Application Programming Interface (API) And Management of Web-Based Accounting Information System (AIS): Security of Transaction Processing System, General Ledger and Financial Reporting System

Efuntade, Olubunmi Omotayo, PhD
Federal University Oye-Ekiti, Ekiti State, Nigeria.
Email: bunmiefuntade@yahoo.com

Efuntade, Alani Olusegun, FCIB, FCA
Federal University Oye-Ekiti, Ekiti State, Nigeria.
Email: alaniefuntadee@yahoo.com

DOI: 10.56201/jafm.v9.no6.2023.pg1.18

Abstract
The paper present exploratory research on application programming interface (API), management of accounting information system, security of transaction processing system, general ledger and financial reporting system. An application programming interface, or API, enables businesses to make the data and functionality of their applications available to external third-party developers, commercial partners, and internal departments inside their own organizations. Using a defined interface enables services and products to interact with one another and benefit from each other's information and features. The interface is used by developers to interact with other software and services; they are not required to understand how an API is developed, and neither are the software's end-users. In short, an API is a contract between pieces of applications serving the main software once integrated into the source code of the main application. These pieces of applications communicate with each other in the language they both understand and over a network if needed. As the name implies, APIs serve as interfaces between programs. The interface is usually between a software developer and software developer's application. Basically, the API allows one software to access some functionalities of another software. In its development, information systems develop and run well using a web-base so that it can be reached via an online computer network. Web applications have become complex and important for many companies, especially when combined with areas such as AIS. All AIS transaction data will be stored in the database management system and will be used if there is a query data request. Based on the IFRS (International Financial Reporting Standard) document, a proper accounting system must at least include the following aspects (IFRS Standard Requirements): Financial position statement, cash flows statement. Web-based AIS application for recording, grouping, processing, and presenting transaction data regarding finance. The data then becomes information that can be used as material for making a decision. Web-Based AIS has an admin page to manage every important data, both master data and data in the form of transactions. It also provides an application programming interface (API) web service, which is access that can be used by external developers who require data access. The

available API serves as an intermediary between the main data provider software system with external developers, where developers develop AIS applications with different platforms such as web-based or mobile-based.

Keywords: *Application programming interface (API), management of accounting information system, security of transaction processing system, general ledger and financial reporting system*
Jel Code: *M40, M41*

1.0 Introduction

An application programming interface (API) is a set of functions that allow developers to access the features and data of the software. An API extends functionalities and facilitates integration with third-party software. API banking is becoming a critical step in helping customers and business partners innovate for new technologies (Adner & Kapoor, 2010).

Information and communication technologies (ICTs) have long been studied as a form of business investment. Application programming interfaces, or APIs, are a newly popular type of ICT. Combining proprietary data from an API management and data, we study, for the time, the impact of APIs on pro and growth. We also examine the contributions of internal versus external developers. A major reason for recent interest in APIs is their role as the foundation of digital ecosystems. APIs make it easy for individuals to write programs that communicate with online services and shared databases.

Although seemingly banal infrastructure, APIs are essential for making the power of systems such as Google Maps, eBay, Amazon, and Twitter available to independent developers. They mediate economic transactions. Their value is not fully determined by the actions of their creators or the preferences of their consumers also critical are the strategic choices of third parties who connect across systems and reuse components in unanticipated innovations (Adner & Kapoor, 2010).

Making these tools and data available to outsiders can be a win-win. Independent developers gain new opportunities to increase agility and speed of deployment, while the API providing gains complementary added value. In the best case scenario for the platform an API becomes the basis of a thriving ecosystem, with exponentially growing revenues and low marginal costs.

The successful design and high performance of APIs are becoming key points for a variety of applications. The performance of APIs can increase when criteria such as programming methods and programming languages and/or environments are selected appropriately. The applications are often built on application programming interfaces (APIs) based on web technologies such as HTTP (Hyper Text Transfer Protocol), REST (Representational State Transfer), SOAP (Simple Object Access Protocol) and JSON (JavaScript Object Notation), which can be used for other operations. This feature enables the development of increasingly complex third -party applications, which repeatedly use existing content and services. These applications, which convert content from various applications into an integrated experience can be created by developers who are not directly associated with the original developers of reuse services (Kemer & Samli, 2019). An application programming interface (API) is a set of functions that allow developers to access the features and data of the software. An API extends functionalities and facilitates integration with third -party software. Well -designed and well - documented APIs help to ensure developers can extend the software coherently and consistently. An advantage of using an API is the connection to the core of the software without the need to understand the underlying structure. It also reduces the need to rewrite code for common functions and t hereby improves productivity as developers can devote more time and effort to extending software functionalities for specific purposes. (Chan et al., 2019)

Application Program Interface gains attraction in the financial markets across the world. Countries like USA, UK, Canada, Singapore and Japan have already adopted the technology at large. APIs are benefiting fintechs in many ways especially with more integrations. Fintechs can integrate existing assets from banks and financ ial institutions to build on top of existing infrastructure to increase the pace of the process. If companies build your own API structure internally, they will be able to deliver products and services much more faster and efficiently on multiple channels like mobile phones, IoT, applications etc and at the same time open it for other partners. Hence, APIs can integrate others to go faster by building on existing infrastructures, they can be built internally to operate on multiple channels and it can enable third parties to build on top of existing infrastructure (Adner & Kapoor, 2010). Currently, APIs are the alternative to FinTech companies that, for some years now, have displaced banks as financial institutions. The digital development provided by APIs allows growing beyond a structure, which is of great help when you want to reach new customers and gain the loyalty of those already established (Adner & Kapoor, 2010).

An application programming interface (API) is a programming style that exposes the inputs, outputs, operations and data types of a piece of code in a standard documented way. In the context of websites, an API is a definition of the generic format of URLs that will return data from a site, as well as information about the structure of the data returned by each URL.

