Association for Information Systems
AIS Electronic Library (AISeL)

AMCIS 2003 Proceedings

Americas Conference on Information Systems (AMCIS)

December 2003

A Reference Architecture Based on Web Components for Ubiquitous Information Systems

Cesar Alvarez
University of South Alabama

J. Harold Pardue
University of South Alabama

A Reference Architecture Based on Web Components for Ubiquitous Information Systems

Cesar E. Alvarez
University of South Alabama
cealvarezo@terr.com.co

Harold Pardue
University of South Alabama
hpardue@jaguar1.usouthal.edu

Abstract

Companies scattered worldwide, the proliferation of network-enabled devices, and advances in telecommunications have fueled a wide range of emerging behaviors that new and legacy systems must attempt to support. An important new usage behavior is ubiquitous computing, defined here as anytime/anyplace access to system functionality and data. Designing and deploying ubiquitous information systems requires a framework or architecture for conceptualizing how to structure and integrate existing and emerging technologies and end systems. The purpose of this research is to propose and describe a reference architecture called the Ubiquitous Information System Architecture. This architecture is a Web-centric synthesis of layered and service-oriented architectural styles with object-oriented patterns designed to provide a blueprint for constructing information systems that must support ubiquitous computing.

Keywords: Reference architecture, Web services, components, ubiquitous computing, software patterns

Introduction

The design, deployment, and maintenance of information systems has undergone many transformations since the early days of electronic data processing systems. Changes in technology and company's worldwide scope precipitate adaptations in user behaviors which in turn place pressure on new and legacy systems to accommodate new information demands.

The proliferation of network-enabled devices such as Personal Digital Assistants (PDA), cell phones, and pervasive computing devices; and advances in telecommunications such as Bluetooth, WI-FI, and 3G have fueled a wide range of emerging user behaviors that new and legacy systems must attempt to support. Examples of these new behaviors include worldwide teams, advertisements available anytime and in multiple locations, worldwide suppliers, international production and operation, sellers querying and updating information from offsite locations (e.g., a street corner), customers buying or ordering around the world, and 7x24x365 global logistics.

These new usage behaviors and information demands are collectively known as *ubiquitous computing*. Although ubiquitous computing can be defined in numerous ways (Weiser 1993), we define ubiquitous computing as the use of heterogeneous devices under a fixed or mobile environment to access an application at anytime and from anyplace. Organizational requirements resulting from globalization and user mobility require ubiquitous applications to exhibit a low level of coupling and ability to be distributed. There are two major requirements that characterize ubiquitous computing as a new paradigm: anytime/anyplace access to applications (ubiquitous application access) and anytime/anyplace access to data (ubiquitous data access).

Ubiquitous application access present two major challenges. How to support device independence and how to offer the same functionality and level of service expected of existing non-ubiquitous applications. Because of the increasing heterogeneity of computing devices, achieving ubiquitous application access requires a high level of device independence (Esler et al. 1999; La Porta et al. 1996). Device independence means that an information system supports equivalent functionality and level of service regardless of the accessing device, e.g., personal computer, laptop, tablets, phones, pagers, and handheld PDAs.

The range of functionality and level of service should not be contingent on the physical location or mobility of the accessing device. Users should experience no restriction on functionality or degradation of service level while moving among local, metropolitan, or wide area networks (La Porta et al. 1996) or when using fixed versus wireless networks.

The primary functionality of most information systems targets data manipulation. Because information systems are often designed to retrieve and manipulate data that are geographically and functionally distributed across multiple heterogeneous databases, achieving ubiquitous data access requires a high level of data access transparency. Transactions and queries against a data source should work independently of the physical location of the database or data persistency mechanism in which the data are stored. Not only should data be available to users anytime and anyplace, the data should be available to both legacy systems and newer multi-tiered systems.

In addition to ubiquitous access to application and data, a system designed to support ubiquitous computing must conform to standard best practices such as maintainability, compatibility and portability. A system that supports ubiquity but is difficult to maintain either because it was not designed in a modular fashion or cannot be scaled or extended, will be a sub-optimal solution.

