Chapter 3

Service Oriented Architecture

3.1 Introduction

A web service is an instance of a more general notion of a service, which is defined as: "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production". The essence of a service, therefore, is that the provision of the service is independent of the application using the service. Service providers can develop specialized services and offer these to a range of service users from different organizations. SOA are a way of developing distributed systems where the system components are stand-alone services, executing on geographically distributed computers. Standard Extensible Markup Language (XML)-based protocols, such as SOAP and WSDL, have been designed to support service communication and information exchange. Consequently, services are platform and implementation-language independent. Software systems can be constructed by composing local services and external services from different providers, with seamless interaction between the services in the system. This chapter presents different topics related to SOA, including: SOA advantages, technologies, different methods of implementation, examples of IS based on SOA, and concludes with an important concept, that is ESB.

3.2 SOA

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].

3.2.1 SOA Advantages

SOA advantages are categorized into implementation, and organizational benefits.

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.

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.

3.3 SOA Technologies

SOA implementations include: Software Agents, and Web services.

3.3.1 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]. IAs 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 functionality. 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.

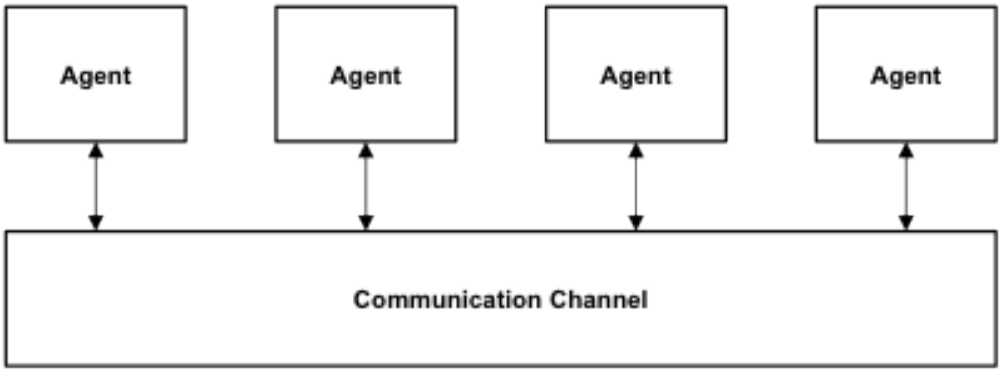

Figure 3.1: Flat MAS Architecture

- 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 3.1 presented in page 60 depicts 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 3.2 presented in page 61 depicts hierarchical MAS architecture.
- Subsumption MAS Architecture: Agent can contain other agents. Highly performance and wide scalability is the main advantage of subsumption architecture. Figure 3.3 presented in page 61 shows subsumption MAS architecture.
- Modular MAS Architecture: Agents are grouped in modules as presented in figure 3.4 presented in page 62. 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.

Multi-Agent architecture standards were attempted in order to force MAS standardization and global integration. Knowledge Query and Manipulation Language (Knowledge Query and Manipulation Language (KQML)) was presented in order to support knowledge sharing among agents [125]. Knowledge Interchange Format

