A System and Method For Building Component Applications Using Metadata Defined Mapping Between Message And Data Domains

BACKGROUND OF THE INVENTION

This application relates generally to communication of services over a network to a device.

There is a continually increasing number of terminal devices in use today, such as mobile telephones, PDAs with wireless communication capabilities, personal computers, self service kiosks and two-way pagers. Software applications which run on these devices increase their utility. For example, a mobile phone may include an application which retrieves the weather for a range of cities, or a PDA may include an application that allows a user to shop for groceries. These software applications take advantage of the connectivity to a network in order to provide timely and useful services to users. However, due to the restricted resources of some devices, and the complexity of delivering large amounts of data to the devices, developing software applications for a variety of devices remains a difficult and time-consuming task.

Currently, devices are configured to communicate with Web Services through Internet based Browsers and/or native applications. Browsers have the advantage of being adaptable to operate on a cross-platform basis for a variety of different devices, but have a disadvantage of requesting pages (screen definitions in HTML) from the Web Service, which hinders the persistence of data contained in the screens. A further disadvantage of Browsers is that the screens are rendered at runtime, which can be resource intensive. Native applications have the advantage of being developed specifically for the type of device platform, thereby providing a relatively optimized application program for each runtime environment. However, native applications have disadvantages of not being platform independent, thereby necessitating the development multiple versions of the same application, as well as being relatively large in size, thereby taxing the memory resources of the device. Further, application developers need experience with programming languages such as Java and C++ to construct these hard coded

native applications. There is a need for application programs that can be run on client devices having a wide variety of runtime environments, as well as having a reduced consumption of device resources.

The systems and methods disclosed herein provide a component based application environment to obviate or mitigate at least some of the above presented disadvantages.

SUMMARY OF THE INVENTION

It is desirable to drive down the complexity involved in developing the wireless application by reducing the need to do any explicit coding, as well as reducing device resources utilized by the application when provisioned. Having an intelligent wireless device runtime environment (Device Runtime) that provides a set of basic services to manage the wireless application as a series if application components, and their interactions, can simplify the development effort and reduce resource allocation. The wireless application is described as a set of components. The data domain for this category of applications is defined using atomic data components. The communication between the device and a server is defined using atomic message components. Both message and data components are described in metadata using a structured definition language such as XML. The relationships between the message and data components are embedded in the XML definitions in the form of message/data mappings. Typically, outgoing messages are derived from some underlying data component and incoming messages affect the current state (or data representation) of the application. It is therefore apparent that the metadata defined mapping relationship is preferable between the expression of data and message components.

According to the present invention there is provided a method for generating a communication message instance based on a data instance for interaction with an application executed on a device, the application including a data component having at least one data field definition and a message component having at least one message field definitition , the component definitions expressed in a structured definition language, the method comprising the steps of: selecting the message component corresponding to the message instance; identifying at least one unique mapping present in the message component, the mapping for specifying a

relationship between the message component and the data component as defined by a unique identifier representing the mapping; selecting the data component mapped by the mapping according to the unique mapping identifier; obtaining a data instance field value corresponding to the data field definitition of the mapped data component; generating a message field value of the message instance to include the data instance field value according to the format of the data field definitition as defined in the mapped data component.

According to a further aspect of the present invention there is provided a method for generating a data instance based on a message instance corresponding to an application executed on a device, the application including a data component having at least one data field definitition and a message component having at least one message field definitition, the component definitions expressed in a structured definition language, the method comprising the steps of: selecting the message component corresponding to the message instance; identifying at least one unique mapping present in the message component , the mapping for specifying a relationship between the message component and the data component as defined by a unique identifier representing the mapping; selecting the data component mapped by the mapping according to the unique mapping identifier; obtaining a message instance field value from the message instance corresponding to the mapped data component; assigning the message instance field value to a data field value of the data instance according to the format of the data field definitition as defined in the mapped data component.

According to a still further aspect of the present invention there is provided a method for generating a communication message instance based on a data instance for interaction with an application executed on a device, the application including a data component and a message component having at least one message field definitition and at least one data field definitition, the component definitions expressed in a structured definition language, the method comprising the steps of: selecting the data component corresponding to the data instance; identifying a unique mapping present in the data component, the mapping for specifying a relationship between the message component and the data component as defined by a unique identifier representing the mapping; selecting the message component mapped by the mapping according to the unique mapping identifier; obtaining a data instance field value corresponding to the

message field definitition of the mapped message component; generating a message field value of the message instance to include the data instance field value according to the format of the data field definitition as defined in the mapped message component.

According to a further aspect of the present invention there is provided a method for generating a data instance based on a message instance corresponding to an application executed on a device, the application including a data component and a message component having at least one message field definitition and at least one data field definitition, the component definitions expressed in a structured definition language, the method comprising the steps of: selecting the message component corresponding to the message instance; identifying a unique mapping present in the data component, the mapping for specifying a relationship between the message component and the data component as defined by a unique identifier representing the mapping; selecting the message component mapped by the mapping according to the unique mapping identifier; obtaining a message instance field value from the message instance corresponding to the data field definitition of the mapped message component; assigning the message instance field value to a data field value of the data instance according to the format of the data field definitition as defined in the mapped message component.

According to a further aspect of the present invention there is provided a device for generating a communication message instance based on a data instance for interaction with an application executed on the device, the application including a data component having at least one data field definition and a message component having at least one message field definitition , the component definitions expressed in a structured definition language, the method comprising the steps of: means for selecting the message component corresponding to the message instance; means for identifying at least one unique mapping present in the message component, the mapping for specifying a relationship between the message component and the data component as defined by a unique identifier representing the mapping; means for selecting the data component mapped by the mapping according to the unique mapping identifier; means for obtaining a data instance field value corresponding to the data field definitition of the mapped data component; means for generating a message field value of the message instance to include

the data instance field value according to the format of the data field definitition as defined in the mapped data component.

BRIEF DESCRIPTION OF THE DRAWINGS

These and other features will become more apparent in the following detailed description in which reference is made to the appended drawings by way of example only, wherein:

Figure 1 is a block diagram of a network system;

Figure 2 is a block diagram of a generic device of Figure 1;

Figure 3 is a block diagram of a component framework of the device of Figure 2;

Figure 4 is a block diagram of a component application program of Figure 2;

Figure 5 shows a representative application packaging and hosting model for the system of Figure 1;

Figure 6 shows an example method of implementing the component application program of Figure 4;

Figure 7 shows a further example method of implementing the component application program of Figure 4;

Figures 8a and 8b are examples of a message level mapping for the application of Figure 4;

Figures 9a and 9b are examples of a field level mapping for the application of Figure 4;

Figures 10a and 10b are examples of a complex mapping for the application of Figure 4;

Figure 11 demonstrates an algorithm for producing an outgoing message with effect of message mappings of Figures 8a,b, 9a,b and 10a,b; and

Figure 12 demonstrates an algorithm for processing an incoming message with effect of message mappings of Figures 8a,b, 9a,b and 10a,b.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