This makes it easy for other software developers to interact with a sites API, as they know how to format requests for information and how to process what is returned (Adner & Kapoor, 2010). The concept of the API originated in the eighties in the software development community. It was used then, and is still used, as a means of exchanging data across organizational boundaries. With the advent of the Web, and particularly since the turn of the millennium, it has also been used to give public access to collection resources, whether for analysis or to allow the development of interface alternatives. Some popular APIs of this kind are available through Flickr, Twitter, and Google Maps. In terms of technology, the most common way of creating a Web-based API is using Representational State Transfer (REST), where the underlying resources are accessed through a URL that returns XML or JSON rather than HTML. Discussions of APIs tend to focus on methods for measuring the usability and quality of both the software and the documentation. (Adner & Kapoor, 2010).

APIs allow technologies to exchange with each other, and the data to flow in real time. They promote innovation as other technologies have allowed in the past, but their impact seems to be predominant today. To the point that banking groups prefer to adopt them rather than being confronted with Data Scraping; indeed, APIs control access to data by allowing access to only certain fields in databases. They are perceived as a lesser evil in the face of wild Data Scraping.

But again, no rearguard action: the likely evolution is that it is the consumer who will decide to give access to some of its data and that this movement is irreversible. (Girard, 2019)

API is the virtual interface between two interworking software functions, such as word processor or spreadsheet. API is the software that is used to support system level integration of multiple commercial-off-the-shelf (COTS) software products or newly-developed software into existing or new applications". A cloud API is basically used to integrate applications in order
to provide inter-cloud compatibility and improving cloud usage experience. They are broadly categorized into two: in-process API and remote API. In-process APIs are used on a regular basis in a typical infrastructure based IT environment (Adner & Kapoor, 2010).
Remote APIs are used to develop cross-border, bridging applications. Web services APIs include SOAP and REST, while remote calls include SUN RPC and JAVA RMI; and application dependent protocol include FTP and SNMP. These APIs communicate based on data structures like JSON and XML and make use of GET, PUT, POST, DELETE request. Cloud applications usually utilize remote APIs.

An API, or application programming interface, is a set of routines, protocols, and tools that standardizes building software applications compatible with an associated program or database. APIs are code. They are also contracts (Jacobson et al., 2011). They govern the type and format of calls, or communications, that any given application can make of another associated program. The associated program is agnostic about the source of the call, and the app need not know anything about the internal workings of the associated program.

APIs or the dual virtues of practical modular design and precise metering. Modular architecture, enabled by APIs, allows designers to independently create, subdivide, modify, and remove components without a other parts of a larger system. It also facilitates partitioning of decision rights (Tiwana et al., 2010) Modularity combines the advantages of standardization typically associated with high volume processes with the advantages of customization typically associated with bespoke processes (Baldwin & Clark, 2000). APIs also enable precise metering of access permissions to these key resources. Metered access permissions ensure that anyone and anything that consumes system resources adheres to technical and economic policies designed to ensure system health. As architecture, APIs provide infrastructure for building platforms. As regulators, APIs partition decision rights and provide scalable mechanisms for governing behavior. These dual roles architecture and governance- provide the foundations for building platforms (Baldwin & Clark, 2000).

APIs come in two main forms: open and closed. This distinction depends on whether access is outward facing (open) or inward facing (closed). This decision should impact the on several levels. A recent survey argues that open innovation occurs across inter-and intra- levels(Baldwin & Clark, 2000).Our research examines openness at a grain level, access to individual byte transfers, facilitating analysis across these levels. We can classify API access as B2B (upstream), internal, and B2C (downstream). Thus, performance di can be analyzed for internal or company-specific" platforms versus /industry-wide platforms Gawer and Cusumano (2014) or even the upstream and downstream ends of the value chain (Adner & Kapoor, 2010).

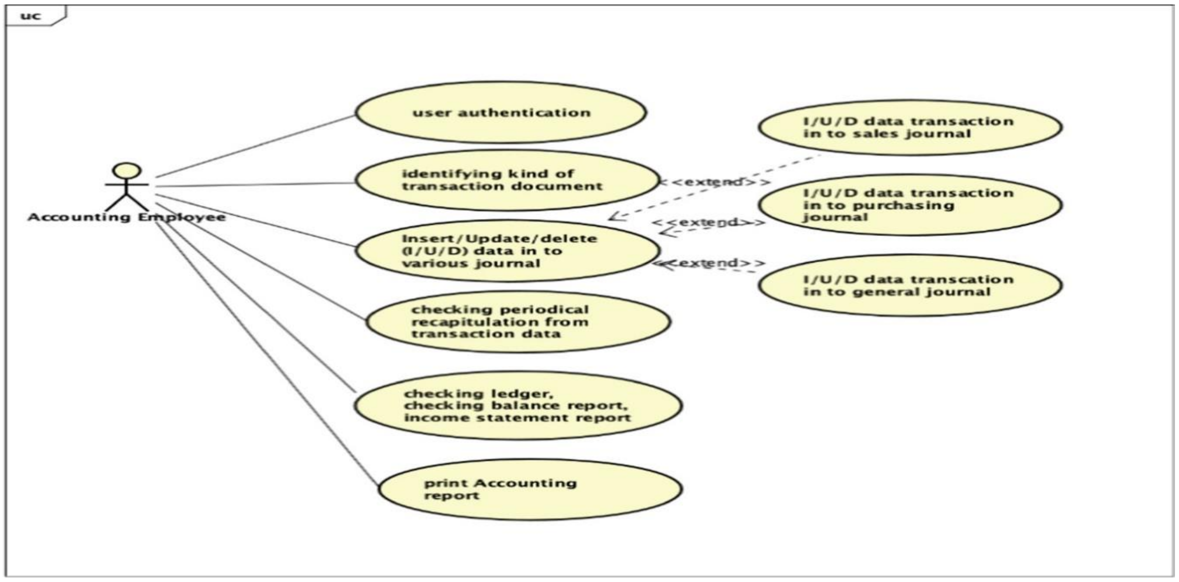
A closed API is only accessible to individuals working for the or close associates. The bulk of APIs are closed (Jacobson et al., 2011). This proprietary usage is designed to enhance internal agility. A that includes well designed APIs in its internal software will likely it easier to modify or create new programs using the same information. Verizon has employees use an app in the course of on-site service calls and to activate new cable lines. If the wants to enable employees to perform the same tasks on a new device, it will only need to create a new app to talk to the API. Similarly, the company could easily add additional functionality to a current app by having it communicate with a new API. An app that communicates to multiple APIs is called a mashup (Adner & Kapoor, 2010).

