are accessed via user interface elements. The devices can be drag and dropped onto pages and properties set on them. Pages are GUI screen representations and are developed using a GUI screen tool. They may be specified declaratively in an XML language such as XAML.

In the example described of a self-service check-in kiosk, access to external systems, in this case airline systems, is via a service interface accessed via the GUI tool. This tool and a service interface can also be used to access other external systems such as targeted marketing, payments and advertising.

The business rules are also declarative and incorporated into the workflow via the GUI tool.

The entire system may be advantageously executed in a development environment on a PC using host and device simulators. At runtime the behaviour of the application can be

adapted by supplying new configuration parameters. The workflow can accept new parameters on each occasion with needing to restart the system.

The example described is a self-service check-in kiosk for an airline. Of course the principles of the invention are not limited to self-service check-in but extend to other check-in scenarios such as mobile check-in, agent based check-in and web-based or on-line check in. Moreover although the check-in described is intended for airline check-in, the invention is applicable to check-in systems intended for any type of travel and even non-travel events. An example of the latter is sports or entertainment events where the user, having already acquired a ticket, is required to authenticate themselves to gain access to the event. This may be by supplying a booking reference and a credit card identification following which an entrance ticket will be printed. This is in effect, very similar to a boarding card. Embodiments of the invention may also be used for hotel check-in systems.

The invention is also applicable more broadly, for example in the financial industry in situations where a large number of legacy systems have evolved over years. One example is Automated Teller Machines (ATMs) which presently are provided by many different banks, each of which has their own code base.

The invention is suitable for any self service transactional application in many industries. The financial services also face a proliferation of channels, customers that range from large multi-national to small regional banks, who seek to differentiate themselves via the self service channel and also are looking for revenue generating opportunities on that channel.resulting in many application code bases and the problems that result therefrom.

In the financial self service environment the customer uses a self service terminal, similar to an airline kiosk. Some non-cash self service terminals are kiosks. Others are kiosks with the addition of a safe which holds cash or other valuable media. The customer is authenticated via the card reader and encrypting pin pad devices. The user is authenticated by a financial host system. When authenticated, the customer will be able to perform transactions on that terminal, including depositing and receiving cash and documents. These transactions have varied business rules depending upon, for example, the bank, the location of the terminal, the customer, or the time of day. This variety results in the need for many different code bases. An embodiment of the present invention applied to such self service terminals and as described above uses a single code base which could be assembled and configured into many applications without writing code. A workflow with declarative rules controls the flow and XAML pages make up the presentation layer. In the

financial industry there are varied host protocols and specialized device interfaces. Therefore the host abstraction layer and the device abstraction layer are necessary to absorb this complexity in the same way as they are required for the airline example described above.

In order to understand the manner in which an embodiment of the invention operates, the workflow designer will now be described in greater detail.

Figure 2 shows the user interface of the workflow designer used to develop the application for a specific installation. The designer includes a GUI object representation of the workflows and activities but not the actual activity objects. The designer creates the workflow programmatically and converts it to an XOML file which is a declarative XML workflow file. Thus in figure 2 the object Identify 200 is linked to the object Select Flight 210. That object is in turn linked to the objects Select seat 220, Select bags 230 and Check in 240. At the right hand side of the screen are displayed the Screens, Business Services, Activities and Scenarios that are available. When the user clicks on one of them, the specification, as described by their schema, and help is displayed to the user. The user can than drag and drop the Screens or Business Services onto a page and the activities and the Scenarios into the workflow. The user does not need to create any code in these steps.

The Workflow contains Scenarios, Pages and Activities. As can be seen from figure 3, page is a composite activity. Figure 3 is a UML object model showing the relationship between activities. Pages are joined by connectors 310 which consist of a link and a destination page. Thus in figure three the page bar 300 has associated activities A to G. The page contains a navigation bar 320 (activity G) which contains the links. The links (Activity E) have a condition property, which is evaluated and if true flow continues to the destination page of the connector. The links are evaluated from left to right across the Navigation bar. Pages that communicate with the client are implemented using a Call External Method Activity 330 (Activity C). An Event Handler activity 340 (activity D) receives the event from the client application. When the event is received the link conditions will be evaluated. These pages are client pages.