The purpose of this research is to propose a reference architecture for the design and deployment of information systems designed to support anytime/anyplace access to information system functions and data in a ubiquitous computing environment. Conceptually, the architecture described here, the Ubiquitous Information System Architecture (UISA), is a synthesis of layered and service-oriented architectures. In practice, the UISA, is a Web-centric specification for combining existing and emerging technologies and object-oriented patterns to provide ubiquitous application and data access for information systems that must support ubiquitous computing.

The Reference Architecture

There are many software architectures designed to address the issues of ubiquitous computing such as Web Services Reference Architecture (W3C, 2002b), Vinci (Agrawal et al. 2002), BEACH (Tandler 2001), and Mobile Agent Architecture for Nomadic Computing (Duda & Perret 1997). Few, however, can be classified as reference architectures and none simultaneously specify a framework for implementing ubiquitous application access and ubiquitous data access. As a result, system architects lack a valuable tool, a reference architecture, for building system architectures for ubiquitous information systems.

Successfully designing and deploying an information system intended to support ubiquitous computing as defined here, requires a framework or blueprint for conceptualizing how to structure and integrate existing and emerging technologies and systems. A reference architecture would provide the needed framework by generalizing and extracting "the infrastructure common to the end systems and the interfaces of components that will be included in the end systems" and the "common functions and configurations" (Gallagher 2002). A reference architecture is a generic architecture that can be applied in several types of industries but that share one or more common domains. Reference architectures should provide architectural guidance, best practice information, and should act as a blueprint for designing information systems. Figure 1 illustrates this definition and the role of reference architecture in information system implementations.

The UISA can be viewed as a framework consisting of four specifications: conceptual, communications, application, and technological. The conceptual specification provides a high-level view of how to construct an information system designed to support ubiquitous application and data access, and device independence. The communications specification defines the nature of data communications among and within the tiers in the information system. The application specification lays out a model for planning the construction of and interactions among application components at each tier. The technological specification defines specific technologies to achieve ubiquity at each tier of the information system.

Conceptual Specification

Conceptually, the UISA can be classified as an N-tier architecture or call-and-return style. The UISA has five tiers: the client's application, the User/Machine Interface (U/MI) tier, the Application Logic (AL) tier, the Data Operation (DO) tier, and the Data Repository (DR) Tier (see Figure 2.).

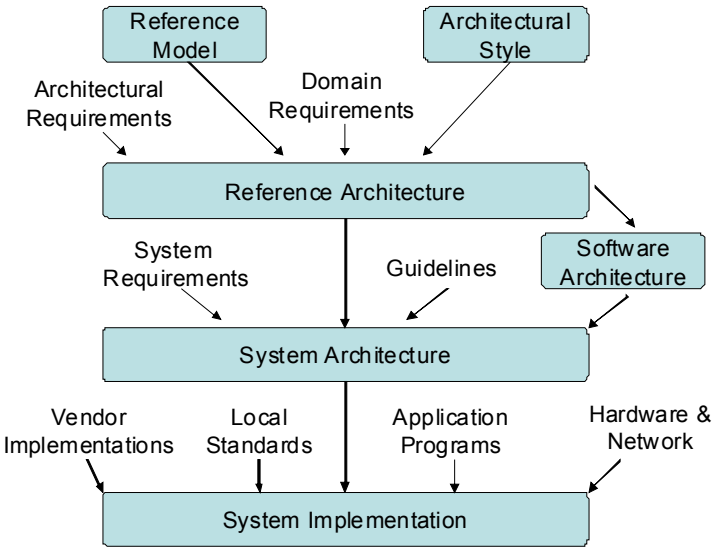

Figure 1. Reference Architecture and its Role in System Implementation (Adapted from Bass et. al. 1998 and Gallagher 2002)

