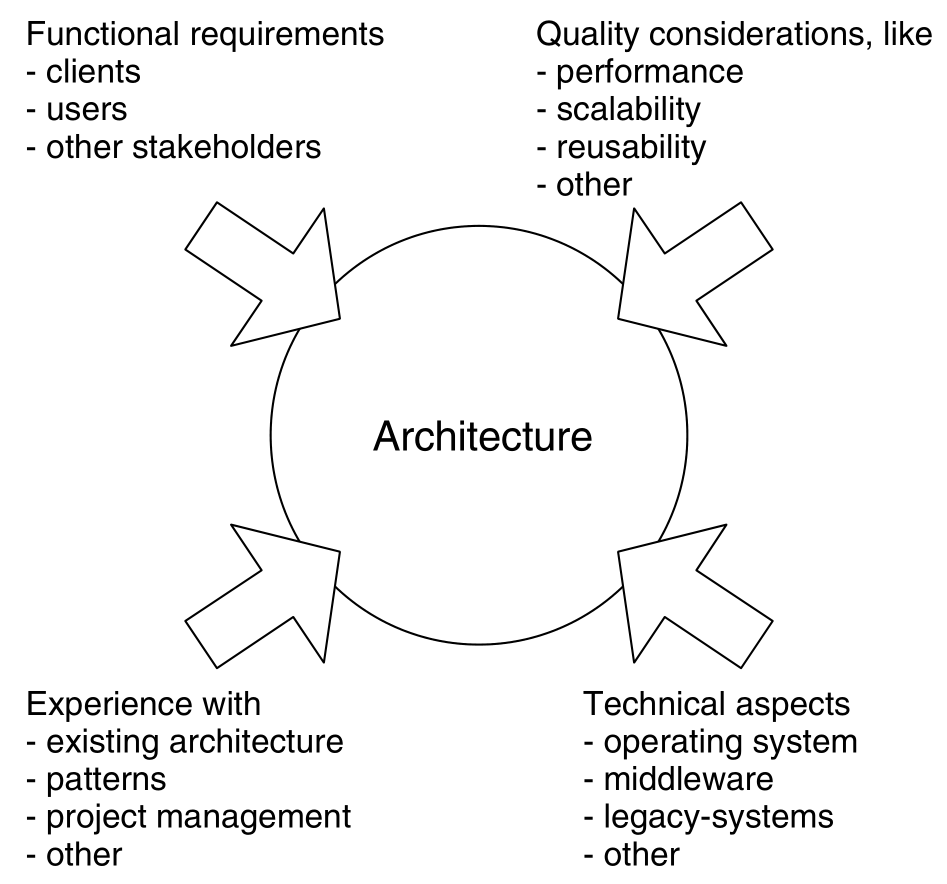
**Architectures for the Web**

Architecture is defined as the structure of components, their relationships, and the principles and guidelines governing their design evolution over time. According to (Bass et al. 1998), the architecture of a software system consists of its structures, the decomposition into components, and their interfaces and relationships.

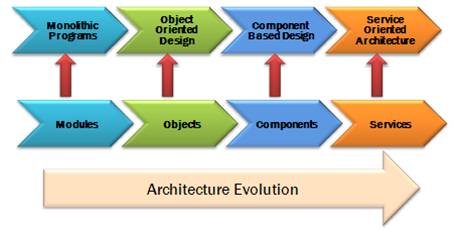
When we create architecture we try to break the functional requirements and quality requirements down into software components and their relationships through interfaces using an iterative approach. Depending on the point of view, we can emphasize and itemize different architectural aspects. Structuring software systems and breaking them down into different perspectives allows us to better manage the complexity of software systems, and the systems become easier to understand. In addition, the abstraction of system aspects facilitates the outline of possible and important architectural issues.

Considering the above properties of architecture we can easily see that architectural decisions are of enormous importance for the development of Web applications. The requirements of software and thus its architecture are subject to change. Technical and organizational constraints change during and after the development of an application. This is the reason why software systems are often referred to as “moving targets”. Due to the changes in capability and performance objectives, interests of different stakeholders, technology improvements, and unanticipated situations, existing architectures may need to evolve to meet new capability requirements and constraints. The following figure shows the different factors and constraints influencing the development of an architecture according to (Jacobson et al. 1999).



Factors influencing the development of an architecture

The evolution of distributed architecture can be considered a sophisticated extension of client/server architecture because the Internet and World Wide Web caused a shift in the nature and ways of business transactions in place today. With this evolution, especially when it concerns to the Web, came a large quantity of services and tools. As these services and tools grew up in uses and complexity they came to be the subject of software engineering, a topic that was well developed by the time this applications became used. The evolution of architectures could be imagined with the following figure:



Evolution of Architectures

A number of architectures for specific requirements in several application domains have been developed in the past few years. The most widely used aspect is the layered view. Layering means that software systems are structured in several tiers to implement the principle of “separation of concerns” within a software system. Many frameworks in the field of distributed systems and Web applications are primarily structured by the layering aspect, which will be discussed in the next chapter.

The increasing distribution of software systems has led to the development of architectures and infrastructures addressing the distribution of data and messages. We could found several proposals to achieve an efficient way in this distributed nature. The most well-known architectures are:

* Distributed Object Middleware (DOM): This type of infrastructure allows to access remote objects transparently. It is based on the Remote Procedure Call (RPC) mechanism. Some DOM systems also enable objects on different platforms to interact (e.g., CORBA). Other examples of this type of system include Microsoft’s DCOM (Distributed Component Object Model), or EJB (Enterprise Java Beans) by Sun Microsystems.
* Message Oriented Middleware (MOM): MOM systems offer functionalities for asynchronous transmission of messages. Asynchronous communication are sent to the receiver regardless of its status, e.g., the receiver may not be available when the message is sent. MOM ensures that messages are delivered nevertheless. Examples of MOM systems include Sun’s JMS (Java Messaging Service) and Microsoft’s MSMQ (Microsoft Message Queue).
* Peer to Peer (P2P): P2P stands for direct communication between two devices – the peers – in a system without using a server, i.e.,they communicate over a point-to-point connection. The peers are basically equal. P2P systems describe how the devices in such a network communicate and how they can “discover” each other.
* Service Oriented Middleware (SOM): SOM enhances DOM systems by the concept of services. A service in this context is a number of objects and their behavior. These objects use a defined interface to make a service available for other systems/services. SOM defines communication protocols between services, and provides for location- and migration-transparent access to services, thus supporting a simple integration of services beyond platform boundaries. One example of a SOM is Sun’s Jini system ( http://www.sun.com/software/jini/ ). Architectures emerging within the field of Web services also belong to this category.

These architectures are applicable to distributed systems in general, which means that they are not limited to Web applications. Similarly, integration architectures address integration aspects on the content level and the application logic level and are commonly summarized under the term Enterprise Application Integration (EAI) architectures. This category also includes architectures which integrate existing applications as a whole. Alternatives to EAI are Web services which support the integration of services, i.e., application logics and contents. On the presentation level, a set of different systems is typically integrated by using portal architectures.