Pages that communicate with the services are implemented using an Invoke Web Service Activity. When the Web Service returns the link condition will be evaluated. These pages are Service Pages. A specialised Iterator Activity iterates over a list and a Set Value Activity sets a value of the Session data.

On the client page the designer has a menu option 'Add Screen' which navigates the user to an XAML file for the screen. The designer leads the user through the necessary steps to add the additional information required such as Pre Conditions, Post Conditions and Return Data Path. The designer then generates the appropriate XML document using the Screen Schema. The Client page sends a message to the client application as specified by the Message Schema. The message contains the name of the screen to show and any properties to set on that screen. The client page will receive an event from the client containing the result and any returned data.

The services page contains a Business Service. The designer has a menu option 'add service' which navigates the user to a WSDL file for the Business Service. The user then provides the necessary additional information including Pre Conditions, Post Conditions and Return Data Path and the designer can then generate an XML document using the Service Schema A pre-condition refers to the data that must be set, that is obtained and validated before the activity can be performed. A post-condition is the data that will be set when the activity completes successfully.

Pre and post conditions allow a flow to be validated, that is it can be determined at design time that an activity will only be activated if its' preconditions are met.
Scenarios are composite Activities and can be invoked from the top level workflow at runtime. Scenarios may contain other scenarios and deliver large grained pieces of functionality such as 'Seat Map', 'Payment' and 'Join Frequent Flyer'. Scenarios can have properties which can be accessed by contained Activities and have Navigation Bars with Links.

The purpose of Scenarios is to simplify the display of large applications into manageable chunks and to provide reusable pieces of functionality which can be imported into other projects.

Scenarios are analogous to function calls in coding and so a change to a scenario in one part of the application will be propagated to the same scenario elsewhere. As it is desirable to improve reusability scenarios can be parameterised with properties which can be referenced by the internal Activities and Scenarios.

The elements of session data can be accessed by using xpath expressions. Xpath is used to navigate through elements and attributes in an XML document. A specialised Activity Set Value can be used to set values within the Session data.

A Page Property is set within xpath to send Session Data values to the client or business service. An example is to send a selected seat to a Change Seat business service. The Change Seat interface is result Change Seat (string passenger, string flight Number, string seat Number). The properties could then be set as follows:

"passenger" = "Session Data/Current Selection/Current Passenger"

"flight Number" = "Session Data/Current Selection/Selected Seat"

"seat Number" = "Session Data/Current Selection/Selected Seat"

The Session data can be updated if the workflow is receiving data from the client or business. For example the presentation page named Seat Selection links to the Select Seat screen on the client and the Select Seat screen returns "Seat=4G". The designer has provided a "Seat" property from reading the xml for the Select Seat screen and updates that property with "4G". For all return values there is an option of using a specialised editor which enables the developer to navigate the session schema and connect the Seat Selection schema element to the Seat property of the Seat Selection Activity. Thus, the session data is updated when the Sear property is set to "4G".

Referring now to figure 4, the accessing of data values will be explained. The Service Page 400 has a Property. This Property 410 is automatically added to the Page by the Designer when a Business Service is dragged and dropped onto the page. The Property matches a parameter in the service call. To help rapid development the name of the Property matches an element of the Session Data and the Designer automatically sets the value of the Property to the path in the data schema. This default behaviour can be overridden by the Designer by entering a string value for the Property and using a Property Editor GUI 420 to navigate to the appropriate place in the Session Data 450. The Property Editor 430 uses a schema navigator 440 to navigate the Session Data 450. It is also possible to select from a list using a Selection Criteria GUI 460. This GUI enable the Developer to navigate to other properties in the workflow.

When a Screen or Business Service is dropped onto a page, the designer will automatically insert Return Data properties according to the schema. For services the return data is a string but the schema that the designer uses extends the WSDL to map the return data to a path in the Session Data.