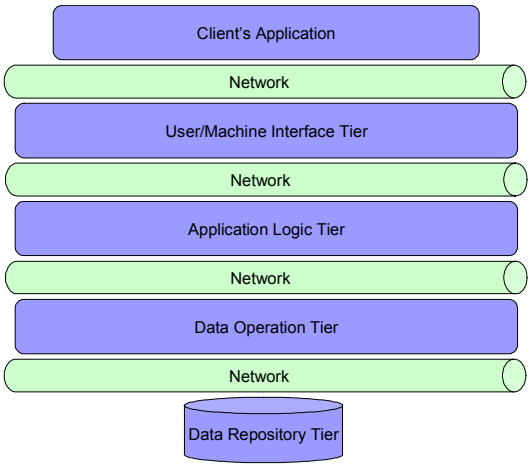

Figure 2. Conceptual Specification

Each tier is defined as a separate functional unit that encapsulates tiers lower in the architecture. Each tier is comprised of components divided into three sub-layers. The upper sub-layer (socket) is the interface with the next higher tier in the UISA and the lower sub-layer (plug) communicates with the next lower tier. The middle sub-layer performs any special processes that the upper or lower sub-layers require that is not directly involved in the communication process with the other tiers, for example generation of graphs based on a set of data or a currency conversion. Each sub-layer must be built using the concept of Web Components. A component can be defined as a one or more program packages that work as a unit to fulfill a clear function and is accessed through a well defined interface. A software component can be deployed independently (Brown & Wallnau 1998). A Web Component, for the purpose of this reference architecture, is defined as a regular component with a façade, Web-enabled component using Web Services (W3C, 2002b) and a plug that is capable of connecting other Web-enabled components. Figure 3 presents a visual representation of a Web component and its sub-layers.

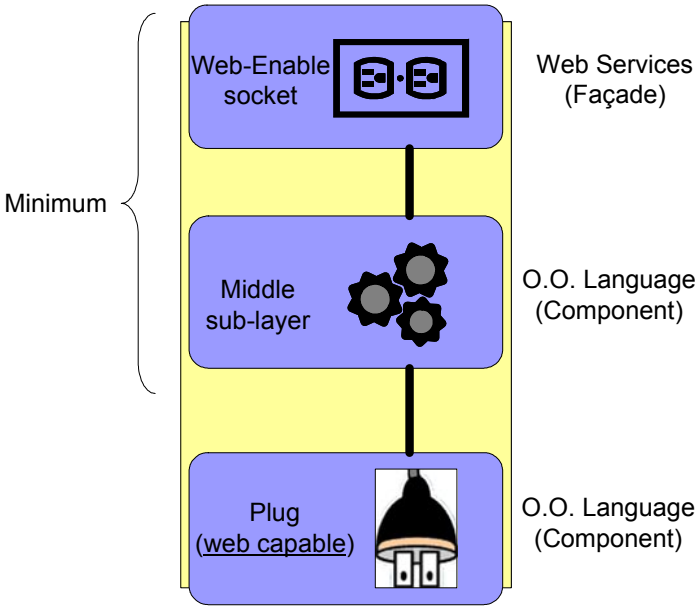

Figure 3. Web Component

The use of Web components allows the designer to materialize a service-oriented architectural (SOA) style because Web components have a Web-enable interface capable of connecting to other Web components. The SOA style allows ubiquitous access to the application and data operation tier because it proposes a design based on services that can use other services regardless the physical location (Brinkschulte & Volgelsang 1998).

The UISA uses Web technologies to create device independence and anyplace application access. Therefore at the client tier, a client application must, at a minimum, be Web enabled and capable of interpreting Web-based protocols and markup languages. Client applications physically reside on a client machine.

The User/Machine Interface tier encapsulates the Application Logic Tier and the Data Operation Tier. The upper sub-layer in this tier is a set of documents dynamically created using an adapter component that creates the document in a specific markup language. Such documents offer services to client applications. The lower sub-layer (plug) is a set of generic objects that provides a connection to the Application Logic or Data Operation tiers. The middle sub-layer is responsible for any transformations required by the upper sub-layer, for example generating a graph based on data or converting an image to another format. The primary function of the U/MI tier is the transformation of the results of the lower tiers into something usable and/or readable by a client application in the form of either presentation pages or formatted data. The U/MI tier ensures ubiquitous application access and device independence using markup languages, components, and Web communication protocols.