**Table of Contents**

[Layered Architecture for Web Applications](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html)

[The Three Layers Model](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60022128)

[The View Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60031632)

[The Business Logic Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60038832)

[The Data Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60050832)

[The MVC pattern - useful but not a silver bullet](https://gyires.inf.unideb.hu/GyBITT/08/ch04s02.html)

[The Layered Architecture and the MVC Design Pattern](https://gyires.inf.unideb.hu/GyBITT/08/ch04s02.html#idp60094672)

[Architectures for Enterprise Level](https://gyires.inf.unideb.hu/GyBITT/08/ch05.html)

[Service-oriented architecture](https://gyires.inf.unideb.hu/GyBITT/08/ch05.html#idp60121168)

[Side note about Web-oriented architecture](https://gyires.inf.unideb.hu/GyBITT/08/ch05.html#idp60158000)

[Representational State Transfer (REST)](https://gyires.inf.unideb.hu/GyBITT/08/ch05s02.html)

[REST and RESTful](https://gyires.inf.unideb.hu/GyBITT/08/ch05s02.html#idp60231376)

[Portal architecture - one of the SOA variants](https://gyires.inf.unideb.hu/GyBITT/08/ch05s03.html)

[Web Services](https://gyires.inf.unideb.hu/GyBITT/08/ch06.html)

[Web Services Description Language (WSDL)](https://gyires.inf.unideb.hu/GyBITT/08/ch06.html#idp60259536)

[Universal Description, Discovery and Integration (UDDI)](https://gyires.inf.unideb.hu/GyBITT/08/ch06s02.html)

[SOAP Web Services](https://gyires.inf.unideb.hu/GyBITT/08/ch06s03.html)

[SOAP vs REST](https://gyires.inf.unideb.hu/GyBITT/08/ch06s04.html)

## Layered Architecture for Web Applications

**Table of Contents**

[The Three Layers Model](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60022128)

[The View Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60031632)

[The Business Logic Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60038832)

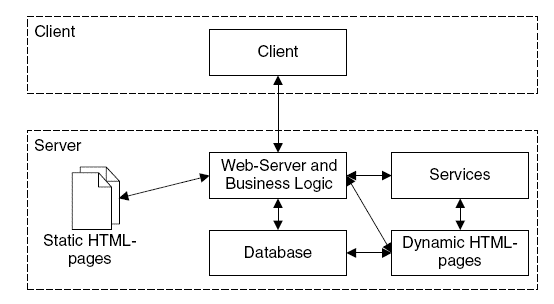
[The Data Layer](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60050832)

[The MVC pattern - useful but not a silver bullet](https://gyires.inf.unideb.hu/GyBITT/08/ch04s02.html)

[The Layered Architecture and the MVC Design Pattern](https://gyires.inf.unideb.hu/GyBITT/08/ch04s02.html#idp60094672)

The first question that should be answered: Why the web is suitable for developing applications? It is not a difficult question, of course. In the World Wide Web Consortium (W3C) Architecture of the Web Recommendation paper various examples are given, notably one about a user who wants to see the weather in a place where she wants to travel to. The nature of the Internet - trafficking data over protocols from network to network - is a powerful resource that make communication between different places very fast and easy. This way computer programs can be made to improve the relation between systems and people, and what is seen today is that it happened.

Web applications are distributed applications, and hence are at least two-tiered. They act as a special kind of client-server applications, where a large portion of the functionality is “pushed” back to the server side despite the fact that the Web does not define what is behind the server.

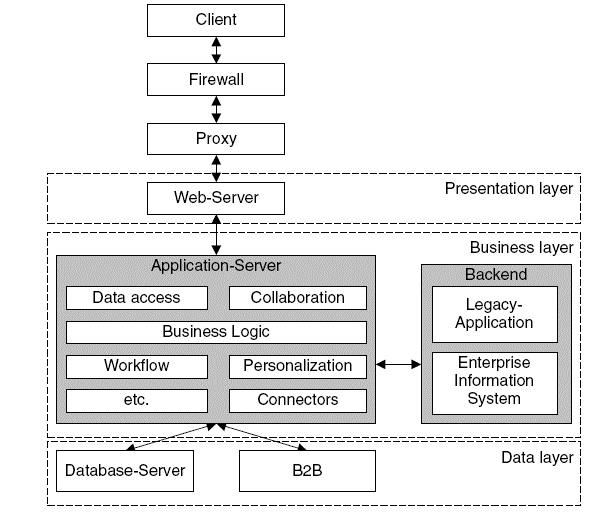


The basic two parts for Web Applications

The Web relies strongly in the client-server model, and it uses markup languages such as HTML and XML to transfer and represent data. Under it there are many programming and scripting languages that can dynamically process, modify and generate data, or give an user interface. This way, the development of Web applications can be put under the cover of software engineering but need to be extended. Web applications are multidisciplinary (software engineering, database modeling techniques, network computing, and effective interface design). They are built in a continuously changing environment where requirements are unstable and the user community is wider than before. Web applications handle information from various sources (text, graphics, video, audio) dealing with structuring, processing, storing and presenting this information.

## The Three Layers Model

The nature of the Web is layered: it has formats over protocols and uses a client-server model. Therefore, it is natural that a layered architecture would be suitable for developing to the Web. We learnt that this model overcame the two layered client-server because of its scalability. Many different approaches to the aim of developing applications with different layers had been used along the years, but a clear pattern seems to appear frequently in various of them: the Three Layers Model (according to Kappel et al. 2006).



The standard three layered architecture for Web Applications

This model of web application development is very similar to the Service Layer/Domain Model/Data Source Layer set of design patterns from Martin Fowler’s collection, but receiving different names . In fact, the idea ( usually named 3-tier architecture, or expanded into n-tier architecture) is very general and widespread, so in this paper only the most common assumptions and uses are examined.[[1]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html" \l "ftn.idp60027408)

Its conception is three layers, one over the other, being the application the set of them working together. The most external of them is the View Layer, that is the visible part of the application, the one that interacts with the user. The layer in the middle is the Business Logic Layer, which serves as an intermediate between the View (or presentation) and the innermost layer, that is the Data Layer. This last one is where all the data used by the application is stored.

The benefits of using layered models in software development are in the way that it is easier to know exactly what each part of the application does. It makes easier to construct the application, to debug it, and to maintain and reuse the code. For example, if it is needed to exchange some details in the presentation of the content but the business rules and data models do not change, only one layer is affected. Even for more complicated changes involving all of the application architecture there are benefices, so a plan can be created in the overall but specifying exactly where the changes need to be done.

### The View Layer

The outermost layer in this kind of model deals with the presentation of the content and interaction with the user. It can be called view, presentation, UI. In this layer the application shows to the user what is needed to be seen and gives the tools for interaction. The exact kind of interaction depends on the application; one can create a web app that only shows information to the user without any kind of interaction, not even hyperlinks to be clicked, but such a case does not need an advanced architecture. In most cases the user will generate some input, send it for processing and then receive a feedback, that can be the final result or a step for further operations.

Following the example by W3C of the user that wants to see the weather in her trip destination, the presentation is where she sees the actual content. The content display is shown, and the user can interact with the provided controls to, for example, see weather in different periods of time or another places, and see pictures of it.

The technologies usually involved in this layer on the web development context are mainly the markup that is processed by the browser (HTML/ XHTML/ ...), the style of the page (CSS) and client-side scripts (Javascript/ Flash/ ...). All of these tools together can produce a rich environment for user interaction and content display. Of course it can be said that server-side scripts can be used to generate content, but at the final level these scripts produce the HTML that will be shown be the browser, so this role of the development can be subdivided: the content generation is created by the business logic layer (the next topic to be discussed) and then it is passed to the view layer, maintaining the logical division of the application. The browser shows the content initially written or produced by the server-side scripts, and the client-side scripts are able to modify that content. A Javascript code, for example, can be used to validate form data or even to create drag and drop interfaces. It interacts with HTML through a DOM tree, that is a representation of the document in memory. HTML5, the present (2014) trend for web development is praised for its flexibility, specially where it touches the concept of responsiveness, that is the ability to change the content disposition according to the screen size. This matters because, in current days, the availability of a page in different screen sizes and devices is extremely important. Having many possibilities like desktops, tablets, smartphones, wearable devices and even augmented reality or voice user interface, the range of technologies and targets for the view layer is very wide, and it shows both the importance of it to the user and reinforce the need of a logical division of the application for supporting such variety.

