**Chapter Two**

**SERVICE ORIENTED ARCHITECTURE**

1. **Introduction**

Service Oriented Architecture (SOA) is one of the software architectures that satisfy most of the non-functional requirements addressed by information systems. SOA principles need to be presented and studied in order to utilize SOA for UMIS and LMS to overcome current LMS shortages, limitations, and deficiencies. SOA as a design pattern is expected to solve many of the current information systems’ problems by easing integration, interoperability, and agility.

1. **Service and Service Orientation**

Objects, components, and services are three main acronyms in software architecture that are interrelated. Objects have lead to components, and services can be thought of as specialized type of components.

* 1. **Software Objects**

Object is the encapsulation of data and behavior in one data type; class; as depicted in figure 2.1. Objects are instances of classes that exist for a certain period of time. Objects communicate with each other via exchanged messages. Messages are instructive, not descriptive. Object based development advances software design by providing more support for hiding behavior and data through objects, however, large number of interconnected objects create dependencies that can be difficult to manage. Object based software development is called Object-Oriented Analysis and Design (OOAD) [94]



Figure 2.1: Objects encapsulate data and behavior

* 1. **Software Components**

Components are more sophisticated software modules than objects and require fundamental changes in systems thinking, software processes, and technology utilization. Software component is a unit of composition with contractually specified interfaces and explicit context dependencies [95]. It is a group of objects with has specified interface, working together to provide an application function, such as depicted in figure 2.2.



Figure 2.2: Software Component

Component may refer to many different software constructs, from single application logic to an entire functional system. In all cases, a component is a software package with one or more well defined interfaces. Components overlap the properties of object orientation, such as encapsulation and polymorphism, except it reduces the property of inheritance. In component thinking, inheritance is tightly coupled and unsuitable for most forms of packaging and reuse [96]. Instead, components reuse the functionality by invoking other objects and components rather than inheriting from them [97]. Reusable components are good reflection of effective software design. The development of software architecture based on component specifications supports parallel and independent building of the system parts. Many platform vendors have already produced software infrastructures which support component oriented technology.

* 1. **Service and Service Models**

Service is a component capable of performing a task [98]. Services can be categorized based on the nature of logic they encapsulate and the manner in which they are typically utilized within SOA as depicted in [99].

1. **Service Oriented Architecture**

SOA is defined as the policies, practices, frameworks that enable application functionality to be provided and consumed as sets of services published at a granularity relevant to the service consumer. Services can be invoked, published and discovered, and are abstracted away from the implementation using a single, standards based form of interface [98 - 101].

* 1. **Advantages of Service Oriented Architecture**

SOA advantages are categorized into implementation, and organizational benefits.

* + 1. **Implementation Benefits**

Implementation benefits satisfy the loose coupling objective. Using a resource only via its published service and not by directly addressing the implementation gives system capabilities of [102]:

* Changes to the implementation by the service provider should not affect the service consumer. Services are exposed via standard interfaces and are thought about as black boxes that changes within it do not affect consumers.
* Service consumer could choose an alternative instance of the same service type without modifying requesting application. As long as new service implements the same interface, theoretically there are no problems.
* Service consumer and service provider do not have to implement same technologies for implementation, interface, or integration when Web services are used.
  + 1. **Organizational / Business Benefits**

Businesses are dealing with two fundamental concerns: the ability to change quickly (agility), and the need to reduce costs [103]. Business must adapt quickly to internal factors such as acquisitions and restructuring, or external factors like competitive forces and customer requirements to remain competitive. Cost-effective, flexible IT infrastructure is needed to support the business. SOA can realize several benefits to help organizations succeed. Organizational/Business benefits of adopting SOA are:

* **Leverage existing assets:**by wrapping existing software applications as services instead or rebuilding new solutions from scratch.
* **Easier to integrate:** integration on service level in order to satisfy business processes integration presents the highest integration technique that solved many problems as depicted in chapter three.
* **More responsive and faster time-to-adapt (Agility):** the ability to compose new services out of existing ones provides a distinct advantage to an organization that has to be agile to respond to demanding business needs. Leveraging existing components and services reduces the time needed to go through the software development life cycle leading to rapid development of new business services and allows an organization to respond quickly to changes. SOA provides the flexibility and responsiveness that is critical to businesses agility.
* **Reduce cost and increase reuse:** with core business service exposed in a loosely coupled manner, they can be more easily used and combined based on business needs. This means less duplication of resources, more potential for reuse, and lower costs.
  1. **SOA Technologies**

SOA implementations include: Software Agents, and Web services.