Network System

Referring to Figure 1, a network system 10 comprises a plurality of generic terminal devices 100 for interacting with one or more generic schema defined services 106, via a coupled Wide Area Network (WAN) 104 such as but not limited to the Internet. These generic terminal devices 100 can be such as but not limited to personal computers 116, wireless devices 101,

PDAs, self-service kiosks and the like. The generic services provided by the service 106 can be Web Services and/or other services such as but not limited to SQL Databases, IDL-based CORBA and RMI/IIOP systems, Legacy Databases, J2EE, SAP RFCs, and COM/DCOM components. Further, the system 10 can also have a gateway server 112 for connecting the desktop terminals 116 via a Local Area Network (LAN) 114 to the service 106. Further, the system 10 can also have a wireless network 102 for connecting the wireless devices 101 to the WAN 104. It is recognized that other devices and computers (not shown) could be connected to the web service 106 via the WAN 104 and associated networks other than as shown in Figure 1. The generic terminal devices 100, wireless devices 101 and personal computers 116 are hereafter referred to as the devices 100 for the sake of simplicity. Web services 106 are selected for the following description of the system 10, for the sake of simplicity. However, it is recognized that other services could be substituted for the web services 106, if desired. Further, the networks 102, 104, 112 of the system 10 will hereafter be referred to as the network 104, for the sake of simplicity.

Referring again to Figure 1, the devices 100 transmit and receive requests/response messages 105, respectively, when in communication with the web services 106. The devices 100 can operate as web clients of the web services 106 by using the requests/response messages 105 in the form of message header information and associated data content, for example requesting and receiving product pricing and availability from an on-line merchant. The web service 106 is an example of a system with which client application programs 302 (see Figure 2) on the communication devices 100 interact via the wireless network 104 in order to provide utility to users of the communication devices 100.

For satisfying the appropriate requests/response messages 105, the web service 106 can communicate with an application server 110 through various protocols (such as but not limited to HTTP and component API) for exposing relevant business logic (methods) to client application programs 302 (see Figure 2) once provisioned on the devices 100. The application server 110 can also contain the web service 106 software, such that the web service 106 can be considered a subset of the application server 110. The application programs 302 of the device 100 can use the business logic of the application server 110 similarly to calling a method on an

object (or a function). It is recognized that the client application program 302 can be downloaded/uploaded in relation to the application server 110, through the messages 105 via the network 104, directly to the devices 100. It is further recognized that the devices 100 can communicate with one or more web services 106 and associated application servers 110 via the networks 104.

Server Environment

Referring to Figure 1, the web service 106 provides the information messages 105 which are used by the client application programs 302 (see Figure 2) on the devices 100. Alternatively, or in addition, the web service 106 may receive and use the information messages 105 provided by the client application programs 302 executed on the devices 100, or perform tasks on behalf of client application programs 302 executed on the devices 100. The web service 106 can be defined as a software service, which can implement an interface such as expressed using Web Services Description Language (WSDL) registered in Universal Discovery Description and Integration (UDDI) in a web services registry, and can communicate through messages 105 with client devices 100 by being exposed over the network 104 through an appropriate protocol such as the Simple Object Access Protocol (SOAP). In some implementations, SOAP is a specification that defines the XML format for the messages 105, including a well-formed XML fragment enclosed in SOAP elements. SOAP also supports document style applications where the SOAP message 105 is a wrapper around an XML document. A further optional part of SOAP defines the HTTP binding (i.e. header), whereas some SOAP implementations support MSMQ, MQ Series, SMTP, or TCP/IP transport protocols. Alternatively, the web service 106 may use other known communication protocols, message 105 formats, and the interface may be expressed in other web services languages than described above.

Client Environment

Referring to Figure 2, the component applications 302 are transmitted via the network 104 and loaded into a memory module 210 of a device infrastructure 204 of the device 100. Alternatively, the component applications 302 may be loaded via a serial connection, a USB connections, or a short-range wireless communication system such as IR, 802.11(x)

BluetoothTM (not shown). Once loaded onto the device 100, the component applications 302 can be executed by a component framework 206 on the device 100, which can convert the component applications 302 into native code, which is executed by a processor 208 in the device infrastructure 204. Alternatively, the component applications 302 may be interpreted by another software module or operating system on the device 100. In any event, the component applications 302 are run in a terminal runtime environment provided by the device 100, such that the runtime environment is an intelligent software framework that provides a set of basic services to manage and execute typical application 302 behaviour (e.g. persistence, messaging, screen navigation and display).

Referring again to Figure 1, the client runtime environment provided by the devices 100 can be configured to make the devices 100 operate as web clients of the web services 106. It is recognized that the client runtime environment can also make the devices 100 clients of any other generic schema-defined services over the network 104. The client runtime environment of the devices 100 is preferably capable of generating, hosting and executing the client application programs 302 (which are in the form of component applications – see Figure 4 and description herein below) on the device 100. Further, specific functions of the client runtime environment can include such as but not limited to support for language, coordinating memory allocation, networking, management of data during I/O operations, coordinating graphics on an output device of the devices 100 and providing access to core object oriented classes and supporting files/libraries. Examples of the runtime environments implemented by the devices 100 can include such as but not limited to Common Language Runtime (CLR) by Microsoft and Java Runtime Environment (JRE) by Sun Microsystems.

The terminal runtime environment of the devices 100 preferably supports the following basic functions for the resident executable versions of the client application programs 302 (see Figure 2), such as but not limited to:

provide a communications capability to send messages 105 to the Web Services 106 or messages 105 to any other generic schema defined services connected via the network 104 to the devices 100;

provide data input capabilities by the user on an input device of the devices 100 to supply

data parts for Web Services' 106 outgoing messages 105 (messages to the service);

provide data presentation or output capabilities for Web Services' 106 response messages 105 (incoming messages) or uncorrelated notifications on the output device;

provide data storage services to maintain local client data in the memory module 210 (see Figure 2) of the device 100; and

provide an execution environment for a scripting language for coordinating operation of the application components 400, 402, 404, 406 (see Figure 4) of the client application programs 302.

Referring to Figures 2, 4 and 5, the client runtime (for example provided by the component framework 206) loads metadata contained in the component 400, 402, 404, 406 definitions and the builds the executable version of the application program 302 on the device 100, via for example an application container 300. There are, such as but not limited to, two operational models for client runtime: template-based native execution and metadata-based execution. With the template-based native execution model the runtime hosts data, message, and screen templates 500 pre-built on the device 100 using the native code. When the application program 302 definition is loaded, the client environment provided by the component framework 206 fills the templates 500 with metadata-defined parameters from the components 400, 402, 404 and builds the executable client application program 302 in the native format. The workflow script (for example ECMAScript) of the workflow component 406 could be either converted to native code or executed using an appropriate script interpreter 502 (e.g., ECMAScript Interpreter) to a native code redirector 504, where the redirector 504 interprets calls to the scripting language into operations on native components through a native runtime engine 506. With the metadata-based execution, the runtime environment of the component framework 206 either keeps component 400, 402, 404, 406 definitions in XML (for example), which are parsed during execution time or uses native representation of XML (for example) nodes. During execution, the native runtime engine 506 operates on definitions of the components 400, 402, 404, 406 rather than on native component entities.