This layer communicates with the business logic layer under it, passing the information from the user and controlling it, then giving back any response it produces, not leaving any decisions of the application’s logic to be resolved by the UI. This passing of information is usually done through forms, like a user log-in in a system by giving username and password, but there are other ways. AJAX is an asynchronous way to pass information to the server and get responses. The cited a-synchronicity comes from the fact that in a form the content needs to be passed and then the response will come after a page refresh, but with AJAX the requested information, that is the result of the user’s action will come in the actual page. It saves time and gives to the user the impression that the application is really interacting with him.

### The Business Logic Layer

The central layer of the model deals with the logic of the program. It receives data from the upper level and transforms it, using in the inner application logics. It also retrieves data from the deepest data level and uses it to the logics. And, by integrating these two processes, it can do modifications in both levels as well.

The Business Logic Layer contains the determinant part of the application logic. It includes:

* performing all required calculations and validations
* managing workflow
  + state management: to keep track of application execution
  + session management: to distinguish among application instances
  + user identification
  + service access: to provide application services in a consistent way
* managing all data access for the presentation layer

The Business Logic Layer is generally implemented inside an Application server (like Microsoft Transaction Server, Oracle Application Server, or IBM WebSphere). The Application server generally automates a number of services like transactions, security, persistence, connection pooling, messaging and name services.

Using the same example of the last session, of the user that wants to see the weather in a specific place, when the information is given by the user the application retrieves it and process. For example, the user wants to see the weather forecast for two days. The application receives its request from the UI and the data is sent to the server. A PHP script catches it and then make the calls for the lower level to get the needed data. When a response comes, being it the desired information or a failed request, it is dealt and then prepared to be sent again to the upper level.

The tools used in this level are usually server-side scripts like PHP,ASP.NET, Ruby or CGI. There is even a solution that uses server-side Javascript, called node.js. These technologies, following the information feeding that comes from the upper level, can do any computational pro- cessing that is needed. The CGI (Common Gateway Interface) scripts are an older technology that defines communication way between a server and the content-generated program, in this context called CGI script. One of the most remembered languages when talking about CGI scripts is Perl. The other languages here cited have a similar approach, by running inside a server. PHP is related to Perl, being as well a scripting language and having similar philosophies. It is one of the most popular languages, being the implementation language of important content management systems as Drupal or Wordpress. Ruby have a large popularity too, especially with the framework Ruby on Rails. Applications as Github or Redmine are built using it. There are many others, of course, and different uses of them, one example being C used as CGI or the Java Server Pages (JSP).

### The Data Layer

The deepest level in the layered architecture, the data layer deals with data retrieval from its sources. It is an abstraction to get the plain data, that can be in a wide variety of forms. Once again, it plays a huge role on the reusability and exchange of technologies: if one data source is changed to another, but the proper data is still the same, a good layered design can help by providing the same data to the upper level with the same interfaces, changing only its inner logic.

In the example given in this paper of the weather forecast, the requirement by the user for the next days forecast will come to this level as a request for the forecasts that it may have. Then a search will be made in the data using the given parameters, and then the data (or some information about not getting it) will be sent again to the upper level.

The technologies used in this layer are database management systems like MySQL or PostgreSQL for relational databases, NoSQL management systems like MongoDB or Apache Cassandra, or even plain XML files or text files. For the management systems usually an API will be used for making queries and retrieving data, and for the plain text ones a script will do the needed operations. Inside it there can be any level of sophistication desired by the application designer, so there can be integrity checks, stored procedures, and virtually anything needed to maintain the data in the desired state.

**Deepest in the Data Layer: NoSQL and NewSQL**

Inside the Data Layer, as it was outlined, many different technologies can be used. Most of the web applications currently active use relational databases, but now the market is seeing a change of paradigm in the form of the NoSQL. NoSQL is a general way to identify non-relational databases. Fowler summarises some common characteristics that NoSQL databases share:

* Not using the relational model
* Running well on clusters
* Open-source
* Built for the 21st century web estates
* Schemaless

The key points NoSQL supporters use to justify the need for it is that relational databases are not the best solution for any kind of problem, being a problem of its own the uses. They say it is heavy, slow, and non-scalable to use the relational databases, so the use of NoSQL can be a good way to solve these kinds of problems. The use of NoSQL nowadays seems related to startups that use innovating new technology and social web services such as Facebook and Amazon, that have a great amount of data[[2]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html" \l "ftn.idp60061760) to deal with and have the need to find new ways to use it.

In fact, that is this demand of large data processing. Under the label of big data lies the concept of large quantities of data generated in the last few years and from different sources and in a variety of different formats. The processing of this kind of data leads to a wide range of uses, from healthcare to criminalistics inferences. Of course, new challenges arises with this perspective. The drawbacks come from the nature of the data - massive, disperse, heterogeneous. This is why NoSQL can be seen as a solution - it thinks about this kind of problem, trying to solve it.

As of 2014, there are four important categories of NoSQL databases:

* key-value stores, that are basically hash tables
* column family stores, which aim is to deal with vast collections of data spread amongst many different machines
* document databases, versioned documents that are collections of other key-value collections
* and graph databases, where the data is presented as a graph, and then it is possible to divide easily into different machines and the queries are more data-specific than the relational ones

A topic that attracts attention when it comes to the issue of scalability and performance of databases is the so-called NewSQL. It is more a way to recognise “ the various emerging database products at this particular point in time”. The authors writing about it use the term as an identification of vendors that provide SQL databases that are high-performance and scalable, in the market that is also aimed by NoSQL providers. The aims of NewSQL are also related to the Big Data paradigm.[[3]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html" \l "ftn.idp60072896)

[[1]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60027408)Sometimes tier is different from layer. For it, layers mean the logical distinction of the applications integral parts, and tiers mean the physical structures in where the application runs (networks, computers, servers). But in this work the two words are used as synonyms, and with the meaning of layer, not being the objective to discuss physical tiers.

[[2]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60061760)“The increasing amount of data in the web is a problem which has to be considered by successful web pages like the ones of Facebook, Amazon and Google. Besides dealing with tera- and petabytes of data, massive read and write requests have to be responded without any noticeable latency”

HECHT, Robin and JABLONSKI, Stefan. NoSQL Evaluation, A Use Case Oriented Survey.

[[3]](https://gyires.inf.unideb.hu/GyBITT/08/ch04.html#idp60072896)“The NoSQL projects were developed in response to the failure of existing suppliers to address the performance, scalability and flexibility requirements of large-scale data processing, particularly for Web and cloud computing applications. NewSQL and data-grid products have emerged to meet similar requirements among enterprises, a sector that is now also being targeted by NoSQL vendors.”

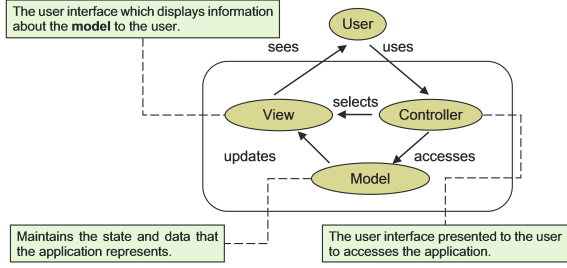
ASLETT, Matthew. NoSQL, NewSQL and Beyond: The answer to SPRAINed relational databases.