* + 1. **Software Agents**

Software Agent is a computer system that is situated in some environment and is capable of autonomous actions in this environment in order to meet its design objectives [104,105].

Different SOA implementations using different software agents are presented in **[**106– 110]. SOA implementation via mobile agents technology can be found in[108]. The main idea is about having one or more software agents perform certain task(s), those tasks can be exposed as services that compose SOA.

Software agents have many classification criteria. Agents can be classified as either Desktop, Internet, or Intranet agents according to their action environment [111]. Many internet related agents can be classified into subcategories as presented in [112]. Intelligent agents can be classified into Simple reflex agents, Model-based reflex agents, Goal-based agents, and Utility-based agents [113]. Agents can be also classified by application types, and by characteristics [114].

Software Agents have characteristics that make them suitable to perform complex functionalities. Characteristics include: Autonomy, Interactivity, Reactivity, Proactivity, Intelligence, and Mobility [114]. Agent is autonomous; it is capable of acting on its own. An agent is goal oriented, collaborative, and flexible, so, it must be autonomous. Agents are designed to interact with other agents, humans, or software programs (Interactivity). Instead of making a single agent conduct several tasks, additional agents can be created to handle un-delegated task. Agents perceive environment via preceptors [113] and respond to changes (Reactivity). Agents do not just act in response to their environment, but agents are able to exhibit goal-directed behavior by taking an initiative (proactive). Agent may need mobility to work on different machines. An agent with this capability is called mobile agent, it can transport itself across different system architectures and platforms, and is far more flexible than those that cannot. Many electronic commerce agents are mobile [114]. Mobile agent is an executing program that can migrate during execution from machine to machine in a heterogeneous network [110].

**Multi-Agent System**

Multi-Agent Systems (MASs) are becoming increasingly important: as a scientific discipline, as a software engineering paradigm, and as a commercially viable and innovative technology [115].

A Multi Agent is any system that contains [116]:

* Two or more agents;
* At least one autonomous agent; and
* At least one relationship between two agents where one satisfies goal of the other.

Many Multi-Agent frameworks have been presented. Some of Multi-Agent frameworks proposed from 2005 till now include works presented in [117 – 122].

MAS characteristics are [123]:

* Each agent has incomplete capabilities to solve problems.
* There is no global system control.
* Data is decentralized.
* Computation is asynchronous.

**Common MAS Architectures**

Agents can be organized in different MAS architectures. MAS architectures include: flat, fixed hierarchy, Subsumption, and Modular [124].

**Flat MAS Architecture:** Agent are directly contact all other agents. The system is either closed, so all agents know the location of all other agents, or open, which requires agent location mechanism. Agent location mechanism can be one of message based architectures. Figure 2.3 depicts Flat MAS Architecture.



Figure 2.3: Flat MAS Architecture

**Fixed Hierarchy MAS Architecture:** Agents communicate only to agents directly above or below them in the hierarchy. Hierarchy is fixed leading to defects in systems scalability. Figure 2.4 depicts hierarchal MAS architecture.



Figure 2.4: Hierarchical MAS Architecture

**Subsumption MAS Architecture:** Agent can contain other agents. Highly performance and wide scalability is the main advantage of subsumption architecture. Figure 2.5 shows subsumption MAS architecture.



Figure 2.5: Subsumption MAS Architecture

**Modular MAS Architecture:** Agents are grouped in modules as presented in figure 2.6. Communication takes place between agents composing the module, and agents of different modules. Each module is specified with one or more task(s). Presenting systems as collection of modules facilitated problem solving.



Figure 2.6: Modular Organization

Multi-Agent architecture standards were attempted in order to force MAS standardization and global integration. Knowledge Query and Manipulation Language (KQML) was presented in order to support knowledge sharing among agents [125]. Knowledge Interchange Format (KIF) is a computer-oriented language for the interchange of knowledge among disparate programs [126]. The OMG group proposed a reference model as an attempt to standardize the development of agent technologies [127]. KAoS is described as "an open distributed architecture for software agents.'' The KAoS architecture describes agent implementations, and elaborates on the interactive dynamics of agent-to-agent messaging communication by using conversation policies [128]. The Foundation for Intelligent Physical Agents (FIPA) is a multi-disciplinary IEEE standardizing group pursuing the standardization of agent technology. FIPA's approach to MAS development is based on a "minimal framework for the management of agents in an open environment'' [129]. Unfortunately, as a result for all the standardization effort, there were no universally accepted commercially supported standard yet.