Therefore, the native client runtime environment provides an interface for the client application programs 302 to the device 100 functionality of the processor 208 and associated

operating system of the device infrastructure 204. The runtime environment preferably supplies a controlled, secure and stable environment on the device 100, in which the component application programs 302 execute. The runtime environment provisions the definitions of the components 400, 402, 404, 406 to create the actual web client specific for each respective device infrastructure 204 of the device 100. It is recognized for the sake of simplicity that the following description hereafter will refer to the client runtime environment being provided by the component framework 206, as an example only.

Communication Device

Referring to again to Figure 2, the devices 100 are devices such as but not limited to mobile telephones, PDAs, two-way pagers or dual-mode communication devices. The devices 100 include a network connection interface 200, such as a wireless transceiver or a wired network interface card or a modem, coupled via connection 218 to a device infrastructure 204. The connection interface 200 is connectable during operation of the devices 100 to the network 104, such as to the wireless network 102 by wireless links (e.g., RF, IR, etc.), which enables the devices 100 to communicate with each other and with external systems (such as the web service 106) via the network 104 and to coordinate the requests/response messages 105 between the client application programs 302 and the service 106 (see Figure 1). The network 104 supports the transmission of data in the requests/response messages 105 between devices and external systems, which are connected to the network 104. The network 104 may also support voice communication for telephone calls between the devices 100 and devices which are external to the network 104. A wireless data transmission protocol can be used by the wireless network 102, such as but not limited to DataTAC, GPRS or CDMA.

Referring again to Figure 2, the devices 100 also have a user interface 202, coupled to the device infrastructure 204 by connection 222, to interact with a user (not shown). The user interface 202 includes one or more user input devices such as but not limited to a QWERTY keyboard, a keypad, a trackwheel, a stylus, a mouse, a microphone and the user output device such as an LCD screen display and/or a speaker. If the screen is touch sensitive, then the display can also be used as the user input device as controlled by the device infrastructure 204. The user interface 202 is employed by the user of the device 100 to coordinate the requests/response

message messages 105 over the system 10 (see Figure 1) as employed by client application programs 302 of a component framework 206, further described below.

Referring again to Figure 2, operation of the device 100 is enabled by the device infrastructure 204. The device infrastructure 204 includes the computer processor 208 and the associated memory module 210. The computer processor 208 manipulates the operation of the network interface 200, the user interface 202 and the component framework 206 of the communication device 100 by executing related instructions, which are provided by an operating system and client application programs 302 located in the memory module 210. Further, it is recognized that the device infrastructure 204 can include a computer readable storage medium 212 coupled to the processor 208 for providing instructions to the processor and/or to load/update client application programs 302 in the memory module 210. The computer readable medium 212 can include hardware and/or software such as, by way of example only, magnetic disks, magnetic tape, optically readable medium such as CD/DVD ROMS, and memory cards. In each case, the computer readable medium 212 may take the form of a small disk, floppy diskette, cassette, hard disk drive, solid state memory card, or RAM provided in the memory module 210. It should be noted that the above listed example computer readable mediums 212 can be used either alone or in combination.

Component Framework of Device

Referring again to Figure 2, the component framework 206 of the device 100 is coupled to the device infrastructure 204 by the connection 220. The client runtime environment the device 100 is provided by the component framework 206, and is preferably capable of generating, hosting and executing the client application programs 302 (which are in the form of component applications – see below) from meta-data definitions. Therefore, component framework 206 provides the native client runtime environment for the client application programs 302 and is an interface to the device 100 functionality of the processor 208 and associated operating system of the device infrastructure 204. The component framework 206 provides the runtime environment by preferably supplying a controlled, secure and stable environment on the device 100, in which the component application programs 302 execute in the application container 300, for example. The application container 300 can be referred to as a

smart host container for the client application program 302, and can be responsible for analyzing message meta-data (of the messages 105 – see Figure 1) and for updating the representation of the meta-data in the memory module 210.

Referring to Figure 3, the component framework 206 can be used to execute the client application programs 302 (such as Web Service client applications) within the terminal runtime environment and can support access to Web Service 106 and associated application servers 110 (see Figure 1), via the request/response messages 105. The component application programs 302 comprise software applications which are executed by the component framework 206. The component framework 206 creates the application container 300 for each component 400, 402, 404, 406 (see Figure 4) of the application program 302, each time that the component application program 302 is executed. The application container 300 loads the components 400, 402, 404, 406 of the application program 302 and can create native code which is executed by the processor 208 in the device infrastructure 204. The component framework 206 therefore provides the host application containers 300 for provisioning the definitions of the components 400, 402, 404, 406 to create the actual web client specific for each respective device infrastructure 204 of the communication devices 100. The application container 300 can provision the component application 302 as per the template-based native execution and metadata-based execution models as described above, by way of example only.

Referring again to Figure 3, the component framework 206 can also provide framework services 304 (a standard set of generic services) to the client application programs 302, in the event certain services are not included as part of the components 400, 402, 404, 406 (see Figure 4) or received as separate components (not shown) as part of the component application program 302. The application program 302 has communications 214 with the application container 300, which can coordinate communications 216 with the framework services 304, as needed. The framework services 304 of the component framework 206 coordinate communications via the connection 220 with the device infrastructure 204. Accordingly, access to the device infrastructure 204, user interface 202 and network interface 200 is provided to the client application programs 302 by the component framework 206 and associated services 304. It is

recognized that a portion of the operating system of the device infrastructure 204 (see Figure 2) can represent the application container 300 and any services of the framework services 304.

The framework services 304 can include such as but not limited to a communication service 306, a presentation service 308, a persistence service 310, an access service 312, a provisioning service 314 and a utility service 316. The communication service 306 manages connectivity between the component application programs 302 and the external system 10, such as the messages 105 and associated data sent/received in respect to the web service (by the communication service 306) on behalf of the component applications 302. As further described below with reference to Figures 8a,b, 9a,b, 10a,b the communication service 306 can be used to implement a series of mappings 800 (see Figure 8a). The presentation service 308 manages the representation of the component application programs 302 as they are output on the output device of the user interface 202 (see Figure 2). The persistence service 310 allows the component application programs 302 to store data in the memory module 210 (see Figure 2) of the device infrastructure 204. It is recognised the persistence service 310 can be used to coordinate the modification/creation of data instances of the data components 400 linked to the message components 404 via the mappings 800 (see Figure 8a). The access service 312 provides the component application programs 302 access to other software applications which are present on the communication device 100. The provisioning service 314 manages the provisioning of software applications on the communication device 100. Application provisioning can include requesting and receiving new and updated component application programs 302, configuring component application programs 302 for access to services which are accessible via the network 104, modifying the configuration of component application programs 302 and services, and removing component application programs 302 and services. The utility service 316 is used to accomplish a variety of common tasks, such as performing data manipulation in the conversion of strings to different formats.