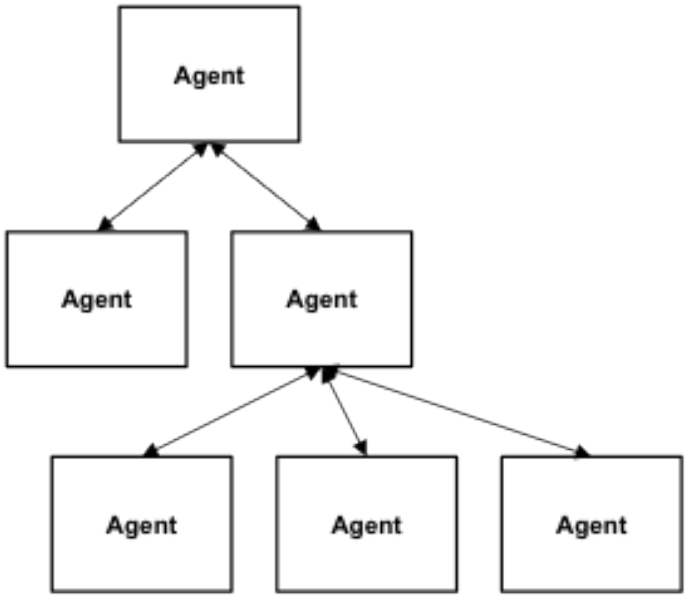

Figure 3.2: Hierarchical MAS Architecture

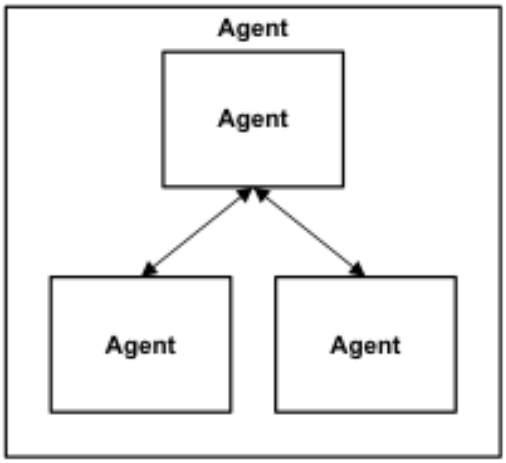

Figure 3.3: Subsumption MAS Architecture

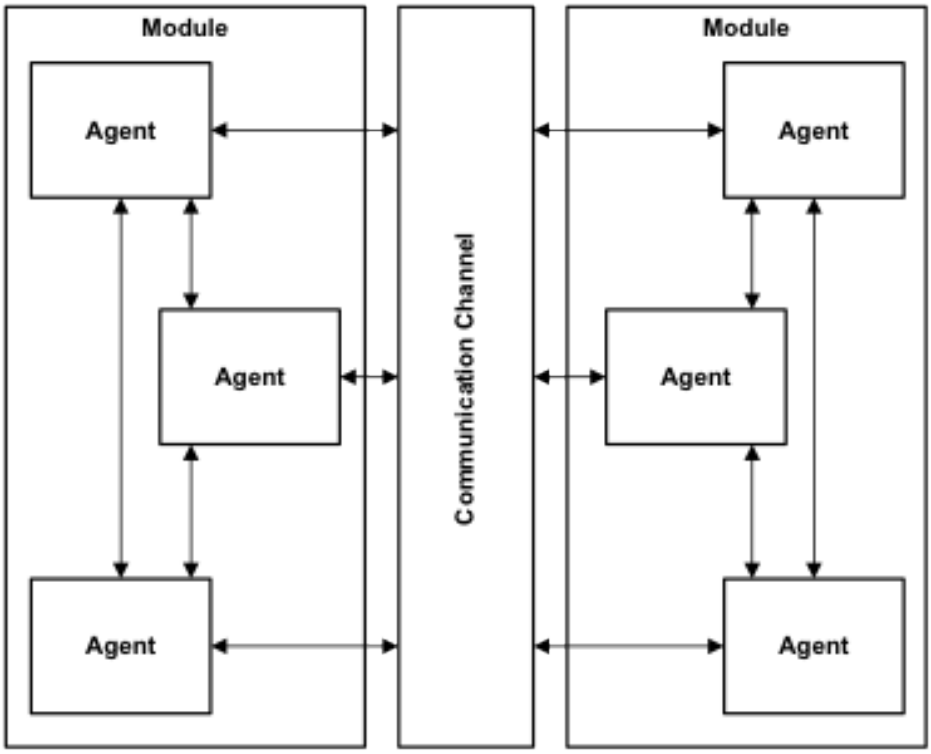

Figure 3.4: Modular MAS Architecture

(Kowledge 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]. Knowledgeable Agent-oriented System (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 (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.

3.3.2 Webservices

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 SCM, virtual enterprise, homeland defense, e-government, and e-business [131]. Software agents can consume Web services as presented in [132].

3.4 Webservices Technology

Web services are applications that use standard transports, encoding, 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 Hyper Text Transfer Protocol (HTTP) with XML serialization in conjunction with other Web-related standards [134].

