Chapter 6

Dependability of Service-Oriented Software

Service-oriented software is distributed and needs to move data over network. Dependability, including security, availability, and reliability impose a bigger challenge to such software than traditional software. This chapter briefly discusses the dependability of service-oriented software, and more emphasis will be put on the security issues.

6.1 Basic Concepts

6.1.1 Dependability

Dependability of a system is defined, by Jean-Claude Laprie in 1985, to be the system’s ability to deliver specified services to the end users so that they can justifiably rely on and trust the services provided by the system. Dependability includes three aspects of a system: attributes, means, and impairments, each of which contains several elements, as shown in Figure 6.1. The dependability attributes, safety, vulnerability, confidentiality, and data integrity are called security attributes.

![Figure 6.1 Dependability definition](image)

---


Property of Kendall Hunt Publishing
**Reliability** ensures continuity of service in a given period of time. More precisely, the reliability of a system is a function $R(t)$, which is the probability that the system has survived in the time interval $[0, t]$, given that it is operational at time 0. A related function $F(t)$ is the failure probability, which is probability that the first failure occurs in the time period $[0, t]$, where $R(t) = 1 - F(t)$. For a non-redundant electronic system, the reliability function can often be characterized by an exponential function $R(t) = e^{-\lambda t}$, where $\lambda$ is called the failure rate of the system.

**Maintainability** of a system is a measure of the ability of the system to undergo maintenance or to return to normal operation after a failure. Maintainability is often characterized by the repair rate $\mu$ of the system.

**Availability** ensures the readiness of service at time point $t$. The availability function $A(t)$ function of a system is the probability that the system is working at time $t$. For a nonredundant electronic system, the availability of a system can often be calculated by

$$A(t) = \frac{\mu}{\lambda + \mu} + \frac{\lambda}{\lambda + \mu} e^{-(\lambda + \mu)t}$$

Notice that, for a large $t$, the second item in the formula will be very small and $A(t)$ becomes a constant. Furthermore, if the repair rate $\mu$ is 0 (the feature of the system surviving to the first failure), $A(t) = R(t)$.

**Vulnerability** describes a problem or weakness, such as a programming error or common misconfiguration, which allows a system to be attacked or broken into.

**Confidentiality** ensures that information is accessible only to those authorized to have access.

**Data integrity** or **message integrity** refers to the validity and consistence of data or message.

**Safety** ensures nonoccurrence of catastrophic consequence on the environment, such as human life lost and major economic impact.

**Security**, including vulnerability, confidentiality, integrity, and safety, deals with malicious attacks, whereas reliability and availability deal with faults, errors, and failures caused by imperfect development and operation environment.

Another related concept is the **quality of service**, which is based on ISO (International Standard Organization) 8402 [1986], which defines quality as “the totality of features and characteristics of a product or service that bears on its ability to meet a stated or implied need.” This is a high-level and generic definition. Different fields have different interpretations of the quality of service; for example:

- Network quality: represents the transmission rates, error rates, and other characteristics that can be measured, improved, and, to some extent, guaranteed in advance.
- Software quality: is the degree to which software conforms to quality criteria. Quality criteria include:
  - Economy, Correctness, Resilience, Integrity, Reliability,
  - Usability, Documentation, Modifiability, Clarity,
  - Understandability, Validity, Maintainability, Flexibility,
  - Generality, Portability, Interoperability, Testability,
6.1.2 Dependability Attributes and Quality of Service

Dependability attributes and quality of service (QoS) are often used interchangeably, although the quality of service is more frequently used from the user’s perspective, whereas the dependability attributes are more frequently used by the system designers, because dependability also includes the concepts of the impairments that impact the dependability attributes and the means that improve the dependability attributes. Figure 6.2 shows the technologies supporting the functionality development, as well as the dependability-related standards defined by the industry and by the standard organizations for service-oriented software development.

Dependability is not a feature that can be added to an existing application. Dependability must be inherently tied to the functions and must be planned as one of the basic requirements in the early design stage. Distributed service-oriented software development requires extensive and additional security, in contrast to centralized software development. Participating parties must be able to identify each other and place limits on what others can do. Communication between parties should be protected against all kinds of malicious attacks.

Figure 6.2 Dependability attributes defined for service-oriented software development

6.1.3 Security Issues in SOA Software

Malicious software, or malware, is software designed to infiltrate or damage a computer system without the owner’s informed consent. There are a variety of forms of hostile, intrusive, or annoying malware. Most Internet attacks are embedded in malware. According to the latest reports from Virus Bulletin (http://www.virusbtn.com/), the number of malicious code incidents is continuously expanding. The attack trends, highlighted by Symantec (http://www.symantec.com/enterprise/threatreport/), show that the SQL-slammer worm during the third quarter of year 2006 was the most common attack, accounting for 45 percent of all attacks.
In general, information security deals with integrity, confidentiality, and safety, as well as availability (service disruption). Some attacks harm integrity of information, others disclose confidential information, and yet other attacks affect the system availability. For example, denial-of-service (DoS) attacks only cause degradation in system availability; however, malicious code attacks can lead to a combined effect. One can use a specific type of malware (e.g., a Trojan horse) to compromise a system, damage the file system (integrity aspect), and at the same time steal user account information, and then extract and crack passwords from the stolen information (confidentiality aspect).

The database slammer worm, often implemented in SQL query command injection, has the ability to steal information and paralyze the database immediately. Rootkits, which are mainly used for the purpose of system administration or protecting licensed systems, threaten confidentiality of assets. Rootkits are used by hackers to hide or protect malicious codes. Sony BMG Entertainment’s notorious rootkit software was originally programmed to protect against fraudulent copying of CDs. Sony was forced to recall millions of CDs after a Windows expert discovered that copy-control software, included in some of Sony’s titles, used controversial Rootkit cloaking techniques to hide itself on the computer. Sony issued the recall after hackers began distributing malicious software that exploited Sony’s cloaking mechanism. According to the earlier reports from Symantec, Klez.A, and Kama Sutra worms cause damage to file systems of compromised systems. Klez.A is programmed to infect systems with the ElKern virus, perform large-scale e-mailing, and truncate file sizes to zero byte. The Kama Sutra worm deletes files. Obviously, these two worms cause integrity violation.

Often, the terms worm and virus are used to denote such hostile codes. Here, worms, viruses, and malware can be used interchangeably. However, not all worms are used in the negative sense. The predator worm, also known as counter worm or killer worm, is used to fight other malicious codes. This good-will mobile code is also a worm that propagates through the backdoor of another worm. They are initially launched to serve as proactive countermeasures among gangs of worm creators.

Worms are spread via worm carrier mechanisms of various types. Some worms are able to replicate themselves and spread automatically through vulnerable nodes after their initial deployment. However, some types of malware are not self-replicating. Some worms, as a subset of malware, are self-replicating hostile codes, while some can be hostile but not self-replicating. Worms are, indeed, self-replicating and self-propagating malware in the sense that they can spread automatically through interconnected but vulnerable systems.

Another subset of malware, inadvertently called viruses, which are somewhat similar to worms, is also in the category of self-propagating codes. Thus, because of their motilities (capable of moving spontaneously), viruses and worms can cause a vast number of incidents. Both worms and viruses often have no specific targets. However, they may scan their targets in a random manner, or even in cooperation from compromised nodes. Table 6.1 lists the common attacks used by malware.

**Authentication** and authorization are common techniques used to ensure basic security. Authentication requires an entity (a person or a computer) to present proof of identity, which verifies “are you who you say you are?”

