SOA and Quality

Yang Qing Fan
Qian Peng
Acknowledgment

First of all, we want to thank my family for their constant support. We also want to express our gratitude to our supervisors, Jesper Andersson and Mathias Hedenborg for their comments and hints that have made this thesis into what it is. We would also like to thank our opponent Abdille Hagi telling me where improvement be needed.
Abstract

This thesis emphasizes on investigating the relationship between the quality attributes and service oriented architecture (SOA). Due to quality attributes requirements drive the design of software architecture, it is necessary to maintain the positive quality of SOA and improve the negative quality of SOA. This thesis gives an introduction to SOA, Enterprise Service Bus (ESB) and MULE. Then, it covers information on quality of systems and tactics for achieving each quality attribute. Finally, we discuss the quality of SOA in detail, and illustrate how to set up a SOA and how to improve its quality using a case of an order for supermarket.

Key-words: Service Oriented Architecture (SOA), quality attributes, Enterprise Service Bus (ESB), MULE.
# Contents

Figures .................................................................................................................. V
Tables ..................................................................................................................... VI

1. Introduction ........................................................................................................... 1
   1.1 Motivation ......................................................................................................... 1
   1.2 Research objectives ......................................................................................... 1
   1.3 Research problem and method ........................................................................ 2
   1.4 Structure of the thesis ..................................................................................... 2

2. Service oriented architecture ............................................................................ 3
   2.1 SOA .................................................................................................................. 3
       2.1.1 Definition of SOA ...................................................................................... 3
       2.1.2 SOA features ............................................................................................ 3
       2.1.3 Why use SOA .......................................................................................... 4
       2.1.4 Layers of SOA ......................................................................................... 5
       2.1.5 SOA infrastructure ................................................................................... 6
   2.2 ESB ................................................................................................................. 6
       2.2.1 Definition of ESB ...................................................................................... 6
       2.2.2 ESB Features ........................................................................................... 7
       2.2.3 ESB patterns ............................................................................................. 8
       2.2.4 ESB product ............................................................................................. 9
   2.3 MULE .............................................................................................................. 10
       2.3.1 The MULE manager ................................................................................. 11
       2.3.2 The model ................................................................................................ 11
           2.3.2.1 UMO components ............................................................................ 12
           2.3.2.2 Endpoints ........................................................................................ 12
           2.3.2.3 Transport provider .......................................................................... 12
           2.3.2.4 Connector ......................................................................................... 13
           2.3.2.5 Router .............................................................................................. 14
           2.3.2.6 Filter ................................................................................................. 15
           2.3.2.7 Transformer ...................................................................................... 15
   2.4 Case study ....................................................................................................... 15
       2.4.1 Design of order system ............................................................................ 15
       2.4.2 Implementation ....................................................................................... 18

3. Quality Attribute of Software Product & tactics ............................................ 21
   3.1 Quality attribute .............................................................................................. 21
       3.1.1 Qualities of the system ............................................................................ 21
       3.1.2 Qualities on conceptual ......................................................................... 27
   3.2 Relationship between Architectural Tactics and Quality attributes ............ 27
   3.3 Architectural tactics ....................................................................................... 28
       3.3.1 Availability tactics .................................................................................... 28
       3.3.2 Modifiability tactics .................................................................................. 29
       3.3.3 Performance tactics .................................................................................. 31
       3.3.4 Security tactics ......................................................................................... 32
3.3.5 Testability tactics 33
3.3.6 Usability tactics 34

4. SOA impact on Quality

4.1 Positive effect
   4.1.1 Interoperability 35
   4.1.2 Flexibility 35
   4.1.3 Reusability 35

4.2 Negative effects
   4.2.1 Performance 36
   4.2.2 Security 36

4.3 Case study
   4.3.1 Interoperability 39
   4.3.2 Flexibility 39
   4.3.3 Performance 40
   4.3.4 Security 40

5. Conclusion

5.1 Evaluation 42
5.2 Conclusion 42
5.3 Future work 43

Reference 44
# Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Layers of SOA</td>
<td>5</td>
</tr>
<tr>
<td>2.2</td>
<td>Infrastructures</td>
<td>6</td>
</tr>
<tr>
<td>2.3</td>
<td>Description of ESB</td>
<td>7</td>
</tr>
<tr>
<td>2.4</td>
<td>Interaction patterns</td>
<td>8</td>
</tr>
<tr>
<td>2.5</td>
<td>Mediation patterns</td>
<td>8</td>
</tr>
<tr>
<td>2.6</td>
<td>Deployment patterns</td>
<td>9</td>
</tr>
<tr>
<td>2.7</td>
<td>MULE architecture</td>
<td>11</td>
</tr>
<tr>
<td>2.8</td>
<td>SEAD-based</td>
<td>11</td>
</tr>
<tr>
<td>2.9</td>
<td>File connector</td>
<td>13</td>
</tr>
<tr>
<td>2.10</td>
<td>Order-flow-charts</td>
<td>16</td>
</tr>
<tr>
<td>2.11</td>
<td>Message</td>
<td>17</td>
</tr>
<tr>
<td>2.12</td>
<td>EntryGateway XML</td>
<td>18</td>
</tr>
<tr>
<td>2.13</td>
<td>Descriptor XML</td>
<td>19</td>
</tr>
<tr>
<td>2.14</td>
<td>ResponseAggregator</td>
<td>20</td>
</tr>
<tr>
<td>3.1</td>
<td>Quality attribute scenario</td>
<td>21</td>
</tr>
<tr>
<td>3.2</td>
<td>Scenario</td>
<td>22</td>
</tr>
<tr>
<td>3.3</td>
<td>Scenario</td>
<td>23</td>
</tr>
<tr>
<td>3.4</td>
<td>Scenario</td>
<td>24</td>
</tr>
<tr>
<td>3.5</td>
<td>Scenario</td>
<td>25</td>
</tr>
<tr>
<td>3.6</td>
<td>Scenario</td>
<td>26</td>
</tr>
<tr>
<td>3.7</td>
<td>Scenario</td>
<td>27</td>
</tr>
<tr>
<td>3.8</td>
<td>Relationship between quality and tactics</td>
<td>28</td>
</tr>
<tr>
<td>3.9</td>
<td>Availability</td>
<td>28</td>
</tr>
<tr>
<td>3.10</td>
<td>Tone set</td>
<td>29</td>
</tr>
<tr>
<td>3.11</td>
<td>Modifiability</td>
<td>30</td>
</tr>
<tr>
<td>3.12</td>
<td>Performance</td>
<td>31</td>
</tr>
<tr>
<td>3.13</td>
<td>Security</td>
<td>33</td>
</tr>
<tr>
<td>3.14</td>
<td>Testability</td>
<td>34</td>
</tr>
<tr>
<td>3.15</td>
<td>Usability</td>
<td>34</td>
</tr>
<tr>
<td>4.1</td>
<td>Unsecured SOA</td>
<td>37</td>
</tr>
<tr>
<td>4.2</td>
<td>Unauthorized machine</td>
<td>38</td>
</tr>
<tr>
<td>4.3</td>
<td>Denial of service attack</td>
<td>38</td>
</tr>
<tr>
<td>5.1</td>
<td>Relationship between the SOA and quality</td>
<td>43</td>
</tr>
</tbody>
</table>
Tables

Table 2-1 Products of ESB .......................................................... 10
Table 2-2 MULE provider .......................................................... 13
Table 2-3 Routers ....................................................................... 14
Table 2-4 UMO components ...................................................... 17
Table 2-5 Message transformer .................................................. 18
Table 2-6 Transports ................................................................. 18
Table 3-1 Availability general scenarios ...................................... 22
Table 3-2 Modifiability general scenario .................................... 23
Table 3-3 Performance General Scenario ................................... 23
Table 3-4 Security General Scenario .......................................... 25
Table 3-5 Testability General Scenario ...................................... 25
Table 3-6 Usability General Scenario ......................................... 27
Table 4-1 Performance .............................................................. 36
1. Introduction

This chapter gives a brief introduction about motivation and objectives of this subject. We present the method that we carry on the investigation. Further, we provide the structure of this thesis.

1.1 Motivation

Currently the software development faces two problems. First, the function mode of software is constantly enlarging and the requirements of customer are increasing. It is a huge investment trap to redesign software. Second, various operating systems, data form and software are interlaced with each other. It is impossible to unify them, but go the enlargement in the existing software, data and system.