3.4.1 Webservices Architecture

Web service technology is based on open technologies; this provides broad interoperability among different vendor solutions [103]. Figure 3.5 presented in page 64 shows the basic Web services architecture, that consists of specifications (SOAP, WSDL, and Universal Description Discovery and Integration (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

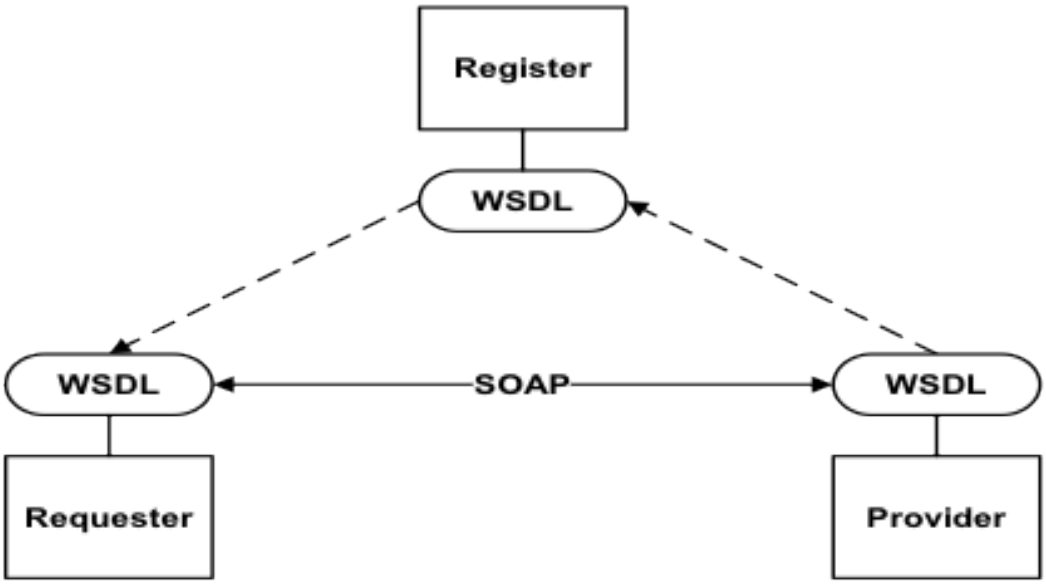

Figure 3.5: Basic Webservices' Architecture

using a UDDI or other type of registry, and requests the execution of the provider's service by sending a SOAP message to it.

Web service protocols cover all aspects of SOAs, from the basic mechanisms for service information exchange (SOAP) to programming language standards (WS-BPEL). These standards are all based on XML, a human and machine-readable notation that allows the definition of structured data where text is tagged with a meaningful identifier. XML has a range of supporting technologies, such as XSD for schema definition, which are used to extend and manipulate XML descriptions. Erl (2004) provides a good summary of XML technologies and their role in web services. Briefly, the key standards for web SOAs are as follows:

1. SOAP This is a message interchange standard that supports the communication between services. It defines the essential and optional components of messages passed between services.

2. WSDL is a standard for service interface definition. It sets out how the service operations (operation names, parameters, and their types) and service bindings should be defined.

3. WS-BPEL This is a standard for a workflow language that is used to define process programs involving several different services. A service discovery standard, UDDI, was also proposed but this has not been widely adopted. The UDDI standard defines the components of a service specification, which may be used to discover the existence of a service. These include information about the service provider, the services provided, the location of the WSDL description of the service interface, and information about business relationships. The intention was that this standard would allow companies to set up registries with UDDI descriptions defining the services that they offered.

3.4.2 Webservices 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.
- Composable: Web services can be aggregated to complex ones.

3.4.3 Webservices Advantages

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 (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.

3.4.4 Enhanced Standards

A number of companies, such as Microsoft, set up UDDI registries in the early years of the 21st century but these have now all closed. Improvements in search engine technology have made them redundant. Service discovery using a standard search engine to search for appropriately commented WSDL descriptions is now the preferred approach for discovering external services. The principal SOA standards are supported by a range of supporting standards that focus on more specialized aspects of SOA. There are a very large number of supporting standards because they are intended to support SOA in different types of enterprise application. Some examples of these standards include the following:

1. WS-Reliable Messaging, a standard for message exchange that ensures messages will be delivered once and once only.
2. WS-Security, a set of standards supporting web service security including standards that specify the definition of security policies and standards that cover the use of digital signatures.

3. WS-Addressing, which defines how address information should be represented in a SOAP message.

4. WS-Transactions, which defines how transactions across distributed services should be coordinated.

Current web services standards have been criticized as being 'heavyweight' standards that are over-general and inefficient. Implementing these standards requires a considerable amount of processing to create, transmit, and interpret the associated XML messages. For this reason, some organizations, such as Amazon, use a simpler, more efficient approach to service communication using so-called RESTful services.