## The MVC pattern - useful but not a silver bullet

Design patterns try to suggest the way of the application design, while methodologies try to give suitable models for the application and its whole lifecycle. The main idea behind design patterns is to extract the high level interactions between objects and reuse their behaviour from application to application. Moreover, design patterns help to clarify the way that we can think about a Web application.

The MVC architecture has its roots in Smalltalk, where it was originally applied to map the traditional input, processing, and output tasks to the graphical user interaction model. However, it is straightforward to map these concepts into the domain of multi-tier Web applications. It can improve the application’s usability, creating reusable code and helping to understand and clarify the functionality of the program. The MVC pattern is very simple, yet incredible useful. It could support:

* Efficient modularity: allows swapping in and out any of the components as the user or programmer desire. Changing one aspect of the program are not coupled to other aspects
* Reusability: it can support the reuse of previously created code if we act sensibly and design carefully.(Reduces risks of bugs coming from refactoring)
* Ease of growth: controllers and views can grow as the model grow
* Centralized controller: a main module is used to make control more manageable

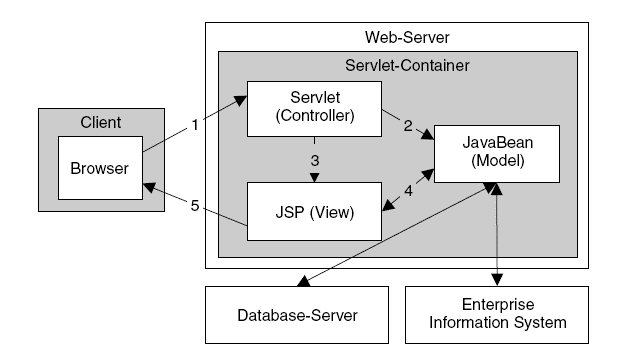


The MVC architecture

* **Model**: The model represents enterprise data and the business rules that govern access to and updates of this data.
* **View**: The view renders the contents of a model. It accesses data through the model and specifies how that data should be presented. It is the view's responsibility to maintain consistency in its presentation when the model changes.
* **Controller**: The controller translates interactions with the view into actions to be performed by the model. In a Web application, they appear as GET and POST HTTP requests. The actions performed by the model include activating business processes or changing the state of the model. Based on the user interactions and the outcome of the model actions, the controller responds by selecting an appropriate view.

Although MVC is undoubtedly a valuable and useful way to design Web applications, but not the only one. The MVC design pattern’s importance lies in how could it help to achieve a clear separation of concerns and functional layers. Create a module for the database portion, create another module for the application code, and create a third module for the presentation code. That way, we can swap and change different elements at will, hopefully without affecting the other parts.

Several vendors have applied this design pattern in their solution. As we have seen that the Web application term appeared in the servlet definition from Sun, here we will show the classic version of the MVC pattern's usage in that context (according to Kappel et al. 2006).



The JSP-Model-2 architecture

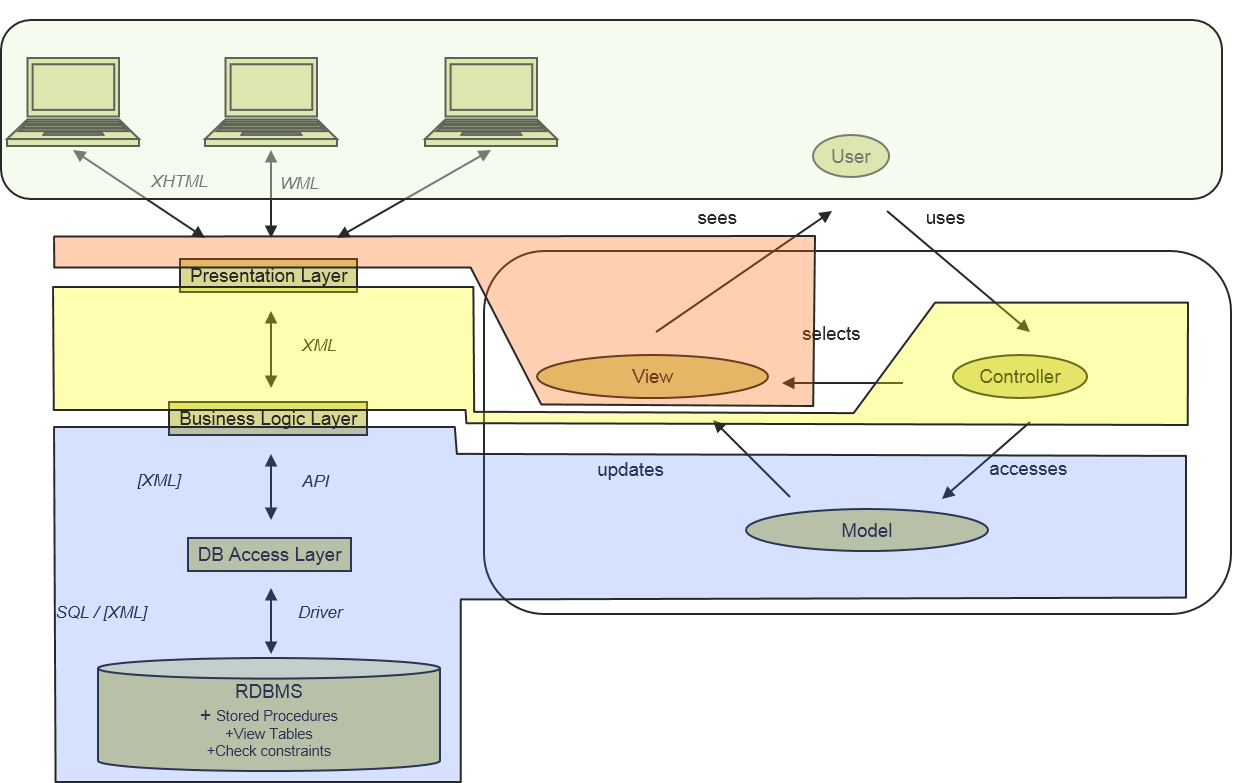
### The Layered Architecture and the MVC Design Pattern

In the context of Web applications, by concentrating too closely on applying the MVC design pattern and nothing else, many other important aspects may be overlooked. This could lead to a fragmented and fragile solution, and as a result, is hard to maintain and execute further development. The MVC is a well known and widely used design strategy, but the problem is, how can we adopt this pattern into the layered model of Web applications?

At first sight the answer could say that we have three layers and three modules, the View module is equivalent with the Presentation layer, the Model module is equivalent with the Data layer, and the Controller module is equivalent with the Business Logic.

But if we take a closer look in the functionality of the Model and the Controller – discussed earlier in the paper – we could find that the Model represents business rules, and the Controller translates the interactions ( HTTP GET, POST requests).

These suggest that the Controller is only a part of the Business Logic Layer and the Model forms the other part of the Business Logic Layer. We must take under consideration this heterogeneous composition of the Business Logic Layer through the development of a Web-based application.



MVC and the Layers

It should be obvious by now that MVC or any other design pattern is no more a silver bullet by itself than object-oriented programming is. It's just one part of a much bigger system. We know that MVC has several advantages:

* Clean separation of different functional layers
* Reduces maintenance costs
* Reduces risks of bugs coming from refactoring, graphics redesign
* Presence of a central controller raises overall software security level
* Controller can centrally perform tasks like access logging – without central controller such a task would affect the source code if all business logic actions.

but we also know the drawbacks:

* More software design cost (short term)
* More implementation cost
* Programmers cannot use some comfortable system features
* Programmers may feel they are forced to program a complicated way instead of quickly implementing.