According to above-mentioned, it is imperative to adjust software structure. Service Oriented Architecture (SOA) is intended to be an effective way to help enterprises to earn more profits with less investment. SOA is defined as an IT system architectural style. It comprises a collection of services which communicate with each other, including either simple data passing or two or more services coordinating some activities.

In order to investigate the relationship between quality (i.e. interoperability, performance and security) and SOA, we present SOA from connectivity layer—Enterprise Service Bus (ESB). Furthermore, we try to realize an implementation through MULE to find how is the impacts of quality on SOA. An effective SOA must meet the quality of service. The SOA style contributes positively to several quality attributes but some are affected negatively. We intend to investigate if and how one can minimize the negative contribution.

1.2 Research objectives

Being confronted with changing market, ever-increasing customer requirements, and abundant information needed to be integrated, advanced information systems are expected by enterprises to stay competitive. And with more and more expensive investment in IT projects, alignment of IT with the business and maximal reuse of IT assets is significant to systems.

Nowadays, many enterprise architects believe that the SOA approach is a good way to help businesses integrate information resources. Is it really true? Once SOA is being exploited, the following questions can be formulated:

1) How can a SOA be set up?
2) Which quality attribute requirements will be positively impacted and which will be negatively impacted by the use of SOA?
3) What can we do to improve quality attribute that are negatively affected by the SOA style?

So this paper starts from concept of SOA and emphasizes service connectivity, using Enterprise Service Bus (ESB) provided by MULE. We will discuss different connection types among different environments and different protocols. Then, after looking into quality attributes, we present relationship between quality attributes and
SOA. Finally, we will exemplify how to improve quality on SOA style architectures.

1.3 Research problem and method
The main problem in theory is to deploy a SOA from connectivity (ESB) aspect. How to analyze relationship between quality attributes (e.g. interoperability, flexibility, reusability, performance and security) and SOA could also be an important problem. At the same time, refer to studying the case; try to find a tradeoff among different quality attribute requirements.

This thesis is separated from 3 parts: a theoretical, a case study and a conclusion. The basic approach is to apprehend reference stffs and get the theoretical foundation. Then, we apply open source-MULE to deploy the case which is clearly and facility to expound technique that we used. Finally, we use the case study and analyze inter-service connections and discuss quality attributes and tactics. Further, we get the conclusion.

1.4 Structure of the thesis
Chapter two has an introduction to SOA, ESB and the open source ESB implementation Mule. An example using Mule shows how to deploy application using the ESB. Chapter three puts forward the quality attributes and tactics in designing and implementing SOA style architecture. Chapter four outlines positive and negative effects that SOA has on a system’s quality attributes. Finally, this thesis ends in Chapter five with evaluation, conclusions and directions for feature research.
2. Service oriented architecture

This chapter introduces SOA and ESB. First, we present the definition of SOA and the characteristic of SOA, why use SOA and the layers of SOA. Second, we outline what is ESB, the ESB patterns, why we need ESB in SOA and the prevalent ESB productions. Finally, we introduce the technique—MULE, and exemplify the implementation of ESB using MULE.

2.1 SOA

In order to support business processes that cover all present and prospective systems requirements needed to realize the business goal, a variety of designs are put forward to, such as, Object Request Brokers (ORBs) based on the Common Object Request Broker Architecture (CORBA) specification. Whereas, a standardized architecture is required to better support the connection of various applications and the sharing of data. SOA is one such architecture.

Meanwhile, more and more companies apply this architecture to serve developing, such as, IBM, Microsoft, and Worldwide Web Consortium.

First, we give the definition of SOA. Then, we introduce the SOA features and why we use SOA. Finally, we present the layers and infrastructure of SOA.

2.1.1 Definition of SOA

There is no official definition of SOA. Different organizations give their definitions for accordingly and expediently adopting. Consequently, SOA is defined in many different ways from different angles, including, [1]

- “A service-oriented architecture (SOA) is an application framework that takes everyday business applications and breaks them down into individual business functions and processes, called services. An SOA lets you build, deploy and integrate these services independent of applications and the computing platforms on which they run.”—IBM Corporation
- “Service-Oriented Architecture is an approach to organizing information technology in which data, logic, and infrastructure resources are accessed by routing messages between network interfaces.”—Microsoft
- An SOA is “a set of components which can be invoked, and whose interface descriptions can be published and discovered.”—Worldwide Web Consortium

Generally speaking, SOA is an architecture style, packaged as services, which easily integrate and reuse existing services. The communication of these services could either transport data from one service to another or coordinate an activity among different services.

2.1.2 SOA features

SOA is a loosely coupled architecture. One service of SOA is an individual module. When the internal structure of each service gradually changed, SOA could continue existing. It could update or add service modules based on existing services to save IT investment.
SOA adopts widely accepted standards (e.g. Extensible Markup Language (XML), Simple Object Access Protocol (SOAP)) and has neutral interface and independence services which lead to flexibility of SOA. Based on the text transport, message transformation between services don’t contain disposing data logic and data type. The receiver just disposes discriminating data and omits other data. It promises the irrespective of protocol and promotes the compatibility. It ensures the entire architecture in a loose-coupled and flexibility system from under stratum.

SOA increases the interoperability. Interoperability is the ability of a collection of communicating entities to share specific information and operate on it according to an agreed-upon operational semantics. For example, Common Object Request Broker Architecture (CORBA), the former of SOA, allows objects to interoperate, no matter the objects written in different languages or implemented in different platform. Nowadays, the prevalent implementation suitable to SOA concepts is Web services, a software system designed to support interoperable Machine to Machine interaction over a network. To improve interoperability, the Web Services-Interoperability Organization (WS-I) released profiles which helped deploying profile compliant Web Service. Just as the services and the interface can be separated, a service could be realized by .NET or J2EE, while these applications could use in different platforms.

There is corresponding quality of services (QoS) for each service. It includes the security, reliability, extensibility, performance and so on. The criterion of web services security ensures the security of message, such as, SAML (as Security Assertion Markup Language). Moreover, in the message transformation, there exists confirmation on how to carry on, such as once-and-only-once delivery or at-most-once delivery. WS-Reliability and WS-ReliableMessaging are standards for solving analogous problems.

2.1.3 Why use SOA

SOA is helping to create flexible business processes through loosely coupled services that could automate apply business functions as changes happened. It make easier to develop, maintain and manage. This is where the real benefit of SOA can be realized.

SOA could get extensible. The widely used protocols’ standard induces that different systems and services could communicate with each other. There is not a limitation on the platform and language. The individual service module insures its extensibility.

SOA is the substitute of object-oriented module. It is impossible to avoid using former module to construct a service, but the whole design is basic on services-oriented. SOA rely on some update development, such as XML. It is not cooped in existed services, but as a flexibility form to adapt dynamic changes.

Although SOA reduces system’s performance and contains redundant data, it improves its extensibility, reusability, interoperability. It simplifies the development based on existed services to reduce the investment.

Due to changefully business requirements and functionality needs are introduced, SOA would be capable of adapt changes and bring more benefit for business.
2.1.4 Layers of SOA

SOA itself is a layered architecture. We depict these layers in Figure 2.1 [3]

- **Layer 1**: Operational systems layer.
  This layer comprises two components: existing systems and application, such as CRM and ERP (Enterprise Resource Planning), object-oriented system. The composite layered architecture of an SOA can evaluate existing systems and integrate them using service-oriented integration techniques.

- **Layer 2**: Enterprise components layer.
  It realizes the functionality and maintains the QoS for the exposed services. It utilizes container-based technologies such as application servers to implement the components, workload management and load balancing.

- **Layer 3**: Services layer.
  This layer could be formed with single service or multiply services. It uses the service descriptions to export the interfaces. It provides for the mechanism to take enterprise scale components.

- **Layer 4**: Business process composition or choreography layer.
  The composition services in this layer are worked as flow and performance is applied as individual applications which support specific business process.

- **Layer 5**: Access or presentation layer.
  This layer provides the interfaces. It contains several of standards, such as WS-ReliableMessaging. The ultimately solution give the selecting individual service or composite service.

- **Layer 6**: Integration (ESB).
  It is from the connectivity aspect. It allows interactions between services with a visually model business process. It provides the capabilities, such as intelligent routing, protocol mediation. It configures the dependencies between services rather than hardwiring them into code at each service directly.

- **Layer 7**: QoS (quality of service).
This layer ensures quality through monitor the health of SOA applications, manage SOA, maintain QoS such as security and performance.