Modern-day software and application development use various API technologies. Some popular API Technologies are Web API, REST Architectural Style, SOAP, RPC, etc. Web API is a term used to describe an application programming interface (API) for a web server or web browser. It is a web development principle that frequently only pertains to the client side of online applications. In a client-server architecture, REST (Representational State Transfer) is a kind of software architecture that defines a consistent interface between physically distinct components, frequently through the Internet. The creation, implementation, and deployment of APIs may be done with greater freedom using the REST architecture, which separates the front and back ends of the API (Huckman et al., 2012).

The high-level communications paradigm utilized by the operating system is provided by the Remote Procedure Call (RPC) protocol. User Datagram Protocol (UDP) or Transmission Control Protocol/Internet Protocol (TCP/IP) are two examples of low-level transport protocols that RPC makes use of to send message data between interacting applications. RPC develops a logical client-to-server communications framework to support network applications. Simple Object Access Protocol (SOAP) is a messaging standard defined by the World Wide Web Consortium and its member editors. To enforce the structure of its payloads, SOAP employs an XML data format to specify both its request and response messages. SOAP serves as an interface for both public and private Application Programming Interfaces (APIs). Despite being more common in large businesses, SOAP APIs are created and used by companies of all sizes.

AIS is a system that can collect, record, store, and process data to maintain its accounting system by utilizing technology. These processes include buying, selling, and other business financial processes. The essential mission of AIS is to allocate a quantitative value of the past, current and future business events. In addition, AIS is also functioned as a data collector and information provider for decision makers such as investors, creditors, and managers of a company to make decisions using documented, repeatable, understandable and feasible processes and procedures Elsharif (2019).

Manual and computerized systems are approaches used to produce accounting information. Approaches or tools used to produce accounting information include manual systems and computerized systems. To ensure its consistency, the system was developed


uc
Accounting Employee
user authentication
identifying kind of transaction document
Insert/Update/delete (I/U/D) data in to various journal
checking periodical recapitulation from transaction data
checking ledger, checking balance report, income statement report
print Accounting report
I/U/D data transaction in to sales journal
I/U/D data transaction in to purchasing journal
I/U/D data transcation in to general journal
<<extend>>
<<extend>>
<<extend>>

Accounting Information System

Source: Elsharif (2019)

Several types of use of accounting information include: First, preparation of external reports to produce specific financial reports that can meet the information needs of stakeholders including investors, creditors, offices, taxes, government agencies and others. Second, support for handling routine operational activities throughout the company's operating cycle. Third, supporting non-routine decision-making processes at all levels of the organization. For example, to determine the performance of products sold, existing distributors and loyal customers. Fourth, budget planning and control. Fifth, internal control facilities include policies, procedures and information systems used to protect company assets from loss or misuse and maintain the accuracy of financial data Elsharif (2019).

In its development, information systems develop and run well using a web-base so that it can be reached via an online computer network. Web applications have become complex and important for many companies, especially when combined with areas such as AIS. All AIS transaction data will be stored in the database management system and will be used if there is a query data request.

Based on the IFRS (International Financial Reporting Standard) document, a proper accounting system must at least include the following aspects (IFRS Standard Requirements): Financial position statement, cash flows statement,

Financial position statement: This is a balance sheet. IFRS affects the way in which components of the balance sheet are reported. Comprehensive profit and loss statement: This can be in the form of a single statement or separated into a profit and loss statement and other income statements, including fixed assets.

Cash flows statement: this report summarizes the company's financial transactions for a certain period, separating cash flows into operating, investing and financing. Change in equity statement.

It is also known as a statement of retained earnings, it documents the changes in a company's earnings or earnings for a specific financial period.

Web-based AIS application for recording, grouping, processing, and presenting transaction data regarding finance. The data then becomes information that can be used as material for making a decision. Web-Based AIS has an admin page to manage every important data, both master data and data in the form of transactions. It also provides an application programming interface (API) web service, which is access that can be used by external developers who require data access so that they can connect with aksyaa.com. The available API serves as an intermediary between the main data provider software system with external developers, where developers develop AIS applications with different platforms such as web-based or mobile-based.

The AIS provides a private-API type, where access is not given openly so that external developers who want to have access to the API must get authorized from the AIS. This is related to the authentication and authorization system of AIS, where to perform proper and correct authentication and authorization, external parties are required to send a valid token. Valid tokens can only be obtained from the service provider. The token given is in the form of a static token which is only created once and can be used repeatedly without any new token creation process so that the token is highly confidential.