## Architectures for Enterprise Level

**Table of Contents**

[Service-oriented architecture](https://gyires.inf.unideb.hu/GyBITT/08/ch05.html#idp60121168)

[Side note about Web-oriented architecture](https://gyires.inf.unideb.hu/GyBITT/08/ch05.html#idp60158000)

[Representational State Transfer (REST)](https://gyires.inf.unideb.hu/GyBITT/08/ch05s02.html)

[REST and RESTful](https://gyires.inf.unideb.hu/GyBITT/08/ch05s02.html#idp60231376)

[Portal architecture - one of the SOA variants](https://gyires.inf.unideb.hu/GyBITT/08/ch05s03.html)

Extensive advancements in enterprise computing has been taking place in the last decade and with the advent of the Web it is now possible for organizations to automate and integrate businesses and computer operations. Almost all enterprise organizations face the problem of integrating different applications and database systems at some point. Research has shown that during software development, a third of the time is dedicated to the problem of creating interfaces and points of integration for existing applications and data stores. In situations like this enterprise application architecture becomes very important. **Enterprise application architecture** allows an enterprise to integrate its existing applications and systems and to add new technologies and applications to the mix.

With the emergence of the Web EAI has gone beyond just merging applications within enterprises. Provision of services through the web has rapidly become a trend. External or internal systems, e.g., existing applications, existing databases and interfaces to external business partners, can be integrated into Web applications. There are two patterns that EAI systems implement:

* Mediation (intra-communication):

Here, the EAI system acts as the go-between or broker between multiple applications. Whenever an interesting event occurs in an application (for instance, new information is created or a new transaction completed) an integration module in the EAI system is notified. The module then propagates the changes to other relevant applications.

* Federation (inter-communication):

In this case, the EAI system acts as the overarching facade across multiple applications. All event calls from the 'outside world' to any of the applications are front-ended by the EAI system. The EAI system is configured to expose only the relevant information and interfaces of the underlying applications to the outside world, and performs all interactions with the underlying applications on behalf of the requester.

Both patterns are often used concurrently. The same EAI system could be keeping multiple applications in sync (mediation), while servicing requests from external users against these applications (federation). Enterprise Application Integration is related to middleware technologies such as message-oriented middleware (MOM), and data representation technologies such as XML. Other EAI technologies involve using web services as part of service-oriented architecture as a means of integration. Enterprise Application Integration tends to be data centric. In the near future, it will come to include content integration and business processes. For our point of view, the service-oriented architecture is the one which plays an important role, while this is the most frequently used in Web context.

## Service-oriented architecture

In the IT industry a frequently used buzzword is ‘Service Oriented Architecture (SOA)’. The Web, IT literature and other resources provide different definitions and interpretations on SOA. Some refer to it as a “Sophisticated Product”; some as an “Architectural Approach” while others treat it as a mere marketing gimmick. Service-oriented architecture is somehow like a design pattern that consists of discrete pieces of software with some functionalities and other applications can utilize these functionalities. This pattern does not require us to use some specific product or a platform. A service provides some functionality and this can be used by other large software applications to complete its use.

In layman terms, Service Oriented Architecture provides a framework where different (Heterogeneous) platforms (Windows, Linux, and UNIX), technologies (.NET, Java) and applications (ERP, CRM, and SCM) operate in synchronization. More specifically, this is achieved through “Services” and “Web based Open Standards”.

Service is the base of this architecture. Services are the structures which have ability to interact with each other. In other words they are the listener of the other side of the phone which is an endpoint.

In the classical layered architecture layers interact with each other. This interaction and hierarchy of this system should be constructed very properly for this system to work proper. Therefore we can say that SOA simply means that these layers are created as services. For example if we make a service which provides us the data, then we have accomplished the functionality of data layer by this. Afterwards we can also use this service from other applications as well. Achieving this we have a flexible architecture to use.

**Why SOA?**

Reusability

This property of SOA is the most important benefit of this architecture. Base of this architecture is constructed with services so that it provides us functionality to multiple applications or clients. The purpose is actually to reuse them.

Connecting Heterogeneous Systems

Nowadays in our software world its almost impossible to create a common infrastructure. What I mean by this is technologies can be different and even if they are same the versions of them might be different as well. Thus SOA provides us a good communication quality.

Loose Coupling

Services are not connected with classes or libraries they are connected with contracts and endpoints. So that they are loosely coupled. A service which obeys the same schema structure can replace another service very easily. To give an example for this we can say that a Java service can be replaced with a .NET service or another service which obeys the same schema.

Facade

This architecture can be use to simplify interface of complex structures or complex services. This type of approach is used when the service is created to simplify technical usage for other developers to use. It can be also used to separate the service from work flow. It is enough to expose only the relevant data to the consumer.

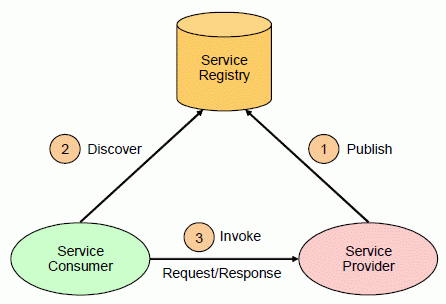
Abstraction

Every service is responsible for their own functionality. For example : If you want your service to produce passwords, than it only produces passwords while another service checks the username and password. In this way application is partitioned into simpler parts. This increases the readability and reduces the time of maintenance.

**Where to use SOA then ?**

This architecture should be preferred when there are multiple heterogeneous systems, applications or consumers. Otherwise achieving flexibility is an additional cost. Consider the scenario where Enterprise systems have different owners and multiple teams working on them. In order to improve visibility and accessibility, centralized control is required. The Service Oriented Approach is appropriate in such a scenario as it provides complete information about the system components through its Service Registry.

The common service-oriented architecture is illustrated in the following diagram:



Service-oriented architecture

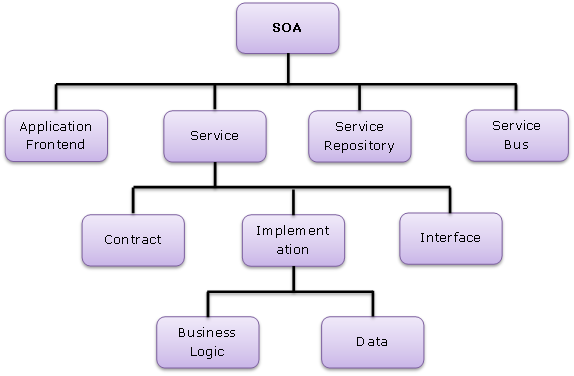
Service provider is responsible for providing the services and the details of the services. The service provider can decide whether the service needs to be secured or can be used by anyone. The cost implication and the traffic of using the service needs to be thought about. There will be an interface provided by the service provider so that any service that needs to be accessed can be achieved with the help of the interface.

The service provider can decide whether the services needs to be listed or not and what should be the agreement that should be set between the consumer for accessing the services. Most of the cases the service provider publishes the interface and the access details with the Service Registry

Service Registry or broker implementers should carefully think and decide on the implementation and access strategy of the services by the service consumers. The ownership of making the service interface public resides with the service registry. The implementers should consider about the scope that is involved. There are pubic service registries that can be accessed over the internet and there are private service registries which is accessible to only restricted or role based consumers

Service consumer ensures that it’s possible to locate the service that is registered in the service registry and binds to the service provider is obtained. The service consumers invoke the service that is defined.