2.1.5 SOA infrastructure
SOA infrastructure is obligatory for enterprise managing. Figure 2.2 shows the infrastructure. [4]

![Diagram of SOA infrastructure]

Figure 2.2 Infrastructures

Web Services Description Language (WSDL) describes the services. Universal Description Definition and Integration (UDDI) provide a registry mechanism and search the services. SOAP as a transport level is a framework for exchanging message between clients and servers. Web Services-Interoperability Basic Profile is provided by Web Services Interoperability Organization. It is the core for SOA testing.

2.2 ESB
With a flexible SOA connection between services, an exiting business process can be integrated easy and external partners can be connected in a security way, and system can be modifiability and extensibility. Thus, ESB is a popular choice for SOA connection. Therefore, this chapter starts with the definition of ESB, introduces the features and patterns of ESB. The current ESB products will be taken on.

2.2.1 Definition of ESB
Enterprise Service Bus is an architecture pattern, a flexible connectivity infrastructure for integrating existing applications and services.

Figure 2.3 is a description of ESB; every component is an equal service on bus.
The mediation is the essential of ESB, it operate on messages in-flight between requesters and providers.

Brief speaking, ESB has four major functions: the first function is message routing which send message to its destination. The detail transmission is transparent to user. Second, the protocol mediation send message by different protocol. Third, the message transformation that package message as a different format when it is being sent. Finally, the invocation supports synchronous and asynchronous transport protocols.

2.2.2 ESB Features
There are both sides for ESB features. On one hand, there are serious advantages of ESB. First, the fast and cheap integration of existing systems is the primary advantage. Then, flexibility, meaning changes of requirement can be managed quickly; third, powerful messages processing, is the one of main strengths of ESB, including message routing, protocol mediation, message transformation, message handling.
Comparing to BPEL (Business Process Execution Language), such as IBM’s WebSphere Process Server, performance is another big strongpoint. An ESB is designed to be able to handle larger volumes of messages than BPEL. That is, if the requirements for a system are data-centric, an ESB is the choice [5]. Ultimately, more configuration rather than integration coding. The ESB provides an additional abstraction layer which removes several of the tedious tasks of making services of different origin and implementation technology work together.

On the other hand, we should look ESB features from proper perspectives. ESB is not easy to integrate many disparate systems on message standards. Moreover, messages traversing the extra ESB layer increase latency, especially as compared to point to point communications. Meanwhile, security is also a problem need to be solved. What’s more, some critics remark that ESB require a significant effort to implement, but produces no value unless SOA services are subsequently created for the ESB.

General speaking, for a enterprise, using ESB as the technique to collaborate the information system, can integrate the developed, developing and will be developed
components into a whole framework, without needing any other equipment, and can supply enterprise with good capability of manage, control and dispatch real-time and large amount information. It can improve use efficiency of whole network and information system, save integrating money for enterprise.

2.2.3 ESB patterns
Building ESB-based solutions are classified as: [6]

- **Interaction patterns**: Enable service interaction points to dispatch messages to, or receive messages from the bus.

  ![Interaction patterns](image)

  **Figure 2.4 Interaction patterns**

- **Request/response**: Handles request/response-style interactions between endpoints.
- **Request/multi-response**: A variant of the above, where more than one response can be sent.
- **Event propagation**: Events may be anonymously distributed to an ESB-managed list of interested parties. Services may be able to add themselves to the list.

- **Mediation patterns**: Enable the manipulation of message exchanges. It is the most widely used model. [6]

  ![Mediation patterns](image)

  **Figure 2.5 Mediation patterns**

- **Protocol switch**: Enables service requesters to dispatch their messages using a variety of interaction protocols or APIs, such as SOAP/HTTP, JMS, and MQ Integrator (MQI). Trans-codes requests into the targeted service provider's format. It can be applied at the requester or the provider end of an interaction, at both ends, or anywhere in between.
- **Transform**: Translates the message payload (content) from the requester's schema to the provider's schema. It may include enveloping, de-enveloping, or encryption.
- **Enrich**: Augments the message payload by adding information from external data sources, such as customization parameters defined by the mediation, or from database queries.
**Route:** Changes the route of a message, selecting among service providers that support the requester's intent. Selection criteria can include message content and context as well as the targets' capabilities.

**Distribute:** Distributes the message to a set of interested parties and is usually driven by the subscribers' interest profiles.

**Monitor:** Observes messages as they pass through the mediation unchanged. Can be used to monitor service levels; assist in problem determination or meter usage for subsequent billing to users; or record business-level events, such as purchases above a certain dollar value. Can also be used to log messages for audit or for subsequent data mining.

**Correlate:** Derives complex events from message or event streams. It includes rules for pattern identification and rules that react to pattern discovery, for example, by generating a complex event derived from content of the triggering event stream.

- **Deployment patterns:** there are many choices, used in complex system, according to reality condition. Figure 2.6 shows four kinds of different deployment patterns:

![Deployment Patterns](image)

**2.2.4 ESB product**

Following is the popular products of ESB:

<table>
<thead>
<tr>
<th>Product</th>
<th>Vendor</th>
<th>Connection</th>
</tr>
</thead>
<tbody>
<tr>
<td>Matrix Business Works</td>
<td>TIBCO</td>
<td>SOAP, EMS, JMS, Rendezvous, MQ, BPEL</td>
</tr>
<tr>
<td>MULE ESB</td>
<td>Open-source, MULE Source, Inc.</td>
<td>SOAP, REST, JMS, MQ, JBI, AQ, Caching,</td>
</tr>
</tbody>
</table>
Table 2-1 Products of ESB

<table>
<thead>
<tr>
<th>Products of ESB</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Open ESB</td>
<td>Open-source, Sun Microsystems</td>
</tr>
<tr>
<td>Sonic ESB</td>
<td>Progress Software</td>
</tr>
<tr>
<td>Websphere ESB</td>
<td>IBM</td>
</tr>
<tr>
<td>AquaLogic Service Bus</td>
<td>BEA</td>
</tr>
</tbody>
</table>

2.3 MULE

MULE is the world’s most widely used open source Enterprise Service Bus, which is designed to adapt to exiting environments, requires lower up-front investment, support high-performance, multi-protocol transactions between various systems. And open access to source code enables developers to customize MULE to meet their needs respectively.

MULE ESB has such features [7]:

- It supports flexible deployment topologies: ESB, ESN, c/s, peer to peer.
- It supports pluggable connectivity such as JMS, VM, JDBC, TCP, UDP, http, servlet, soap, SMTP, POP3, file, XMPP.
- It supports JBI Integration.
- It supports for asynchronous, synchronous and request-response event processing over any transport.
- It supports Web Services using XFire, Axis and Glue.
- It includes End-to-End support for routing, transport and transformation of events.
- Spring framework Integration, means spring framework can be used as the ESB container and MULE can be easily embedded into spring applications. And default container is SEAD processing model, which is described in more detail below.
- Rest APIs are provided for technology agnostic and language neutral web based access to MULE Events

The MULE architecture is described in Figure 2.7. [7]
2.3.1 The MULE manager
The MULE Manager is central to a MULE server instance (also known as a Node or MULE Node). Its primary role is to manage the various objects such as connectors, endpoints and transformers for a MULE instance.

2.3.2 The model
The basic part of MULE framework is MULE model, also can be called as container. The container provides a range of services for UMO components such as transaction management, transformation of events, routing, event correlation, logging, auditing and management.

The default MULE model is SEAD-based. [8]

Figure 2.7 MULE architecture

Figure 2.8 SEAD-based
SEAD stands for Staged Event-Driven Architecture. SEAD-based is developed to use an efficient event-based queuing model to maximize performance and throughput. From the picture, we can see that in SEDA, a request to system or service can be split in to a chain of stages each doing its part of the total workload of the request. Each stage is connected to the others by request queues, and by controlling the rate at which each stage admits requests, the service can perform focused overload management. And each stage consists following part:

- Incoming Event Queue
- Admission Controller
- Dynamically sized Thread Pool
- Event Handler
- Resource Controller

In SEAD based mode, MULE is treating each component as a stage, with its own thread pool and work queue. Incoming Event Queue is provided as an inbound router or endpoint, and the Event Handler is the component itself. Admission controller is associated with incoming event queue, and the most common way is implemented as an Inbound Router which "can be used to control how and which events are received by a component subscribing on a channel". As resource controller, MULE allow user to configure most aspects of how stages are initialized and this is also true for the number of threads and the size of the object pools.