2.0 API Security:

API security protects data transferred through APIs, often between clients and servers linked over public networks. Organizations use APIs to connect services and convey data. Using Transport Layer Security (TLS) to implement data encryption and digital signatures is one of the many approaches to securing an API. Transport Layer Security safeguards data sent between you and a server and the privacy of your internet connection. As a means of authentication, OAuth is used in combination with Open ID as a technique to secure APIs. OAuth, which provides authorization services, may also be used to control who has access to which enterprise resources (Red Hat, Inc., 2019). To authenticate users and devices using OAuth 2.0, OpenID may use a third-party authentication system. This combination is currently one of the more secure authentication-authorization options on the market. If an API is breached, exposed, or hacked, it can disclose personal, financial, or other sensitive data, causing immense mayhem (Red Hat, Inc., 2019).

2.1 Authentication

Security involves policies and policy implementations that keep a user's data from being accessed by unauthorized individuals, and protect a user's transactions from being altered or imitated. Security mechanisms are used to enforce the privacy policies chosen by the user. Privacy here refers to what data a user will let other entities view, and under what conditions. Regardless of the strength of the security and privacy policies, this device will not be useful if its users do not trust it, or the transactions it performs. Thus, trust needs to be established so that users will feel as comfortable using this device to perform online transactions as they feel when using an automated teller machine at a bank.

Security and privacy policies play a large part in gaining a user's trust. As such, not only should

the policies be complete, but they should also be customizable to each individual. Thus, a user should be able customize their security policy in order to accommodate exceptions to the general rules. For example, a user might state that their prescription drug history can be released only to a
pharmacy with whom a transaction is being negotiated.

Additionally, the security and authentication mechanisms for this device need to take into consideration that the target population is functionally illiterate. As such, the end user should not be expected to perform complicated interactions.

Thus the system must be simple, yet protect the user against exploitation by others.

In order to access the device, the user must authenticate themselves. Authentication can be broken down into four components: what you have (e.g. tokens such as smart cards); what you know (e.g. passwords); what you are (e.g. biometrics such as fingerprints), and where you are.

Given the target population, it is not reasonable to expect passwords to be used. Location can be used in some circumstances, but is not always a unique identifier. While the device can itself be used as a token, this is not secure in that it can easily be stolen. This leaves biometric techniques as the preferred method of authentication.

Multiple biometric techniques are available, such as fingerprinting, retinal scans, speaker recognition, iris scans, facial recognition, and signatures, where each of these techniques has its own advantages and disadvantages. Finger-printing, for example, involves matching a user's fingerprint to one stored in a database for authentication. However, they have been found to produce variable results as they are subject to noisy or useless data. The image quality for the fingerprint plays an important role, with poor quality increasing error rates.

Additionally, some fingerprints have ridges which are too fine to be captured well by a photograph, while others are obscured by cuts or scars. The advantage of fingerprints is that they are a non-invasive technique with which most people are comfortable.

Another non-invasive authentication method is facial recognition. The most commonly employed technique for this is eigen faces, which are a derivation of eigenvectors applied to images of faces. An advantage to this method is that it mimics our own recognition processes, thus people are already accustomed to this procedure. Yet, this method is not very robust and can encounter problems based on spatial orientation and illumination properties.

A final non-invasive technique involves voice pattern recognition. Similar to the previous techniques, this method is not entirely secure. There are two possible approaches to this method: (1) having a user say a predetermined phrase, or (2) using a text-independent method such as conversational speech. While the second option is more difficult to implement it is clearly the preferred method. However, voice recognition suffers from many variables such as changes with age, health or even mood.

The more invasive techniques include retinal scans and iris recognition. Both of these methods require that an image be taken of a user's eye. The patterns in the iris or retina are then matched against a database to authenticate the user. While these methods are potentially the most invasive, they are also the most secure. It is anticipated that as biometric measurement techniques are perfected, and as we learn more about the biometrics themselves, that ultimately a user's DNA will be used for authentication.

2.2 Security Mechanisms

Web-Based ASI requires implementation of security mechanisms due to the sharing of a large amount of data among distributed agents and the presence of mobile code. The required security services range from authentication to integrity and also include non-repudiation and access control. Two common elements in these services are cryptographic techniques and intrusion detection.

Public key cryptography has been used for checking the integrity of software obtained through insecure channels. When a piece of electronically signed software is submitted for execution on another node of an insecure network, there are two implications: Software that migrates over the network may not be changed by active attackers without being noticed at the receiving end and the entity that has prepared and sent the code over the network will not later be able to deny this fact.

In anomaly detection that aims at avoiding illegal access or operations, the current behavior is observed and compared to see if it corresponds to some past behavior. The rules used in the identification and detection can either be generated by experts or automatically.

To address the security issues in Web-based ASI, we have two security agents at each host: a message security agent (MSA) and a controller security agent (CSA). The MSA deals with services related to the exchange of messages, such as electronic signature and cryptography, and the CSA provides services to check adequate use of resources by detecting anomalies. The security agents perform the following actions: digital signature of the message. The MSA-send uses the sender's private key to sign the message, and then sends to the receiver the certified public key and signature check. The MSA-receive uses the sender's public key to check the electronic signature.

– Sender reliability check. The MSA-receive accesses a list of all the nodes considered to be unreliable (supplied by a separate authority) and if the sender is on the list, it refuses to execute the code received.

– Control and execution of code. The CSA execute the code received after having checked for any anomalies.

The received code might be unreliable due to two reasons: (1) the sender may have changed its policies and may have planned attacks; (2) the software may contain bugs that endanger local security. The CSA adopts rule-based anomaly

detection and intrusion identification mechanisms to provide access control and protect the system's integrity.

A Multimedia Session Manager Service for the Collaborative Browsing System Browsing the Web is usually a lonely task. People visit sites, collect information and are not aware of other people looking at the same material, people with whom they could exchange experiences and ideas about the subject they are looking at. A Collaborative Browsing System (CoBrow) has been developed to bring awareness to the World Wide Web. CoBrow users can see other people looking at the same Web pages they are browsing. The Multimedia Session Manager (MSM) is one of the components that currently form CoBrow. It is responsible for initiating and managing multimedia sessions required by CoBrow users.