The Application Logic tier (AL) encapsulates the Data Operation (DO) Tier. The upper sub-layer of the AL tier provides services to the U/MI tier and the lower sub-layer provides connectivity services to the DO tier. The purpose of the AL tier is to enforce business rules, manage application security, communicate with legacy or external systems, and perform data manipulations and transformations. For performance reasons, the AL tier can be omitted if it serves merely as a bridge between the U/MI and DO tiers and adds no value. The lower sub-layer (plug) can not instantiate, inherit, or navigate U/MI, Data Repository Operations or upper sub-layer components.

The Data Operation tier (DO) encapsulates the Data Repository Tier. The DO tier enables ubiquitous data access by providing the Web-methods for interaction with the data repository. Such methods are located in the upper sub-layer (socket). The DO, with the middle sub-layer, also provides data type compatibility and data transformation to specific data types. The lower sub-layer (plug) provides the connection with the data repository. The DO tier contains data business rules or data manipulation/ transformation logic just when performance require it. The DO tier provides security over the data repository because the data repository is encapsulated; the data structures remain unknown for upper tiers or external applications and direct access to the data is not allowed. The data is accessed by specific operations offered by the DO. The data repository tier represents any

mechanism to store data. The number and type of databases or files depend on the information system requirements. The DO tier encapsulates heterogeneous data and enables easy data distribution.

Communication Specification

Component-to-component communication follows a request-answer format. To keep encapsulation and modularity, a component can only issue a request to another component in its own tier or a lower tier but not to a higher tier. All between-tier communication utilizes the network. When the AL tier functions merely as a bridge, components in the U/MI and DO tiers can communicate directly.

Component-to-component communication is a combination of simplex and half duplex connections. Between-tier requests and answers flow in a simplex fashion. Requests flow from higher to lower tiers and answers flow from lower to higher tiers. Between-tier connections permit bi-directional communication, but only one way at a time. A between-tier request precedes the subsequent answer. This arrangement is visually depicted in Figure 4.

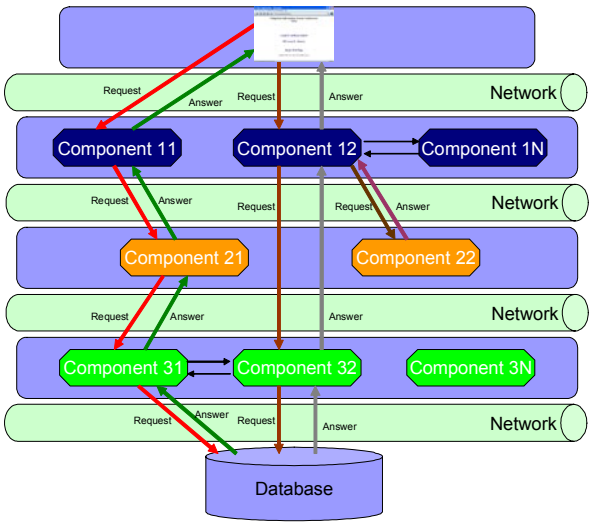

Figure 4. Layer Communication Example

Application Specification

User/Machine Interface

The upper sub-layer in the U/MI tier supports interaction with the client's application. The upper sub-layer consists of server documents (e.g. Java Server Pages – JSP, Active Server Pages – ASP, Servlets) that render the results of data requests into a Web-based markup language. The middle sub-layer provides data transformation services required by the upper sub-layer. The lower sub-layer acts as a bridge between the server pages and the AL and DO tiers. Figure 5 illustrates the organization of the three sub-layers.