To aid the developer the Return Data property value is automatically set to the path specified. The return data may be a complex type but the data schema type is made compatible with the services schema type to avoid the need for internal mapping. If an internal mapping is required the mapping will be contained in the services schema for that Business Service. The type compatibility will be validated as part of design time validation.

The Designer checks for errors from the Web Service and set the Status accordingly. In the case of error the session data does not get updated. Values within the Session Data can also be updated by the developer using the Set value Activity. This Activity allows the developer to override any value within the Session Data with an alternative value.

The validator ensures that all required properties are set and that all return values cause a transition. It validates before a Service Page or Client Page is activated and its specified preconditions have been met.

The values of the workflow dependency properties can be edited by the configuration tool. The tool creates a dictionary object (path=value) which will be passed to the workflow runtime when the workflow is executed. This will dynamically set Workflow properties. Workflows are developed with default properties which can be updated at runtime by the dictionary configuration object.

To create these Configuration files the developer will edit the properties of the workflow. This is done by choosing a menu item 'New....Configuration'; and associating the Configuration with a workflow. The configuration tool opens a designer window containing the workflow and the designer constrains updates to only dependency property updates. The developer then edits the workflow properties to create a new configuration. The developer can use the schema navigator tool to set values to Session Data properties and can edit the condition property (rules). The configuration tool creates a path=value pairs for the dictionary configuration object Workflow.Scenario.Property.Attribute=*string*.

The tool will automatically import WSDLs and enable the application to connect to any web-service without coding. The screens on the client are specified using XML schema and are analogous to the WSDLs, the tool will automatically import the screen specification and enable the screen to be shown at any point in the workflow. The tool provides for declarative business rules and the tool will automatically extend the application data mode.

In these ways the tool can integrate web-services, screens and extend the data model, modify or add business rules without writing code.

Various modifications to the embodiments described are possible and will occur to those skilled in the art without departing from the invention which is defined by the following claims.

Claims

1. A method of developing a software application in which a plurality of devices communicate with external systems and services, the method comprising:

providing a common code base;

describing data and messages used by the application using a declarative data description language;

defining a library of system components including a device abstraction layer for interactions between devices and the application and a host abstraction layer for interactions between a host and the application;

providing a graphical (GUI) tool to model the workflow of the application, the workflow including screens and services described declaratively by a declarative data description language; and

assembling the application using the graphical tool, declarative rules and customisations of system components selected from the library.

2. A method according to claim 1, comprising defining user interface elements for accessing devices, whereby devices can be dragged and dropped onto pages of composite activities using the graphical tool.

3. A method according to claim 2, wherein the pages are screen representations and are specified declaratively.

4. A method according to claim 1, wherein the graphical tool provides access to external systems.

5. A method according to claim 1, wherein business rules are incorporated into the workflow using the graphical tool.

6. A method according to claim 1, wherein the library of system components comprise complex, generic and custom components.

7. A method according to claim 1, wherein at least one of the data and messages, and the screens and services used by the application are described using XML schemas.

8. A method according to claim 1, wherein data and messages described using a declarative data description language are at least partially provided from a library of data and messages.

9. A method according to claim 6, wherein generic components comprise re-usable components for a domain and comprising parameterising at least one generic component to increase re-usability.

10. A method according to claim 6, wherein custom components are created without coding by the developer.

11. A method according to claim 1, comprising defining a common data model, the data model having data specified using schemas.

12. A method according to claim 11, wherein the data schemas specify the data format of session data and the values of the schemas are set during runtime execution of the workflow.

13. A method according to claim 1, wherein the application is a check-in client.

14. A method according to claim 13, wherein the check-in client is a self-service check-in client.

15. A method according to claim 14, wherein the self-service check-in client is a kiosk.

16. A method according to claim 14, wherein the self-service check-in client is a web based application.

17. A method according to claim 14, wherein the self-service check-in client is a mobile communications device.

18. A method according to claim 13, wherein the host abstraction layer provides access for the application to airline systems.