This paper aims to describe the work performed by MSM in managing those users and their sessions. MSM is a CoBrow component that gives the users of the system the ability to communicate with each other. MSM provides the necessary support within the CoBrow environment for users to join sessions and automatically open the appropriate external tool set.The MSM software has been designed to allow different communication tools to be easily integrated within the CoBrow system.

As different people use different operating systems and software, support for different conferencing tools is required. MSM is specifically designed to be extensible by allowing additional tools and services to be defined. Modules known as drivers are implemented as self-contained code blocks, using standard library routines and following a defined structure, for each new session type.

Implemented in Java, MSM provides an ideal topology for installing a driver on the fly without the need to alter any existing code or drivers. Java's ability to load custom classes is exploited for services offered by the server and by drivers handling the various session types. In order to support a new conferencing tool within CoBrow, the only alteration required is to include a new driver on MSM that copes with the tool's requisites. Placing the driver within the defined driver directory, and configuring the server to include the class, immediately allows the server to handle the new session type.

The Multimedia Session Manager, an application that manages multimedia sessions for Web based users. We presented the protocols involved and best practices. We also have shown the integration between the MSM and the user interface, which is the component that makes use of the facilities provided by MSM. MSM, as part of the CoBrow project, allows collaborative browsing on the Web, which may enable innovative applications in the future. Finally, despite the fact that MSM has been developed to work within the CoBrow project, MSM was implemented as a standalone application, has a very simple mechanism and employs well known protocols, therefore it can be used for other systems which need a manager for multimedia sessions.

3.0 Adaptive Portals with Wireless Components

Data access on the web is moving quickly past concerns with reliability and speed to concerns of usefulness and timeliness of information. Public portals, such as web search engines and corporate portals that facilitate access to enterprise information within a company, normally through the web, have been available for the last few years. Such portals are made up of "channels" of information and the purpose of these portals is to provide an interface that presents an organized view of the data to which the user has access, i.e., a straight forward means of access to this data.

The next evolutionary step in internet information management is to provide support for tasks, which may be collaborative and may include multiple target devices, from desktop to handheld. This means that the software supports the processes of the task, recognizes group interaction, and lets users migrate seamlessly among internet-compatible devices without losing the thread of the session. This extends the notion of portal from an organized view of web data to a managed view of task data.

The requirements of the software to support this level of task-dependent device-independent, internet/intranet services are significant. Such software must be generic enough to support a wide range of complex tasks in a variety of application areas.

Group interaction and concurrent transaction support are needed for "real time" group support. Additionally, the state of both group and task must persist as members join and leave a task session. If users are free to migrate amongst devices during the course of a session then intelligent transformation of data is required to exploit the display and input characteristics of the appliance with minimal loss of content and context. Finally, such software must be standards based, rather than language or operating system based.

In this paper we describe an agent architecture that supports a class of portals,which we are calling adaptive portals, in the broad sense. That is, adaptive portals support environments characterized by real tasks, collaboration in completion of such tasks, and a wide variety of appliances, from desktop to wireless. As an example, a physician portal may be defined for managing information related to patient rounds.

Members of the group may include the primary physician, attending nurses, physiotherapists, pharmacists, and other specialists. Individuals may join in the session from a desktop at the nursing station or office or from a wireless device while on rounds or in transit.

The emergence and domination of the web as the common information provider encourages the development of portals that support tasks and collaboration because of its deployment and acceptance in all sectors: education, home, entertainment, business, and government. Portals provide much needed support for communities of users to access content within domains, defined by corporate, community, or private interests. Although the content of given web sites and databases may change, the "portal" provides a constant view for the users. Portals provide most of their value currently by organizing a domain of sites and data access for the user community. To be useful in more dynamic situations, the content must reflect the context of the current state of the task and the reality of the display characteristics of the particular device. This represents a convergence of task and device (Agrawal et al.,1995).

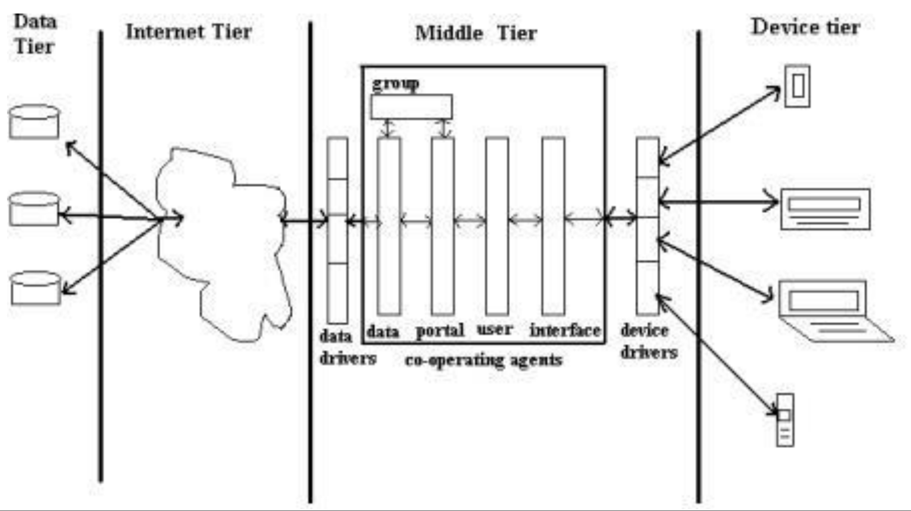


Fig. 2. Adaptive Portal Architecture

Source: Jacobson (2011)

The Adaptive Portal is based on a multi-tier architecture, as shown above: data tier, internet tier, middle tier, and device tier. Data Tier The data tier is the domain of documents, databases, and other data available to the system. The data may be proprietary or public, in files or in database management systems, real-time, triggered or generated as needed. Middle Tier The middle tier is server(s) side software that provides the adaptive portal functions. This tier manages the administration functions of maintaining user, group and task profiles including definition and updates. The software in this tier manages task sessions, from both the individual and the group perspectives and includes data and device drivers. Data drivers facilitate access to particular data sources, providing a degree of data independence. This includes the specifics of access and updates to distributed databases and/or database management systems. Device drivers facilitate communication with particular devices, both wired and wireless.