**Authorization** confirms that an entity, after its identification, is entitled to access a resource, or it answers the question: what are you allowed to do, after you are confirmed who you are?
Table 6.1 Common attacks used by malware

<table>
<thead>
<tr>
<th>Attack</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Cross-site scripting (XSS)</td>
<td>Script programs are used as input data and are executed in browser on client side if the browser does not have the ability to filter the script.</td>
</tr>
<tr>
<td>Denial of service (DoS)</td>
<td>The attacker floods the network with fake requests, overloading the system and blocking regular traffic.</td>
</tr>
<tr>
<td>Eavesdropping</td>
<td>The attacker uses a sniffer to read unencrypted network packets as they are transported on the network.</td>
</tr>
<tr>
<td>Rootkit and hidden-field tampering</td>
<td>The attacker compromises unchecked (and trusted) hidden fields stuffed with sensitive data in the Web forms</td>
</tr>
<tr>
<td>One-click</td>
<td>Malicious HTTP posts are sent via script when a link is clicked.</td>
</tr>
<tr>
<td>Session hijacking</td>
<td>The attacker guesses, dictionary attack, or steals a valid session ID and connects over another user’s session.</td>
</tr>
<tr>
<td>Database query / SQL slammer / injection</td>
<td>The attacker inserts malicious input that the code blissfully concatenates to form dangerous SQL commands.</td>
</tr>
</tbody>
</table>

A typical XSS attack consists of the following steps:
1. Enter script code, for example, `<script>command(parameter)</script>`, in the textbox where user name is expected.
2. If an error message complains that user name cannot contain special characters such as “<” and “/”, then, use browser menu command to View Source of the Web page.
3. If the page uses JavaScript to check whether the entered user name contains special characters, replace the JavaScript code and disable the check.
4. Use the modified HTML code to create a new Web page and now enter the script code again: `<script>command(parameter)</script>`. This time, the special characters will be accepted by the modified page and script is sent to the browser.
5. If the browser’s security level is set to allow script in input, then, the attacker’s script will execute.

The SQL connection stream injection uses a similar idea. When a malicious user enters SQL commands in the user input part, the SQL may consider the data as command if the security properties are not properly configured.

Encryption is the major technique to prevent eavesdropping. There are two major encryption techniques: secret key system and public key system.

**Secret key systems:** In such security systems, a secret key is used to encrypt the data and message. For example, cipher encryption adds an integer (key) to each character. The problem with secret key systems is how one can safely transfer the key to the receiver.

**Public key systems:** In such security systems, the sender or receiver creates two keys, and one key is published (public) while the other key is kept secret (private). The public-key systems can be used for two purposes: encryption and digital signature. In the public-key encryption system, anyone who wants to send a confidential document to the receiver uses the public key to encrypt the
document. Only the receiver with the private key can decrypt the document. Consider an analogy.
A bank wants its customers to send confidential documents to the bank without security concerns. The bank delegates the post office to distribute unlocked safe boxes, which only the bank has the key to open. Anyone can buy the safe box, put the confidential document in the box, lock (encrypt) the box, and mail the box to the bank.

The other public-key system is digital signature, in which encryption is held private, whereas the decryption key is made public. Nobody, except the sender, can create the encrypted document, but everyone can decrypt the document.

Currently used public key systems are secure if implemented properly. Without a vulnerability (weakness), it would take over 100 years to break in the system using systematic exploiting, such as dictionary attack. However, due to the complexity of software and possibly hardware, vulnerabilities often exist. For example, Microsoft Security Advisory (http://technet.microsoft.com/en-us/security/bulletin/MS10-070) in September 2010 acknowledged that a vulnerability was found in ASP.Net, which could allow encrypted information in ASP.Net applications to be disclosed through padding oracle attacks. Padding oracle attack is an approach that generates a large number of incorrectly encrypted messages of different types and sends these messages to the system for decrypting. The system may return a different type of error message in responding to different errors in encryption. The attacker may use the error messages types to figure out the secret key used for decryption. The ASP.Net vulnerability applies to .Net 1.1 through .Net 4.0. The vulnerability was fixed through a security update. A user can also fix the problem by enabling the customErrors mode in the Web.config file:

```xml
<system.web>
  <customErrors mode="On" defaultRedirect="customErrorPage.aspx" />
</system.web>
```

This element will return the same error message to all kinds of errors, preventing the attackers to exploiting the secret key used for decryption.

## 6.2 Access Control in Web Applications

Most Web sites offer public portions and restricted portions. The restricted potions can be further authorized to different users. For example, some portions are allowed for all employees, while some portions are allowed for administrators only. When a Web site stores its customers’ private information, such as credit card numbers, access to the information must be restricted. This section will discuss different security considerations and options, including forms security, IIS, and Windows-based security.

### 6.2.1 IIS and Windows-Based Security Mechanisms

Security of ASP.Net Web applications and services deployed on IIS (Internet Information Services) and windows are managed by IIS security and Windows security systems, as shown in Figure 6.3. Any Web services running in IIS root directory, or in a virtual directory created in the IIS directory are projected by the IIS security mechanisms. All accesses to a Web application or Web service must go through IIS, which assigns every request an access token. The access token enables the
Windows operating system to perform ACL (Access Control List) checks on resources targeted by the request. Each file can be given an access group (anonymous or with credential).

IIS also supports IP address and domain name restrictions, enabling requests to be granted and denied based on the IP address or domain of the requestor. IIS supports encrypted HTTP connections using the Secure Sockets Layer (SSL) family of protocols. SSL does not protect resources on the server. Instead, it prevents eavesdropping on conversations between Web servers and remote clients.

IIS supports multiple levels of access control, which can be selected in the administrative tool in the control panel of the Windows computer:

1. Anonymous: No access control.
2. Basic authentication: The security token is passed to the Windows operating system, and Windows user name and password are used to authenticate users. The password is sent in clear text.
3. Integrated Windows authentication in NTLM or Kerberos security methods: It uses the Windows login credential to authenticate users. To access a service in IIS, the service requester must have a Windows user account created.
4. Forms authentication: Developers can design their own security mechanisms outside IIS. If Forms authentication is deployed, all unauthenticated requests are redirected to an HTML Form using HTTP client-side redirection. A client provides credentials and submits the Form.
5. Passport authentication: A centralized authentication service provided by Microsoft that offers a single logon and core profile services for member sites.

Application of different security options can be specified in the Web.config file, as shown below. If the mode = “Windows” is defined, the <identity> element can be defined to specify further security options.

```xml
<system.web>
  <authentication mode="[Windows|Forms|Passport|None]" />
  <identity impersonate ="[true|false]" />
  <authorization> ... </authorization>
</system.web>
```

If the <identity> element is not present, the default will be applied, which is the stronger (the most secure) security option: The application will inherit the identity of the worker process (aspnet_wp.exe), which runs using an account (defined in machine.config) with weaker privileges than the local system account. By doing so, an intruder will not have the administrative access even if security is breached (have the administrator’s password). This is because the local system account has access to almost all resources on the local computer not specifically denied to it. The option may be too strong, leading to denying a user from accessing any disk file.
If the `<identity>` element is not defined, its impersonate attribute can take a true or a false value.

- `<identity impersonate = "false" >` ASP.NET impersonates the token passed to it by IIS, which is either an authenticated user or an anonymous Internet user account. This is the common use of Web applications.
- `<identity impersonate = "true" username = "domain\user" password = "password" />`

In this case of `impersonate = "true"`, ASP.NET impersonates the token generated using an identity specified in the Web.config file given in the identity element. This feature is useful for developers to test the program using a different account.

### 6.2.2 Forms-Based Security

Forms-based security uses the Web.config file to define the detailed security policies. The second element, `<system.web>` in the Web.config file can be used to define, among many other functions, the authentication and authorization to a Web application. Table 6.2 lists a part of the sub elements of `<system.web>` in Web.config. The code template that follows shows some of the sub elements listed in Table 6.2.