19. A method according to claim 1, wherein the application is a ticketing client.

20. A software application development tool for assembling applications in which a plurality of devices communicate with external systems and services, comprising:

data and messages used by the application described using a declarative data description language; A library of system components including a device abstraction layer for interactions between devices and the application and a host abstraction layer for interactions between a host and the application;

a graphical (GUI) tool to model the workflow of the application, the workflow including screens and services described declaratively by a declarative data description language; and

an assembler for assembling the application using the graphical tool, declarative rules and customisations of the system components selected from the library.

21. A software application development tool according to claim 20, comprising user interface elements for accessing devices, whereby devices can be dragged and dropped onto pages of composite activities within the graphical tool.

22. A software application development tool according to claim 21, wherein the pages are screen representations and are specified declaratively.

23. A software application development tool according to claim 22, wherein the pages are specified in an XML language.

24. A software application development tool according to claim 20, wherein the graphical tool provides access to external systems.

25. A software application development tool according to claim 20, comprising business rules specified declaratively and available for incorporation into the workflow using the graphical tool.

26. A software application development tool according to claim 20, wherein the library of system components comprise complex, generic and custom components.

27. A method according to claim 20, wherein at least one of the data and messages, and the screens and services used by the application are described using XML schemas.

28. A method according to claim 20, wherein data and messages described using a declarative data description language are at least partially provided from a library of data and messages.

29. A software application development tool according to claim 26, generic components comprise re-usable components for a domain and wherein at least one generic component is parametered to increase re-usability.

30. A software application development tool according to claim 26, wherein custom components are created without coding by the developer.

31. A software application development tool according to claim 20, comprising a common data model, the data model having data specified using schemas.

32. A software application development tool according to claim 31, wherein the data schemas specify the data format of session data, and the values of the schemas are set during runtime execution of the workflow.

33. A software application development tool according to claim 32, wherein the graphical tool provides a gui object representation of workflow and activities which are converted to a declarative XML workflow by a workflow designer.

34. A software application development tool according to claim 20, wherein the application is a check-in client.

35. A software application development tool according to claim 34, wherein the check-in client is a self-service check-in client.

36. A software application development tool according to claim 35, wherein the self-service check-in client is a kiosk.

37. A software application development tool according to claim 36, wherein the self-service check-in client is a web based application.

38. A software application development tool according to claim 35, wherein the self-service check-in client is a mobile communications device.

39. A software application development tool according to claim 34, wherein the host abstraction layer provides access for the application to airline systems.