The middle tier software has three main functions: administration, task management, and session management. The administration function is to manage the personal characteristics, task parameters and processes, and group characteristics. The task management function is to control the flow of task processes, including sequencing and completion of sub processes. The session management function maintains the current state for a task over the period of a session, where a session may involve members joining and leaving the session during its course.

A Resource Classification System for the WEB AIS There are many standardized communication protocols that define the interaction between entities, but often they fall short in providing clues about the meaning of the data exchanged and the service provided. In order to enhance interaction between WOS nodes, an agreement on the meaning of data and services is required.

The impossibility to classify resources on the Web in a reasonable amount of time asks for support of multiple classifications that can evolve locally and independently.

The design of a WOS warehouse based upon classifiers and ontologies is presented. There are many standards in use on the Internet for data transmission between active entities such as RPC, CORBA, RMI, HTTP and other protocols, but once communication is established, what should the entities talk about? In fact, if they do not agree on the meaning, is interaction even possible?

Communication protocols like the WOSP define very well the interaction between the entities for establishing communication as well as specifying the structure of the data exchanged. However, they often fall short in providing clues about the meaning of the content or explicitly do not address content at all. What is needed for higher level interaction is an agreement on that meaning.

This is, what is usually captured by the notion of ontology. Both data and active entities are referred as "resources", because they are addressed in the same way by URI's (Universal Resource Identifiers) (Agrawal et al.,1995).

The rapid development and the heterogeneous nature of the Web ensures that it is impossible to develop a complete catalog of all resources. In this context, we would like to extend this, statement. Building a catalog is only concerned about instances (physical versions) of resources. But in the ever changing environment of the Web, we claim that it is even impossible to build a complete classification of resources on the Web, at least within a reasonable time. As a consequence, we need support of many different (versioned) ontologies that can emerge and evolve independently of each other, logically close to the resources that are related to the problem domain they are concerned with.

Having versioned ontology support at hand helps building and structuring WOS warehouses as well as specifying the interaction requirements of active entities. The WebCom system for example requires the possibility to express a precondition in order to validate the data used to trigger an action. These preconditions are expressed in an agreed vocabulary of terms that form an ontology.

In many computer applications, we deal with multidimensional information, for example, time series, spatial information, and data warehousing data. It is not surprising, therefore, that a recent development in database systems is the emergence of multidimensional database systems to store multi-dimensional information in order to provide support for decision support systems.

Most of the recent approaches to multidimensional data models offer hyper-cube-based data models. For instance, (Agrawal et al.,1995) cube data model oriented towards a direct SQL implementation into a relational database. Libkin et al. (1996) defined a query language based on multidimensional arrays, which is oriented towards physical implementation. Cabibbo and Torlone (1995) proposed a multidimensional data model based on f-tables, a logical

abstraction of multidimensional arrays. These approaches, among others, are steps towards a systematic approach to modelling mutidimensional information in the database context. However, the current multidimensional databases are still not as powerful and widely accepted as relational databases, partly because they are not based on an application-independent formal basis, in which each dimension is treated uniformly, whose model generalized the relational model, and which enables the declarative specification and optimization of queries.

4.0 Multidimensional XML

The basic component in XML is the element, which is a piece of data bounded by matching tags (markup) of the form element-name and element-name, called start-tag and end-tag respectively. Element names in XML are defined at will so that they best represent data domains. Inside an element we may have other elements, called subelements. Markup encodes a description of the storage layout and the logical structure of the document. An XML document consists of nested element structures, starting with a root element. The data in the element are in the form of character data, sub elements, or attributes.

XML allows us to associate attributes with elements. Attributes in XML are declared within element start tags and have the form of a sequence of name-value pairs. The value of an attribute is always a string enclosed in quotation marks. Unlike sub elements, where a sub element with the same name can be repeated inside an element, an attribute name may only occur once within a given element.

The Extensible Markup Language (XML) tends to become a widely accepted formalism for the representation and exchange of data over the Web. A problem that often arises in practice is the representation in XML of data that are context-dependent (for example, information that exists in many different languages, in many degrees of detail, and so on). In this paper we propose an extension of XML, namely MXML or Multidimensional XML, which can be used in order to alleviate the problem of representing such context-dependent (or multidimensional) data. Apart from its usefulness in the semistructured data domain, MXML also reveals some new and promising research directions in the area of multidimensional languages.

The Extensible Markup Language (XML) is a data description language which tends to become standard for representing and exchanging data over the Web. Although elegant and concise, the syntax of XML does not allow for convenient representation of multidimensional information. Suppose that one wants to represent in XML information that exists in different variations (for example in different languages, in various degrees of detail, or in different time points). With the current XML technology, a solution would be to create a different XML document for every possible variation. Such an approach however is certainly not practical, because it involves excessive duplication of information (especially if the number of variations is high and there exist large identical parts that remain unchanged between variations). In this paper we propose a solution to the above problem, based on ideas that originate from the area of multidimensional programming languages

Application Programming Interface for Web-Based Accounting Information System (ASI) The Web Operating System (WOS) for software services to be distributed over the Internet. It supports applications and users with mechanisms to offer own resources and to locate and use accessible remote resources while taking advantage of the ever changing software and hardware infrastructure of global systems. The WOS approach to support distributed computing relies on the widespread use of version control techniques. It is a decentralized system, for which there is no complete catalog of available services and resources. Such knowledge is rather dynamically built up and managed. Therefore, the central component of the WOS is a generic communication framework allowing for versioned protocols, used to support communication between components or nodes. Each node operates as a server and a client at the same time and maintains warehouses with information about other nodes (Agrawal et al.,1995).