3.4.5 RESTful Webservice

The set of web services protocols that have been developed support a very general model of web services and have taken into account important enterprise issues such as security, reliability and transactions. They allow services from different providers to be dynamically linking and executed.

However, an inevitable consequence of this generality is that there is significant processing overhead in interpreting the XML documents that are exchanged by web services. This can mean that the performance of service-oriented systems is reduced. This has led to alternative approaches to service implementation which use simpler protocols and which are usable in situations where the generality offered by 'big' web services is not required.

REST (REpresentational State Transfer) is an architectural style based on transferring representations of resources from a server to a client. It is the style that underlies the web as a whole and has been used as a much simpler method than SOAP/WSDL for implementing web services [246]. A RESTful web service is identified by its URI (Universal Resource identifier) and communicates using the HTML protocol. It responds to HTML methods GET, PUT, POST, and DELETE and returns a resource representation to the client. Simplistically, POST means create, GET means read, PUT means update, and DELETE means delete. RESTFul services involve a lower overhead than so-called 'big web services' and are used by many organizations implementing service-based systems that do not rely on externally provided services.

So-called RESTful web services follow the REST (REpresentational State Transfer) architectural style, with all communications based on the simple HTTP protocol. Services are modeled as resources (e.g. a parts catalogue) and respond to HTTP methods GET, POST, PUT and DELETE. The service is addressed using its URI (Universal Resource Identifier.

The information exchanged by RESTful services is the resource representation which may be XML, but, in principle, other representations such as JSON (Javascript Object Notation) may also be exchanged.

The HTTP methods are usually interpreted as [244]:

- POST=Create
- GET = Read

- PUT = Update or Create
- DELETE = Delete

The key benefit of RESTful services is that they are simpler and more efficient than 'Big' web services. The amount of processing overhead is reduced, which is particularly important for simple services than implement small increments of functionality.

SOAP and REST are not incompatible and it is possible for web services providers to offer both SOAP and REST service interfaces. Major providers such as Amazon offer both and their experience is that the REST interface is often preferred by developers.

3.5 An in-car Information System

SOAs are loosely coupled architectures where service bindings can change during execution. This means that a different, but equivalent version of the service may be executed at different times. Some systems will be solely built using web services and others will mix web services with locally developed components. To illustrate how applications that use a mixture of services and components may be organized, consider the following scenario: An in-car IS provides drivers with information on weather, road traffic conditions, local information, and so forth [246]. This is linked to the car radio so that information is delivered as a signal on a specific radio channel. The car is equipped with GPS receiver to discover its position and, based on that position, the system accesses a range of information services. Information may then be delivered in the driver's specified language. Figure 3.6 presented in page 69 illustrates a possible organization for such a system. The in-car software includes five modules. These handle communications with the driver, with a GPS receiver that reports the car's position and with the car radio. The Transmitter and Receiver modules handle all communications with external services. The car communicates with an external mobile information service that aggregates information from a range of other services, providing information on weather, traffic information, and local facilities. Different providers in different places offer these services, and the in-car system uses a discovery service to locate appropriate information services and bind to them. The discovery service is also used by the mobile information service to bind to the appropriate weather, traffic, and facilities services. Services exchange SOAP messages that include GPS position information used by the services to select the appropriate information. The aggregated information is then sent to the car through a service that translates that information into the driver's preferred language.