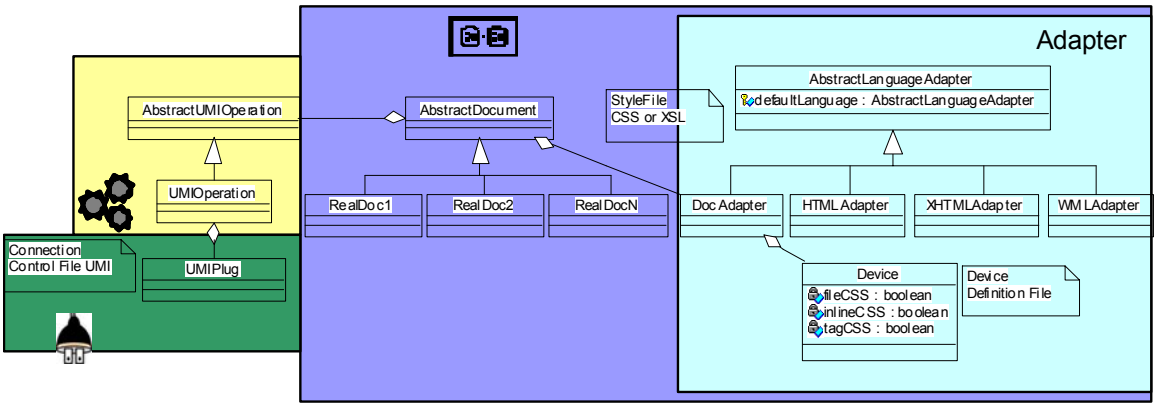

Figure 5. User/Machine Interface

A purpose of the U/MI tier is to provide device independence. To date, most solutions have been based on providing scaling, transducing, and transformation mechanisms and manual authoring (Trevor et al. 2001; Chen 2001; Huang & Sundaresan 2000; Kaasinen et al. 2000; Kirda 2001). These mechanisms rely on the existence of an XML or HTML page that is used to create a new document in a new language. These approaches present two problems; first, they require the additional task (manual or automatic) of recreating the document; second, there is a creation of duplicate documents to support the same functionality. The UISA relies on the creation of an adaptor component that is capable of writing a document object into any defined markup language the moment the document is invoked. By picking specific methods from the adaptor component, document objects can be created dynamically through semantic conversion and author intention. Current mechanisms do no consider the meaning and functionality of the pages (author intention), so converted documents can be difficult to understand and navigate (Chen et al. 2001; Huang & Sundaresan 2000).

To achieve device independence, server documents should not contain markup languages or inline style tags (e.g. color, size, or bold) or commands. Formatting is defined by constructing Cascading Style Sheets (e.g. CSS, CSS2) or Extensible Style Sheets (XSL); a style file is created for each type of device that will access the application. To ensure a common language and device independency, the basic tags or commands of each language are encapsulated using an adapter object that adapts each language. In this way, the server document uses the commands defined in the adapter object and does not use the tags or commands of the native language directly. The UISA allows any number of server documents.

A device definition file is used to define which language object is used based on the type of client software (e.g. browser). This information must be retrieved at the beginning of a session. The device definition file should be a XML file with entries that relate the type of client software and the language that must be used. An abstract page class is created to implement the bridge pattern (Gamma et al. 1995) with the middle sub-layer (operations) and the proxy pattern (Gamma et al. 1995) with languages classes as depicted in Figure 4.

It is important to note that a RealDoc object replaces the AbstractDocument when the server page language does not support inheritance. This is important because no abstract class will exists to create the real pages, so developers must declare manually the adapter and the middle sub-layer classes. Any requirements from the upper sub-layer to a lower tier must be developed using the middle sub-layer components in the U/MI tier.

The lower sub-layer components (plugs) issue requests, receive answers from the AL and DO tiers, and provide the received information to the middle sub-layer. Plugs can be built using any component language. Similar to the upper sub-layer, a control file is used to maintain configuration information. The control file identifies the server or servers (Web services) to connect to use the AL or DO tier and can be used to define a pool of servers to connect to in order to find either the fastest server or an available server. There is no theoretical limit to the number of UMIOperation and UMIPlug classes.

Application Logic Tier

At the AL tier, the upper sub-layer offers services (answers data requests) to the U/MI tier. The lower sub-layer communicates with the DO tier. Web services provided by the upper sub-layer are implemented as a Façade pattern (Gamma et al. 1995). The Façade reduces the number of objects that the upper tier needs to interface with, ensures a unique interface with the upper layers, and reduces coupling. The façade is a group of methods offered by the Web services in this layer (see Figure 6).