40. A software application development tool according to claim 20, wherein the application is a ticketing client.

41. A software application developed according to the method of any of claims 1 to 19.

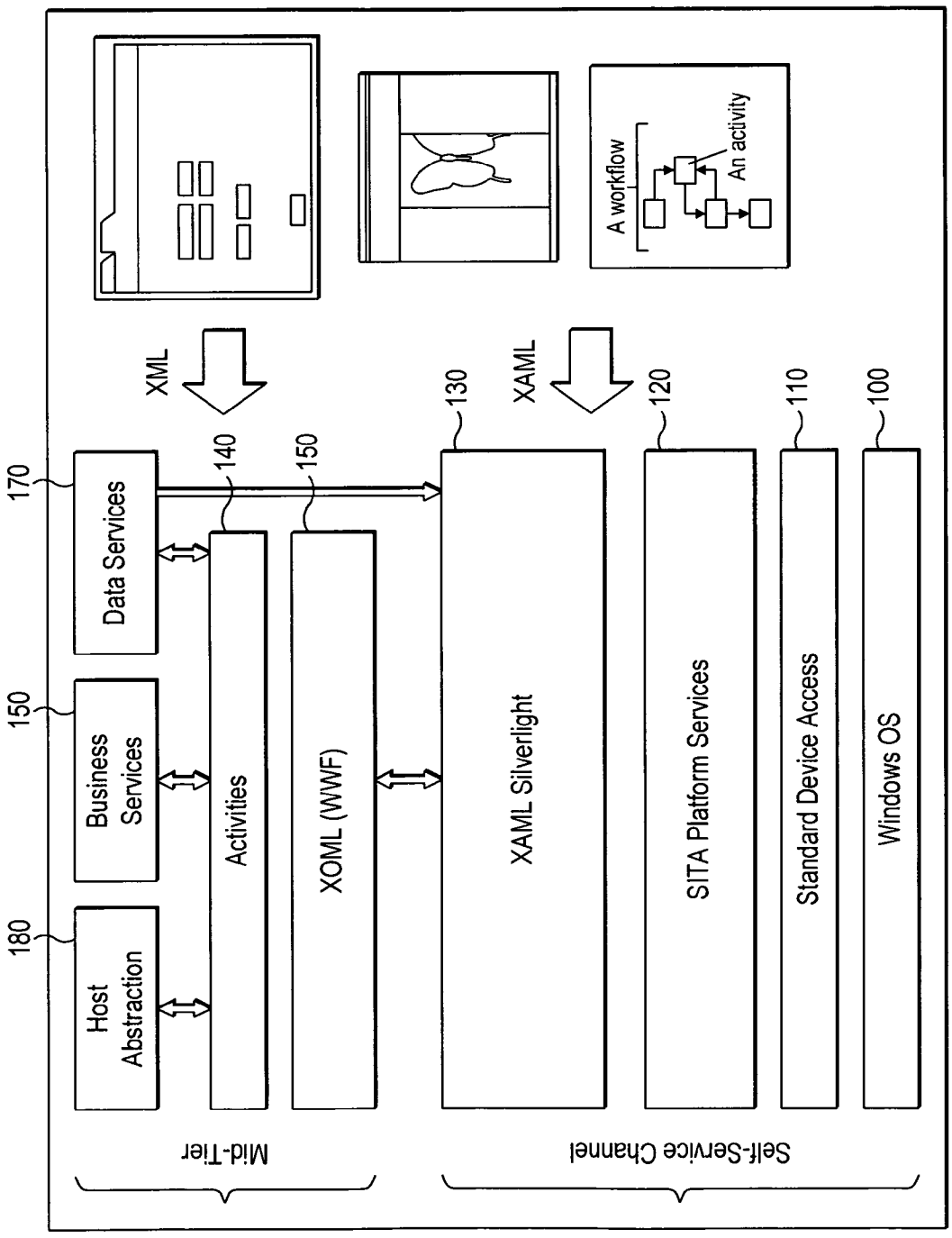
A workflow
An activity
XML
XAML
130
120
110
100
140
150
170
150
180
Data Services
Business Services
Host Abstraction
Activities
XOML (WWF)
XAML Silverlight
SITA Platform Services
Standard Device Access
Windows OS
Mid-Tier
Self-Service Channel