3.6 Webservices 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

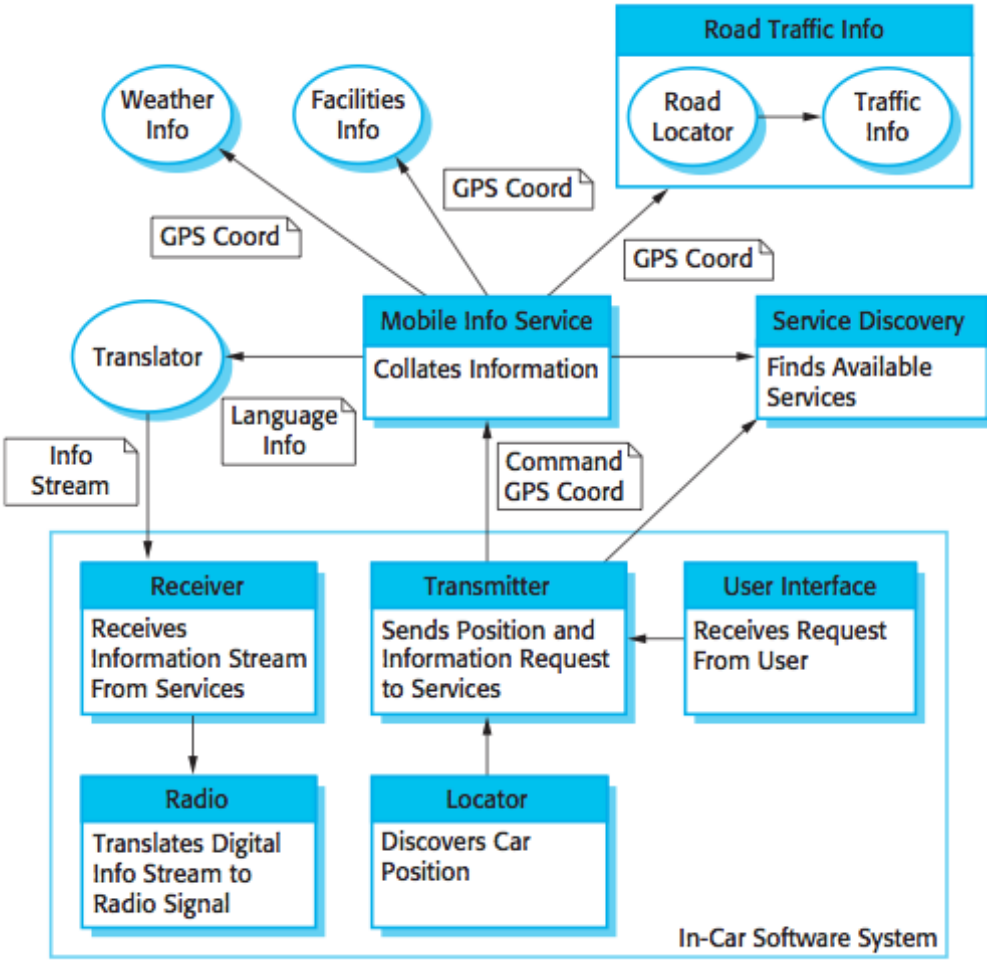

Figure 3.6: Service-based in-car IS [246]

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.

3.7 Enterprise Service Bus

ESB provides a comprehensive, scalable way to connect a large number of applications without the need for each pair of applications to make a direct connection. Such a direct connection between two applications is called a point-to-point connection. Note that even in the case of Web Services, the connection between the service consumer application and the service provider application is "point to point." The point-to-point connection approach does not scale well because the number of applications involved in the integration increase; therefore, this integration approach is not suitable for a large enterprise where a large number of applications need to be integrated.

3.7.1 ESB

ESB can be defined in two contexts: design and run-time. As a design-time context, ESB is an architectural pattern that has multiple motivations:

- Supports large numbers of service interactions in a manageable way
- Provides support for advanced service interaction capability, such as transactions, store and forward, infrastructure services, security, quality of service, and so on
- Supports a variety of interaction styles such as synchronous request/response, messaging, publish/subscribe, and events
- Provides a robust, manageable, distributed integration infrastructure consistent with the principles of SOA
- Supports service routing and substitution, protocol transformations, and other message processing
- Supports both Web Services and traditional EAI communication standards and technologies

The ESB pattern can be implemented in one or more of the following hybrids:

- EAI technology

- Messaging technology
- Technology rebranded or classified as an ESB product
- "Gateway" technology
- Appliance technology
- Bespoke

The ESB mediates characteristics of interactions between service requestors and service providers. The ESB enables the substitution of service providers or implementations transparent to service requesters. The ESB supports a variety of means to attach requesters and providers, and it allows intermediary services to be sequenced between requesters and providers. The ESB may provide a broad set of capabilities dependent on business needs and implementation in several areas, including network communications, integration, security, message processing, quality of service, and service management.