**Elements of SOA:**



SOA elements

These architectures have a number of benefits, including quicker and more cost-effective responses to changing market conditions. SOAs can also simplify connections to legacy IT systems. SOAs also have a number of challenges in implementation, such as managing the large number of services and how they interact with each other. There are also concerns about lack of testing and security levels of services. Overall, service-oriented architecture is a widely-used technology that is changing the way large businesses function today.

**Principles of SOA:**

1. Explicit Boundaries

2. Shared Contract and Schema, not Class

3. Policy-driven

4. Autonomous

5. Wire formats, not Programming Language APIs

6. Document-oriented

7. Loosely coupled

8. Standards-compliant

9. Vendor independent

10. Metadata-driven

The following figure is a good description about how could we imagine a completely service-oriented model:

|  |
| --- |
|  |

SOA overview

### Side note about Web-oriented architecture

### Note

According to Gartner's definition of Web-Oriented Architecture (WOA) from 2008:

WOA is an architectural substyle of SOA that integrates systems and users via a web of globally linked hypermedia based on the architecture of the Web. This architecture emphasizes generality of interfaces (UIs and APIs) to achieve global network effects through five fundamental generic interface constraints:

1. Identification of resources
2. Manipulation of resources through representations
3. Self-descriptive messages
4. Hypermedia as the engine of application state
5. Application neutrality

Resource identification could be done by uniform resource identifier (URI), resource manipulation by HTTP, application state by links and self-describing messages could be identified by Multipurpose Internet Messaging Extensions (MIME) types.

If we familiar with REST than we can feel this is essentially a rehash of REST except for the fifth point on "application neutrality". After some back-and-forth discussions on the REST-discuss list [WOAvsREST], it turned out that "application neutrality" meant using generic media types such as the Atom Syndication Format for representations. Media types should be able to convey the type of representation, and by using generic types, applications lose describability.

## Representational State Transfer (REST)

Representational State Transfer is an architectural style that build on certain principles using the current web fundamentals. It generally runs on HTTP. It makes a stateless transfer. REST ignores the details of component implementation and protocol syntax in order to focus on the roles of components, the constraints upon their interaction with other components, and their interpretation of significant data elements. REST has been applied to describe desired web architecture, to identify existing problems, to compare alternative solutions, and to ensure that protocol extensions would not violate the core constraints that make the Web successful. Fielding used REST to design HTTP 1.1 and Uniform Resource Identifiers (URI).

The REST architectural style is also applied to the development of web services as an alternative to other distributed communication types such like SOAP. REST is often used in mobile applications, social networking Web sites, mashup tools and automated business processes.

There are 5 basic fundamentals of REST services which are created for the Web.

1. Everything is a Resource.
2. Every Resource is Identified by a Unique Identifier.
3. Use Simple and Uniform Interfaces.
4. Communication is Done by Representation.
5. Every Request is Stateless.

Everything is a Resource

The first important point in REST is to think in terms of resources rather than physical files. You access the resources over some URI. for example:

* http://www.mysite.com/pictures/logo.png - Image Resource;
* http://www.mysite.com/index.html - Static Resource;
* http://www.mysite.com/Customer/1001 - Dynamic Resource returning XML or JSON content;
* http://www.mysite.com/Customer/1001/Picture - Dynamic Resource returning an image.

Unique Identifier

Informations are reached by using unique identifiers. In REST, we add one more constraint to the current URI: in fact, every URI should uniquely represent every item of the data collection. For example you can see below unique URI format for each customer is defined.

http://www.mysite.com/Customer/dupont

http://www.mysite.com/Customer/smith

and orders of customer ” dupont ” is defined like following:

http://www.mysite.com/Customer/dupont/Orders

Simple and Uniform Interfaces

To request and send data to those resources some HTTP methods are used. These are the HTTP methods :

GET - List the members of the collection (one or several)

PUT - Update a member of the collection

POST - Create a new entry in the collection

DELETE - Delete a member of the collection

Then, at URI level, you can define the type of collection, like this:

http://www.mysite.com/Customer to identify the customers or

http://www.mysite.com/Customer/1234/Orders to access a given order.

This combination of HTTP method and URI replace a list of English-based methods, like GetCustomer / InsertCustomer / UpdateOrder / RemoveOrder.

Representations

What you are sending via the network is actually a representation of the actual resource data.

The main representation schemes are XML and JSON but it could be CVS as well.

For example, here is how a customer data is retrieved with a GET method:

An example with XML format :

<Customer>

<ID>1234</ID>

<Name>Dupond</Name>

<Address>Tree street</Address>

</Customer>

Below is a simple JSON snippet for creating a new customer record with name and address:

{Customer: {"Name":"Dupont", "Address":"Tree street"}}

As a result to this data transmitted with a POST command, the RESTful server will return the just-created ID.

Clarity of this data format is one of the reasons why preferably to use JSON format instead of XML or any format.

Stateless

In REST concept every request is independent and each request doesn’t have any information about the previous request. Every request should be an independent request so that we can enhance the requests being sent. As REST doesn’t use much memory like SOAP so that we can make more independent requests. This Scalability in terms is important because a server request is unaware of any state we store and makes resource management easier.

Of course, there are some disadvantages of being stateless. Client has to add all the necessary information in request' which increases network traffic. It also highlights consistency in the behavior of the application server which could be handled difficultly because there may be many different Clients, the requests may come from the different content.

Independent request means that each request doesn’t carry out any state information about another. A classic example of statelessness is the use of the HTTP protocol. HTTP protocol does not carry out data between requests so that this is usually achieved by using some kind of server variables to carry out the data using some programming languages. For example in Asp.Net it can be achieved by using variables such as viewstate, session, caching, query string vice versa.

There is one more important fact that need to be outlined: **Cacheable**. HTTP response by the client "cache" can be entertainment, so it sent Server RESPONSE to indicate whether the case is cacheable, it is important in terms of performance.

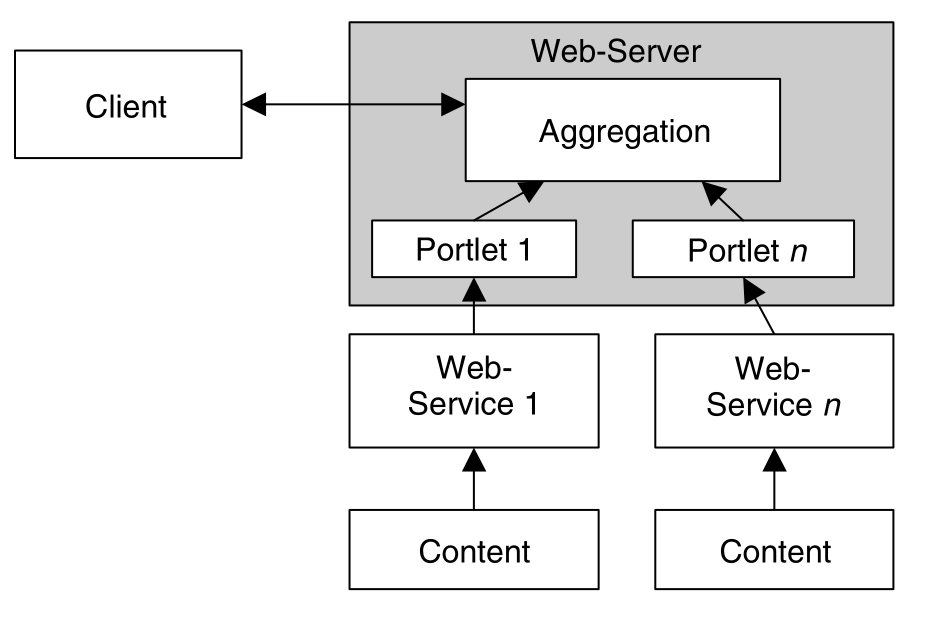
What are the advantages of rest;

* It is lightweight, can be easily extended.
* Inbound, outbound data size is very small.
* It is easy to design and easy implementation, it does not need any extra tools.
* Works over HTTP, platform independent.