<table>
<thead>
<tr>
<th>Element</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anonymous</td>
<td>Configures anonymous identification for application authorization. This is required to identify entities that are not authenticated when authorization is required.</td>
</tr>
<tr>
<td>Identification</td>
<td></td>
</tr>
<tr>
<td>Authentication</td>
<td>Configures ASP.NET authentication support.</td>
</tr>
<tr>
<td>authorization</td>
<td>Configures ASP.NET authorization support.</td>
</tr>
<tr>
<td>browserCaps</td>
<td>Configures the settings for the browser capabilities component.</td>
</tr>
<tr>
<td>compilation</td>
<td>Contains all compilation settings that are used by ASP.NET.</td>
</tr>
<tr>
<td>Globalization</td>
<td>Configures the globalization settings of an application.</td>
</tr>
<tr>
<td>healthMonitoring</td>
<td>Configures an application for health monitoring.</td>
</tr>
<tr>
<td>hostingEnvironment</td>
<td>Defines configuration settings that control the behavior of the application hosting environment.</td>
</tr>
<tr>
<td>httpCookies</td>
<td>Configures properties for cookies that are used by a Web application.</td>
</tr>
<tr>
<td>httpHandlers</td>
<td>Maps incoming URL requests to IHttpHandler classes.</td>
</tr>
<tr>
<td>httpModules</td>
<td>Adds, removes, or clears HTTP modules within an application.</td>
</tr>
<tr>
<td>httpRuntime</td>
<td>Configures ASP.NET HTTP run-time settings. This section can be declared at the machine, site, application, or subdirectory level.</td>
</tr>
<tr>
<td>identity</td>
<td>Controls the application identity of the Web application.</td>
</tr>
<tr>
<td>roleManager</td>
<td>Configures an application for role management. This element is new in the .NET Framework version 2.0.</td>
</tr>
<tr>
<td>securityPolicy</td>
<td>Defines valid mappings of named security levels to policy files. This section can be declared at the machine, site, or application levels.</td>
</tr>
</tbody>
</table>
The authentication will be checked against the provided user name and password. This option will allow a user to use different accounts on the Windows to perform different operations.

The problem with the Windows-based security is that the clients of the Web page must have a Windows account of the server that runs the Web applications or Web services. One can also create a Web application that has a separate security system from the Windows and the application maintains a separate list of credentials (user names and passwords). The following template shows the format of the Web.config that enables Forms security.
<configuration>
  <system.web>
    <authentication mode="Forms">
      <forms name="[name]" loginUrl="[url]"
        protection="[All|None|Encryption|Validation]"
        path="[path]" timeout="[minutes]"
        requireSSL="[true|false]"
        slidingExpiration="[true|false]">
        <credentials passwordFormat="[Clear|MD5|SHA1]">
          <user name="[UserName]" password="[password]"/>
        </credentials>
      </forms>
    </authentication>
  </system.web>
  <authorization>
    <allow users="[comma separated list of users]"
      roles="[comma separated list of roles]"/>
    <deny users="[comma separated list of users]"
      roles="[comma separated list of roles]"/>
  </authorization>
  <system.web>
    <authorization>
      <allow users="[comma separated list of users]"
        roles="[comma separated list of roles]"/>
      <deny users="[comma separated list of users]"
        roles="[comma separated list of roles]"/>
    </authorization>
  </system.web>
</configuration>

The upper part of the template (the first system.web element) defines the authentication through user names and password. The lower part of the template (the second system.web element) defines the authorization of users that have been authenticated.

Let us consider an example shown in Figure 6.4. The application is organized in three directories. The folder “FormsSecurity” is the root, and it has two subdirectories named Staff and Admin. The root directory contains three files: Default.aspx, Login.aspx, and Web.config.

![Figure 6.4 Organization of directories and files of the authentication and authorization example](image)

The page design of the Default.aspx, Login.aspx, and Staff.aspx are shown in Figure 6.5. The Admin folder shown in Figure 6.4 is not implemented in the example.
The source code of the Login.aspx page is shown as follows. C# code is embedded in the HTML as the server-side script.

```html
<html>
<body>
<h1>Login to access Staff page</h1>
<form runat="server">
<table cellpadding="4">
<tr>
<td>User Name: </td>
<td><asp:TextBox ID="txtUserName" RunAt="server" /></td>
</tr>
<tr>
<td>Password: </td>
<td>
<asp:TextBox ID="txtPassword" TextMode="password" RunAt="server"/>
</td>
</tr>
<tr>
<td></td>
<td>
<asp:Button Text="btnLogin" OnClick="LoginFunc" RunAt="server"/>
</td>
</tr>
</table>
<asp:Label ID="Output" RunAt="server"/>
</form>
</body>
</html>

<script language="C#" runat="server">
void LoginFunc(Object sender, EventArgs e) {
    if (FormsAuthentication.Authenticate(txtId.Text, txtPwd.Text))
        FormsAuthentication.RedirectFromLogin(UserName.Text, false);
    else
        Output.Text = "Invalid login";
}
</script>
```

The code behind the Default.aspx page is simple. One line of code is behind each button; for example, when the button “Staff Page,” is clicked on the Default.aspx page, the corresponding C# function is called, which uses

```csharp
Response.Redirect("Protected/Staff.aspx");
```
to access the StaffPage.aspx. The function will first check the Web.config file in the directory for authentication. The Web.config file in the root directory contains authentication setting given in Figure 6.4.

To access the pages protected by the Web.config file, the access will be redirected to the page address given in the loginURL attributes in the “forms” element if user is not authenticated yet. Once a user is authenticated, the login page will not be activated in the following accesses to the protected page.

The authorization is defined in the Web.config files in the directories “Staff” and “Admin,” respectively. A sample code for the Staff.aspx page is given as follows:

```
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">  <title>staff page</title> </head>
<body>
  <form id="form1" runat="server">
    <h1>Staff Page of the Camp</h1>
    <div>
      <% Response.Write("Hello " + Context.User.Identity.Name + ", "); %>
      This page contains the information about staff members who will teach and manage the camp. Only authenticated users can access this page.
    </div>
  </form>
</body>
</html>
```

The access control list in the Web.config is implemented using two subtags <allow> and <deny> of specific user name. For the <allow> element, a question mark (?) allows anonymous users; an asterisk (*) allows all users. For the <deny> element, a question mark (?) indicates that anonymous users are denied access; while an asterisk (*) indicates that all users are denied access.

Figure 6.6 illustrates flow of accesses, including authentication and authorization processes.

The example is implemented and can be tested at:

http://venus.eas.asu.edu/WSRepository/FormsSecurity/

You can test the example using the credentials shown in Figure 6.4. The example presented here has several problems. The password is stored in “Clear” text. The other options in: credentials passwordFormat="[Clear|MD5|SHA1]"> can secure the password:

- MD5: Specifies that passwords be encrypted using the MD5 hash algorithm.
- SHA1: Specifies that passwords be encrypted using the SHA1 hash algorithm.
The authentication and authorization are checked sequentially in the given order. The process will be slow if the list is long. Furthermore, if the list needs to be changed from time to time, the maintenance process can be difficult and error-prone. In Visual Studio 2012 and .Net 4.5, FormsAuthentication.Authenticate method is obsolete. The solution is to use a database to store the user names and passwords. The access and maintenance processes become database read and write issues, which are well addressed in database domain. Part II of the book will discuss database accesses and database-related Web applications. In this section, we show how we can use an XML file to store the user names and passwords, as the continuation of XML discussion in Chapter 4. In the LoginFunc part, we can call a user defined authentication function, say, myAuthenticate(), and we write our own myAuthenticate() to access an XML file. The code is given as follows:

```csharp
<script language="C#" runat="server">
    void LoginFunc(Object sender, EventArgs e) {
        if (myAuthenticate(txtId.Text, txtPwd.Text))
            FormsAuthentication.RedirectFromLogin(UserName.Text, false);
        else Output.Text = "Invalid login";
    }

    bool myAuthenticate (string username, string password) {
        string fLocation = Path.Combine
            (Request.PhysicalApplicationPath, @"App_Data\Users.xml");
        if (File.Exists(fLocation)) {
            FileStream FS= new FileStream(fLocation, FileMode.Open);
            XmlDocument xd = new XmlDocument();
            xd.Load(FS);
            XmlNode node = xd;
            XmlNodeList children = node.ChildNodes;
            foreach (XmlNode child in children) {
                // check if the username and password exist in the XML file;
                // code to access XML file and check authentication
            }
        }
    }
</script>

Figure 6.6 Authentication process

The authentication and authorization are checked sequentially in the given order. The process will be slow if the list is long. Furthermore, if the list needs to be changed from time to time, the maintenance process can be difficult and error-prone. In Visual Studio 2012 and .Net 4.5, FormsAuthentication.Authenticate method is obsolete. The solution is to use a database to store the user names and passwords. The access and maintenance processes become database read and write issues, which are well addressed in database domain. Part II of the book will discuss database accesses and database-related Web applications. In this section, we show how we can use an XML file to store the user names and passwords, as the continuation of XML discussion in Chapter 4. In the LoginFunc part, we can call a user defined authentication function, say, myAuthenticate(), and we write our own myAuthenticate() to access an XML file. The code is given as follows:
6.2.3 User Registration and Account Management

Forms-based security allows users to register and to create accounts. A typical account contains user name, user id, password, email, phone, a security question and its answer, and so on. In such applications, we will use a database to store the user account information. We will extend the example in the previous section to include the account registration and management functions.

Figure 6.7 shows the solution after a folder named Account and a number of ASPX pages are added into the project. To create a folder, right-click the project name and select “New Folder.” To add a new page, right-click the folder Account and select Add New Item, and then choose Web Form.

Once the pages in the Account folder are added, we can start to add GUI controls and code behind the control. The middle window in the figure shows the extended GUI design, in which two more buttons are added: Student Registration and Student Login, which will allow students (customers) to create their own credentials through registration and then use their credentials to login.

As user access control and account management are rather standard features. ASP.Net has created Web control and the complete code behind the controls. A few drag-and-drop operations will complete the entire design. For example, drag the Web control “CreateUserWizard” in the Toolbox, see the left window of Figure 6.7, and drop it into the Register.aspx page; the GUI is already
implemented in the control. Double click the button “Create User,” and the code that performs the registration function is given.