3.7.2 ESB vs. Integration Technology

An ESB is a connectivity infrastructure for integrating applications and services, in contrast to EAI, which focuses on the integration of applications. ESB infrastructure differs from EAI in the following aspects:

- ESB infrastructure is more than integration because it performs routing of messages between services, converts transport protocols between consumers and providers, transforms message formats between requesters and providers, and distributes business events from disparate sources. Although EAI solutions can address all of these aspects, integration technologies are usually much more narrowly focused. ESB handles a variety of interaction patterns, including events.
- ESB requires management such that the status of a business transaction can be assessed. Has the transaction completed? How long did it take? Did the process step complete? Although the ESB will not be the only technology to assist in business activity management, it will be a part.
- ESB product technologies will be federated such that various technologies (e.g., gateways and appliances) can be used to fulfill a single purpose and provide a single interface to applications. Heterogeneous platforms can be supported allowing different ESB technologies to operate as a single logical ESB.

3.7.3 ESB Functionality

Business-and IT-level events are used by service-level automation to enforce the policies associated with the business services they host. A service bus can be used to collect, aggregate, and evaluate those events for presentation to business process participants in business activity management scenarios. The ESB provides a broad set of capabilities dependent on scenarios:

- Communications: Routing, addressing, protocol, publish/ subscribe interactions, asynchronous interactions, event handling, and other features.
- Service Interaction: Interface definition, SOAP, REST, and other features
- Integration: DB, legacy, and middleware connectivity; service aggregation, application server connectivity, protocol transformation, and other features
- Quality of Service: Transactions and delivery assurance, and other features
- Security: Authentication, authorization, non-repudiation, confidentiality, standards support, and other features
- Service level: Performance, throughput, availability, scalability, and other features
- Message Processing: Message and data transformations, intermediaries, content-based routing, and other features
- Management and Autonomic: Service provisioning and registration, logging, metering, monitoring, system management, and other features
- Appliances: Parsing, compression, routing, and other features
- Infrastructure Intelligence: Business rules, policy-driven behavior, pattern recognition, and other features

3.7.4 BizTalk Server

Microsoft BizTalk Server uses adapter technology to connect disparate entities and enable the integration of data, events, processes, and services. An entity may be an application, department, or even an altogether different organization that needed to be able to share information with. A software adapter is typically used when we need to establish communication between two components that do not natively collaborate. BizTalk Server adapters are built with a common framework which results in system integration done through configuration, not coding. Traditionally, BizTalk Server has solved problems in three areas. First, BizTalk Server acts as an EAI server that connects applications that are natively incapable of talking to each other. The applications may have incompatible platforms, data structure formats, or security models. For example, when a new employee is hired, the employee data in the human resources application needs to be sent to the payroll application so that the new employee receives his/her paycheck on time. Nothing prevents you from writing the code necessary to connect these disparate applications with a point-to-point solution. Many organizations choose to insert a communication broker between these applications. Some of the benefits that you would realize from such an architectural choice include:

- Loose coupling of applications where one does not have a physical dependency on the other

- Durable infrastructure that can guarantee delivery, and queue messages during destination system downtime
- Centralized management of system integration endpoints
- Message flow control such as in-order delivery
- Insight into cross-functional business processes through business activity monitoring

BizTalk Server solves a second problem by filling the role of business-to-business (B2B) broker that facilitates communication across different organizations. BizTalk supports B2B scenarios by offering Internet-friendly adapters, industry-standard EDI message schema, and robust support for both channel- and message-based security. The third broad area that BizTalk Server excels in is Business Process Automation (BPA). BPA is all about taking historically manual workflow procedures and turning them into executable processes. For example, consider the organization that typically receives a new order via email and the sales agent manually checks inventory levels prior to inserting the order into the Fulfillment System. If inventory is too low, then the sales agent has to initiate an order with their supplier and watch out for the response so that the Inventory System can be updated.

3.8 SOA Selected Topics

3.8.1 SOA service vs. Webservice

SOA services can be realized as web services, but not all web services are equal to SOA services. Web services represent the use of both a published standard and a set of technologies for invocation and interoperability. SOA services are services that fulfill a key step or activity of a business process and can be described as business services and are often exposed as web services. We can distinguish between SOA service and Webservice in design time, and runtime.

- In design, we identify and specify a service that provides the design, or we identify and specify interfaces that include method specifications. The combination of the definition of the method and the interface at design time is what we refer to as a service from an SOA perspective. Use cases can be used to capture the functional requirements for a service.
- In runtime, In an SOA, business processes, activities, and workflow are broken down into constituent functional elements called services. They can be accessed and used directly by applications, or they can be mixed and matched with other services to create new business capabilities. Business services or SOA services are reusable business capabilities. Examples in banking include open account or change address. For transportation, it might be get reservation or hold reservation, and with loan processing, get loan, apply for loan, and update address are examples of business services.