It is recognized that the framework services 304 of the communication device 100 can provide functionality to the component application programs 302, which can include the services described above. Further, the framework services 304 can be integrated with the components 400, 402, 404, 406 of the application 302 rather than provided as a separate framework 304. In

any event, the component application programs 302 can have access to the functionality of the communication device 100 through integrated and/or separate framwork services 304.

Example Component Application Program

Referring to Figure 2, the Web Service (for example) client application programs 302 are executed within the terminal runtime environment of the component framework 206 and support access to Web Service operations provided by the service 106 (see Figure 1). WSDL and SOAP protocol definitions clearly imply a messages/data pattern. In a WSDL Web Service definition, the operations are defined using the notion of messages and data parts, which are used to define the Web Service client application programs 302 as a set of the related data 400 and the message 404 components (see Figure 4).

Referring to Figure 4, a block diagram of the component application program 302 comprises the data components 400, the presentation components 402 and the message components 404, which are coordinated by workflow components 406 through communications 214 with the application container 300. The structured definition language can be used to construct the components 400, 402, 404 as a series of metadata records, which consist of a number of pre-defined elements representing specific attributes of a resource such that each element can have one or more values. Each metadata schema typically has defined characteristics such as but not limited to; a limited number of elements, a name of each element, and a meaning for each element. Example metadata schemas include such as but not limited to Dublin Core (DC), Anglo-American Cataloging Rules (AACR2), Government Information Locator Service (GILS), Encoded Archives Description (EAD), IMS Global Learning Consortium (IMS), and Australian Government Locator Service (AGLS). Encoding syntax allows the metadata of the components 400, 402, 404 to be processed by the device infrastructure 204 (see Figure 2), and encoding schemes include such as but not limited to XML, HTML, XHTML, XSML, RDF, Machine Readable Cataloging (MARC), and Multipurpose Internet Mail Extensions (MIME).

Referring again to Figure 4, the data components 400 define data entities which are used by the component application program 302, including application data represented in for

example native code or XML. Examples of data entities which data components 400 may describe are orders, users, and financial transactions. Data components 400 define what information is required to describe the data entities, and in what format the information is expressed. For example, the data component 400 may define such as but not limited to an order which is comprised of a unique identifier for the order which is formatted as a number, a list of items which are formatted as strings, the time the order was created which has a date-time format, the status of the order which is formatted as a string, and a user who placed the order which is formatted according to the definition of another one of the data components 400. Since data parts (elements) are usually transferred from message 105 to message 105 according to Web Services' 106 choreography rules, preferably there is persistence of data components 400. Data components 400 may be dynamically generated according to Web Services' 106 choreography definitions (if available) or defined by the application designer based on complex type definitions and/or message correlation information. It is recgnised that the message components 404 can be linked via the mappings 800 to the data components 400 (see Figure 8a), as further described below.

Referring again to Figure 4, the message components 404 define the format of messages used by the component application program 302 to communicate with external systems such as the web service 106, and include message data represented in for example native code or XML. For example, one of the message components 404 may describe such as but not limited to a message for placing an order which includes the unique identifier for the order, the status of the order, and notes associated with the order. Message component 404 definitions written in the structured definition language can uniquely represent (and map to) WSDL messages, and can be generated dynamically at runtime. Accordingly, the dynamic generation can be done for the component definitions for client application messages 105, and associated data content, from standard Web Service metadata in the definition language used to express the web service interface, for example such as but not limited to WSDL and BPEL. Web Service messages 105 are defined within the context of operation and there is defined correlations between the message components 404 in the component application program 302 definition. This correlation could be done using predefined message parameters and/or through separate workflow components 406, as further defined below.

Referring again to Figure 4, the presentation components 402 define the appearance and behavior of the component application program 302 as it displayed by the user interface 202. The presentation components 402 can specify GUI screens and controls, and actions to be executed when the user interacts with the component application 302 using the user interface 202. For example, the presentation components 402 may define screens, labels, edit boxes, buttons and menus, and actions to be taken when the user types in an edit box or pushes a button. The majority of Web Service consumers use a visual presentation of Web Service operation results, and therefore provide the runtime environment on their devices 100 capable of displaying user interface screens.

Referring again to Figure 4, the workflow components 406 of the component application program 302 define processing that occurs when an action is to be performed, such as an action specified by a presentation component 402 as described above, or an action to be performed when messages 105 (see Figure 1) arrive from the system 10. Presentation workflow and message 105 processing are defined by the workflow components 406. The workflow components 406 are written as a series of instructions in a programming language or a scripting language, such as but not limited to ECMAScript, and can be compiled into native code and executed by the application container 300, as described above. An example of the workflow components 406 may be to assign values to data, manipulate screens, or send the message 105. The workflow component 406 supports a correlation between the messages 105 and defines application flow as a set of rules for operations on the other components 400, 402, 404. Multiple workflow components can be defined with respect to a given application program 302.

ECMA (European Computer Manufacturers Association) Script is a standard script language, wherein scripts can be referred to as a sequence of instructions that is interpreted or carried out by another program rather than by the computer processor. Some other example of script languages are Perl, Rexx, VBScript, JavaScript, and Tcl/Tk. The scripting languages, in general, are instructional languages that are used to manipulate, customize, and automate the facilities of an existing system, such as the devices 100. In such systems, useful functionality is already available through the user interface 202 (see Figure 2), and the scripting language is a

mechanism for exposing that functionality to program control. In this way, the device 100 is said to provide the host runtime environment of objects and facilities which completes the capabilities of the scripting language.

Accordingly, referring to Figure 4, the client application programs 302 can be defined as a set of platform-neutral component definitions, namely for data 400 and message 404 components, and presentation components 402 using XML (or any other suitable structured definition language). The workflow components 406 can be defined using ECMAScript (or any other suitable platform-neutral scripting language). The client runtime environment of the component framework 206 (see Figure 2) can generate component templates based on meta-definitions, as further described below, when the components 400, 402, 404, 406 of the component application program 302 are provisioned on the device 100. With a large variety of terminal runtime environments, the cross-platform standards such as XML or ECMAScript can be used to define application component metadata instead of pre-building the component application programs 302. This delayed binding can allow generic application definitions of the component application programs 302 to be run on a wide variety of terminal system environments, represented by various different devices 100.