```csharp
using System;
using System.Web.Security; // Contains FormsAuthentication class
public partial class Account_Register : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
        RegisterUser.ContinueDestinationPageUrl = Request.QueryString["ReturnUrl"];
    }
    protected void RegisterUser_CreatedUser(object sender, EventArgs e) {
        FormsAuthentication.SetAuthCookie(RegisterUser.UserName, false /* createPersistentCookie */);
        string continueUrl = RegisterUser.ContinueDestinationPageUrl;
        if (String.IsNullOrEmpty(continueUrl)) {
            continueUrl = "~/";
        }
        Response.Redirect(continueUrl);
    }
}
```

Similarly, the other pages in the Account folder corresponding to the other Web controls in the Login part of the Toolbox can be created. For the example, the code behind the CustomerLogin.aspx is given as follows:

```csharp
using System;
using System.Web; // contains HttpUtility class
public partial class Account_Login : System.Web.UI.Page {
    protected void Page_Load(object sender, EventArgs e) {
    }
}
```

Notice that CustomerLogin.aspx is different from the Login.aspx we discussed in the previous section. The CustomerLogin page will use the user created credentials, while the Login.aspx serves as the Staff login, which uses a different authorization mechanism.

When the Web controls in the Login part of the Toolbox is used, a database storage object will be created automatically and placed in the App_Data folder of the project. The database object is named ASPNETDB.MDF, which is configured to work with SQL Server Express, with the data source name = SQLEXPRESS, which is the default name when one installs SQL Server Express on a Windows machine or on one a Windows server. If your computer or the server is using these defaults, the account registration and management pages created in the simple steps above will work. You can test the pages by adding a few accounts and login into your system using these accounts.

You need to install SQL Express on your computer in order to use the database-based account management in the aforesaid example. When you install SQL Express, do not use the default name SQLEXPRESS. If you do, you will have a connection string conflict if the server also used the default name to install the SQL Express. On the other hand, if you have installed your SQL Server Express using a different data source name, or you are using the full version of the SQL server, you
need to modify the connection string in the <connectionStrings> element in the `machine.config` file, which is located in your Visual Studio’s “Config” directory, for example, `C:\Windows\Microsoft.NET\Framework\v4.0.30319\Config\machine.config`.

The element `<connectionStrings>` of the `machine.config` file is shown as follows:

```xml
<connectionStrings>
  <add name="LocalSqlServer" connectionString="data source=.\SQLEXPRESS; Integrated Security=SSPI; AttachDBFilename=|DataDirectory|aspnetdb.mdf; User Instance=true" providerName="System.Data.SqlClient"/>
</connectionStrings>
```

You can also manage your database schema by opening the Server Explore in the View menu of the Visual Studio, as shown in Figure 6.8, which shows the Server Explore and the schema of the user account database. Database access and management will be further discussed in the second part of the book.

![Server Explorer and User Account Database](Figure 6.8)

The full example with student registration page and database is implemented and can be tested at:

http://venus.eas.asu.edu/WSRepository/FormsSecurity/

In this example, we used server controls to create the database, read the database (when login) and write the database (when register). We will further discuss the creating of database, defining the schema, and reading and writing the data in the database the database chapter in Part II of the book.
6.3 Encryption and Decryption

There are many situations in which we want to encrypt and decrypt data. For example, if the user IDs and passwords are stored in the Web.config file, text file, or database in plain text, the system administrators and anyone who has the administrative access to the server can see the user IDs and passwords. It is not acceptable for an organization that cares about protecting its user privacy. This section discusses encryption and decryption in data storing and transmitting for confidentiality and data integrity.

6.3.1 Developing Encryption and Decryption Service in WCF

Having understood the foregoing simple example, we can start to develop real service follow the pattern. We will create an encryption/decryption service.

Many encryption and decryption algorithms are of matters of national security and are classified. In this section, we will use Data Encryption Standard (DES) as an example to create basic encryption and decryption services. DES was developed by IBM in 1970 and selected by the National Bureau of Standards as an official Federal Information Processing Standard (FIPS) for the United States in 1976, which subsequently became an international standard. It is based on a symmetric-key algorithm that uses a 56-bit key encryption. As the key length is short, DES is used for low-level security purpose only [http://en.wikipedia.org/wiki/Data_Encryption_Standard]. DES uses a secret key system and block cypher encryption technique. The data and the secret key are mixed first. The mixed code are divided into eight blocks, with 6 bits in each block. Cypher encryption is applied in each block. The encrypted blocks are mixed again. The process repeats 16 times to obtain the final encrypted data. The encryption and decryption are safe if the secret key does not need to be transferred to a remote site. For example, we can use the encryption service to encrypt data and save the data into a file. Then we use the decryption service to decrypt the data before using the data. In this application, the secret key and both services stay in the same server without having to be transferred to a remote place.

Visual Studio .Net Class Library implemented a namespace called System.Security.Cryptography, which includes a number of security-related classes [http://msdn.microsoft.com/en-us/library/system.security.cryptography.aspx]. The example that follows uses the DES SymmetricAlgorithm class to create a Web service with a “string Encrypt(string)” method and a “string Decrypt(string)” method. To separate the interface from the implementation, we put the service in two files: Service.cs contains the Web interface and the Cryption.cs contains the code of the encryption and decryption methods. Both files should be placed in the App_Code folder of your ASP .Net project. Following are the steps to create the service.

**Step 1.** Use WCF Service Application template to create a service. Name the project EncryptionWcf. You will see two files in the App_Code folder: IService.cs and Service.cs.

**Step 2.** Copy the following code to replace the ServiceContract and OperationContract in IService.cs file.

```csharp
[ServiceContract]
public interface IService {
    [OperationContract]
    string Encrypt(string text);
    string Decrypt(string encryptedData);
}
```
Step 3. Copy the following class to replace the service class in Service.cs file.