* + 1. **Web services**

Web services are not just an integration solution. Web services technology is currently the major implementation of SOA [130]. All service models are specific to the WS-Coordination specification and related protocols [99]. Web services adoption by organizations solved many problems. Web services is a general framework that expedites the sharing of heterogeneous data and software resources dispersed on the internet. The standard-based resource sharing and platform-neutral characteristics of web services have motivated many organizations to apply the technology in diverse areas, such as supply chain management, virtual enterprise, homeland defense, e-government, and e-business [131]. Software agents can consume Web services as presented in [132].

1. **Web services Technology**

Web services are applications that use standard transports, encodings, and protocols to exchange information [133]. A Web service is a software system designed to support interoperable machine-to-machine interaction over a network. Web service has an interface described in a format that machines can process (specifically WSDL), Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with XML serialization in conjunction with other Web-related standards [134].

* 1. **Web services Architecture**

Web service technology is based on open technologies; this provides broad interoperability among different vendor solutions [103]. Figure 2.7 shows the basic Web services architecture, that consists of specifications (SOAP, WSDL, and UDDI) that support the interaction of a Web service requester with a Web service provider and the potential discovery of the Web service description [135]. The provider typically publishes a WSDL description of its Web service, and the requester accesses the description using a UDDI or other type of registry, and requests the execution of the provider's service by sending a SOAP message to it.



Figure 2.7: Basic Web services Architecture

* 1. **Web services Key Features**

Some of Web services key features are [103]:

* **Self-contained:** On the client side, a programming language with XML and HTTP client support is the only requirement. On the server side, a Web server and a servlet engine are required.
* **Self-describing:** format and content of request and response messages (loosely coupled application integration) is what only matters. Messages are descriptive, not instructive.
* **Modular:** Web services are a technology for deploying and providing access to business functions over the Web; J2EE, CORBA, and other standards are technologies for implementing Web services.
* **Web published, located, and invoked:** by applying the previously mentioned standards SOAP, XML, WSDL, and UDDI, Web services can be published, located, and invoked via internet.
* **Language independent:** interaction between a service provider and a service consumer is designed to be completely platform and language independent. Interaction requires a WSDL document to define the interface and describe the service, along with a network protocol (usually HTTP).
* **Interoperable:** Because service provider and service consumer do not share same platforms or languages; only communicate via standard protocols, interoperability is achieved.
* **Inherently open and standards based:** XML, SOAP, WSDL and HTTP are the technical foundation for Web services. A large part of the Web service technology has been built using open source projects.
* **Dynamic:** dynamic e-business can become a reality using Web services because, with UDDI and WSDL, the Web service description and discovery can be automated.
* **Compassable:** Web services can be aggregated to complex ones.
  1. **Advantages of Web services**

Web services advantages arise from Web services key features. Web services were designed to satisfy all software requirements arose over the years, and avoid all drawbacks of previous technologies. Web services advantages include [133,136, 137, 138, 139]:

* **Interoperability:** Software interoperability is the capability to communicate, execute programs, or transfer data among various functional units in a manner that requires the user to have little or no knowledge of the unique characteristics of those units. Web services are interoperable.
* **Language agnostic:** Web services are neither based on a programming language nor on a programming data model. Any Programming language can be used to implement Web services.
* **Relatively simple:** Web services are simple to build, expose, and consume because they are:
* Based on Web technologies: so, they are scalable, and easily to control security features.
* Do not necessarily require a huge framework in memory: A small amount of code could expose a service as a Web Service, besides Web services can be used easily in today's Web Interface.
* **Loosely coupled applications:** applications based on Web services are loosely coupled applications by default. Loose coupling is a design goal that describes a resilient relationship between two or more applications, systems, or organizations with some kind of exchange relationship [140]. In Web services, exchange relationship is established via request/response messages.
* **Support of software industry leaders:** Hundreds of IT vendors have participated in the Web services standardization process under the sponsorship of the World Wide Web Consortium (W3C), Organization for the Advancement of Structured Information Standards (OASIS) and Web Services Interoperability Organization (WS-I). IT vendors include, not only: Microsoft, Oracle, bea, SAP, IBM, and Sun..
* **Integration with the World Wide Web:** Web services tended to integrate with World Wide Web from the very first beginning in order to use internet as the infrastructure. Internet integration provides the most advantage of adopting Web services within applications.
  1. **Web services as SOA enabler**

Web services are relatively new technology that have received wide acceptance as an important implementation of SOA [103]. Web services is not SOA, and SOA is not Web services. It is important to differentiate between Web services; as a technology, and SOA as a design pattern. Web services enabled the maximum advantages of SOA, this is mainly why Web services is the main implementation and enabler of SOA. SOA presents systems as collection of services to be published, and consumed when required. When exposed services are Web services, system gains the advantage of Web services standardization, like interoperability. But still systems need to be studied carefully, and analyzed intensively to determine services will be exposed, suitable services interfaces, and design the required databases. SOA is not about exposing and consuming Web services to facilitate interoperability or integration or other requirements, but it is rather the science, art, and practice of building loosely coupled fine granular services to form applications.