Expressing the data 400, message 404, and presentation 402 components using XML or its derivatives, and the workflow component 406 using the ECMAScript language or its subset, can allow an application developer to abstract the Web Service client from any specific platform or environment and implement in principle "develop once run everywhere" applications. The following example shows how a Web Services client application program 302 could be expressed using a structured definition language, such as but not limited to XML, and a platform neutral scripting/programming language, such as but not limited to ECMAScript, defined components:

example XML data components 400

```
<data name="Order">
        <item name="orderId" type="Number" key="true"/>
        <item name="items" type="String" array="true"/>
        <item name="user" comp="true" compName="User"/>
```

```
                <item name="orderStatus" type="String"/>
        </data>
        ...
```

example XML message components 404

```
        <msg name="ordConfirmation" type="response" action="mhConfirmation">
                <part name="orderId" type="String" />
                <part name="status" type="String" />
        </msg>
        ...
```

example XML presentation components 402

```
        <screen name="scrConfirmation" title="Order Confirmation" param="Order">
                <layout type="vertical">
                <widget type="label" value="Order Confirmation Result:"/>
                        < widget type="edit" value="@Order.orderStatus"/>
        </layout>
        ...
                <menu>
                        <item label="Continue"  navigate="@scrMain"/>
                        ...
                </menu>
        </screen>
        ...
```

example ECMAScript workflow components 406

```
        <actions>
                <function name="mhConfirmation">
                        key = ordConfirmation.orderId;
                        order = Order.get(key);
                        order.orderStatus = ordConfirmation.status;
```

```
            scrConfirmation.display(order);
        </function>
        ...
    </actions>
```

Referring to Figure 4, as given above, it can be seen that the message components 404 relay the required data for the input and output of the messages 105. The corresponding data components 400 coordinate the storage of the data in the memory module 210 (see Figure 2) of the device 100 for subsequent presentation on the user interface 202 (see Figure 2) by the presentation components 402. The workflow components 406 coordinate the transfer of data between the data 400, presentation 402, and message 404 components.

Further, the component application architecture can provide a relatively small application download size (consisting of component definitions only) as compared to hard coded native applications, and an effective data storage and persistence model. The client runtime is capable of storing and updating atomic data entities directly vs. manipulating rendered presentations such as HTML pages for browser applications.

Example Operation of Component Application Model

Referring to Figures 1, 3 and 6, for example, operation 600 shows when the device 100 receives 602 the response message 105 containing message data, the appropriate workflow component 406 interprets 604 the data content of the message 105 according to the appropriate message component 404. The workflow component 406 then processes 606 the data content and inserts 910 the data into the corresponding data component 400 for subsequent storage 612 in the memory module 210 (see Figure 2). Further, if needed, the workflow component 406 also inserts 608 the data into the appropriate presentation component 402 for subsequent display 614 on the user interface 202 (see Figure 2).

Referring to Figures 1, 3 and 7 operation 700 shows data input 702 for an action, such as pushing a button or selecting a menu item, which the user performed 703 on a user-interface element through the user interface 202 (see Figure 2). The relevant workflow component 406

interprets 704 the input data according to the appropriate presentation component 404 and creates 706 data entities which are defined by the appropriate data components 400. The workflow component 406 then populates 710 the data components 400 with the input data provided by the user for subsequent storage 712 in the memory module 210 (see Figure 2). Further, the workflow component 406 also inserts 708 the input data into the appropriate message component 404 for subsequent sending 714 of the input data as data entities to the web service in the message 105, as defined by the message component 404.

Mapping between Data and Message Components

As described above with reference to Figure 4, the wireless applications 302 are expressed as a collection of message 404, data 400 and components 402, 406, including information that specifies how these components interact. The application 302 is expressed using a structured definition language such as XML. It is noted that the expression of both messages 404 and data 400 as components bear certain similarities:

- each component 400, 404 is identified by a unique name; and
- each component 400, 404 specifies one or more subfields consisting of name and declared type.

In practice, typically the expression of the components 400, 404 by the developer can be almost identical, while the behaviour of each of the components 400, 404 of the application 302 is distinct. Therefore, by recognizing the fact that message 105 (see Figure 1) content is often generated from some underlying data element, and in light of the similarities between expression of these components 400, 404, it is convenient to introduce certain mappings 800 (see Figure 8a) to the expression of message components 404, as further described below. These mappings 800 are essentially shortcuts to the expression of the message 105 that specify how the message's definition is obtained regarding the message component 404, and how the message component 404 behaves at runtime during execution of the application 302. The mapping 800 is a stated relationship between the message component 404 definition and the data component 400 definition. In relation to expression of the message component 404, using the mapping 800 can reduce the amount of metadata required to describe the component 404. Thus use of the mapping 800 can have a direct effect on the amount of "code" required to describe the application 302. In

relation to how the component 404 behaves at runtime, the mapping 800 specifies how linked data elements (described by the data component 400) are resolved and affected by message state. In this regard, specifying the mapping 800 can reduce the need for the developer to provide additional specific message handling code in the application 302.

Mapping Resolution Contract

Referring again to Figures 1 and 8a, the application and corresponding services 304 rely upon a mapping resolution contract or mapping rule having a unique identifier 802 (see Figure 8a). This mapping rule states that any mapping 800 attached to the data component 400 will map exactly one key field 804 per mapped data type. This mapping rule provides for unique identification and modification of the data instance affected by the mapping 800. The mapping rule states that the mapping 800 isolates an instance of the data component 400 to which the message content of the corresponding message component 404 is linked. Data component 400 instances are resolved by the unique identifier 802 (e.g. a key). It is noted that the composition of this identifier 802 could be such as but not limited to a simple primary key or a composite key arising from more than one field. A single field 804 (such as a component name) in the Data definition of the data component 400 is identified as representing this identifier 802. The mapping resolution contract provides that exactly one primary key field 804 is involved in the mapping 800 to each linked data component 400. This one to one property of the mapping 800 provides for the unique the resolution of data instances to which incoming message data applies, as further described below. A particular data instance is represented as a selected data component 400 that is assigned data values to each of the field names 808. A message instance 806 is represented as a selected message component 404 that is assigned data values to contained message field(s) through the mappings 800.

Two types of mappings 800 are described: field level mappings 901, and message level mappings 801. The following elaborates on how message 404 to data 400 component mappings 800 may be expressed, and specify a runtime resolution contract that exists to determine uniquely where message content is to be applied.

Message Level Mappings 801

Referring again to Figure 8a, the Message level Mapping 801 is a mapping 800 from the message component 404 directly to the named data component 400 definition, such that message 806 field properties (message instance) are identical to those on the mapped data component. Message level mappings 801 state that the message instance 806 derives its complete specification from the linked data element of the data component 400. All fields described in the linked data component 400 will be present in the message instance 806, observing both field names 808, type declarations 810 and field order. For example, this type of message level mapping 801 can be convenient when the incoming or outgoing message instances 806 exactly duplicate the information represented by the data instance of the data component 400. Referring to Figure 8b, a sample message level mapping 801 between the **Order** data component 400 and the **submitOrder** message component 404 is illustrated. The mapping resolution contract for the mapping 801 is satisfied by the implicit linking of orderId primary key field 802. A sample structured definition language description (e.g. XML) of this relationship is provided in Figure 8b. It is apparent from the XML expression that the size of the application 302 definition (see Figure 4) can be reduced by introducing this mapping 801, as the listing of arguments 812 of the data component 400 is not repeated in the linked message component 404.

