UNIT - IV

4.1. SOA platform basics

Before we begin to look at the specifics of the J2EE and .NET platforms, let's first establish some of the common aspects of the physical development and runtime environments required to build and implement SOA-compliant services.

4.1.1. Basic platform building blocks

Taking a step back from SOA for a moment, let's start by defining the rudimentary building blocks of a software technology platform. The realization of a software program puts forth some basic requirements, mainly:

- We need a development environment with which to program and assemble the software program. This environment must provide us with a development tool that supports a programming language.
- We need a runtime for which we will be designing our software.
- We need APIs that expose features and functions offered by the runtime so that we can build our software program to interact with and take advantage of these features and functions.
- Finally, we need an operating system on which to deploy the runtime, APIs, and the software program.

Fig 4.1. Fundamental software technology architecture layers.

4.1.2. Common SOA platform layers

The contemporary SOA is a distributed architectural model, built using Web services. Therefore, an SOA-capable development and runtime platform will be geared toward a distributed programming architecture that provides support for the Web services technology set. As a result, we have two new requirements:
- We need the ability to partition software programs into self-contained and composable units of processing logic (components) capable of communicating with each other within and across instances of the runtime.
- We need the ability to encapsulate and expose application logic through industry standard Web services technologies.

**Fig 4.2. The common layers required by a development and runtime platform for building SOA.**

![Diagram of SOA layers and technologies](image)

### 4.1.3. Relationship between SOA layers and technologies

When we introduce components and Web services to our architecture model, we end up with a number of different relationships forged between the fundamental architecture layers and the specific technologies involved.

**Fig 4.3. A logical view of the basic relationships between the core parts of a service-oriented architecture.**

![Diagram of relationships between core parts](image)
technologies introduced by the Web services framework (namely, WSDL, SOAP, UDDI, and the WS-*
specifications).

4.1.4. Fundamental service technology architecture

So far we've established the overall pieces that comprise a fundamental, abstract service-
oriented architecture. What is of further interest to us are the specifics behind the relationship between
the Web Technology and Component Technology layers. By studying this relationship, we can learn
how service providers and service requestors within an SOA can be designed, leading us to define a
service-level architecture.

Service processing tasks

Service providers are commonly expected to perform the following tasks:

- Supply a public interface (WSDL definition) that allows it to be accessed and invoked by a
  service requestor.
- Receive a SOAP message sent to it by a service requestor.
- Process the header blocks within the SOAP message.
- Validate and parse the payload of the SOAP message.
- Transform the message payload contents into a different format.
- Encapsulate business processing logic that will do something with the received SOAP message
  contents.
- Assemble a SOAP message containing the response to the original request SOAP message from
  the service requestor.
- Transform the contents of the message back into the format expected by the service requestor.
- Transmit the response SOAP message back to the service requestor.

Service providers are designed to facilitate service requestors. A service requestor can be any piece of
software capable of communicating with a service provider. Service requestors are commonly expected
to:

- Contain business processing logic that calls a service provider for a particular reason.
- Interpret (and possibly discover) a service provider's WSDL definition.
- Assemble a SOAP request message (including any required headers) in compliance with the
  service provider WSDL definition.
- Transform the contents of the SOAP message so that they comply with the format expected by
  the service provider.
Service processing logic

Looking at these tasks, it appears that the majority of them require the use of Web technologies. The only task that does not fall into this category is the processing of business logic, where the contents of the SOAP request are used to perform some function that may result in a response. Let’s therefore group our service provider and requestor tasks into two distinct categories.

- **Message Processing Logic** The part of a Web service and its surrounding environment that executes a variety of SOAP message processing tasks. Message processing logic is performed by a combination of runtime services, service agents, as well as service logic related to the processing of the WSDL definition.

- **Business Logic** The back-end part of a Web service that performs tasks in response to the receipt of SOAP message contents. Business logic is application-specific and can range dramatically in scope, depending on the functionality exposed by the WSDL definition.

**Fig 4.4. A service provider consisting of message processing and business logic.**

**Table 4.1. Service provider logic categorization.**

<table>
<thead>
<tr>
<th>Message Processing Logic</th>
<th>Business Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP message receipt and transmission.</td>
<td>Application-specific business processing logic.</td>
</tr>
<tr>
<td>SOAP message header processing.</td>
<td></td>
</tr>
<tr>
<td>SOAP message payload validation and parsing.</td>
<td></td>
</tr>
<tr>
<td>SOAP message payload transformation.</td>
<td></td>
</tr>
</tbody>
</table>
Fig 4.5. A revised service provider model now including an endpoint within the message processing logic.