```csharp
public class Service : IService {
    public string Encrypt(string text){
        EncryptionWcf.Cryption encrypt = new EncryptionWcf.Cryption();
        return encrypt.Encrypt(text);
    }
    public string Decrypt(string text){
        EncryptionWcf.Cryption decrypt = new EncryptionWcf.Cryption();
        return decrypt.Decrypt(text);
    }
}
```

Step 4. Write the Cryption.cs class

Right-click the project folder App_Code and choose Add New Item... Then, choose the Class item and name the class Cryption.cs.

The code of the Cryption.cs is given as follows. A number of library classes are used. Their key class is the CryptoStream class. It combines the encryption algorithm and a seed (array of binary numbers) to create an encryption block. The MemoryStream object provides an internal storage for the CryptoStream object to store the encryption block. Then, the StreamWrite class mixes the encryption block with the plaintext to be encrypted. As the CryptoStream object and the StreamWrite object use unstructured memory, it is safe to flush the leftover characters and terminators that can be left in the memory stream when the program uses both structured accesses (e.g., read an entire line, including the line-end terminator) and unstructured accesses (read a byte or a character at a time). Such problem needs to be dealt with when one reads a stream buffer to a file system.

```csharp
// file name: Cryption.cs
using System; using System.IO; using System.Text;
namespace Encryption {
    public sealed class Cryption {
        byte[] seed = ASCIIEncoding.ASCII.GetBytes("cse44598"); // A seed binary array for encryption
        public string Encrypt(string plainString) { // encryption using DES
            if (String.IsNullOrEmpty(plainString))
                throw new ArgumentNullException("The input string for encryption cannot be empty or null!");
            SymmetricAlgorithm saProvider = DES.Create(); // Lib class
            MemoryStream mStream = new MemoryStream();
            CryptoStream cStream = new CryptoStream(mStream,
                saProvider.CreateEncryptor(seed, seed), CryptoStreamMode.Write);
            StreamWriter sWriter = new StreamWriter(cStream);
            sWriter.Write(plainString);
        }
    }
}
```
Step 5. Build and start the service.

The aforesaid service is implemented in both ASP .Net service and in WCF service, and they are deployed for public access at:

WCF service: http://venus.eas.asu.edu/WSRepository/Services/EncryptionWcf/Service.svc

6.3.2 Developing a Secure Hashing Service in WCF

The decryption service allows the users to obtain the original data after encryption. There are situations where the encrypted data never needs to be decrypted, for example, the session ID used for identifying the browser that revisits the same session, as we discussed in Chapter 5. In such applications, we should use the one-way hashing service, instead of the encryption service. We show a simple example here of creating a secure hashing value.

The contract definition of the service is

```csharp
using System.ServiceModel;
using System.ServiceModel.Web;
[ServiceContract]
public interface IService {
    [OperationContract]
    string Hash(string data, string salt);
}
```

The service implementation is

```csharp
using System;
using System.Text;
```
public class Service : IService {
    public string Hash(string data, string salt) {
        using (var sha = new SHA512CryptoServiceProvider()) {
            var hashedString = sha.ComputeHash(Encoding.Default.GetBytes(data + salt));
            return Convert.ToBase64String(hashedString);
        }
    }
}

In the example, the parameter salt is any string that the user can use to make it more difficult to crack the hashed data, such as password and user ID. The salt is appended to the data to be hashed.

The service is deployed at:

http://venus.eas.asu.edu/WSRepository/Services/HashSha512/Service.svc

The service can be tested by the service tool deployed at:

http://venus.eas.asu.edu/WSRepository/services/wsTesterTryIt/

### 6.3.3 WCF Service Client

Different platforms can be used for creating WCF service clients, including Console, ASP .Net, Workflow Foundation, Presentation Foundation, and Silverlight. Creating a client accessing a WCF service is not much different from accessing other services. The idea is to create a proxy to connect to the endpoints and the proxy will contain the address, binding, and contract information for the client to invoke the remote service. In order to take the advantage of the additional binding option, we need to use “Add Service Reference,” instead of “Add Web Reference” in the ASP .Net when creating the proxy. Figure 6.9 shows a simple ASP .Net client that tests the WCF encryption/decryption service. A string is entered in the textbox. Once submitted, the encrypt operation is called and the string is encrypted, as shown in the line below the textbox. Then, the encrypted string is sent the decrypt function, which decrypts the encrypted string back to the original text.

**Figure 6.9** ASP .Net client accessing the WCF encryption/decryption service

The C# code behind the “Submit” button of the GUI design is given as follows.

```csharp
protected void btnSubmit_Click(object sender, EventArgs e) {
    try {
        lblEncrypted.Text = myClient.Encrypt(txtInput.Text);
    } catch (Exception ec) { lblEncrypted.Text = ec.Message.ToString(); }
    try {
        lblDecrypted.Text = myClient.Decrypt(lblEncrypted.Text);
    } catch (Exception dc) { lblDecrypted.Text = dc.Message.ToString(); }
```
Instead of writing a client to test the encryption and decryption service, the service can also be tested by the service tool deployed at:

http://venus.eas.asu.edu/WSRepository/services/wsTesterTryIt/

The Web.config file of the test client is given as follows. In the <client> element, the proxy of address, binding, and contract is stored.