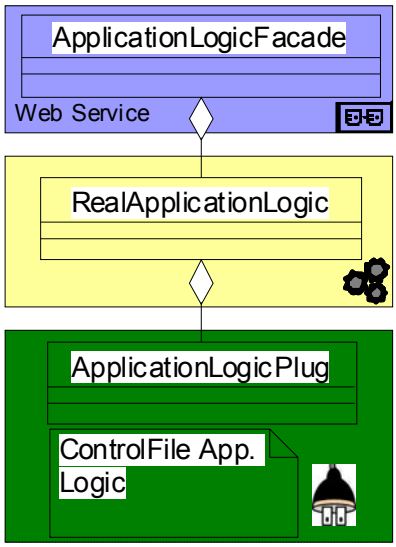

Figure 6. Application Logic

The middle sub-layer is a group of components to perform operations and calculations. The lower sub-layer (plug) is a group of components to interact with the DO Tier. As in the previous tier, a control file is used to define the server (Web services) to be connected to and can be used to define a pool of servers to connect to in order to find either the fastest server or an available server.

Data Operation Tier

The DO tier supplies results or answers to the AL and U/MI tiers via Web services in the upper sub-layer (socket). These Web services must be implemented according to the Façade pattern to reduce the number of objects that the upper tier need to interface with, to ensure a unique interface with the upper tiers, and to reduce coupling (see Figure 7). The use of Web services in this tier ensures ubiquitous data access.

The U/MI or AL tier communicates with DO objects by calling methods in the socket (DO upper sub-layer – Web services). The U/MI or AL tier never accesses directly objects in DO middle or lower sub-layer. The upper or middle sub-layer do not access the Data Repository directly. DO lower sub-layer objects retain no knowledge (reference) of the upper sub-layer (socket), middle sub-layers, or upper tiers.

The lower sub-layer is a group of objects that access the Data Repository. These objects must be implemented using the pattern shown in figure 6. The implementation of this pattern ensures maintainability and flexibility or extensibility. Access is accomplished via SQL instructions, stored procedures invocations, read or write commands into a file either using JDBC, ODBC, ADO commands or data manipulation instructions in the implementation language.

Calculations or transformations are allowed in this tier (middle sub-layer) only for performance considerations. Data formatting is allowed in this tier (middle sub-layer) to ensure data type compatibility, so the data are formatting as need. For instance, if there are two data sources to retrieve salaries, one text file and the other one a database, this layer will format the values to decimal, so the upper layers will retrieve a decimal values all the time, no matter the source of the data. No automatic mechanisms are offered to perform such formatting in this architecture.