Field Level Mappings 901

The Field level Mapping 901 (see Figure 9a, 9b) provides a mapping 800 from a particular field 904 of the message component 404 definition to the named field 808 of the named data component 400 definition. Field level mappings 901 may arise where a more flexible arrangement of mappings 800 is required. In this configuration, each field mapping 901 specifies a linkage between each selected field 904 of the message instance 906 and the field 808 of the data instance corresponding to the data component 400. There may be any number of such field mappings 901. Field mappings 901may involve only one target data component 400 (one-to-one linkage) or multiple data components 400 may be linked to the message instance 906 through separate field mappings 901 (one-to-many linkage). In order to satisfy the mapping resolution contract, the key field 802 is included for every data component 400 that is linked to the message component 404.

Referring to Figure 9a, one-to-one mapping 901 arrangements incorporate a link to a single data component 400. One field mapping 901 is made to the field representing the primary key 802 of the data component 400, thus linking the message instance 906 with the data instance of the data component 400. Other mappings 901 are made between the selected message fields 904 of the component 404 and corresponding data fields 808 of the component 400. Figure 9a depicts a typical field level mapping 901 relationship where a subset of the **Part** fields 808 are linked to the **priceReduction** message field 904. The mapping resolution contract is satisfied by making the link 901 to the partNo field which is identified as the key field 802 of **Part**. A sample XML expression for these relationships is provided in Figure 9b, where Key field mapping 901 is shown in bold. It is recognised that the message instance 906 can have more than one message field 904, each mapped 901 to a respective data field 808 under the same key 802 (i.e. the message component 404 can be linked to two or more data fields 808 of the data component 400 using the same key 802).

Complex Mappings 1001

Referring to Figures 10a and 10b, a complex mapping 1000 arrangement consists of field level mappings 901 to two or more data components 400. As with the one-to-one mapping case of Figure 8a,b, different unigue ones of the primary key field 802 mapping is provided for every data component 400 linked through the set of field mappings 901. Figure 10b shows an XML representation of the relationships between the **orderUpdate** message 404 and the **Order** and **Inventory** data components 400. For each of the two data components 400 linked, a respective primary field mapping 901 with keys 802 is in place; orderId field key 802 for Order 400 and partNo field key 802 for Inventory 400. This satisfies the mapping resolution contract. These primary key field mappings 901 are shown in bold.

In view of the examples shown in Figures 8a,b, 9a,b, and 10a,b, other mapping 800 configurations are possible. Examples of such include such as but not limited to Extended Definition, Message Prototyping, and Arrival Event Processing, as further described below.
An Extended Definition is a message component 404 that extends the message 801 or field mapping 901configuration by defining additional un-mapped fields 904. This extended message instance 906 may extend its definition, in the presence of mappings 801, 901, by adding fields

904 that are not mapped to a respective data component 400 but rather complete their own specification whithin the message component definition 404. These fields 904 may be added to either the message 906 that has one or more field mappings 801, 901, or the message 906 that is mapped 801,901 to a respective data component 400. Extended definition can provide an additional measure of flexibility to the specification of the mapped message 906. Message Prototyping can be defined as the ability to extend the stated definition of another message component 404. This mechanism has a similar effect as in object oriented inheritance; all the declared fields 904 of the parent message 906 will be available to the extending message 906. With regard to mapping 801, 901 relationships, the extending message mappings 801,901 could override any mapping specifications stated on the parent message 906. For Message Arrival Event Processing, the mapping mechanism can be further enhanced by permitting the association of additional processing code to the message reception. The body of processing code can be identified through the specification of the message component 404 through application XML. The processing code may be a script (such as ECMAScript) embodied in the application 302 (e.g. a workflow component 406), or may identify a standard native transformation offered by the Device Runtime environment of the component framework 206 (see Figure 2). Handling of message arrival event processing is further discussed below.

Message Generation 1100

Referring to Figures 8a,b, 9a,b, and 11, for the origination of message instances 906, the message to data mappings 801, 901 define the source and format for content that the message instance 906 will carry. The message format may be derived identically from the linked data component 400 (message mapping 801) or may be defined by the aggregate effect of multiple data component 400 relationships (field level mapping 901). Finally, some message fields can carry their own declarations (extended definition). In order to generate the originating message instance 906 that specifies either type of mapping 801, 901, all dependent data component 400 instances are supplied to the input of the generation process 1100. In addition the use of the extended declaration of fields 904 implies that the supplied message instance 906 provides the necessary field value.

Referring to Figures 9a,b and 11, the ultimate generation 1114 of a message instance **M** (corresponding to message instance 806,906) is done using a message declaration **Ms** (corresponding to message component 404). In the case of direct message mapping 801, the mapped data component definition **Ds** (corresponding to data component 400) is used with input data instance **d** to determine the ultimate message instance **M**. In step 1102, the message component **Ms** is obtained and in step 1104, a decision is made by the device runtime environment (for example the comunication service 306 – see Figure 3) by examining the component **Ms** to see whether message level mapping 801 is present. In the YES case, the metadata declaration **Ds** is obtained 1106 as defined by the mapping 801. The service 306 inputs 1108 the data instance **d** values from the persistence service 310 and writes 1112 in the message field 904 of the message instance **Ms**. The communication service 306 then analyses 1110 the data component **Ds** to see if there are any more data fields 808 for relating to the message instance **M** implied by the mapping 801. In the YES case, steps 1106, 1108, and 1112 are repeated until all of the fields 808 are included in the message instance M. The message instance **M** is then generated at step 1114.

In the case of field mapping 901 the data component definition $\mathbf{Ds_p}$ and subordinate data field declarations $\mathbf{Ds.f_n}$ are extracted by interrogating the message field level mapping 901 by the communications service 306. The appropriate field value 808 ($\mathbf{d.f_n}$) from input data instance $\mathbf{d_i}$ is added into the output message instance **M**. In step 1104 in the NO case, at step 1116 the communication service 306 determines whether there are field mappings 901 present, if no then the current contents of the message instance **M** are generated at step 1114, as noted above. Otherwise, the message field 904 descriptions are analysed 1118 and where field mappings 901 are in place 1120, the mapped data component type is obtained 1124, the mapped field type declaration is obtained 1126 from the data component **Ds**, and the data value of the data instance corresponding to the declaration is obtained 1128 from the persistence service 310. The values are then written in the corresponding message field 904 using the format of the data component **Ds**. If there are no further mappings 901, then the message instance **M** is generated at step 1114. Finally, in the case of an extended definition, the field format is taken directly 1122 from **Ms**, and it is expected that the field value is available through input message instance **M**. It is recognised that a combination of the message and field mappings 801, 901 as described above

could be provided, in particular where more than one type of mapping is provided in the message instance **M** (i.e. multiple mappings 801, 901 for multiple components **Ds** linked to the message component **Ms**).