FIG. 1

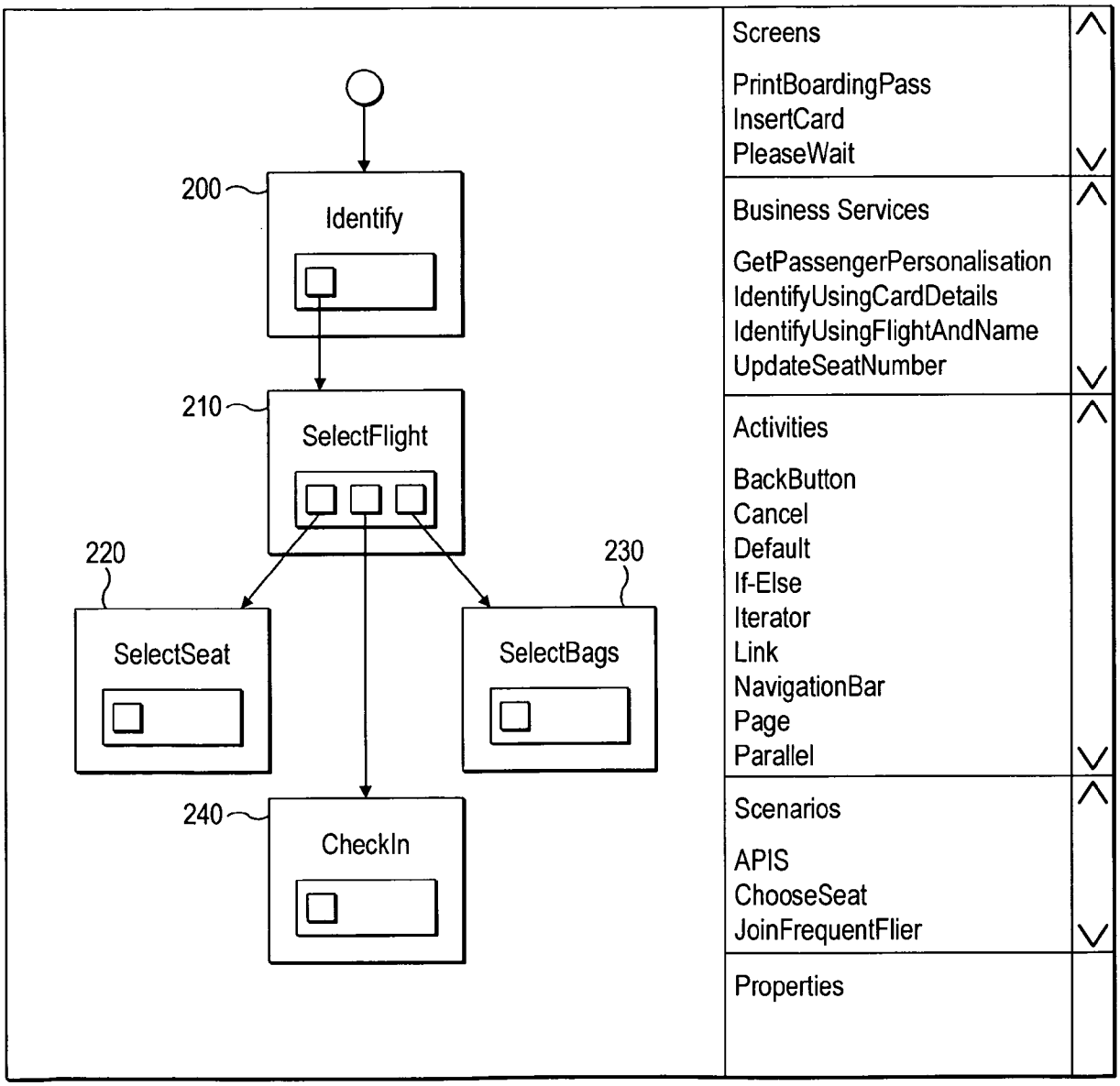
200
Identify
210
SelectFlight
220
SelectSeat
230
SelectBags
240
CheckIn
Screens
PrintBoardingPass
InsertCard
PleaseWait
Business Services
GetPassengerPersonalisation
IdentifyUsingCardDetails
IdentifyUsingFlightAndName
UpdateSeatNumber
Activities
BackButton
Cancel
Default
If-Else
Iterator
Link
NavigationBar
Page
Parallel
Scenarios
APIS
ChooseSeat
JoinFrequentFlier
Properties