This allows to considerably reduce the response time to requests because expensive search procedures may often be avoided while relying on the information present in the warehouses for fulfilling service requests. The information in these warehouses are continuously updated when new knowledge about other nodes becomes available through service requests issued by the node itself or by other nodes. A collection of such nodes is managed in a completely decentralized manner.

The WOS communication layer uses a two-level approach. The first layer offers discovery/location services and the second one is used for service invocation. The Web-Based Accounting Information System (ASI) allows for a user to submit a service request without any prior knowledge about the service and to have it fulfilled according to the user's desired constraints/requirements. Such services may be specialized hardware or software, or both. The Web-Based Accounting Information System (ASI) considers the communication layer to be the centralized part. The communication protocols may thus be seen as the "glue" of the Web-Based Accounting Information System (ASI) architecture. This paper presents an Application

Programming Interface (API) to access Web-Based Accounting Information System (ASI) communication services. In order to present how the communication layer works, we introduce all the concepts related to the communications in the Web-Based Accounting Information System (ASI) and will show how these components are put together to support communication.

In other words, the WOS is designed to enable transparent usage of network-accessible resources, whenever a user requires a service, wherever the service is available. These services may be specialized hardware or software, or a combination of both. A user needs only to understand the WOS interface, and does not need to know how the service request is fulfilled. Therefore the WOS provides a computation model and the associated tools to enable seamless and ubiquitous sharing and interactive use of software and hardware resources available on the Internet.

The WOS is designed as a fully distributed architecture of interconnected nodes where the communication protocols are considered to be the centralized parts. The communication protocols may thus be seen as the "glue" of the WOS architecture. We identified two majors problems with this approach: The service classes could not be developed independently from the communication layer, because they had to specialize WOSP Analyzer and The communication layer was controlling the flow of processing for the whole system, since WOSP Parser was explicitly calling the specialized WOSP Analyzer.

In addition, the initial design of WOSRP/WOSP assumed that a synchronous dialog was required between two nodes in the connection-oriented mode. It turns out that this requirement imposes too much constraints on the service class developer. For instance, the application developer must guarantee that all communications between clients and servers be synchronized.

The new API presented in this paper alleviates these problems by removing every aspects of the semantics processing from the communication layer. This should provide more flexibility to the service class developer. It also supports asynchronous communications between clients and servers. In order to present how the new communication layer API works, we first introduce all the concepts related to the communications in the WOS. We will then show how these components are put together to support these communications.

4.1 Design and Implementation of a Distributed Agent Delivery System

Among the most significant changes that have affected the domain of computer networking is the proliferation of distributed applications and services, particularly within wide-area networks such as corporate intranets and most notably within the Internet. As the demand for such applications and services continues to expand, the need for a generic, open solution facilitating the distribution of data and services becomes increasingly apparent. Researchers have recently begun to investigate the feasibility of using the Mobile Agents Paradigm as an integral part of distributed computing infrastructures. In addition to facilitating the exchange of data and the access to services, agents serve as abstractions that separate the communication of data from the location and format of data that is transferred among the nodes of the distributed environment. This paper discusses the goals, design and implementation of a particular multilingual mobile agent development kit, the Distributed Agent Delivery System (DADS). DADS supports multiple agent languages types, and is deemed sufficiently lightweight to be deployed in performance-sensitive environments. DADS thus provides the fundamental mechanisms for the development of distributed applications that would scale well with the ever-increasing size and complexity of modern distributed infrastructures.

Mobile agents have begun to radically alter the way that we view the exchange of information across distributed environments. Modern communication infrastructures, such as Internet2 and the Next Generation Internet, give rise to very large distributed systems that may consist of hundreds or even thousands of computing nodes. The spatial and temporal nature of the distributed infrastructure requires the underlying mechanisms to scale easily with the size and dynamics of the computational environment. Mobile agent systems are deemed to provide the necessary scalability to solve many problems related to managing Mobile agents have begun to radically alter the way that we view the exchange of information across distributed environments. Modern communication infrastructures, such as Internet2 and the Next Generation Internet, give rise to very large distributed systems that may consist of hundreds or even thousands of computing nodes. The spatial and temporal nature of the distributed infrastructure requires the underlying mechanisms to scale easily with the size and dynamics of the computational environment. Mobile agent systems are deemed to provide the necessary scalability to solve many problems related to managing Distributed Agent Delivery System (DADS).

DADS can be viewed as a collection of lig htweight daemon processes, which by means ofthe Agent Delivery Protocol (ADP), exchange agents that will perform domain specific tasks. The ADP represents a multi-lingual solution as it is oblivious to the implementation language ofthe agent being transported. The support for multiple agent languages solely depends on the availability of the corresponding interpreters or runtime systems at the nodes where the agents are to execute. ADP is a simple, lightweight protocol that does not use any cumbersome encoding scheme, thereby allowing the DADS to perform equally well in performance-sensitive environments as well as in distributed environments such as the Internet.