```xml
<configuration>
  <system.serviceModel>
    <bindings>
      <basicHttpBinding>
        <binding name="BasicHttpBinding_IService" >
          <security mode="None">
            <transport clientCredentialType="None"
            proxyCredentialType="None" realm="" />
            <message clientCredentialType="UserName" algorithmSuite="Default" />
          </security>
        </binding>
      </basicHttpBinding>
    </bindings>
    <client>
      <endpoint address="http://venus.eas.asu.edu/WSRepository/Services/EncryptionWcf/Service.svc" bindingConfiguration="BasicHttpBinding_IService" contract="EncryptService.IService" name="BasicHttpBinding_IService"/>
    </client>
  </system.serviceModel>
</configuration>
```

By default, basicHttpBinding is used. In order to use other binding protocols, you can use Visual Studio menu: Tools → WCF Service Configuration Editor. Browse to the location of the application and select the Web.config file of the test client. Then, you can choose other binding protocols shown in Table 6.3.

**Table 6.3 Bindings supported in WCF**

<table>
<thead>
<tr>
<th>Binding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>basicHttpBinding</td>
<td>HTTP, text encoding, basic security</td>
</tr>
<tr>
<td>wsHttpBinding / wsDualHttpBinding</td>
<td>HTTP, HTTPS, text or MTOM encoding, SSL over HTTP, WS-reliable, WS-transaction</td>
</tr>
<tr>
<td>netTcpBinding</td>
<td>TCP, binary, SSL over TCP, WS-reliable, WS-transaction</td>
</tr>
<tr>
<td>netNamedPipeBinding</td>
<td>Named pipe, binary, security, sessions, reliable, transactions</td>
</tr>
<tr>
<td>netMsmqBinding</td>
<td>MSMQ, binary, security, sessions, reliable, transactions</td>
</tr>
<tr>
<td>nsmqIntegrationBinding</td>
<td>MSMQ, text, sessions, reliable, transactions</td>
</tr>
<tr>
<td>netPeerTcpBinding</td>
<td>Peer-to-Peer communications</td>
</tr>
</tbody>
</table>
6.3.4  Error Control Code

Communication between distributed and networked nodes is neither reliable nor secure. Efforts must be made in all layers of communication in order to achieve reliable and secure communication. Error control codes are widely used at the data link layer and above.

Coding theory is based on Shannon’s theorem in 1948: Even in a noisy channel, errors in data transmission can be reduced to any desired level, if a certain minimum percentage of redundancy is maintained by means of proper encoding and decoding of the data.

This theorem founded the theories of error control codes as well fault tolerant computing. Shannon's theorem did not suggest any methods for constructing such codes. Golay [1949], Hamming [1950], and many other pioneers in the area have developed Shannon's theorem into Coding Theory. Most fault tolerance techniques can be viewed as implementations of error control codes.

Coding is the representation of information (signals, numbers, messages, etc.) by code symbols or words of code symbols. Let W be the set of all possible words in a code. In order to control errors, W is divided into two subsets, C and W – C. Only the words in C are used for representing information, while words in W – C are redundant, indicating incorrect words. C is the code space, the words in C are called codewords.

As an example, we can use two lines to transmit the same bit of information. In this case, the entire output space is W = \{00, 01, 10, 11\}, and the code space \( C = \{00, 11\} \).

We can define the distance between two words \( d(x, y) \) to the bit positions in which the words x and y differ. We can further define Hamming distance to be the minimum distance between any pair of codewords in the code space. Let \( d_m(C) \) be the Hamming distance of a code space C:

\[
    d_m(C) = \min(d(x, y)), \text{ where } x \in C \text{ and } y \in C.
\]

Hamming distance determines the error detection and correction capability:

1. If \( d_m(C) = p + 1 \), the code C can detect all errors in which up to p bits are erroneous.
2. If \( d_m(C) = 2l + 1 \), the code C can correct all errors in which up to l bits are erroneous; for example, if \( d_m(C) = 3 \), C can detect up to 2 bit errors and correct 1 bit error. If \( d_m(C) = 5 \), C can detect up to 4 bit errors and correct 2 bits errors.

A few codes have been widely used in communication and in storage, among them, parity check code, checksum, arithmetic code, and combinational code.

For a given n-bit word, a parity check code appends one additional bit to each word to make a codeword. The additional bit makes the total number 1 of all codewords an even number (even party code) or an odd number (odd party code).

Assume an n-bit word is represented as \( w = (b_0, b_1, ..., b_{n-1}) \), and the encoded word is \( (b_0, b_1, ..., b_{n-1}, b_n) \), where the check bit is calculated by a codeword, where it is calculated through \( \oplus \) (XOR) operations:

\[
    b_n = (b_0 \oplus b_1 \oplus ... \oplus b_{n-1}) \quad \text{(Even-parity code)}
\]

\[
    b_n = (b_0 \oplus b_1 \oplus ... \oplus b_{n-1} \oplus 1) \quad \text{(Odd-parity code)}
\]
The Hamming distance in parity code is 2, which can detect any single bit error. Parity code is simple in encoding and decoding. It uses n XOR operations. The cost of redundancy is 1/n, and encoding and decoding delays are log(n).

Parity check code is often used at low level in the communication stack, such as Data Link layer. Checksum can be used at low level and high level. Instead of dealing with a stream of bits, it deals with a sequence of words. Let \((x_0, x_1, x_2, ..., x_{m-1}, x_m)\) be a vector of m words, with an additional check word appended to the end, where,

\[ x_m = \sum_{i=0}^{m-1} \text{mod} \ (2^n). \]

The appended word \(x_m\) is the checksum for words \(x_0, x_1, x_2, ..., x_{m-1}\). We use modulo operation to make sure that the sum does not take more space than any words in the code.

When applying checksum in communication, the sender first calculates the checksum \(x_m\), and then transmits \(m+1\) codewords \((x_0, x_1, x_2, ..., x_{m-1}, x_m)\).

The receiver recalculates the sum and checks whether the calculated sum is equal to the received checksum.

Checksum coding can be applied to blocks of consecutive words in memory or communication. If used in memory, each write operation leads to recalculation of the checksum. Checksum coding cost is low, but the detection diagnostic resolution is low. When an error is detected, no information is available to help identify which word is incorrect.

Both parity code and checksum code are separable code, which means that the check bit/word is not mixed with the data. This type of code allows the data to be used without delay before the error check is completed. If error check detects an error, the used data needs to be rolled back.

Not all codes are separable code. The arithmetic code and combinational code discussed next are not separable code.

Let \(A\) be a function. \(A\) defines an arithmetic code, if

\[ (\forall a)(\forall b)(a \in W \land b \in W \Rightarrow A(a \otimes b) = A(a) \otimes A(b)) \]

where, \(\otimes\) is one of arithmetic operations given, for example, \(+, -, \ast\).

The most widely used arithmetic code is the AN Code defined by \(A \ast N\), where, \(A\) is an integer, normally, is a prime number of greater than or equal to 3, and the operation \(\otimes = \ast\). We can easily verify that the condition of arithmetic code is met:

\[ A \ast (a + b) = A \ast a + A \ast b \quad \text{and} \quad A \ast (a - b) = A \ast a - A \ast b \]

The characteristics include: It is non-separable code, For \(A = 3\), the code uses 2 bits redundancy, and can detect a single error.

Error control code can also be defined based on combination theory. An m-out-n code or m/n code requires that each codeword consists of n bits, in which there are exactly m ones. The code is a nonseparable code. The code has \(n!/(m!(n-m)!\) codewords in \(2^n\) words. The code contains a large amount of redundancy and the cost is high. The usage of codewords is \(n!/(m!(n-m)!)/2^n\).
For example, if \( n = 16 \) and \( m = 3 \), then, \( \frac{n!}{m!(n-m)!} = 3360 \) and \( 2^{16} = 65536 \). Only about 5% of words are codewords.

The code can detect all single errors. It can also detect all unidirectional errors that change from 0 \( \rightarrow \) 1 or from 1 \( \rightarrow \) 0 errors, but not both types. The ability can be used in the situation where unidirectional errors are common; \( m \)-out-of-\( n \) code is a nonseparable code.

6.3.5 Secure Sockets Layer Cryptographic Protocols

Error control code is normally used to protect fault and errors caused by noises from network and the environment. It does not project malicious attacks. Encryption and decryption are needed to protect data over the Internet from malicious attacks. Secure Sockets Layer (SSL) defines cryptographic protocols that provide data confidentiality and data integrity (digital signature) for communications over HTTP and TCP/IP networks.

The SSL consists of a stack Protocols split into two communication layers. The first layer is the higher layer, used in the management of SSL exchanges. It consists of SSL Handshake Protocol, SSL Change Cipher Spec Protocol, SSL Alert Protocol, and HTTP. The second layer is the lower layer, consisting of SSL Record Protocol, TCP, and IP.

An SSL session is a connection between a client and server. Multiple sessions are possible between a client and server. An SSL connection is a transport that provides a type of service. Connections are of peer-to-peer relationship.

IIS uses X.509 key system in its SSL protocol to provide HTTPS secure access to the Web applications that it hosts. To apply HTTPS to your Web server, you need to request the security certificate, which consists of an open key and a secret key. After installing the certificate on the server, clients can use https connection to access the server.

Certificate of a server is similar to the driver’s license of a person. It certifies that the server has a certain level of trustworthiness in providing services. However, it does not guarantee the quality of services. Certificate is offered by independent organizations, such as GeoTrust, GlobalSign, Thawte, and VeriSign.

To install the SSL certificate, you can follow these steps in Windows server:

- Open the Internet Information Services (IIS) Manager.
- Right-click the Default Web site and select Edit Bindings.
- Select HTTPS from the Type drop-down list.
- Select the certificate from the SSL certificate drop-down list and click OK.

To test if your certificate is installed successfully, you can access a service in a browser by using the HTTPS address, for example: https://localhost/secureConnectionServices/service1.svc.

6.4 Dependable Computing in Windows Communication Foundation

Windows Communication Foundation (WCF) is the latest service-oriented computing development environment. It extends ASP .Net to better support service-oriented software development, particularly in reliability and security. WCF is a part of WinFx, which is a new
programming environment in Windows Vista, Windows 7, and Windows Server 2008 and 2010. We will use WCF to develop services in the next chapter. In this section, we will discuss the security, reliability, and transaction features offered in WCF.

6.4.1 WS-Security

WCF supports multiple security methods at multiple layers. The security methods include authentication, authorization, confidentiality, and integrity. The layers in which security mechanisms are deployed include message layer and transport layer.

Authentication: WCF supports all the authentication methods we discussed in ASP.Net sections, including IIS security, integrated Windows security, and Forms-based security. In addition, it supports the X.509 certificate, which employs the public key infrastructure (PKI) to encrypt a message to generate digital signature as credential. X.509 specifies standard formats for public key certificates and a certification path validation algorithm.

Authorization: WCF supports all the authorization methods we discussed in ASP.Net sections, including access control list and role-based authorization. In addition, it supports XSI standard, which is a claim-based authorization method.

Transport security protects data during their transportation on the network. SSL (Secure Socket Layer) is the common mechanism used for encrypting the data in HTTPS protocol. Transport security depends on the mechanism for the binding the user has selected. For example, if wsHttpBinding is used, the security mechanism is Secure Sockets Layer (SSL), as shown in Table 6.3.

Message security ensures confidentiality of message by encrypting and signing message before sending them to the transport layer, regardless if the transport layer will encrypt the data to be transported. It means that every message includes the necessary headers and data to keep the message secure. Because the composition of the headers varies, you can include any number of credentials. This becomes a factor if you are interoperating with other services that demand a specific credential type that a transport mechanism cannot supply, or if the message must be used with more than one service, where each service demands a different credential type.

Transport and message security can be defined in the Web.config file. It can be used in the program code. The following code segment from WCF online tutorial shows how to create an object of wsHttpBinding and initialize the field in the object:

```csharp
// Create the binding for an endpoint.
NetTcpBinding b = new NetTcpBinding();
// Create the ServiceHost for running a calculator service.
Uri baseUri = new Uri("net.tcp://MachineName/tcpBase");
Uri[] baseAddresses = new Uri[] { baseUri };
ServiceHost sh = new ServiceHost(typeof(Calculator), baseAddresses);
// Add an endpoint using the binding and a new address.
Type c = typeof(myService);
sh.AddServiceEndpoint(c, b, "MyEndpoint");
// Set a certificate as the credential for the service.
sh.Credentials.ServiceCertificate.SetCertificate(
```
6.4.2 WS-Reliability

Web services and SOC applications are largely Web- and Internet-based. Not only security, but also reliability is a major concern.

WS-Reliability defined by OASIS [OASIS 2004], is a SOAP-based specification that fulfills reliable messaging requirements critical to SOC applications of Web Services. SOAP over HTTP is not sufficient when an application-level messaging protocol must also guarantee some level of reliability and security. This specification defines reliability in the context of current Web Services standards. This specification has been designed for use in combination with other complementary protocols and builds on previous experiences, such as ebXML Message Service (ebMS).

WS-Reliability defines WS-ReliableMessaging protocol with the following reliability features:

- Guaranteed message delivery, or At-Least-Once delivery semantics;
- Guaranteed message duplicate elimination, or At-Most-Once delivery semantics;
- Guaranteed message delivery and duplicate elimination, or Exactly-Once delivery semantics;
- Guaranteed message ordering for delivery within a group of messages.

WS-ReliableMessaging supports Reliable Messaging, which is the execution of a transport-agnostic, SOAP-based protocol providing quality of service in the reliable delivery of messages. Reliable Messaging has the following features:

- Reliable Sessions are suitable in the scenarios where both parties of communication are online. They deal with faults at the message level, including
  - Lost messages
  - Duplicated messages
  - Messages received out of order
- Message Queuing, for example, MSMQ, which deals with the situation that the receiver of the message may be offline at the time the message is sent. It ensures reliable communication between the sender and the receiver.
Figure 6.10 shows the steps of establishing and execution of the reliable messaging protocol in dealing the message loss.

The reliable sessions are implemented by message identify, sequence number, and message acknowledgement upon receiving. The following example given in [OASIS 2004] specifies the request that includes such information so that reliable sessions can be enforced.

```
<Request
xmlns="http://docs.oasis-open.org/wsrn/2004/06/ws-reliability-1.1.xsd"
xmlns:soap12="http://www.w3.org/2003/05/soap-envelope"
soap12:mustUnderstand="1">
  <MessageId groupId="mid://20040202.103832@wsr-sender.org">
    <SequenceNum number="0" groupExpiryTime="2005-02-02T03:00:33-31:00"/>
  </MessageId>
  <ExpiryTime>2004-09-07T03:01:03-03:50</ExpiryTime>
  <ReplyPattern>
    <Value>Response</Value>
  </ReplyPattern>
  <AckRequested/>
  <DuplicateElimination/>
  <MessageOrder/>
</Request>
```

WCF Reliable Sessions is an implementation of SOAP reliable messaging as defined by the WS-ReliableMessaging protocol.
WCF SOAP reliable messaging provides a reliable session between two endpoints, where the reliability provided is end-to-end, regardless of the number or type of intermediaries that separate the messaging endpoints. This end-to-end channel includes any transport intermediaries that do not use SOAP, such as HTTP proxies, or intermediaries that use SOAP that are required for messages to flow between the endpoints. A reliable session channel is implemented by “interactive” communication so that the services connected by such a channel run concurrently and exchange and process messages under conditions of low latency, that is, within relatively short intervals of time. This coupling means that these components make progress together or fail together, so there is no isolation provided between them.

The following configuration file shows the settings of using reliableSessionOverHttps in the communication between the client and service.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<configuration>
  <system.serviceModel>
    <client>
      <!-- this endpoint has an https: address -->
      <endpoint name="" address="https://localhost/servicemodelsamples/service.svc" binding="customBinding" bindingConfiguration="reliableSessionOverHttps" contract="Microsoft.ServiceModel.Samples.myService" />
    </client>
    <bindings>
      <customBinding>
        <binding name="reliableSessionOverHttps">
          <reliableSession />
          <httpsTransport />
        </binding>
      </customBinding>
    </bindings>
  </system.serviceModel>
</configuration>
```

### 6.4.3 Transactions

A transaction usually means a collection of actions, including information exchange and status modification. The collection of actions must be treated as a unit or atomic operation for the purposes of satisfying a request and for ensuring database integrity. For a transaction to be completed and database changes to be made permanent, a transaction must be completed in its entirety. A typical example of transaction is an online payment of electricity bill. You request will activate at least two actions: a withdrawal from your account and a deposit to the electricity company’s account. Obviously, it is not acceptable to have taken the money from your account, but the deposit does not go through, or vice versa.

In computer hardware design, we have a similar situation called indivisible operations, where we need to design a hardware instruction to perform two operations in a single step: check the flag, and lock the flag if the flag is not locked already. This indivisible instruction will allow the program to perform exclusive operation on a resource.
Transactions are at application level, which often consists of sequences of long operations. It is not possible to take a hardware solution to make sure the transactions complete in entirety or not at all. Thus, the software solution to perform transaction must meet the following requirements:

**Atomic:** The actions involved in a transaction must be all-or-nothing, that is, an action is either complete 100%, or not at all. Even if a failure occurs, such as hardware, software, and power failures, the atomic property must be guaranteed.

**Consistent:** The result of the transaction must be consistent and preserve the data integrity of all systems involved. For example, in case of an account transfer, if $100 is withdrawn from one account, the deposit into the other account must be $100 too.

**Isolated:** The principle of implementing atomic and consistent transaction is to define a set of prior-to-commit actions that are tentative and not visible to other activities. When all parties involved in the actions have successfully completed their parts, a coordinator (independent mechanism) will ask all parties to commit at the same time. At the end of the commitment, the coordinator will check the consistence of the results, and it will make the results visible to other activities only if the results are consistent. If any party cannot perform its action, or the results of the collective actions are not consistent, the coordinator will ask all parties to abort and to roll back to their previous status prior to making the tentative actions, which makes the tentative actions appear as if the actions never happened. The entire operations of transaction must be isolated before it is completed.

**Durable:** Once the transaction is marked completed by the coordinator, the results should have been placed into their final destination, such as database, and will be permanent. Any party involved cannot revoke its action that has been committed. Another transaction needs to be initiated if a change needs to be made.

WCF implements WS-AtomicTransaction (WS-AT), which is an interoperable transaction protocol developed by BEA, IBM, and Microsoft and supported by many other companies and organizations. It is currently an OASIS standard (http://docs.oasis-open.org/ws-tx/wsat/2006/06). WCF's implementation of WS-AtomicTransaction is based on Microsoft Distributed Transaction Coordinator (MSDTC) transaction manager. WS-AT enables distributed transactions to be flowed using Web service messages, and coordinates in an interoperable manner among heterogeneous transaction infrastructures. WS-AT uses the Two-Phase Commit protocol to drive an atomic outcome between distributed applications, transaction managers, and resource managers.

WCF supports multiple transaction standards, including WS-AT, OleTrasaction, and TransactionNego. When flowing a transaction between a client application and a server application, the transaction protocol used is determined by the binding that is exposed by the server on the endpoint selected by the client. Some WCF system-provided bindings default to specifying the OleTransactions protocol as the transaction propagation format, while others default to specifying WS-AT. The choice of transaction protocol inside a given binding can also be modified programmatically.