At same time, MULE separates object construction from management. So other IoC/DI containers such as spring, PicoContainer or Plexus can be used to construct UMO components.

2.3.2.1 UMO components

UMO is Universal Message Object that can receive and send events from anywhere. UMO Components are business objects which execute business logic on an incoming event. These components are standard JavaBeans, without MULE-specific code. MULE handles all routing and transformation of events to and from your objects based on the configuration of your component.

2.3.2.2 Endpoints

An Endpoint defines a communication channel between two or more components, applications or repositories, and they can be configured with message filters, security interceptors and transaction information to control who, what and how messages are received or sent via the endpoint.

2.3.2.3 Transport provider

Transport is the critical part of ESB, and MULE provider support for communication over a lot of protocols consisting WebService (Axis,XFire)/REST, Ejb/Rmi, Email, Jdbc, Tcp/Udp, Xmpp, VM, Ftp, File and so on with Transport providers which are a set of objects can be chosen by users. [7], we list some common ways.

<table>
<thead>
<tr>
<th>Provider</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ejb</td>
<td>Allows Ejb invocations to be made using outbound endpoints.</td>
</tr>
<tr>
<td>Email</td>
<td>This Provider supplies various email connectivity options.</td>
</tr>
</tbody>
</table>
This connector allows files to be read and written to directories on the local file system. The connector can be configured to filter the file it reads and the way files are written, such as whether binary output is used or the file is appended to.

This provider supplies Http and Https transport of MULE events between applications and other MULE servers.

A MULE provider for JDBC Connectivity.

A MULE provider for JMS Connectivity. MULEs itself is not a JMS server but can use the services of any JMS 1.1 or 1.02b compliant server such as ActiveMq, OpenJms, Joram, JBossMQ and commercial vendors such as WeblogicMQ, WebsphereMQ, SonicMQ, SeeBeyond, etc.

Provides connectivity to the MULE server via servlets, provides facilities for MULE components to listen for events received via a servlet request. There is also a servlet implementation that uses the Servlet provider to enable REST style services access. Note this transport is now bundled with the Http transport.

Enables your components to be exposed as web services and to act as soap clients. The Soap provider supports XFire, Web Methods Glue and Apache Axis.

This provider allows connectivity to Streams such as System.in and System.out and is useful for testing.

A MULE provider that enables event sending and receiving over VM or embedded memory or persistent queues.

The WSDL transport provider can be used to for invoking remote web services by obtaining the service WSDL. MULE will create a dynamic proxy for the service then invoke it.

Table 2-2 MULE provider

2.3.2.4 Connector
A connector is the object that sends and receives messages on behalf of an endpoint. Connectors are bundled as part of specific transports or providers. For example, the FileConnector can read and write file system files.

```xml
<connector name="FileConnector"
  className="org.mule.providers.file.FileConnector">
  <properties>
    <property name="pollingFrequency" value="1000"/>
    <property name="moveToDirectory" value="/test-data/output"/>
    <property name="binaryMode" value="false"/>
  </properties>
</connector>
```

Figure 2.9 File connector
2.3.2.5 Router
A router is the object that does some thing on messages once they have been received by a connector, or prior to being sent out by the connector.

- **Inbound Routers**
A single event is received via a endpoint and the router controls how and if this event gets routed into the system.

- **Outbound Routers**
Once a message has been processed by a component and outbound router can used to determine which components get the result event.

- **Response Routers**
Response routers are used in request/response scenarios where event traffic is triggered by a request and the traffic needs to be consolidated before a response is given. The classic example of this is where a request is made and tasks are executed in parallel. Each task must finish executing and the results processed before a response can be sent back to the caller. [7]

<table>
<thead>
<tr>
<th>Inbound Routers</th>
<th>Outbound Routers</th>
<th>Response Routers</th>
<th>Nested Routers</th>
<th>Related</th>
</tr>
</thead>
<tbody>
<tr>
<td>Idempotent Receiver</td>
<td>Filtering Outbound Router</td>
<td>Response Aggregator</td>
<td>Nested Router</td>
<td>Filters</td>
</tr>
<tr>
<td>Selective Consumer</td>
<td>Recipient List</td>
<td></td>
<td></td>
<td>Using ReplyTo Endpoints</td>
</tr>
<tr>
<td>Aggregator</td>
<td>Multicasting Router</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Resequencer</td>
<td>Chaining Router</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Forwarding Consumer</td>
<td>Message Splitter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>WireTap Router</td>
<td>Filtering List Message Splitter</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Filtering Xml Message Splitter</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table 2-3 Routers**

- Filtering Outbound Router: A content based router uses filters to determine whether the content of the event meets to some requirements and if so, it will be routed to the endpoints configured on the router.
- Multicasting Router: The Multicasting router can be used to send the same event over multiple endpoints.
- Message splitter: A message splitter can be used to breakdown a message into parts and dispatch those parts over different endpoints configured on the router.
- Chaining router: The Chaining router can be used to send the event through multiple endpoints using the result of the first invocation as the input for the next.
2.3.2.6 Filter
A filter optionally filters incoming or outgoing messages that are coming into or going out from a connector. Filters are used in conjunction with Routers. For example,

```xml
<filter expectedType="java.lang.String"
        className="org.mule.routing.filters.PayloadTypeFilter"/>
```

It is a default filter of MULE.

2.3.2.7 Transformer
A transformer optionally changes incoming or outgoing messages in some way. For example, the `StringToByteArray` transformer String objects into byte arrays.

2.4 Case study
In this case study, we give the design concepts and implementation of order system.

2.4.1 Design of order system
Give an example about the order of supermarket. The system gets the productions’ sale-amount of last 3 months from file. According to the data, the computer automatically calculates to predict the amount of stock for next month and transfer the data to the services bus. The buyer will give the price in accordance with the amount and affirm the order. Then the order will be transferred to the services bus. Through Java Bean, the system sets the address of suppliers.

The providers receive the information (order-name, order-price, and order-amount) and issue their information (supply-name, supply-price) through SOAP to the supermarket. Finally, comparing with the price of providers, the system gives the feedback to supermarket which cooperator is appropriate.

Figure 2.10 will show the order-flow-chart.
The system gets the file containing the sale-amount. It transports the data to bus using transformer that translates the inputting message. Through calculating using the entry-service, it gets the first order from the system. The first order will be changed into java message form, and sent to buyer. Buyer has the right to decide the ordering account, change it or keep it. Next, the new order will be sent to a java bean named supply service, addresses of different suppliers will be added into another new order. According to the address, order will be sent to each supply, and each supply will give out his price. After comparison of these prices, the system will choose out the cheapest one and respond to buyer.

Table 2-4 explain the details of UMO components:
**UMO components:**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>File</td>
<td>It stores the information of productions</td>
</tr>
<tr>
<td>Entry gateway</td>
<td>Collect productions’ information from database</td>
</tr>
<tr>
<td>Entry service</td>
<td>Predict the amount</td>
</tr>
<tr>
<td>Order client</td>
<td>The buyer will give the price of the production. The order contains the information (e.g. name, amount, price, color, size).</td>
</tr>
<tr>
<td>Order gateway</td>
<td>It is responsible for aggregating the order into response for the request.</td>
</tr>
<tr>
<td>Supply service</td>
<td>Give the address of providers</td>
</tr>
<tr>
<td>Supply gateway</td>
<td>It is responsible for aggregating the providers into response for the request.</td>
</tr>
<tr>
<td>Gateway</td>
<td>Dispatches to one or more providers</td>
</tr>
</tbody>
</table>

Table 2-4 UMO components

- **Message:** through the whole system, there are three kinds most important message style:

```java
public class Order_First implements Serializable {
    private static final long serialVersionUID = 36140177016469133110L;
    private String name;
    private long[] amount;
    private long budget;
}

public class Order implements Serializable {
    private static final long serialVersionUID = -32932311913325365731L;
    private String name;
    private long amount;
    private double price;
    private boolean feedback;
    private Supplier[] suppliers;
}

public class Order_second implements Serializable {
    private static final long serialVersionUID = -32932311913325365731L;
    private String name;
    private long amount;
    private double price;
    private String supply;
    private long supplyprice;
}
```

Figure 2.11 Message

- **Message transformer:**

<table>
<thead>
<tr>
<th>Transformer</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>FileToShort</td>
<td>Change file into string</td>
</tr>
<tr>
<td>Ord_Sysin_OrderClass</td>
<td>Change string into orderfirst class</td>
</tr>
<tr>
<td>Entry_OrderFirst_Order</td>
<td>Change orderfirst class to order class</td>
</tr>
<tr>
<td>Gateway_order_ordersecond</td>
<td>Change order class to ordersecond class</td>
</tr>
<tr>
<td>----------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>ObjectToJMSMessage</td>
<td>Change object to jms message</td>
</tr>
<tr>
<td>SetSupplyAsRecipients</td>
<td>Set supply as recipients</td>
</tr>
</tbody>
</table>

Table 2-5 Message transformer

Ord_Sysin_OrderClass, Entry_OrderFirst_Order and Gateway_order_ordersecond, these three transformers all extends AbstractTransformer class, override doTransform(Object src, String arg1) function. And both FileToString and ObjectToJMSMessage supplied by MULE API. However, SetSupplyAsRecipients extends AbstractEventAwareTransformer, override transform(Object arg0, String arg1, UMOEventContext arg2) function.