The primary difference between how service logic is used in requestors and providers is related to the role of business logic. The business logic part of a service requestor is responsible for initiating an activity (and the resulting SOAP message exchange), whereas the business logic within a service provider responds to an already initiated activity.

Fig 4.6. A service requestor consisting of message processing and business logic.

Table 4.2. Service requestor logic categorization.

<table>
<thead>
<tr>
<th>Message Processing Logic</th>
<th>Business Logic</th>
</tr>
</thead>
<tbody>
<tr>
<td>WSDL interpretation (and discovery).</td>
<td>Application-specific business processing logic.</td>
</tr>
<tr>
<td>SOAP message transmission and receipt.</td>
<td></td>
</tr>
<tr>
<td>SOAP message header processing.</td>
<td></td>
</tr>
<tr>
<td>SOAP message payload validation and parsing.</td>
<td></td>
</tr>
<tr>
<td>SOAP message payload transformation.</td>
<td></td>
</tr>
</tbody>
</table>

M.Senthil Kumar, Asst. Prof. Department of Information Technology, SMEC
Message processing logic

The characteristics of the message processing logic of a service provider and service requestor consists of functions or tasks performed by a combination of runtime services and application-specific extensions. It is therefore not easy to nail down which elements of the message processing logic belong exclusively to the service. Among the processing layers are tasks, such as header processing, that are generic and applied to all service providers. Validation or transformation tasks, on the other hand, may involve service-specific XSD schemas and XSLT stylesheets and therefore may be considered exclusive to the service provider.

Fig 4.7. An example of the types of processing functions that can comprise the message processing logic of a service.

Although the message processing logic for service requestors and service providers may be similar, there is an important implementation-level difference. The service provider supplies an endpoint that expresses an interface and associated constraints with which all service requestors must comply. Vendor platforms accomplish this by supporting the creation of proxy components. These components exist as part of the message processing logic and are commonly auto-generated from the service provider WSDL definition (and associated service description documents). They end up providing a programmatic interface that mirrors the WSDL definition but complies to the native vendor runtime environment.
Fig 5.8. The message processing logic part of a service requestor includes a proxy component.

Proxies accept method calls issued from the regular vendor platform components that contain the service requestor business logic. The proxies then use vendor runtime services to translate these method calls and associated parameters into SOAP request messages. When the SOAP request is transmitted, the proxy is further able to receive the corresponding SOAP response from the service provider. It then performs the same type of translation, but in reverse.

**Business logic**

Business logic can exist as a standalone component, housing the intelligence required to either

Fig 4.9. The same unit of business logic participating within a service provider and a service requestor.
invoke a service provider as part of a business activity or to respond to a request in order to participate in such an activity. As an independent unit of logic, it is free to act in different roles. If units of business logic exist as physically separate components, the same business logic can be encapsulated by different service providers.

**Fig 4.10. One unit of business logic being encapsulated by two different service providers.**

Because units of business logic can exist in their native distributed component format, they also can interact with other components that may not necessarily be part of the SOA. This, in fact, is a very common model in distributed environments where components (as opposed to services) are composed to execute specific tasks on behalf of the service provider.

**Fig 4.11. The same unit of business logic facilitating a service provider and acting on its own by communicating independently with a separate component.**
In this case the second component can be considered as belonging to the overall automation logic encapsulated by the service provider.

**Service agents**

A type of software program commonly found within the message processing logic of SOA platforms is the *service agent*. Its primary role is to perform some form of automated processing prior to the transmission and receipt of SOAP messages. As such, service agents are a form of intermediary service. Service agents usually address cross-cutting concerns, providing generic functions to alleviate the processing responsibilities of core Web service logic. Examples of the types of tasks performed by service agents include:

1. SOAP header processing
2. filtering (based on SOAP header or payload content)
3. authentication and content-based validation
4. logging and auditing
5. routing

*Fig 4.12. Service agents processing incoming and outgoing SOAP message headers.*
An agent program usually exists as a lightweight application with a small memory footprint. It typically is provided by the runtime but also can be custom developed. What's the difference between a service agent intermediary and an intermediary Web service? The determining factor is typically the availability of a WSDL endpoint. Service agents don't generally have or require one, as they are designed to intercept message traffic automatically. An intermediary that is also a Web service will supply a published WSDL definition, establishing itself as a legitimate endpoint along the message path. Note that a service agent intermediary can be designed to also be a Web service intermediary.