3.8.2 SOA vs. DCE and CORBA

The difference between SOA and earlier approaches, such as DCE or CORBA, has a lot to do with standardization. This was initially brought about through the use of Web services and Web services standards. WSDL, SOAP, and other Web services standards enable the industry to converge around a common infrastructure model for runtime. Different vendors would have different implementations, but they would generally conform to the base models. The Web Services Interoperability Organization (WS-I) is an example of an effort to pull things together so that different vendor implementations are consistent and compliant to the degree possible. Web services, unlike earlier approaches, built upon existing and deployed infrastructure as it took advantage of the Internet, resulting in less cost for adoption and reduced risk. Agreed upon versus *defacto* standards is a huge difference between SOA and earlier approaches. Web services standards have been the genesis of SOA, and Web services are more language independent than object-oriented technology integration approaches, which are often language specific (e.g., Java, C, or Smalltalk). XML is language neutral and renders naturally into languages of choice: COBOL, C++, Java, or others. XML and Web services standards, such as WSDL, have improved flexibility than approaches such as CORBA or RPC found in DCE, where changes and additions to the data structures often resulted in breakages of the code that used such structures. In contrast, XML does not use offsets, and it is therefore possible to reorder or add data elements without a break in older versions. Web services also use one type space for interfaces, and that type is XML. The other approaches use one type for databases (e.g., SQL), another for in-flight messages (e.g., Internet Inter-Orb Protocol, IIOP), and another Interface Definition Language (IDL; e.g., CORBA). One approach versus three creates an easier-to-use developer toolkit and application programming interface (API) set, and it makes the code base less brittle and easier to change. Contracts that provide a valid sequence of interaction with a service and policies that govern the nonfunctional characteristics of a service have augmented the notion of the interface found in earlier approaches. One of the most important advances that SOA provides is that the set of services that are required by organizations and captured in the service portfolio provides a business language between business and IT to discuss fulfillment of business needs. A service portfolio is governed as an enterprise asset and used by business and IT stakeholders. This represents a big change from CORBA and RPC, which are something IT uses. SOA and services provide much more than programmatic notions; they provide services at a granularity that the business understands and can use.

3.8.3 SOA vs. CBA

The differences between a service-oriented approach and a component-oriented approach to distributed systems architectures are [245]:

1. Services can be offered by any service provider inside or outside of an organization. Assuming these conform to certain standards (discussed below), organizations can create applications by integrating services from a range of providers. For example, a manufacturing company can link directly to services provided by its suppliers.

2. The service provider makes information about the service public so that any authorized user can use the service. The service provider and the service user do not need to negotiate about what the service does before it can be incorporated in an application program.

3. Applications can delay the binding of services until they are deployed or until execution. Therefore, an application using a stock price service (say) could dynamically change service providers while the system was executing.

4. Opportunistic construction of new services is possible. A service provider may recognize new services that can be created by linking existing services in innovative ways.

5. Service users can pay for services according to their use rather than their provision. Therefore, instead of buying an expensive component that is rarely used, the application writer can use an external service that will be paid for only when required.

6. Applications can be made smaller (which is important if they are to be embedded in other devices) because they can implement exception handling as external services.

7. Applications can be reactive and adapt their operation according to their environment by binding to different services as their environment changes.

3.8.4 WSDL vs. REST

SOA is an architectural style, and that the use of Web services is only one way to implement or realize this architectural style. In fact, there may be a set of different implementations of a service portfolio. Suppose, for instance, that the service portfolio has 100 services; 50 of them may be implemented using Web services, and the rest may be implemented using a combination of Representation State Transfer (REST) and Simple Object Access Protocol (SOAP). Or, an architectural decision can be made to use other Java or .NET mechanisms that do not use REST or SOAP. SOA does not constrain a solution to using Web services, SOAP, or REST. However, using Web services is a best practice, and using WSDL is the fundamental aspect of what makes a Web service not SOAP or REST. There are different patterns for SOA implementation patterns. Not all of them promote SOA best practices. Pattern 1 is a tightly coupled interaction, which does not use a Services Layer but may use a service interface in the operational system. In this pattern, it's unlikely that SOAP or REST is used, and most likely a proprietary messaging interface is the transport of choice. In pattern 2, a service component is used with a service interface but no Services Layer is in place. In pattern 3, a Services Layer is present, but the service maintains state in its interactions with the service component and Operational Systems Layer. In pattern 4, a packaged solution provides a service interface and most likely makes REST or SOAP options available. In pattern 5, a business state machine might be in use directly interacting with service interfaces. In pattern 6, a Business Process Layer and Services Layer are leveraged. The primary reason