- Transports:

<table>
<thead>
<tr>
<th>File</th>
<th>provider file of history sell-amount information to system</th>
</tr>
</thead>
<tbody>
<tr>
<td>JMS</td>
<td>Transfer order information over JMS</td>
</tr>
<tr>
<td>VM</td>
<td>Invoke services over VM</td>
</tr>
<tr>
<td>Soap</td>
<td>Expose supply as web services on axis, and using soap to send order information</td>
</tr>
</tbody>
</table>

Table 2-6 Transports

2.4.2 Implementation

Followings are some important points with introductions.

**Points 1:** Figure 2.12 is entrygateway XML configure:

```xml
<MX:descriptor name="EntryGateway">
    <implementation>umo.EntryGateway</implementation>
    <inbound-router>
        <endpoint address="Entry"/>
    </inbound-router>
    <outbound-router>
        <router className="org.mule.routing.outbound.ChainingRouter">
            <endpoint remote="true" address="EntryService"/>
            <endpoint address="Order" transformers="Entry_OrderFirst_Order, ObjectToJMSMessage"/>
        </router>
    </outbound-router>
</MX:descriptor>
```

Figure 2.12 EntryGateway XML

UMO entryGateway is a javabeen called umo.entryGateway. And it gets message from JMSBUS, then after ingrate information within UMO, send it as OrderFirst to VM, and anther UMO called EntryService receive the information, and after calculate out predicting amount for next month and fill back to OrderFirst, OrderFirst will transformed by transformers both Entry_OrderFirst_Order and ObjectToJMSMessage and go to JMS, addressed as order: value="jms://.order" because of ChainingRouter.

**Points 2:**
From above xml, we can see `<reply-to address="Record"/>` and `<response-router>`

```xml
<response-router>
  <endpoint address="Record"/>
  <router className="router.ResponseAggregator"/>
</response-router>
```

They always can be used together. And reply-to is used to ensure all messages sent got sent back to the same destination for Recipient List.

**Points 3:**

Following is ResponseAggregator implements a single method that works out the lowest price of suppliers after response-router receive all information of orders supplied by different suppliers.
public class ResponseAggregator extends ResponseCorrelationAggregator {

    protected UMOEvent aggregateEvents(EventGroup events)
            throws RoutingException {
        UMOEvent event;
        Order last=null;

        for (Iterator iterator = events.iterator(); iterator.hasNext();)
        {
            event = (UMOEvent) iterator.next();
            Order or;
            or = (Order) event.getTransformedMessage();
            System.out.println("supply-name:"+or.getSupply()+" supply-price:"+or.getSupplyPrice());

            if (last == null) {
                last = or;
            }
            else if (last.getSupplyPrice() > or.getSupplyPrice()) {
                last=or;
            }

            System.out.println("order: name: "+last.getOwner()+" amount of next month: "+last.getAmount());
            System.out.println("price: "+last.getSupplyPrice()+" supply: "+last.getSupply());
            return new NULLMessage(last);
        }
    }
}

Figure 2.14 ResponseAggregator
3. Quality Attribute of Software Product & tactics

This section discusses software quality attribute first. We introduce their definitions and characters in detail by quality attribute scenarios. Then it talks about tactics, that is, how to implement to achieve expected quality attribute. Finally, it presents some common styles of architecture which are the combination of some tactics and design.

3.1 Quality attribute

According to Software Architecture in Practice written by Len Bass, Paul Clements [18], Software product qualities can be categorized into three parts: system quality, business quality and architecture quality. And we emphasize on system qualities which include availability, modifiability, performance, security, testability and usability. Scenarios are used to describe quality attribute of a system detailed only after relevant requirements have been identified. And then some tactics are selected to achieve a given quality. [17]

- Source of stimulus is some entity (a human, a computer system, or any other actuator) that generated the stimulus.
- Stimulus is a condition that needs to be considered when it arrives at a system.
- Environment means the conditions where the stimulus arrives. The system may be in an overload condition or may be running when the stimulus occurs.
- Artifact is stimulated by stimulus. This may be the whole system or some pieces of it.
- Response is the activity undertaken because of the arrival of the stimulus.
- Response measure means the system should be measurable, when the response occurs.

Quality attribute scenario is tools used to specify each quality attribute required.

![Quality attribute scenario](image)

3.1.1 Qualities of the system

- **Availability**

  Availability of a system is the probability that it will be operational when it is needed.

  \[
  a = \frac{Mean \_time \_to \_failure}{Mean \_time \_to \_failure + Mean \_time \_to \_repair}
  \]

  This is typically defined as

<table>
<thead>
<tr>
<th>Source</th>
<th>Internal to the system; external to the system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Fault: omission, crash, timing, response</td>
</tr>
<tr>
<td>Artifact</td>
<td>System's processors, communication channels, persistent storage, processes</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal operation; degraded mode (i.e., fewer features, a fall back solution)</td>
</tr>
<tr>
<td>Response</td>
<td>System should detect event and do one or more of the following: record it notify appropriate parties, including the user and other systems disable sources of events that cause fault or failure according to defined rules be unavailable for an interval, where interval depends on criticality of system continue to operate in normal or degraded mode</td>
</tr>
<tr>
<td>Response Measure</td>
<td>Time interval when the system must be available Availability time Time interval in which system can be in degraded mode Repair time</td>
</tr>
</tbody>
</table>

Table 3-1 Availability general scenarios

For example, within HR system, if electrical power goes off suddenly, standby electrical source should be applied to start up system again and information should be resumed according to persistent storage. Availability is the time that the system failure is observable by user / time that the system failure is observable adding time to repair.

Scenario: [18]

![Figure 3.2 Scenario](image)

- **Modifiability**
  Modifiability is about the cost of change. It brings up two concerns: what can change and when the change is made and who makes it. And once change has been specified, many actions should be taken, which all should be measured.

  Modifiability General Scenario Generation:[18]

<table>
<thead>
<tr>
<th>Source</th>
<th>End user, developer, system administrator</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Wishes to add/delete/modify/vary functionality, quality attribute,</td>
</tr>
<tr>
<td>Artifact</td>
<td>System user interface, platform, environment; system that interoperates with target system</td>
</tr>
<tr>
<td>----------</td>
<td>--------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Environment</td>
<td>At runtime, compile time, build time, design time</td>
</tr>
<tr>
<td>Response</td>
<td>Locates places in architecture to be modified; makes modification without affecting other functionality; tests modification; deploys modification</td>
</tr>
<tr>
<td>Measure</td>
<td>Cost in terms of number of elements affected, effort, money; extent to which this affects other functions or quality attributes</td>
</tr>
</tbody>
</table>

Table 3-2 Modifiability general scenario

For example, if within the performance subsystem of the HR system, another requirement is added that managers’ performance should be evaluated by his employees also. We should consider the cost of change. How to make the change, which makes the change, how long time will spend on the change and the time to test the changed system, all should be measured.

Figure 3.3 [18] shows the scenario,

![Figure 3.3 Scenario](image)

- **Performance**
  Performance is almost about the response time to events occurred. Such as the number of events and arrival patterns will affect performance a lot.