4.2 DADS Components

The DADS manifests a service that resides on the set of participating nodes. Henceforth, these nodes will be referred to as patrons, as they provide the necessary support for the agents to execute a domain specific task and at the same time, act as customers that may utilize the agents on behalf of the system. In general, the DADS is not formally coupled with applications that interact with the agents that are transferred. Hence, the DADS provides an ideal solution for the development of heterogeneous distributed applications. The canonical DADS architecture consists offive components, namely Mobile Agents, an Agent-Based ADP, a Patron-Based ADP, an Agent Interpreter, and Domain-Specific Protocols (Boudreau, 2010).

– Mobile Agents: The agents are a crucial element towards the development of an agent-based system. Without exhibiting a specific behavior, the infrastructure supporting the mobilization of agents is useless. Hence, each agent is the realization of the domain-specific task that is to be performed.

– Agent-Based ADP : Agents and nodes must support a common protocol to support the migration of agents. In order for an agent to move itself, this protocol must be supported from the perspective of the agent. Hence, associated with each agent is an Application Program Interface (API) that provides the agent with specific functions for using the ADP.

– Patron-Based ADP : The patron refers to a node which utilizes agent services. For an agent to migrate to a patron, the patron must support the Agent Delivery Protocol. Hence, a patron-specific API for using ADP functionality is imperative.

– Agent Interpreter (on every participating node) : Because agents move to and from patrons that execute on a variety of different architectures, a platform-independent, interpreted language must be selected and supported on every patron.

– Domain-Specific Protocols (to be used between agents and patrons) : Because most (but not all) agents will need to communicate with the patrons, an extensible protocol should be developed.

5.0 Conclusion and Recommendations

An application programming interface, or API, enables businesses to make the data and functionality of their applications available to external third-party developers, commercial partners, and internal departments inside their own organizations. Using a defined interface enables services and products to interact with one another and benefit from each other's information and features. The interface is used by developers to interact with other software and services; they are not required to understand how an API is developed, and neither are the software's end-users.

In short, an API is a contract between pieces of applications serving the main software once integrated into the source code of the main application. These pieces of applications communicate with each other in the language they both understand and over a network if needed. As the name implies, APIs serve as interfaces between programs. The interface is usually between a software developer and software developer's application. Basically, the API allows one software to access some functionalities of another software. In its development, information systems develop and run well using a web-base so that it can be reached via an online computer network. Web applications have become complex and important for many companies, especially when combined with areas such as AIS. All AIS transaction data will be stored in the database management system and will be used if there is a query data request.

Based on the IFRS (International Financial Reporting Standard) document, a proper accounting system must at least include the following aspects (IFRS Standard Requirements): Financial position statement, cash flows statement. Web-based AIS application for recording, grouping, processing, and presenting transaction data regarding finance. The data then becomes information that can be used as material for making a decision. Web-Based AIS has an admin page to manage every important data, both master data and data in the form of transactions. It also provides an application programming interface (API) web service, which is access that can be used by external developers who require data access. The available API serves as an intermediary between the main data provider software system with external developers, where developers develop AIS applications with different platforms such as web-based or mobile-based.

References

Adner, R. & Kapoor, R. (2010). Value creation in innovation ecosystems: How the structure of technological interdependence a performance in new technology generations. *Strategic Management Journal,31*(3), 306-333.

Agrawal, S., Gupta, A., & Sarawagi, S. (1995). Modeling multidimensional databases. Technical report, IBM Almaden Research Center, San Jose, California, 1995.

Baldwin, C. & Clark, K. (2000). *Design Rules: The Power of Modularity*. Volume 1. The MIT Press.

Boudreau, K. (2010). Open platform strategies and innovation: Granting access versus devolving control. *Management Science,56*(10), 1849-1872.

Cabibbo, L. & Torlone, R. (1995). Querying multidimensional databases. In Proceedings of the Sixth Workshop on Database Programming Languages, 1997.

Chan, S. W., Schilizzi, S., Iftekhar, M. S., & Da –silva, R. (2019). Web-based experimental economics software: How do they compare to desirable features? *Journal of Behavioral and Experimental Finance, 23*,138-160.

Elsharif, T. A. (2019). The elements of accounting information systems and the impact of their use on the relevance of financial information in Wahda Bank-Benghazi, Libya. *Open Journal of Business and Management, 07*(03),1429-1450.

Gawer, A. & Cusumano, M. A. (2014). Industry platforms and ecosystem innovation. *Journal of Product Innovation Management,31*(3),417-433.

Girard, R. (2019). Opinion | L'open banking à l'ère des API, de l'intelligence artificielle et du cloud. Le Cercle Les Echos, 26(2),12-22.

Huckman, R., Pisano, G., Chen, D. & Kind, L. (2012). Amazon web services. *Harvard Business School Case,9-609-048*.

International Financial Reporting Standards IAS 1 Presentation of Financial Statements, https://www.ifrs.org/issued-standards/list-of-standards/ias-1-presentation-of-financial-statements/, last accessed 2023/05/21.

Jacobson, D., Brail, G., & Woods, D. (2011). *APIs: A strategy guide*. O'Reilly Media, Inc. API Technologies and Protocols:

Kemer, E., & Samli, R. (2019). Performance comparison of scalable rest application programming interfaces in different platforms. *Computer Standards & Interfaces, 66*(1), 20-30.

Libkin, L. & Wong, L. (1996). A query language for multidimensional arrays: Design, implementation and optimization techniques. In Proceedings of the 1996 ACM SIGMOD International Conference on Management of Data ,228-239. ACM Press, 1996.

Red Hat, Inc., 2023. What is an API? [Online] Available at: https://www.redhat.com/en/topics/api/what-are-application-programming-interfaces

Tiwana, A., Konsynski, B., & Bush, A. A. (2010). Research commentary-platform evolution: Coevolution of platform architecture, governance, and environmental dynamics. *Information Systems Research,21*(4),675-687.