FIG. 2

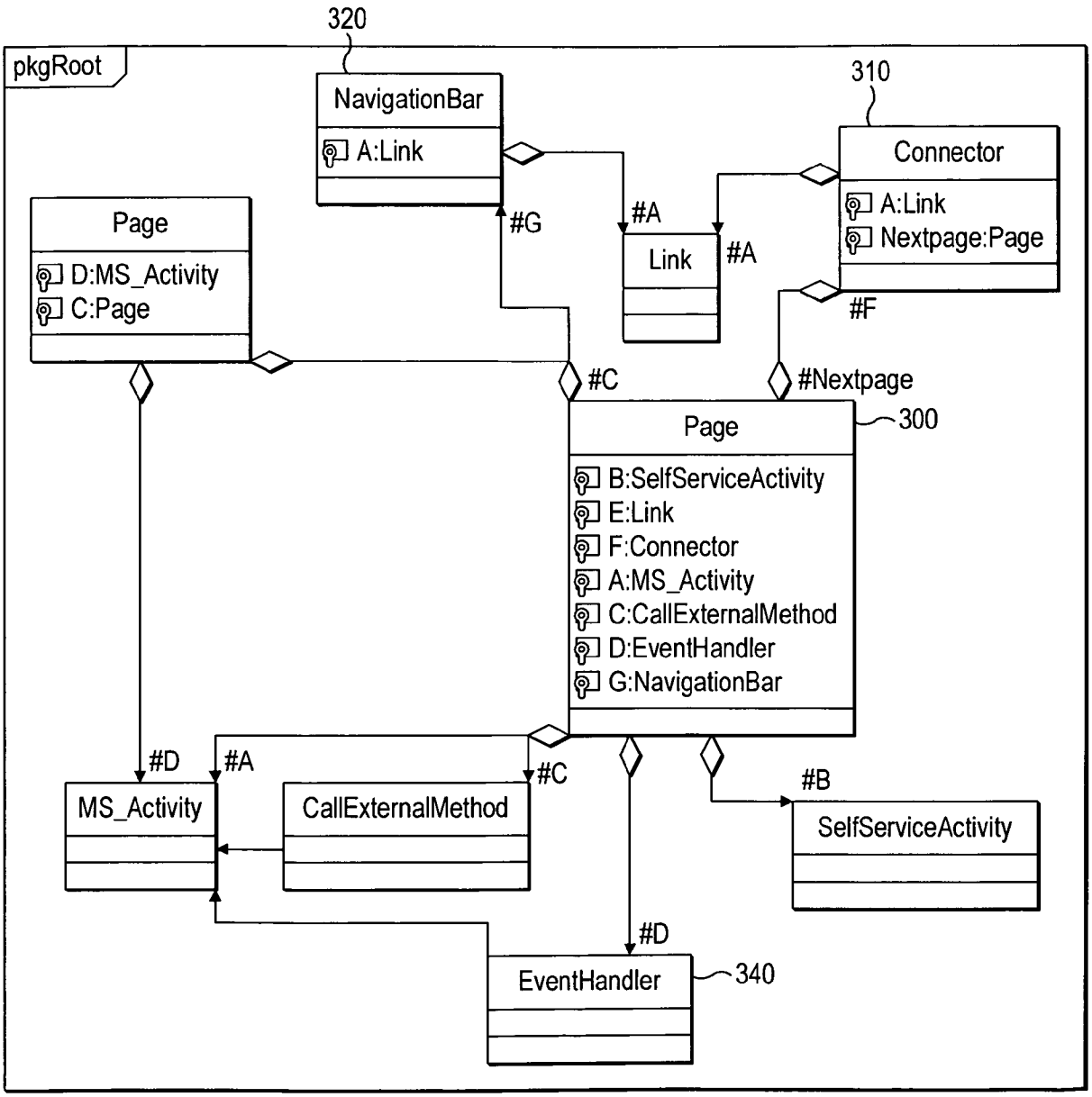
320
pkgRoot
NavigationBar
A:Link
310
Connector
A:Link
Nextpage:Page
Page
D:MS_Activity
C:Page
#G
#A
Link
#A
#F
#C
#Nextpage
Page
300
B:SelfServiceActivity
E:Link
F:Connector
A:MS_Activity
C:CallExternalMethod
D:EventHandler
G:NavigationBar
#D
#A
#C
#B
MS_Activity
CallExternalMethod
SelfServiceActivity
#D
EventHandler
340