Performance General Scenario Generation: [18]

<table>
<thead>
<tr>
<th>Source</th>
<th>One of a number of independent sources, possibly from within system</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Periodic events arrive; sporadic events arrive; stochastic events arrive</td>
</tr>
<tr>
<td>Artifact</td>
<td>System</td>
</tr>
<tr>
<td>Environment</td>
<td>Normal mode; overload mode</td>
</tr>
<tr>
<td>Response</td>
<td>Processes stimuli; changes level of service</td>
</tr>
<tr>
<td>Response Measure</td>
<td>Latency, deadline, throughput, jitter, miss rate, data loss</td>
</tr>
</tbody>
</table>

Table 3-3 Performance General Scenario
For example, if one HR wants to query all information of one employee by his ID, the response time to this operation shows performance attribute.

Figure 3.4 [18] shows scenario,

![Scenario Diagram]

- **Security**
  Security is a measure of the system ability to keep unauthorized usage out while providing its services to legitimate users.
  Security General Scenario Generation: [18]

<table>
<thead>
<tr>
<th>Source</th>
<th>Individual or system that is correctly identified, identified incorrectly, of unknown identity who is internal/external, authorized/not authorized with access to limited resources, vast resources</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stimulus</td>
<td>Tries to display data, change/delete data, access system services, reduce availability to system services</td>
</tr>
<tr>
<td>Artifact</td>
<td>System services; data within system</td>
</tr>
<tr>
<td>Environment</td>
<td>Either online or offline, connected or disconnected, firewalled or open</td>
</tr>
<tr>
<td>Response</td>
<td>Authenticares user; hides identity of the user; blocks access to data and/or services; allows access to data and/or services; grants or withdraws permission to access data and/or services; records access/modifications or attempts to access/modify data/services by identity; stores data in an unreadable format; recognizes an unexplainable high demand for services, and informs a user or another system, and restricts availability of services</td>
</tr>
<tr>
<td>Response Measure</td>
<td>Time/effort/resources required to circumvent security measures with probability of success; probability of detecting attack; probability of identifying individual responsible for attack or access/modification of data and/or services; percentage of services still available under denial-of-services attack; restore data/services; extent to which</td>
</tr>
</tbody>
</table>
For example, in hospital system, that one employee wants to change his attendances information is not allowed. The system ability to prevent itself from modification by the illegal operation is security. It can be described as scenarios as follow: [18]

Testability
Testability means the ease with which software can be made to demonstrate its faults through testing.

Testability General Scenario Generation: [18]

Still take the hospital system for instance, if software engineers want to test the performance of the system to know whether it works well or not, time to prepare the test environment and time to performance the test are both response measure of
testability, to show the testability of this system. And the scenario show system users want to test the whole system whether work well or not when system is being delivered.

Figure 3.6 [18] shows scenario,

Figure 3.6 Scenario

- **Usability**

  Usability refers to how easy it is for the user to accomplish a desired task.

  Usability General Scenario Generation: [18]

<table>
<thead>
<tr>
<th>Source</th>
<th>End user</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Stimulus</strong></td>
<td>Wants to learn system features; use system efficiently; minimize impact of errors; adapt system; feel comfortable</td>
</tr>
<tr>
<td><strong>Artifact</strong></td>
<td>System</td>
</tr>
<tr>
<td><strong>Environment</strong></td>
<td>At runtime or configure time</td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td>System provides one or more of the following responses: to support &quot;learn system features&quot; help system is sensitive to context; interface is familiar to user; interface is usable in an unfamiliar context to support &quot;use system efficiently&quot;: aggregation of data and/or commands; re-use of already entered data and/or commands; support for efficient navigation within a screen; distinct views with consistent operations; comprehensive searching; multiple simultaneous activities to &quot;minimize impact of errors&quot;: undo, cancel, recover from system failure, recognize and correct user error, retrieve forgotten password, verify system resources to &quot;adapt system&quot;: customizability; internationalization to &quot;feel comfortable&quot;: display system state; work at the user's pace</td>
</tr>
<tr>
<td><strong>Response</strong></td>
<td>Task time, number of errors, number of problems solved, user</td>
</tr>
<tr>
<td>Measure</td>
<td>satisfaction, gain of user knowledge, ratio of successful operations to total operations, amount of time/data lost</td>
</tr>
<tr>
<td>---------</td>
<td>---------------------------------------------------------------------------------------------------------------</td>
</tr>
</tbody>
</table>

Table 3-6 Usability General Scenario

In hospital system, HR can query one employee’s information according to many ways such as his name, his ID, choosing him from the whole list, which makes using system efficiently. It is a simple example of usability.

Figure 3.7 [18] shows scenario,

3.1.2 Qualities on conceptual
Qualities, such as conceptual integrity, those are about the architecture itself.

3.2 Relationship between Architectural Tactics and Quality attributes
Architecture tactics are used to examine the design decisions. They affect and control the quality attribute response. This response may be the requirement (e.g., latency requirement). Thus, tactics are leverage for achieving quality attribute requirements. Every tactic is a design option for various architectures. Analyzing the quality of an application, we would use and refine a collection of tactics which named architectural strategy. For sake of enhancing quality, to different faults and exceptions, there are a corresponding tactics for detecting and recovering. There are six tactics for achieving the six quality attributes.
3.3 Architectural tactics
In order to achieve different quality, we use correspondingly tactics to handle.

3.3.1 Availability tactics
The goal of availability tactics is to detect fault at first, then recovery and get fault prevention. And ways to approach availability involve redundancy, health monitoring to detect a failure and recovery. Figure 3.9 [18] shows the availability.

During running time, if code containing a fault is executed, such as, overflowing, divider as 0, and so on. How can system recognize faults? What measurements should
be took by system?

The prevalent means for fault detection are Ping/Echo, heartbeat and exception. When a fault happened, a signal is received through above ways. Then the signal is sent to monitor to get recovery.

For recovery, taking the voting gets the single processor to sure the failure of processor. Active redundancy namely hot restart is used in a client/server configuration for quick responses. Passive redundancy ensures the backup state is fresh before reusing services. Spare configures the failed components.

Shadow operation prevents the fault components restoring to service. After the component being restored and upgraded, complex the software, state resynchronization periodically update messages. Checkpoint/rollback is a recording of the statement and the response of the specific events.

In order to prevent the failure, removal from service, such as rebooting the component is avoid memory leaking. While dealing with the data which being influencing in the failure process, transactions could prevent collisions among various threads accessing the same data. Process monitor deletes the nonperforming process and initializes some adaptive state in the spare tactic. Through those steps, we get the fault masked and escape from the same fault.

Thus, the system could throw exception to detect the fault and apply for “try/catch” to recover. If a fault is executed but the system is able to recover from the fault without it being observable, there is no failure and availability for user.

### 3.3.2 Modifiability tactics

Cell phone is wide used in our daily life. Producers would like to absorb much more customers; they design kinds of tone in call, message, and so on. Cell phone has 3 tones for each function (e.g., call, message and email). And each function integrates tones in itself set. As a result of tone could be an individual function. For the sake of limit the module, we could integrate tones into tone set.

![Figure 3.10 Tone set](image)

We could get a modifiability product that is advantage for designers imploding and integrating new functions.

The goal for modifiability tactic is hiding information, that is, limit the public responsibilities and make them visible through an interface.

Figure 3.11 [18] shows modifiability,
Maintain semantic coherence is to make sure that these responsibilities work together without extra reliance on other modules. Assuming that expected changes will be semantically coherent, it needs a way to get a special assignment of responsibilities. For getting a generally module which require a widely input, the various changes can be made by modulating the input rather than by modifying the module. What’s more, limiting possible options could abate the modifications simultaneously. During the modification, there exist ineluctability ripple effects. The syntax, semantics, sequence of data and service’s changes lead to the modification generating the ripple effects. Making the avoidance of ripple effect, hide information, maintaining existing interface, restrict communication paths and using an intermediary. Reducing the number of modules to be changed—time to deploy and allowing non-developers to make changes, defer binding time supply the cost of requiring additional infrastructure to support the late binding. In the runtime registration supports plug-and-play operation at the cost of additional overhead to manage the registration. Configuration files are intended to set parameters at startup; Polymorphism allows late binding of method calls; Component replacement allows load time binding; Adherence to defined protocols allows runtime binding of independent processes.

Figure 3.11 Modifiability
3.3.3 Performance tactics
In general, what we are desirous to is a good performance product. It is well-known that maximum CPU utilization obtained with multiprogramming. So allocate limited number of CPU (now we could use 2 CPUs to improve utilization) to processes that are ready to execute affect performance of the system. Effective scheduling could get maximum throughput, minimum turnaround time, minimum waiting time, minimum response times which all reduce latency. When many processes coming at the same time, we could adopt following scheduling policies to allocate CPU: FIFO (first-in/first-out); SJR (shortest-job-first); RR (round robin); multilevel queue scheduling (a process can move between the various queues; aging can be implemented this way). And resource contention could be seen in many aspects, such as memory allocating, IO, file systems and so on.