for illustrating the various SOA interaction patterns is to illustrate that there is no right or wrong choice when picking SOAP or REST. Instead, choices may be architecturally weak or strong based on the interaction pattern needed, the quality of service (QoS) attributes that must be achieved, the examination of any constraints imposed by existing operational systems, and the determination of where the greater flexibility lies. One can then make an architectural choice: SOAP, REST, or another interaction. The use of SOAP often is equated as using Web services when in fact Web services can also be REST-based Web services. In SOA adoption, the interaction patterns are architectural decisions along with whether to use SOAP, REST, or other service.

3.8.5 Fundamental Types of Services

There are three fundamental types of service that may be identified [122]:

1. Utility services These are services that implement some general functionality that may be used by different business processes. An example of a utility service is a currency conversion service that can be accessed to compute the conversion of one currency (e.g., dollars) to another (e.g., euros).

2. Business services These are services that are associated with a specific business function. An example of a business function in a university would be the registration of students for a course.

3. Coordination or process services These are services that support a more general business process which usually involves different actors and activities. An example of a coordination service in a company is an ordering service that allows orders to be placed with suppliers, goods accepted, and payments made.

3.8.6 REST

REST is an architectural style that uses the analogy of the state transition diagram and maps that to an application having a set of resources that are connected by state transitions or links. A network of nodes such as the World Wide Web provides an ability to click a link and be transferred to the resource for that link. For example, if you are on a Web page that is a representation of a resource, you are then considered to be in a particular state when you are on that Web page. By clicking a link within that initial Web page that represents the current state, you convey the intent to transition to another Web page, which is really a resource representation of the next state. Thus, a network of states and transitions is traversed by selecting the appropriate link to the appropriate resource. This resource representation puts you, accessing the resource, into a particular state. When you click a link on that page, you get a representation of another resource. When you continue to access various resources by following the links, you keep changing state. This is essentially what is meant by REST.

3.8.7 Web services vs. REST

There are three main differences between Web services and REST:

- Service oriented versus resource oriented. With Web services, you are requesting a service (and so this is service oriented). With REST, however, you are implying you are looking for a specific type of resource (and so this is considered to be resource oriented).
- Use of HTTP. REST typically uses HTTP as the transport, whereas SOAP has no restriction on the use of a particular transport. Many people use REST-style interactions using HTTP.
- Quality of service. When you use REST, all the QoS parameters must be provided by the transport itself. In Web services, a significant number of specifications must be written to support QoS options, using the WS-*set of standards.

3.9 Summary

A web service is an instance of a more general notion of a service, which is defined as: "an act or performance offered by one party to another. Although the process may be tied to a physical product, the performance is essentially intangible and does not normally result in ownership of any of the factors of production". The essence of a service, therefore, is that the provision of the service is independent of the application using the service. Service providers can develop specialized services and offer these to a range of service users from different organizations. SOA are a way of developing distributed systems where the system components are stand-alone services, executing on geographically distributed computers. Standard XML-based protocols, such as SOAP and WSDL, have been designed to support service communication and information exchange. Consequently, services are platform and implementation-language independent. Software systems can be constructed by composing local services and external services from different providers, with seamless interaction between the services in the system. This chapter presented different topics related to SOA, including: SOA advantages, technologies, different methods of implementation, examples of IS based on SOA, and concluded with an important concept, that is ESB.

3.10 Review Questions

1. What is SOA?
2. What are SOA advantages?
3. What are SOA main technologies?
4. What is Webservice?

5. What are Webservices advantages?
6. What is Webservice Enhanced Standards?
7. What is ESB?
8. What is REST?
9. What are the fundamental types of Services?

3.11 Exercises and Labs

In Labs, you shall be getting familiar with Java programming language, as it is the programming language we will be using to build, deploy, and consume different online Webservices and SOA systems presented in this book.