The following code segment shows the idea of performing transactions in C#. The code implements two database actions, imitating an account transfer transaction. The actions involved must be
quoted in “TransactionScope.” All the actions defined in the scope will be tentative, until the statement "transScope.Complete();" is executed before exiting the scope.

```csharp
using (TransactionScope transScope = new TransactionScope()) {
    // Create an connection channel
    using (SqlConnection connection1 = new SqlConnection(connectString1)) {
        // Opening connection1 to prepare withdrawal from one account
        // The connection will be listed in the TransactionScope
        connection1.Open();
        // modify database here.
        connection1.BeginTransaction(IsolationLevel.Serializable);
        // Database operation will be discussed in the next chapter.
        // Create another connection channel
        using (SqlConnection connection2 = new SqlConnection(connectString2)) {
            // Opening connection2 to prepare deposit into another account.
            // The connection will be listed in the TransactionScope
            connection2.Open();
            connection2.BeginTransaction(IsolationLevel.Serializable);
            // modify database here.
        }
    }
    // The actual method that commits the transaction.
    transScope.Complete();
}
```

The TransactionScope is the coordinator that performs tentative actions in isolation. Each time a database connection opens, the connection object is added into the TransactionScope. When all connection objects are added, the TransactionScope will perform all actions in isolation through transScope.Complete(); If the isolated execution is successful, the TransactionScope will make the transaction permanent. Otherwise, it will roll back the actions to make nothing has happened. More detailed example of transactions will be discussed in the next chapter in collection with database operations.

### 6.5 Discussions

Having discussed the development of the functionality of service-oriented software, this chapter presented the dependability issues of distributed service-oriented software. After a brief introduction to the basic concepts, we studied the security mechanism in IIS, ASP .Net, and Windows, and how these mechanisms can be integrated into the SOC software developed in ASP .Net. Structure of ASP .Net applications is studied to better understand how security is managed and a developer can implement and manage the security features of the applications.

Then, WCF is discussed, which is extended from ASP .Net. One of the main purchases of WCF is to strengthen the dependability capacity of ASP .Net. We discussed the security extension, the reliability consideration and implementation, and the transaction implemented in WCF.

This chapter may overlap with other computer science courses on computer security and computer reliability. We try to associate the topics to the context of service-oriented software development, and use code example to support the concepts presented.
This chapter also covered materials that are often covered in fault-tolerant computing and dependability theories. However, the focus here is on dependability design for service-oriented software, rather than the computing principles. Reliability evaluation of service-oriented software is still a new research area with few papers available today.

Another important problem in service-oriented software is the reliability and integrity of data. As services will be continuously evaluated, eventually services selected will be of high quality. In this case, the reliability of data will determine the reliability of service-oriented applications. Reliability of data will be determined by reliability of data source, reliability of any intermediate data routing and processing by services and networks. Data provenance in service-oriented software is also an important research topic as it addresses the history of data including the data source, intermediate processing, and final data processing at the destination [Tsai 2008].
6.6 Exercises and Projects

1. Multiple choice questions. Choose one answer in each question only, unless otherwise specified.

1.1 Reliability ensures
   (A) continuity of service in [0, t].
   (B) the readiness of service at time point t.
   (C) nonoccurrence of catastrophic consequence.
   (D) the validity and consistence of data and message.

1.2 Which of the followings belong(s) to security attributes? Select all that apply.
   [ ] Availability
   [ ] Confidentiality
   [ ] Safety
   [ ] Vulnerability

1.3 In what circumstance should such an encryption system be used, where the decoding key is public and the encoding key is private?
   (A) Reliability is needed.
   (B) Confidentiality is needed.
   (C) Digital signature is needed.
   (D) All of the above.

1.4 What security mechanism(s) does IIS support?
   (A) Access control list
   (B) IP address restrictions
   (C) Domain name restrictions
   (D) Encrypted HTTP connections
   (E) All of the above

1.5 What type of file overrides its occurrence in the parent directory?
   (A) ASAX file (Global)
   (B) ASCX file (User controls)
   (C) ASPX file (Web form)
   (D) Web.config
   (E) DLL file

1.6 What is (are) the problem(s) associated with the standard Windows Forms Security mechanism?
   (A) Passwords are stored in clear text
   (B) Sequential comparisons of user name and password
   (C) Unmanageable if accessibility needs to be changed frequently
   (D) All of the above
1.7 Web.config file in ASP.Net application structure is used for
(A) Authentication (B) Authorization
(C) Both (A) and (B) (D) None of (A) and (B)

1.8 Which of the following sequences does not make sense logically or semantically?
(A) <allow users="*"/>
(B) <deny users="?"/>
(C) <allow users="Bob"/> <deny users="*"/>
(D) <deny users="*"/> <allow users="Bob"/>

1.9 In order to tolerate (or correct) one bit error, the hamming distance \(d_m(C)\) of code C must be at least
(A) One (B) Two (C) Three (D) four

1.10 What error control codes are separable codes?
[ ] parity check [ ] checksum
[ ] Arithmetic Code [ ] m-of-n Code

1.11 DES (Data Encryption Standard) is used for low-level security purpose only, because its
(A) secret algorithm has been published. (B) encryption key is short.
(C) algorithm complexity is too high. (D) code is open source.

1.12 What is (are) the major dependability feature(s) added into the Windows Communications Foundation?
(A) WS-Security (B) Reliable Sessions (WS-R)
(C) Interoperability (WS-I) (D) All of the above

1.13 What reliability features does WS-RM specification define?
(A) At-Least-Once delivery, At-Most-Once delivery, and Exactly-Once delivery
(B) Guaranteed message ordering for delivery
(C) Both (A) and (B)
(D) None of the above

1.14 In Windows Communication Foundation, the scope of a transaction flow is
(A) in the entire program by default
(B) defined using an object of TransactionScope class
(C) quoted by a pair of special of tags <transaction> … </transaction>
(D) left to the user to write a rollback method that commits the transaction calls simultaneously

1.15 What security features are supported in WCF’s SSL protocol?
(A) Data confidentiality   (B) Data integrity
(C) Both (A) and (B)     (D) Neither (A) nor (B)

1.16 What reliability features does WS-ReliableMessaging specification define? Please check all that apply.
[ ] Lost messages
[ ] Duplicated messages
[ ] Messages received out of order
[ ] Guaranteed Secure Socket Layer data confidentiality

2. What is the dependability of a system?

3. What is quality? What are criteria for software quality?

4. What is quality of service and what are dependability attributes?

5. What attributes are called security attributes?

6. How are digital signature and confidentiality implemented in a public key security system?

7. What types of files are used in the ASP.Net Web application? What are their functions? How they are organized?

8. How is the Windows-based security implemented in a Web application?

9. How is the form-based security implemented in a Web application?

10. What dependability functions are implemented in WCF (Windows Communication Foundation)?

11. What is WS-* Specification? What components are included in WS-* Specification?

12. What is WS-Reliability? What issues the WS-Reliability specification address?

13. What is WS-transaction? What steps are involved in order to ensure the correctness of transactions?
14. Compare and contrast the indivisible instruction supported in all computer architectures and the atomic transactions supported in most distributed software development environments.

15. Discuss what security mechanisms are supported by WCF and at what layers they are implemented.

Project

In this project, you will develop a Web application with functionality and security considerations. You must use WCF services, instead of using ASP .Net services, as the building block of your application.

1. Draw the diagram that shows the overall system, its components (services), and the relationship among the components of the online shopping application.

2. Elaborate each component with sufficient detail so that a peer in this class can understand and can follow the explanation to find or implement (code) each of the components. You may use the workflow (flowchart) or pseudo code with comments to explain each component. You must give the type of the file, for example, aspx, ascx, html, text, config, asax, dll, and so on, that implements the component.

3. Develop at least two services in WCF and use the service in your application.

4. Give the part of code that implements the access control of the Web application. You must use the Forms-based security.