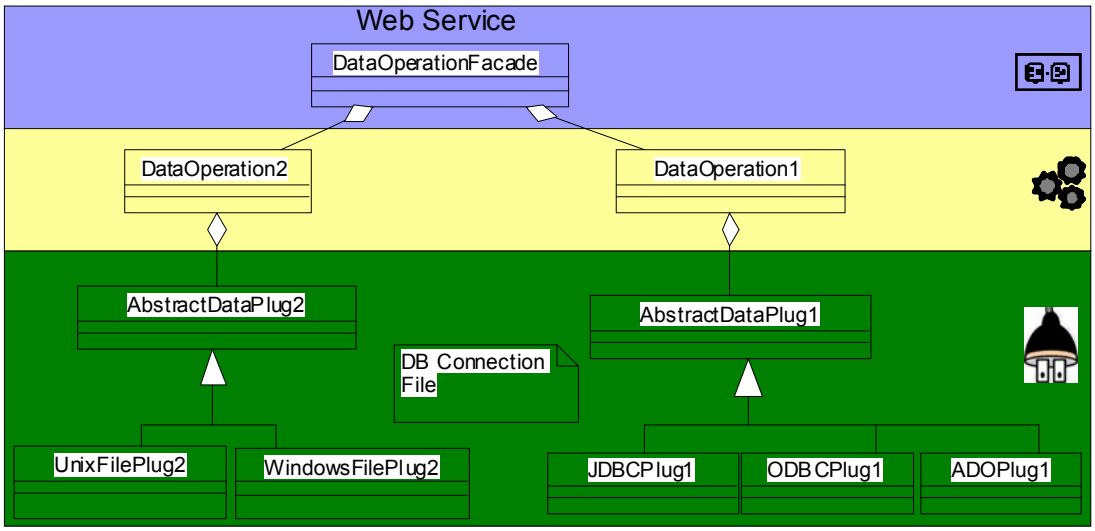

Figure 7. Data Operation

Data Repository

The UISA does not place restrictions on data types in the data repository. Data can be stored as either files or databases allowing database heterogeneity and integration with legacy systems. Database heterogeneity is achieved through the encapsulation implemented at the DO tier. Stored procedures can be implemented in the database to improve performance.

Technological Specification

An information system must be implemented using specific technologies. Given the component Web-centric focus of the UISA, some technologies are prescribed, others are recommended. The specific technologies at each tier are depicted in Table 1. Because the Web services reference architecture is based in SOAP and XML standards that are supported in any platform, the use of Web services supports interoperability compatibility. Object oriented languages allows conformance to the UISA requirements because they allows easily create the patterns required in the architecture.

As a Web-centric architecture, the UISA specifies that the client application have markup language capabilities.

Conclusions

Current work in IS architectures is not accomplishing the ubiquitous application access and ubiquitous data access simultaneously, which is required in ubiquitous information system and few can be classified as reference architectures. As a result, system architects lack a valuable tool, a reference architecture, when designing system architectures for ubiquitous Information Systems. The proposed reference architecture provides ubiquitous application access by implementing a User/Machine interface tier that contains a Web-enabled pattern to transform markup languages capable of connecting to lower tiers in the architecture across Web environments. Ubiquitous data access is provided through a combination of layered and service-oriented architectural styles that allows an application to access data worldwide. By using Web technologies and encapsulating data in a layer using Web services as a façade, the UISA avoids direct unsecured access to data. The use of Web services in the UISA overcomes two limitations in CORBA, DCOM, and RMI: the ability to access distributed applications and the differences in programming languages when developing ubiquitous applications.

Table 1. UISA Specifications

Conceptual	Technological	Communication	Application
Client Application Tier	Markup Language	Hypertext Transfer Protocol (HTTP)	Web enabled device; Markup Language compatibility
Network	Wireless/Wire Protocol + TCP/IP required		
User/Machine Interface Tier	Markup Language and Style Sheets; Server Side & Component O.O. Language required	Simple Object Access Protocol (SOAP) and HTTP	Interface Adapter (Device configuration file); Interface Operations Methods; Plug (Control file)
Network	TCP/IP required		
Application Logic Tier	Web Services required; Component Object Oriented Language	SOAP and HTTP	Logic Methods Socket; Logic Operation Methods; Plug (Control file)
Network	TCP/IP required		
Data Operation Tier	Web Services required; Component O. O. Language required; JDBC, ODBC, ADODB	SOAP and HTTP	Database Socket; Database Methods; Plug (Database Control file)
Network	Any Wire Protocol		
Data Repository Tier	File, Relational and/or Object Oriented Database, XML	SQL, OQL, File Operations	XML, Stored Procedures, Tables and Views, Files

The growth of the Internet has demonstrated that Web technologies such as HTTP and markup languages are flexible enough to provide anytime/anyplace access to applications. As a result, the use of HTTP and markup languages in the U/MI tier ensures ubiquitous application access. Device independence is reached implementing the language adapter pattern and the use of it when creating the documents that interact with the client. Ubiquitous application access and ubiquitous data access is reached by implementing the façade pattern using Web services. The façade, using Web services, creates a layer that is accessible worldwide. Maintainability requirements are reached through the use of components; the system can be seen as a collection of small components, each with specific functions. Compatibility is reached because of the use of well-known patterns, and Web standards. Finally, Portability is reached because the architecture implements a layered architecture using Web services that allows moving the tiers independently without impact the other tiers.