Message Processing 1200

Upon arrival of the message instance **M** (see Figure 11), mappings 800 have one of two effects:

- cause field $\mathbf{d.f_n}$ updates in the mapped data component instance(s) **d**; and
- cause creation of a new data component instance(s) **d** with fields $\mathbf{d.f_n}$ as determined by further mapping relationships 800.

The data instance(s) **d** to which message field values $\mathbf{M.f_n}$ apply is determined by virtue of the message to data mapping resolution contract. Data instances **d** are resolved by taking the message field values $\mathbf{M.f_n}$ that are mapped to data primary key fields 802 (see Figure 8a). Depending on the type of mappings 800 employed, there may be one (message mapping 801), or one or more (field mapping 901) data instances **d** affected by message **M** arrival at either the device 100 or the web service 106. For finding and updating data, the message field value $\mathbf{M.f_n}$ corresponding to the message field 904 (see Figure 9a), mapped to data primary key field, is not unique. A data instance **d** of the mapped typed already exists with the primary key value 802. The data instance **d** is resolved by this primary key value 802, and all other mappings 800 to this data type update the fields $\mathbf{d.f_n}$ on this data instance **d**. It is noted that there may be more than one data instance **d** affected by this process when field mappings 901 are employed. For creating new data, the message field value $\mathbf{M.f_n}$ corresponding to the message field mapped to data primary key field, is unique. As a result, a new instance of data **d** is created with the associated primary key field value 802. All subsequent mappings to this data type update the fields $\mathbf{d.f_n}$ on the newly created data instance **d**. As with "find and update", there may be more than one data instance **d** created when using field mappings 901. For the data resolution and update process, the processing of arrival messages **M** and effect on existing/new data instances **d** is depicted in Figure 12. Extended definition (unmapped) fields have no bearing in the data resolution and update process. For simplicity, the case of detecting this field type is not depicted in the following. Figure 12 shows the resolution and update process for data components 400 associated to the message instance **m** through mappings 800.

At step 1202 the metadata definition for the message instance **m** is obtained from the message component **Ms**, where the message instance **m** is related to the message component 404 at step 1204. In the case of a direct message mapping 801 at step 1206, the data component specification **Ds** is obtained 1208 from the mapping information. The primary key field of **Ds** is determined 1210 and the value to be applied to this field ($\mathbf{m.f_{pk}}$) is obtained 1212 from the corresponding field of **m**. At step 1214 an existing version of the instance **d** is searched in the memory 210 (see Figure 2). Using this value, the instance of **Ds** may either be found 1216 or created 1218, thus producing **d**. At step 1220 more fields of the data component **Ds** are determined. If yes, the data component definitions for the corresponding fields are analysed 1222 and the remaining field values of **m** are applied 1224 to each of the corresponding fields of the resolved, thus generating 1226 the updated/created data instance **d**.

At step 1228 in the case of field mappings 901, primary keys for each of the field mappings 901 are determnined. The data component specification **Ds** is obtained 1230 from the mapping information 901. The resolution process relies on determining all field level mappings 901 from **Ms** that are mapped to data component primary fields: $\mathbf{Ms.f_{pki}}$. The component data type of the field mapping 901 can be determined from $\mathbf{Ms.f_{pki}}$ as $\mathbf{Ds_i}$. Using 1232 the key value 802 from the input message **m** for this field ($\mathbf{m.f_{pki}}$), the data instance of $\mathbf{Ds_i}$ may be either found 1236 or created 1234, thus producing the data instance $\mathbf{d_i}$ corresponding to $\mathbf{Ds_i}$. The data instances $\mathbf{d_i}$ are resolved and updated one at a time by selecting 1238 each field $\mathbf{di.f_n}$ in sequence. The mapped message field $\mathbf{Ms.f_p}$ is obtained 1240 and the corresponding mapped field definition $\mathbf{Dsi.f_n}$ is obtained 1242. Remaining fields of **Ms** that are mapped 901 to fields of $\mathbf{Ds_i}$ have their values taken from the message instance field $\mathbf{m.f_p}$ and applied 1244 to $\mathbf{d_i.f_n}$ using the corresponding field format of $\mathbf{Ds_i}$; $\mathbf{Ds_i.f_n}$. This process continues until all mapped data instances are updated, thus generating **d** at step 1226.

As stated previously, referring to Figure 1, it may be desirable to associate a script or other executable unit (e.g. the workflow component 406) – see Figure 4) to perform additional processing or transformations on the application 302 when the message 105 arrives. To this end, the message component definition 404 may specify the address of a custom processing unit (part

of the framework 206) that will be invoked when a message 105 of this type is received. In addition to executing the script, the intelligent Device Runtime makes available the message 105 and all its field values to the scope of the processing unit. Further, all data instances $\mathbf{d_i}$ affected through mappings 800 are made available to the scope of the processing unit.

Although the disclosure herein has been drawn to one or more exemplary systems and methods, many variations will be apparent to those knowledgeable in the field, and such variations are within the scope of the application. For example, although XML and a subset of ECMAScript are used in the examples provided, other languages and language variants may be used to define the component applications 302. Further, it is recognised as an alternative to the above described mapping 800 and operation 1100, 1200, the definition of the data fields 808 (see Figure 8) could be contained in the message component 404 (see Figure 4). Therefore, the generation of the message instance would be based on data field definitions included in the message component 404 definitions, and the data component 400 would be mapped 800 to the corresponding message component 404 having included data field definitions. Accordingly, generation of the data instances would rely upon data field definitions contained in the mapped message component 404.

CLAIMS:

1. A method for generating a message field value of a communication message instance, the form and content of the message instance being defined by a corresponding message component of a component based application comprising a data component, the message component, a screen component and a coordinating workflow component, each of the components having respective field definitions expressed in a structured definition language, the method comprising the steps of:

selecting the message component corresponding to the message instance, the message component defining a message field value of the message instance;

identifying in the message component a unique identifier representing a mapping between the message component and the data component, the unique identifier mapping the message field value definition to a data field definition of the data component, the data field definition defining a data instance field value;

selecting the mapped data component;

obtaining the data instance field value corresponding to the data field definition of the mapped data component; and,

generating the message field value of the message instance, the message field value incorporating the data instance field value defined by the mapped data component.

2. The method as claimed in claim 1, wherein a plurality of the data field definitions of the data component is shared between the message component and the data component as represented by the mapping.

3. The method as claimed in claim 2 further comprising the step of linking the plurality of data field definitions to corresponding ones of the message fields of the message component as represented by the unique identifier.

4. The method as claimed in any of claims 1 to 3 further comprising the step of identifying additional processing code coupled to the message component for assisting in processing of the message instance.

5. The method as claimed in claim 4, wherein the processing code specifies an address of a provisioning unit for handling the processing code.

6. The method as claimed in any of claims 1 to 5 further comprising the step of identifying additional unmapped ones of the message field definitions in relation to the unique identifier.