A processing is touched off; the system should allocate resources. But the processing is blocked by resources contention. If resources could not be allocated propriety, the system may generate a deadlock. Therefore, we can classify performance tactics into three: resource demand, resource management, and resource arbitration.

Figure 3.12 [18] shows performance,

![Figure 3.12 Performance](image)

When two event streams are touched off, two stream traits demand: how often each requirement is made, that is, arrival rate. And how much of a resource is consumed by each request, that is, execution time. Those lead to latency. In order to reduce latency, we could adopt following measurements for resource demand:

- Increase computation efficiency
- Reduce computational overhead
- Manage event rate
- Control frequency of sampling

For resource management:

- Introduce concurrency
- Maintain multiple copies
- Increase available

And for resource arbitration:

- Scheduling policy

Response generated within time constraints
demand:
● Control the demand of resources. Declare the system’s available resources and each event’s maximum requested resources.
● Manage the frequency of sampling.
● Manage event rate. Reserve enough time for each event’s execution, then spring the next event.
● Bound queue sizes. Controls the maximum number of queued arrivals and arrival rates.
● Bound execution times. Place a limit on how much execution time is used to respond to an event.
● Increase computational efficiency. Improving the algorithms used in critical areas reduces the demand for processor time.

When resources demand may not be controllable, the management of resources affects response time. As a mature system, single resource obviously dissatisfies with so many requirements. Thus exploiting and managing multiply resource to achieve performance goals brings challenges to manage single resource. Introducing concurrency which get process running in parallel to reduce the blocked time is an effective way to load balance of resources allocating. Moreover, increase available resources, such as additional processors, additional memory, to reduce latency. Maintain multiple copies can help reduce the overhead associated with moving data around.

Even though controlling resources demand and managing resources, the resource must be scheduled. We consider scheduling in two parts, priority assignment and dispatching. Some common scheduling policies are: first-in/first-out; fixed-priority according with semantic, deadline and rate; dynamic priority scheduling associated with round robin, earliest deadline first; a static cyclic.

3.3.4 Security tactics
When you login your email, you must input both your address and code correctly. It could be keep security. What we want to get are security architectures. Therefore, the avoidance of attacks is important for architectures. Ensuring the security architectures require detecting the latency attacks, then resisting attacks and recovering from attacks. In this way, validating the users, keeping data confidentiality and integrity, limiting access and exposure is an insurance of architectures.

Figure 3.13 shows security, [18]
3.3.5 Testability tactics

Through evaluating architectures, we could get the architecture being good or not. The interface is the key access for testing. Through controlling the input and output capture information about interface and separating or specializing interface specialize variable values for a component get the result of a test. Figure 3.14 shows testability. [18]
3.3.6 Usability tactics
The target of architecture is usability for user. When the user gives a request, the designer should generate a model of the task, a model of the user, a model of the system. In the testing tactics, the user interface is expected to change frequently. The user will give appropriate feedback and assistance to improve the quality of architecture.
4. SOA impact on Quality

This section analyzes SOA impact on quality attributes from pros and cons. What are the reasons for the pros and cons, and what can be done to improve disadvantages? How can it get the tradeoffs to satisfy quality attribute requirements? There are two sorts of quality: positive and negative. We divide interoperability, flexibility, reusability into positive aspects. The rest are negative aspects.

4.1 Positive effect

Compared to other architecture, the obvious advantages of SOA are interoperability, flexibility, reusability. It also can be said in this way, SOA is build up for reaching these qualities, to meet impending requirements of enterprises and software development.

4.1.1 Interoperability

Interoperability means the ability of a collection of communication entities to share specific information and operate on it according to an agreed upon operational semantics. [2] Interoperability is the most benefit of SOA, especially now, web service is used to build SOA, with help of XML, soap, WSDL. Business Process Execution Language [BPEL] is also developed over the last years. Meanwhile, techniques such as ESB always adopted, information can be transported in a good way.

4.1.2 Flexibility

Flexibility for architecture is important because the business environments always change, and software system also should be changed to suit the business environment to supply good service quickly and cost-effectively. And flexibility is improved a lot in SOA because SOA is loosely-coupled, meaning there are few dependencies between services and also because of use of various Web standards. First, SOA can add new services easily; Second, Some services can be changed that do not require changes in the service interface, so other services will not be affected. But, if service has been published and used by applications, it is difficult to change interfaces again, if interface has to be changed, a lot other applications should be changed too.

4.1.3 Reusability

Maximal reuse of IT assets is another important benefit of SOA. With effective service-based software reuse programs in place, IT organizations can build up libraries of functionality that can be used directly or just modified a litter to meet new requirements. [9] Also, it means using service-enable exiting assets to build up SOA to ensure consistency throughout your company and cut development time, to save time and money with help of techniques of integration and information transportation.

4.2 Negative effects

It is inevitable to bring the negative attributes in SOA. The effective way is to analyze the negative attributes and improve the SOA.
4.2.1 Performance
Performance is always an important quality attribute that is negatively affected in SOA. The key factors contribute to performance issues are:

- Distribute computing. Services and clients always distribute at different machines, and different containers. It will take a lot of time to communicate over the network.
- For Interaction protocol, a call is often used to locate the desired service, which wastes extra time to accomplish a transaction.
- Another performance cost is to make services interoperate on different platforms. All intermediaries, such as stubs, skeletons, SOAP engines, proxies and other kinds of elements or framework, will have a negatively impact on performance.
- The use of a standard messaging format also increases the time to process a request.

Just as referred before, nowadays, the technical used to realize distribute computing and make service interoperate is adopting simple object access protocol (SOAP) as the standard of transporting encapsulation data. And SOAP is a collection of XML, which is the standard messaging format for web service. So, here is the entry point to improve performance.

<table>
<thead>
<tr>
<th>Factors</th>
<th>Effect</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>SOAP</td>
<td>SOAP binding proxy, create many SOAP object, sequence data</td>
<td>Use its own XML environment to supply SOAP binding</td>
</tr>
<tr>
<td>Resource</td>
<td>Each request needs a set of new resource (e.g. CPU, bandwidth, object)</td>
<td>Resource schedule, cache services</td>
</tr>
<tr>
<td>XML Database</td>
<td>Correlative database, no flexibility</td>
<td>Use local XML durative</td>
</tr>
<tr>
<td>Transport</td>
<td>Message transport</td>
<td>Optimizing transport</td>
</tr>
<tr>
<td>XML</td>
<td>Base-script language of SOA, prolixity, latency</td>
<td>Service-mesosphere, message parse, decompress XPath and XQuery</td>
</tr>
<tr>
<td>Multi-processors concurrent</td>
<td>SOA extending services</td>
<td>Apply Bean asynchronous</td>
</tr>
</tbody>
</table>

Table 4-1 Performance

4.2.2 Security
Security is a critical problem should be solved in SOA system to meet business requirements. The major reasons to impact security directly contain:

- XML, SOAP, WSDL and UDDI are open standards, construct up the whole SOA system. However, these standards do not consider security problems. For instance, messages often contain data in text format, such as XML, and metadata is embedded, which means someone can intercept a message and fetch out personal information.
- Because of losing-couple, web services were designed to be able move efficiently through firewalls while this actually decreases their usability in this
Because the SOA stresses machine-to-machine interaction, while most IT security is based on human-to-machine interaction, security is a pressing problem for SOA systems.

Third-party services also built into SOA approach. So identity of external service provider has to be authenticated and at some time, when a system sends classified data to the external service, the data should be protected for both transmission and storage.

An SOA solution may look up services in a public directory. It is important to ensure that information in the directory is up to date and was added by valid publishers.

SOA system can be attacked by these three normal ways:

- In authorization, you establish whether a particular user has the permission to proceed with the task it is requesting. In authentication, you prove that the user is actually the user it claims to be. As the figure 4.1 [16] shows, in an unsecured SOA, the unauthorized machine may fake its identity to connect system.