### REST and RESTful

Representational state transfer (REST) is a style of software architecture. As described in a dissertation by Roy Fielding, REST is an "architectural style" that basically exploits the existing technology and protocols of the Web.

RESTful is typically used to refer to web services implementing such an architecture. Some says that "REST" is an architectural paradigm. "RESTful" describes using that paradigm. More precisely the term Representational State Transfer was introduced and defined in 2000 by Roy Fielding in his doctoral dissertation. Conforming to the REST constraints is referred to as being ‘RESTful’.



## Web Services

**Table of Contents**

[Web Services Description Language (WSDL)](https://gyires.inf.unideb.hu/GyBITT/08/ch06.html#idp60259536)

[Universal Description, Discovery and Integration (UDDI)](https://gyires.inf.unideb.hu/GyBITT/08/ch06s02.html)

[SOAP Web Services](https://gyires.inf.unideb.hu/GyBITT/08/ch06s03.html)

[SOAP vs REST](https://gyires.inf.unideb.hu/GyBITT/08/ch06s04.html)

Web services are the standardized way of communicating different web-based applications via network. By using Web services, your application can publish its function or message to the rest of the world. Web services can communicate with each other using XML file format without knowing how each other is implemented. Web based applications are communicating using the concepts as XML, SOAP, REST, WSDL and UDDI.

In the context of Service Oriented Architecture, Web Services are used to facilitate communication between service providers and service consumers.

W3C defines web service as a “software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format (specifically WSDL). Other systems interact with the Web service in a manner prescribed by its description using SOAP messages, typically conveyed using HTTP with an XML serialization in conjunction with other Web-related standards.”

Different applications might be built with different programming languages, therefore there is a need for a method of data exchange that doesn't depend upon a particular programming language. In other words a web service is a collection of open protocols and standards used for exchanging data between applications or systems. Software applications written in various programming languages and running on various platforms can use web services to exchange data over computer networks like the Internet in a manner similar to inter-process communication on a single computer.

Here are the common properties of web services:

* Web services are application components
* Web services communicate using open protocols
* Web services are self-contained and self-describing
* Web services can be discovered using UDDI
* Web services can be used by other applications
* HTTP and XML is the basis for Web services

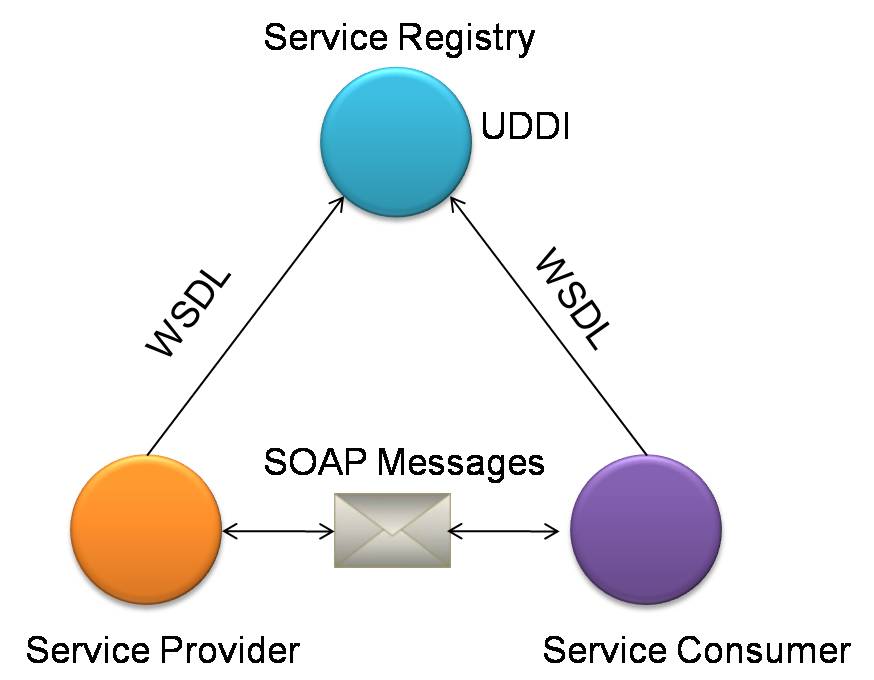
Usage of web services can be mentioned in two major parts. First one is ”reusable application components” and second one is to ”connect existing software”.

Reusable application components:There are sometimes some informations which many applications need to use. So why do we need to do these again and again. We can reuse the application components. Web services can offer some components like weather reports, currency conversion or language translation.

Connect existing software:You can connect your existing software to a Web service to utilize the data from which a web service provide.

**Web Service Roles**

Web service architecture can be expressed by mentioning some major roles of it. There are three major roles within the web service architecture:



Web Services overview

Service provider

This is the provider of the web service. The service provider implements the service and makes it available on the Internet.

Service requestor

This is any consumer of the web service. The requestor utilizes an existing web service by opening a network connection and sending an XML request.

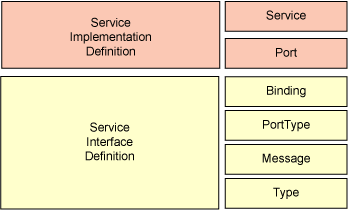
Service registry

This is a logically centralized directory of services. The registry provides a central place where developers can publish new services or find existing ones. It therefore serves as a centralized trade center for companies and their services.

## Web Services Description Language (WSDL)

WSDL describes the functionality of a web service and it is an XML based description language. A WSDL file contains information about how a web service can be called, what parameters it expects and what data structure it returns. The WSDL describes services as collections of network endpoints, or ports. The abstract definitions of ports and messages are separated from their concrete use.

This is good for allowing the reuse of these definitions. As you can see in the Figure ... it describes the structure of a WSDL file graphically. Concrete and Abstract sections are separated for reuse. I am going to mention about some major information given in a WSDL file.



structure of a WSDL file

Service:Contains a set of system functions that have been exposed to the Web-based protocols.

Endpoint:Defines the address or connection point to a Web service. It is typically represented by a simple HTTP URL string.

Binding:Specifies the interface and defines the SOAP binding style and transport ( SOAP Protocol). The binding section also defines the operations. There may be any number of bindings for a given portType.

PortType: Defines a Web service, the operations that can be performed, and the messages that are used to perform the operation. It is the most important element in WSDL. I can be compared to a module or a class in traditional programming.

Message: Typically, a message corresponds to an operation. The message contains the information needed to perform the operation. Each message is made up of one or more logical parts. Each part is associated with a message-typing attribute. The message name attribute provides a unique name among all messages.

Types:The types element defines the data types that are used by the web service. WSDL 1.1 allows the use of any typing system, but it uses W3C XML schema for maximum platform neutrality.