1. **Service Layers**

Layers of abstraction are the main SOA advantages enabler [99]. By leveraging the concept of composition, specialized layers of services can be built. Each layer can abstract a specific aspect of the solution as presented in figure 2.8. Layers of abstraction differ from solution to another according to system requirements. Presented layers do not have to exist as an entity in all solutions. Architecting is the science and art of designing software systems, not the science of applying well defined steps. System architects are responsible for determine and defining the required layers of abstract to satisfy system requirements. Layering gives the enterprise capabilities to achieve enterprise-wide loose coupling by physically separating layers of services to collections of services that represent corporate business logic and other that represent technology-specific application logic. Each domain of the enterprise is freed of direct dependencies on the other [99]. Presented model depicts four abstraction layers, namely: presentation, orchestration, application services, and application layer.



Figure 2.8: Service Layers Model

A Bottom-Up discussion of the presented model is presented to highlight the tremendous layering advantages.

* 1. **Application Layer**

Application layer holds current running applications within organizations. Application layer include legacy systems, software agents, single applications, and Enterprise Resource Planning applications (ERP). Legacy systems are computer systems that have been in operation for a long time, and whose functions are too essential to be disrupted by upgrading or integration with another system despite its poor competitiveness. Specific task software agents are required to serve and support system. Software agents can be thought of as the optimum solution for complex, complicated, long running, tracking, analyzing, and intelligent tasks. Software agents are black boxes that hide intelligence and functionality features as black boxes, unfortunately without a standard interface to be accessed. Single applications and ERP were not designed with integration in mind, so it is almost impossible for nowadays educational institutions running all those different kinds of applications to integrate them, so, the need to leave those applications just running is almost a necessity.

* 1. **Application Services Layer**

Application services layer holds applications exposed as services, newly added services, and legacy applications wrapped by standard Web services interface. Legacy systems compatibility with modern equivalents has been facilitated via wrapper services. Wrapper service is a type of integration service that encapsulates and exposes logic residing within a legacy system via standard Web services interface to be integrated in the new SOA based systems. Application Services are set of stateless Web services that perform certain task(s). Web services are stateless; they can not maintain business logic, operation flow, or user state. System process is the summation of tasks performed by one or more service of application services layer at the sequence maintained by orchestration layer services. Application Services are reusable among different processes, can be integrated in new applications, and can be extended address new system processes. Service granularity level reflects the extent to which entity related services are decomposed into.

* 1. **Orchestration Layer**

Web services are stateless services that can not maintain process logic, operation flow, or user state; so, the need of an orchestration layer to include process logic is addressed. Orchestration layer holds orchestration and choreography engines to maintain system process workflow logic, performance requirements, and system/user state. Orchestration expresses a system process that is typically owned by the organization, while choreography engine addresses the realm of collaboration between multiple services from different enterprises. Orchestration layer manages interaction details required to ensure that service operations are executed in a specific sequence. Orchestration layer services maintain process logic that refers to mapping supported systems' processes into set of rules and sequences to be followed and maintained. Sequences are determined by supported system processes.

##### **Presentation Layer**

Presentation layer presents system’s direct contact and interactions with users. Each user category shall have a separate portal that provides different functionalities. Process logic shall not be embedded within presentation layer. Interface design and implementation shall be separated from process logic. This separation has proven many advantages for business architectures and is recognized as a recommended design pattern. Portals provide flexibility to add new services and remove existing services from the interface upon need. Portal consumes Web services, and provides means of user interactions with the system either via displaying results or acquiring inputs.

1. **Summary**

Service Oriented Architecture is the software architecture that achieved all systems functional and non-functional requirements. This chapter presented:

* Intensive SOA overview
* SOA characteristics and advantages
* SOA enablers, like CORBA, Software Agents, and Web services
* Web services as main SOA enabler, advantages and key features of Web services