Many organizations can benefit through the use of this reference architecture when developing information system oriented to ubiquitous computing. Organizations with global strategies that require data base centralization to have a global vision of the organization but at the same time require a distributed implementation can take advantage of this reference architecture to support ubiquitous data access and keep control over a global repository that is accessible worldwide. Companies that require device independence access to enhance their business can use the UISA to allow multiple devices with only one application. Many organizations can benefit from the "build once use in any-device" approach. The use of Web services in the UISA can reduce development times and cost and reduce the integration efforts.

References

Agrawal, R., Bayardo, R. J. Jr., Gruhl, D., and Papadimitriou, S. "Vinci: a Service-Oriented Architecture for Rapid Development of Web Applications," *Computer Networks,* (39), 2002, pp. 523-539.

Bass, L., Clements, P., and Kazman, R. *Software Architecture in Practice*. Reading, Mass: Addison-Wesley, 1998.

Brinkschulte, U. and Volgelsang, H. "A Distributed Scaleable Real-time Add-on for Operating Systems," In *4th IEEE Real-time Technology and Application Symposium, Work in Progress session* Denver, 1998.

Brown, A. W. and Wallnau, K. C. The Current State of CBSE. *IEEE Software*, (15), 1998, pp. 37-46.

Chen, J., Zhou, B., Shi, J., Zhang, H., and Fengwu, Q. "Function-Based Object Model towards Website Adaptation," In *Tenth International Conference on World Wide Web,* Hong Kong: ACM Press, 2001, pp. 587-596.

Duda, A. and Perret, S.. "Mobile Agent Architecture for Nomadic Computing." In *International Conference on Computer Communications* Cannes, France, 1997.

Esler, M., Hightower, J., Anderson, T., and Borriello, G. Next Century Challenges: Data-Centric Networking for Invisible Computing: The Portolano Project at the University of Washington. In *Fifth Annual ACM/IEEE International Conference on Mobile Computing and networking,* Seattle, Washington, United States, 1999, pp. 256-262.

Gallagher, B. P. *Using the Architecture Tradeoff Analysis Method℠ to Evaluate a Reference Architecture: A Case Study* (Rep. No. CMU/SEI-2000-TN-007). Carnegie Mellon University, 2002.

Gamma, E., Helm, R., Johnson, R., and Vlissides, J. *Design Patterns: Elements of Reusable Object-Oriented Software*. Reading, MA: Addison-Wesley, 1995.

Huang, A. W. & Sundaresan, N., "Aurora: A Conceptual Model for Web-Content Adaptation to Support the Universal Usability of Web Services." In *Conference on Universal Usability,* Arlington VA, USA, 2000, pp. 124-131.

Kaasinen, E., Aaltonen, M., Kolari, J., Melakoski, S., and Laakko, T.Two Approaches to Bringing Internet Services to WAP Devices. *Computer Networks, 33*, 2000, pp. 231-246.

Kirda, E., "Web Engineering Device Independent Web Services," In *23rd International Conference on Software engineering* Washington, DC, USA, 2001.

La Porta, T. F., Sabnani, K. K., and Gitlin, R. D., Challenges for Nomadic Computing: Mobility Management and Wireless Communications. *Mobile Networks and Applications, 1,* 1996, pp. 3-16.

Tandler, P., "Software Infrastructure for Ubiquitous Computing Environments: Supporting Synchronous Collaboration with Heterogeneous Devices," In *Ubicomp 2001,* pp. 96-115.

Trevor, J., Huang, A. C., Schilit, B. N., and Koh, T. K. "From Desktop to Phonetop: a UI for Web Interaction on Very Small Devices," In *14th Annual ACM Symposium on User Interface Software and Technology,* Orlando, Florida, 2001, pp. 121-130.

Weiser, M. "Some Computer Science Issues in Ubiquitous Computing," *Communications of the ACM*, 36, 1993 pp. 75-84.

W3C (2002b). Web Service Architecture. World Wide Web Consortium [On-line]. Available: **http://www.w3.org/TR/ws-arch/**.