7. The method as claimed in any of claims 1 to 6, wherein the mapping is a field level mapping linking a first one of the message field definitions to a first one of the data field definitions and a second one of the message field definitions to a second one of the data field definitions.

8. The method as claimed in claim 7, wherein the number of message field definitions of the message component is less that the number of data field definitions of the mapped data component.

9. The method as claimed in claim 7 or claim 8, wherein the first message field definition is mapped by a first one of the unique identifiers to a first one of the data components and the second message field definition is mapped by a second one of the unique identifiers to a second one of the data components difference from the first data component.

10. The method as claimed in claims 7 or claim 8, wherein the first message field definition and the second message field definition are mapped to the same data component using respective unique identifiers.

11. The method as claimed in any of claims 1 to 10, wherein the structured definition language is XML based.

12. The method as claimed in any of claims 1 to 11, wherein the application is a wireless application and the device is a wireless device.

13. The method as claimed in any of claims 1 to 12, wherein the unique identifier is a simple primary key.

14. The method as claimed in any of claims 1 to 13, wherein the unique identifier is a composite key.

15. The method as claimed in any of claims 1 to 14, wherein the message component defines a parent one of the message components coupled to a dependent one of the message components.

16. The method as claimed in claim 15 further comprising the step of overriding one of the mappings of the parent message component by respective one of the mappings of the dependent message component.

17. The method as claimed in any of claims 1 to 16, wherein generation of a communication message instance is based on a data instance for interaction with an application executed on the device.

18. A method for generating a data instance field value of a data instance of a data component, the data instance being derived from a message instance, the form and content of which being defined by a corresponding message component of a component based application comprising the data component, the message component, a screen component and a coordinating workflow component, each of the components having respective field definitions expressed in a structured definition language, the method comprising the steps of:

selecting the message component corresponding to the message instance, the message component defining a message field value of the message instance;

identifying in the message component a unique identifier representing a mapping between the message component and the data component, the unique identifier mapping the message field value definition to a data field definition of the data component, the data field definition defining a data instance field value;

selecting the mapped data component;

obtaining the message instance field value from the message instance corresponding to the mapped data component; and,
assigning the message instance field value to the data field value of the data instance.

19. The method as claimed in claim 18, wherein a plurality of the data field definitions of the data component is shared between the message component and the data component as represented by the mapping.

20. The method as claimed in claim 19 further comprising the step of linking the plurality of data field definitions to corresponding ones of the message fields of the message component as represented by the unique identifier.

21. The method as claimed in any one of claims 18 to 20 further comprising the step of identifying additional processing code coupled to the message component for assisting in processing of the message instance.

22. The method as claimed in claim 21, wherein the processing code specifies an address of a provisioning unit for handling the processing code.

23. The method as claimed in any one of claims 18 to 22 further comprising the step of identifying additional unmapped ones of the message field definitions in relation to the unique identifier.

24. The method as claimed in any one of claims 18 to 23, wherein the mapping is a field level mapping linking a first one of the message field definitions to a first one of the data field definitions and a second one of the message field definitions to a second one of the data field definitions.

25. The method as claimed in claim 24, wherein the number of message field definitions of the message component is less that the number of data field definitions of the mapped data component.

26. The method as claimed in claim 24 or claim 25, wherein the first message field definition is mapped by a first one of the unique identifiers to a first one of the data components and the second message field definition is mapped by a second one of the unique identifiers to a second one of the data components difference from the first data component.

27. The method as claimed in claims 24 or claim 25, wherein the first message field definition and the second message field definition are mapped to the same data component using respective unique identifiers.

28. The method as claimed in any one of claims 18 to 27, wherein the structured definition language is XML based.

29. The method as claimed in any one of claims 18 to 28, wherein the application is a wireless application and the device is a wireless device.

30. The method as claimed in any one of claims 18 to 29, wherein the unique identifier is a simple primary key.

31. The method as claimed in any one of claims 18 to 30, wherein the unique identifier is a composite key.

32. The method as claimed in any one of claims 18 to 31, wherein the message component defines a parent one of the message components coupled to a dependent one of the message components.

33. The method as claimed in claim 32 further comprising the step of overriding one of the mappings of the parent message component by respective one of the mappings of the dependent message component.

34. A method for generating a message field value of a communication message instance, the form and content of the message instance being defined by a corresponding message component of a component based application comprising a data component, the message component, a screen component and a coordinating workflow component, each of the components having respective field definitions expressed in a structured definition language, the method comprising the steps of:

selecting the data component corresponding to a data instance, the data component defining a data field value of the data instance;

identifying in the data component a unique identifier representing a mapping between the data component and the message component, the unique identifier mapping the data field value definition to a message field definition of the message component, the message field definition defining the message instance field value;

selecting the mapped message component;

obtaining the data instance field value corresponding to the message field definition of the mapped message component; and,

generating the message field value of the message instance, the message field value incorporating the data instance field value defined by in the mapped data component.

35. A method for generating a data instance field value of a data instance of a data component, the data instance being derived from a message instance, the form and content of which being defined by a corresponding message component of a component based application comprising the data component, the message component, a screen component and a coordinating workflow component, each of the components having respective field definitions expressed in a structured definition language, the method comprising the steps of:

selecting the data component corresponding to the message instance;

identifying in the message component a unique identifier representing a mapping between the message component and the data component, the unique identifier mapping the message field value definition to a data field definition of the data component, the data field definition defining the data instance field value;

selecting the mapped message component;

obtaining the message instance field value from the message instance corresponding to the data field definition of the mapped message component; and,

assigning the message instance field value to the data field value of the data instance.

36. A device for generating a message field value of a communication message instance and/or a data field value of a data instance, the form and content of the data instance and the message instance being defined by a corresponding message component and data component, respectively, of a component based application executable on the device, the component based application comprising the data component, the message component, a screen component and a coordinating workflow component, each of the components having respective field definitions expressed in a structured definition language, the device comprising:

means for selecting the component corresponding to one of the message instance and the data instance;

means for identifying at least one unique identifier in the selected component, the unique identifier representing a mapping between the selected component and the other of the data component and message component, the unique identifier mapping the message field value definition of the message component to the data field definition of the data component, the data field definition defining the data instance field value and the message field definition defining the message instance field value;

means for selecting the mapped component;

means for obtaining the instance field value corresponding to the mapped component's field definition; and,

means for generating the selected component's field value incorporating the instance field value defined by the mapped component.

37. The device as claimed in claim 36, wherein the device is arranged whereby generation of a communication message instance is based on a data instance for interaction with an application executed on the device.

38. The device as claimed in claim 36 or claim 37, wherein the device is arranged whereby generation of a data instance is based on a message instance corresponding to an application executed on the device.

39. A machine readable medium comprising program code executable on a device as claimed in any of claims 36 to 38 for implementing the method as claimed in any of claims 1 to 35.

40. A machine readable medium as claimed in claim 39, wherein said machine readable medium comprises a part of the device.