![Figure 4.1 Unsecured SOA](image)

- In an unsecured SOA, as shown in figure 4.2 [16], an unauthorized machine can intercept into system and pass the messages to other unauthorized users for the purpose of fraud or malicious mischief. For example, information about inventories, delivery dates, materials and so on would be stolen or modified.
In figure 4.3 [16], unauthorized users "flood" system with service requests and render it inoperable, such as a denial of service (DoS) attack.

Many solutions are brought forward.

- SOAP message monitoring utilizes a SOAP interceptor model to intercept and monitor SOAP messages as they travel from invoking systems to web services. It is the foundation of SOA security because it gives your security solution the ability to stop and examine each message for user authentication and authorization by checking user data contained in the SOAP message header.

- Network infrastructure level is also to be considered to achieve security. For example, Web servers that host Web services can be configured to use Secure
Sockets Layers (SSLs) and digital certificates to encrypt data transmission and authenticate the communicating parties.[10]

- Security Assertions Markup Language (SAML) is a proposed standard, which provides a standard, XML-based format to exchange security information between different security agents over the Internet. It allows services to exchange authentication, authorization, and attribute information without organizations and their partners having to modify their current security solutions. [11]

- Federated authentication is a process by which multiple parties agree that a designated set of users can be authenticated by a given set of criteria. Federated authentication approach creates a Federated Identity Management System, which is a sort of pool, of authenticated users. The SOA security solution can authenticate a user by checking with the Federated Identity Management System.[12]

- A web service application proxy keeps all users away from the actual service by receiving and processing SOAP requests in place of the actual web service.

- Contract management is an SOA management feature that governs the use of a web service to eliminate usage by non-contracted parties to aid security for SOA.

- Certificates, keys, and encryption [11] are also essential for a truly secure SOA. For example, XML encryption is very effective. If XML is encrypted, the message is still in XML format, but the content is not apparent as it is hidden by using of an encryption algorithm. System that receives the message can accept it, decrypt it, and process it as XML without relying on custom or proprietary messaging standards. Also, an ongoing audit log of all SOAP message can be kept to check security.

4.3 Case study
We analyze the SOA impact on quality in the case study. We present the positive effects and give the solution to improve the negative effects.

4.3.1 Interoperability
In our project, we use web service to communication between buyer and suppliers, using SOAP. It does not matter buyer and suppliers built on different platforms, different computers, different OS, different developing tools and even in different languages. For instance, buyer can be developed in J2EE while suppliers are developed by .NET. They can communicate very well on the standards of WSDL, SOAP, and XML, which are the standard communication protocol used effectively on a worldwide scale, and make SOA to be an effective concept.

Although semantic interoperability is not mature enough and need to be developed further, the interoperability is still a positive effect in our project.

4.3.2 Flexibility
Flexibility contains extendibility and modifiability. Extending an SOA means making changes, including:

- Adding new services. It is easy for users adding new services with loosely coupled architecture. The users update their code to being in accordance with these
new services. For example, we can add a stock module into the application of order. The stock module is used to record the amount of stocks and implement the order. It could add a new UMO of implementing the order without affecting other modules.

- **Update existing services.** Updating existing services could change the independent interfaces. The entire business is deployed with various changing of interface. But the service users’ application must be able to handle any changes to the interface. Thus, it could keep the effective of SOA.

  Modifiability is the ability to make changes to a system quickly and cost-effectively [13]. The dependent, self-contained, modular characteristics of services bring the modifiability of SOA.

  Let’s take a look at the flexibility in the example. First, we can add various services into the system, such as auditing, stock and so on. These new function modules extend this system. Second, update existed services. Third, the system could dispose and integrate different services, for instance, http, file, email.

### 4.3.3 Performance

In order to promote SOA, it is inevitable to reduce some quality (e.g. performance, security) to compensate others (e.g. Interoperability, flexibility). In general, performance involves three aspects: the response time to events occurred; throughput (how many requests overall can be processed per unit of time); efficiency.

There is no limitation of platform, languages in SOA. Thus, the protocol standard should be universal. In the communication, the call of protocol to a directory of services to locate the desired service increase the total time needed to perform the transaction. The WS-I protocol is widely apply in the application of order example, such as, SOAP. However, when the message transport in the communication, we adopted the JMS message and VM channel which should call different protocol.

The complex communication of services increases the response time through network, typically internet. There is no fixed range for communication latency. What’s more, in case of a block occurring in the communication of network which results in the collapsing of system.

XML is widely adopted and used throughout the Web Services infrastructure as the data format. XML documents may embed metadata describing the structure of the data. However, not all data documents are available for different module. The redundant of XML reduce the throughput. Abundant XML generate when the SOAP effected.

### 4.3.4 Security

No measures have been taken for security problem in our project. It is completely open. In reality system, it is critical. The major reasons and solutions in market have been list out above and now we are looking into our project and give out some tips that we can do to improve security.

- **Encryption:**

  The message transported between buyer and suppliers in XML format, including the information of name, price and so on. If someone intercepts a message, he can read it clearly. So encryption should be placed to protect privacy. Security manager of
MULE can be configured with one or more encryption strategies that can then be used by encryption transformers, security filters or secure transport providers such as http provider. [5]

```xml
<security-manager>
  <encryption-strategy name="PBE"
    className="org.mule.impl.security.PasswordBasedEncryptionStrategy">
    <properties>
      <property name="password" value="mule"/>
    </properties>
  </encryption-strategy>
</security-manager>

• Authentication:
  Trust must be built into the security of supplier service, that it, the identity of supplier should be authenticated.

MULE has a default security implementation that uses Acegi Security, which is Spring-based implementation and also provides interceptors that can enable method-level authorizations on your service components. [http://acegisecurity.org/] It supports HTTP BASIC authentication, HTTP Digest authentication and various authentications. Alternatively, user can implement the single-method UserDetailsService interface and obtain authentication details from other place he likes. Besides Acegi Security, user can use other security technologies, such as Java Authentication and Authorization Service (JAAS) [14] and Pretty Good Privacy (PGP) [15].
5. Conclusion

In this chapter we will evaluate the relationship between SOA and quality and the thesis as whole. We conclude the experience through the analyzing the relationship. Finally, we present the future work.

5.1 Evaluation

In this paper, we want to research the relationship between quality attributes and SOA. Quality attribute requirements drive the design of software architecture. An architecture meeting the quality attribute requirements are vital to the success of those systems. An SOA comprises a set of services. It is important to check how an SOA supports quality attributes. This report examines several quality attributes, such as interoperability, flexibility, and performance, explores how an SOA supports them, and identifies issues related to them. The test for check how this information impacts the use of SOA is another topic that will be the subject of future reports.

From the connectivity aspect, ESB, we use the open source-MULE to implement the case study. Through the experiment we analyze the quality attributes in detail.

In the beginning of this thesis, we start with the introduction Service Oriented Architecture, its features and discuss its advantages and disadvantages briefly. Second, we pay attention to Enterprise Service Bus, one effective solution for connectivity of SOA. Next, we introduce MULE, an open source of ESB product, and then implement a project named Order using MULE.

As we exemplify above, we deploy the ESB. Through the ESB, we discuss the quality attributes (e.g. interoperability, performance, security, etc.) of SOA in detail. Furthermore, we give some advices to improve quality of SOA, especially on performance and security, and explain and analysis those on our case study.

All in all, the advantages of SOA/ESB overwhelm its drawbacks, for the modern business systems to reduce developing cost and time.

5.2 Conclusion

Our goal for this thesis is to find the relationship between the quality attributes and SOA. We analyze some quality attributes in a SOA. First, we give the background knowledge to bring forward the SOA, ESB, the use of MULE and the quality attributes. Second, we configure the SOA from ESB aspect and present quality attributes’ impacts through case study.

From learning, configuring the SOA and analyzing the relationship between the quality and SOA, we get the following experience:
SOA is an architecture style, packaged as services. Because of its feature, this architecture could cut the development time and cost. From the connectivity layer, ESB bring the simple management of business processes and support the interoperability. However, the SOA features impact on quality attributes. Although there exists positive and negative quality attributes for a SOA, it is necessary to maintain typical features of SOA. Therefore, it should compromise other quality attributes. For the negative quality, we put forward to the solution to improve quality.

5.3 Future work
There is a long way to go to implement SOA worldwide and effectively. And we should find a tradeoff between different qualities to fully meet business goals, and security is another big issue which should be researched more. What’s more, how to test the quality impacting on the use of SOA is another interesting topic.
Reference

Security in a Loosely Coupled SOA Environment By Eric Pulier and Hugh Taylor
[17] Software architectures.com est.2002