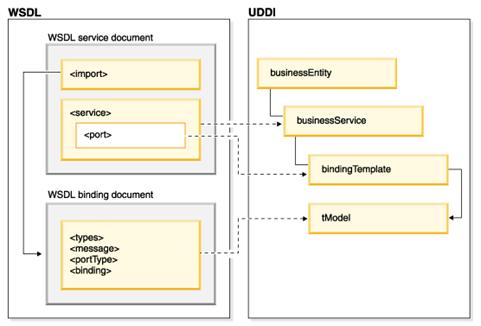
## Universal Description, Discovery and Integration (UDDI)

UDDI is a web based distributed directory. It enables businesses to list themselves on internet and they also discover each other. It is similar to a traditional phone book’s yellow and white pages. Companies that use the UDDI protocol can extend their market reach and find new customers while also finding other businesses that offer useful services to them. Because UDDI uses a standard format for describing business services, it is easy to search and find useful services offered from other businesses.

A UDDI registration consists of three components:

* White Pages — address, contact, and known identifiers.
* Yellow Pages — industrial categorizations using the standard classifications.
* Green Pages — technical information about services published by the business.

The Relation between WSDL and UDDI is shown in the Figure



UDDI

## SOAP Web Services

Web services can be implemented with different technologies such as SOAP or REST. Simply, SOAP web services provide a business logic exposed via a service to a client. And it is done by a loosely coupled interface using XML. The interface to which a message is sent defines the format of the message request and response, and mechanisms to publish and to discover web service interfaces.

SOAP usually relies on several technologies and protocols such as HTTP or SMTP for transferring and transforming data. Some of this technologies are:

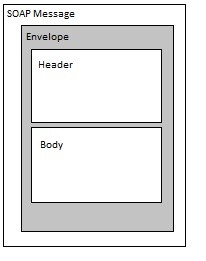
* Extensive markup language (XML) is the foundation of SOAP which is used to build and define the web services.
* Web Services Description Language (WSDL) describes the protocols, message types, interface and interactions of SOAP web services.
* Simple Object Access Protocol (SOAP) is a protocol that is used for exchanging data using XML.
* HTTP is the commonly used transport protocol for exchanging messages. Other protocols such as SMTP or JMS are also used.
* Universal Description Discovery, and Integration (UDDI) is optionally used to store and categorize SOAP web service interfaces (WSDL).

**SOAP**

Simple Object Access Protocol is the standard web services application protocol. It provides the communication mechanism to connect web services, exchanging formatted XML data across a network protocol. SOAP relies heavily on XML for defining an extensive messaging framework. This framework is independent of any particular programming model.

**SOAP Messages**

SOAP message is a one-way message such as a request from a client, or a response from a server. Every SOAP message contains an Envelope element, an optional Header element and a Body element.



SOAP message

**Envelope**

The SOAP XML Envelope defines specific rules for encapsulating data being transferred between computers. This includes application-specific data, such as the method name to invoke, method parameters, or return values. It can also include information about who should process the envelope contents

Envelope element has:

* a local name for the envelope
* a namespace name used for differentiating versions ("http://www.w3.org/2003/05/soap-envelope" for SOAP 1.2),
* Header and Body elements in its children property

An example:

<?xml version="1.0"?>

<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

...

Message information goes here

...

</soap:Envelope>

The encodingStyle attribute is used to define the data types used in the document. This attribute may appear on any SOAP element, and applies to the element's contents and all child elements.

**Header**

The optional Header element offers a flexible framework for specifying additional application-level requirements (like authentication, payment, etc.) about the SOAP message.

An example:

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2003/05/soap-envelope"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Header>

<m:Trans xmlns:m="http://www.example.org/transaction/"

soap:mustUnderstand=”true” >

5

</m:Trans>

</soap:Header>

...

</soap:Envelope>

**Body**

Body element contains the actual message. For example:

<?xml version="1.0"?>

<soap:Envelope

xmlns:soap="http://www.w3.org/2003/05/soap-envelope"

soap:encodingStyle="http://www.w3.org/2003/05/soap-encoding">

<soap:Body xmlns:m="http://www.example.org/stock">

<m:GetStockPrice>

<m:StockName>IBM</m:StockName>

</m:GetStockPrice>

</soap:Body>

</soap:Envelope>

A GetStockPrice request is sent to a server. The request has a StockName parameter, and a Price parameter that will be returned in the response

## SOAP vs REST

SOAP is a protocol specification for exchanging structured information in the implementation of Web Services. It uses XML for the message format. It is independent of the transport protocol (could be HTTP, FTP, TCP, UDP, or named pipes). SOAP based services strictly define the format of messages passed back and forth. A SOAP message contains the data, the action to perform on it, the headers, and the error details in case of failure. Security is provided by WS-Security standards and is end-to-end. It supports identity through intermediaries, not just point to point (SSL).

Representational State Transfer (REST) is an architecture style for designing networked applications. REST recognises everything a resource (e.g. User, Course, etc.) and each resource implements a standard uniform interface (typically HTTP interface), resources have name and addresses (URIs), each resource has one or more representation (like JSON or XML) and resource representations move across the network usually over HTTP.

RESTful web APIs (or RESTful web service) is a web API implemented using HTTP and principles of REST. RESTful API separates user interface concerns from data storage concerns. It improves portability of interface across multiple platforms and simplifies server components by making them stateless. Each request from client contains all the state information and server does not hold client context in the session.

One of the major benefits of SOAP is that you have a WSDL service description, like a contract between the two parties. You can pretty much discover the service automatically and generate a useable client proxy from that service description (generate the service calls, the necessary data types for the methods and so forth). With REST you’ll have to implement such contract yourself in your code.

Note that with version 2.0, WSDL supports all HTTP verbs and can be used to document RESTful services as well, but there is a less verbose alternative in WADL (Web Application Description Language) for that purpose.

With RESTful services, message security is provided by the transport protocol (HTTPS), and is point-to-point only. It doesn’t have a standard messaging system and expects clients to deal with communication failures by retrying. SOAP has successful/retry logic built in and provides end-to-end reliability even through SOAP intermediaries.

One of the major benefits of RESTful API is that it is flexible for data representation, for example you could serialize your data in either XML or JSON format. RESTful APIs are cleaner or easier to understand because they add an element of using standardised URIs and gives importance to HTTP verb used (i.e. GET, POST, PUT and DELETE).

RESTful services are also lightweight, that is they don’t have a lot of extra XML markup. To invoke RESTful API all you need is a browser or HTTP stack and pretty much every device or machine connected to a network has that. With web services that are generally SOAP-based, the request and response are hidden. SOAP requests must be interpreted as they are received at the server to determine the operation to perform and the arguments required to perform that operation. They are generally passed through as a parameter, which is essentially a function/method call. SOAP does not currently provide a mechanism for caching results, where as REST can (through the standard HTTP caching).

Essentially, the Traditional SOA way of doing things and the RESTful way of doing things would both yield similar results. However you might find that the Traditional SOA would allow you to do more powerfull things (and as a result is more complex) and the RESTful oriented architecture would allow you to easily do simple things (and this being much easier to implement) REST becomes an architecture style applicable to systems that would want operate with a Resource view. SOA, on the flip-side, has a different view of the web. It uses the web as a application infrastructure between service providers and consumers otherwise known as the Service view.

In conclusion, we can say that web services standards provide an open standards based communication framework that operates within the purview of W3C guidelines. Web Services provide a platform/technology independent communication methodology while SOA is an overall IT strategy framework that aims to provide business agility. Finally, which ever architecture you choose make sure its easy for developers to access it, and well documented.

Link:-

https://gyires.inf.unideb.hu/GyBITT/08/pt02.html