FIG. 3

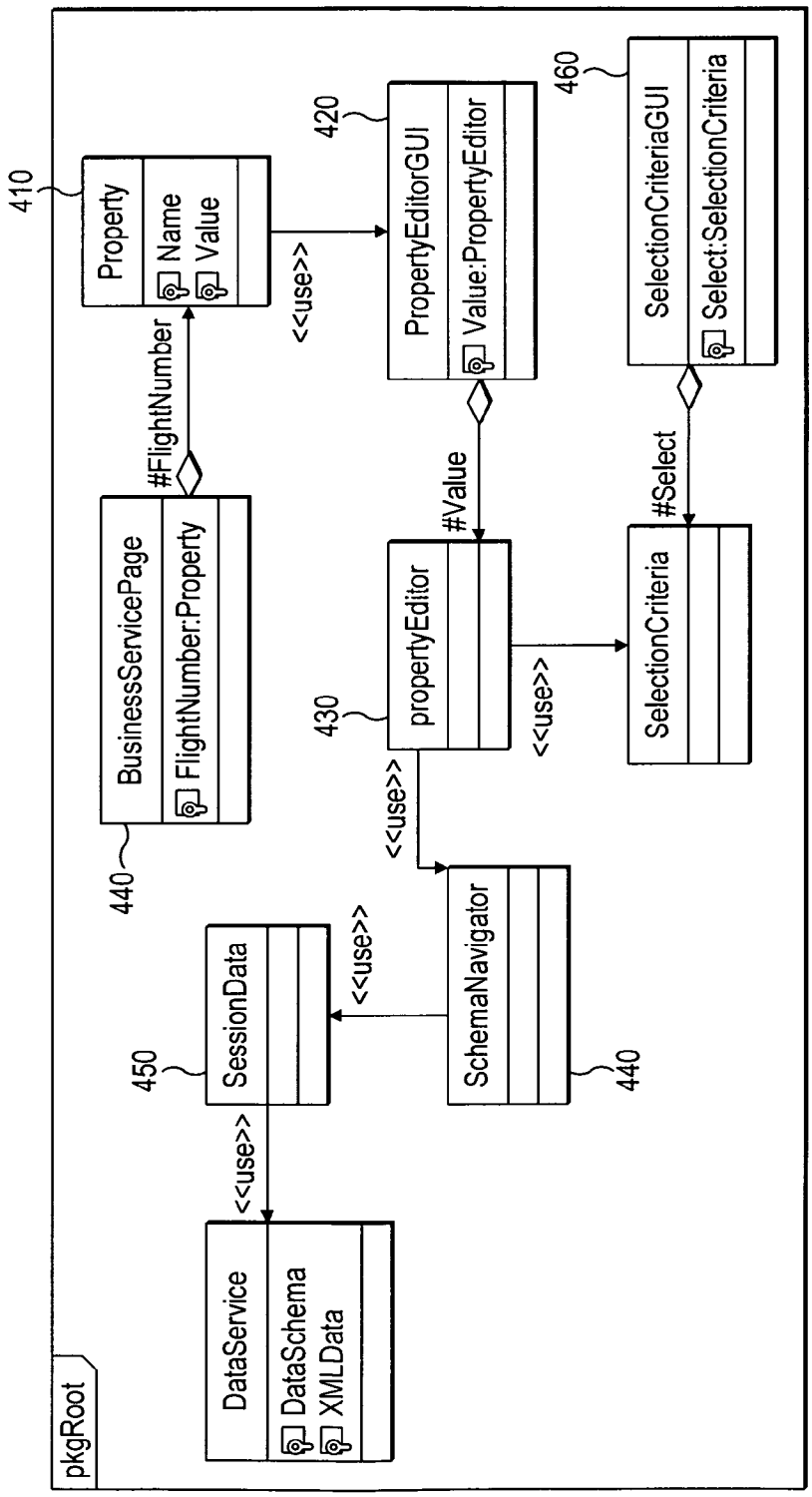
pkgRoot
Property
Name
Value
410
PropertyEditorGUI
Value:PropertyEditor
420
SelectionCriteriaGUI
Select:SelectionCriteria
460
<<use>>
#FlightNumber
BusinessServicePage
FlightNumber:Property
440
#Value
propertyEditor
430
#Select
SelectionCriteria
<<use>>
<<use>>
<<use>>
SessionData
450
SchemaNavigator
440
<<use>>
DataService
DataSchema
XMLData

FIG. 4