Detail Implementation of FT-SOAP

Deron Liang[1,2] , Chen-Liang Fang[3] , Chien-Cheng Lin[2]

[1] Institute of Information Science, Academia Sinica, Taipei, Taiwan, 11529, R.O.C.

[2] Department of Computer and Information Science, National Taiwan Ocean University, Keelung, Taiwan, R.O.C.

[3] Department of Information Management, Jin-Wen Institute of Technology, Taipei, Taiwan, R.O.C.

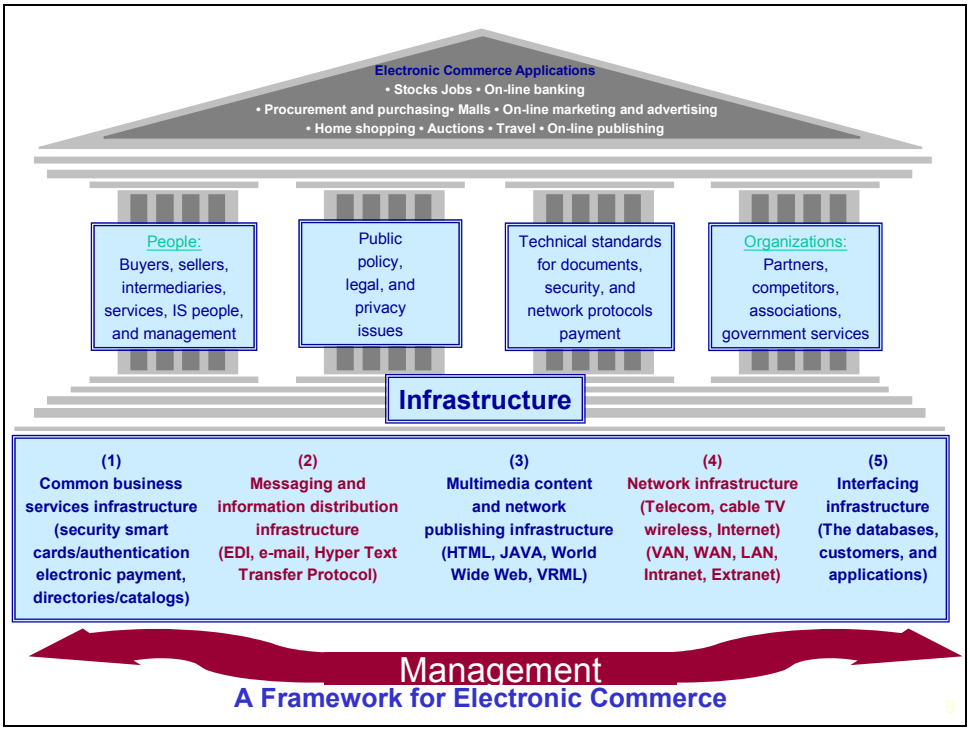
Abstract

Security is the major concern that refrains people from conducting commerce electronically. The security concerns related to electronic commerce (EC) includes transaction security and system security. We can partially address the transaction security issues by message distribution middleware such as simple object access protocol (SOAP), which is one of the four information technology (IT) infrastructures that facilitate EC. It requires a comprehensive set of IT in order to address the system security issues where fault-tolerance technology is one of the core technologies to enhance the system survivability after attack. Based on our preliminary investigation, we conclude that the current SOAP architecture is lack of mechanism to build a highly reliable EC system. We propose a comprehensive fault-tolerance framework based on the current SOAP architecture in order to address the system security issues for EC. We consider this research is the continuing effort of our previous work on fault-tolerant CORBA, thought there are similarity and difference between these two technologies. We plan to propose a standard recommendation that outlines a set of interfaces named fault-tolerant SOAP. We also expect to implement two prototypes, one based on Windows 200 and the other is based on Linux, in order to prove the concepts.

1. Introduction

電子商務(Electric Commerce)近年來以驚人的聲勢顛覆了人們從事商業行為(Business Conducts)的方式。根據 Forrester Research Inc.等公司的研究報告，企業間電子商務(B2B EC)佔全體企業商業交易之比率將從 1997 之 0.2%快速成長至 2003 之 9.5%產值高達 1 兆 3 千億美金($1.33.Billion)[Free98]，[RC98]。而台灣是以製造業出口導向的經濟，近年來由以高科技製造業為主力，根據同樣的調查報告顯示，高科技交易依賴電子商務之比率遠高於一般傳統產業，2003 預計 39.3%之產值來自於 EC。

根據 Zwass 之研究顯示[Zwas96]，電子商務之建置主要依賴 5 組底層基礎技術(Infrastructure)，如圖表 1 所示，他們分別為(1)Common Business Service Infrastructure，(2)Messaging and Information Distribution Infrastructure，(3)Multimedia Content and Network Publishing Infrastructure，(4)Network Infrastructure，and (5)Interface Infrastructure。



圖表 1 電子商務之建置五大基礎技術

其中第(2)項即是為一般資訊業界所稱之分散式中介軟體(Distributed Middleware)，近年來具代表性的中介軟體及技術包含了 OMG 的 CORBA[OMG98]，Microsoft 之 DCOM[MSFT98]，IBM 之 MQ[IBM01]以及 SOAP[W3C00] 等等。在 B2B 之電子商務中，又以 SOAP 技術最被看好，我們在下文會詳細探討其被看好的原因。EC 之推廣目前也面臨了許多技術性或非技術性的瓶頸。根據 Internet Week 最近的網路民調(online survey)顯示，阻礙人們上網消費的原因中，排名第一的是消費者擔心網路的交易安全(transaction security)[CHI01]。因之應用在電子商務的 middleware 技術選擇需考慮分散環境的

異質性以及安全性。在安全性的考量上，我們也許會考慮虛擬私有網路(VPN)的技術，另外在異質性的考量上，也許我們會考慮 CORBA 標準中介軟體。雖然 VPN+CORBA 是絕佳的搭配，但有幾個理由使得我們不得不另外以其他的技術來實現安全的電子商務環境。

a、 VPN 的軟體尚未普及：目前的商業環境 VPN 的使用並未達普遍使用的程度。

b、 VPN 的標準及功能未完備及統一：目前的 VPN 產品相容程度是一個問題，不同產品間的相容性尚待加強。目前大都只用在同一企業內，因此不適合在企業間應用。

c、 法令問題：有些國家的法令並不允許秘密通信，因此應用 VPN 的技術在部分國家是不可行的。

d、 防火牆阻隔中介軟體間的服務：現在的網路環境已經有很多企業非常重視資訊安全而架設防火牆以阻隔外界惡意的入侵行為。很不幸的，有些防火牆的設定也使得 CORBA 的通訊管道變的不是很通暢。相對的 SOAP 標準容易配合防火牆的使用：SOAP 的標準直接透過常用的 HTTP 或 SMTP 協定直接作信息傳送。因此非常適合在有防火牆的環境。

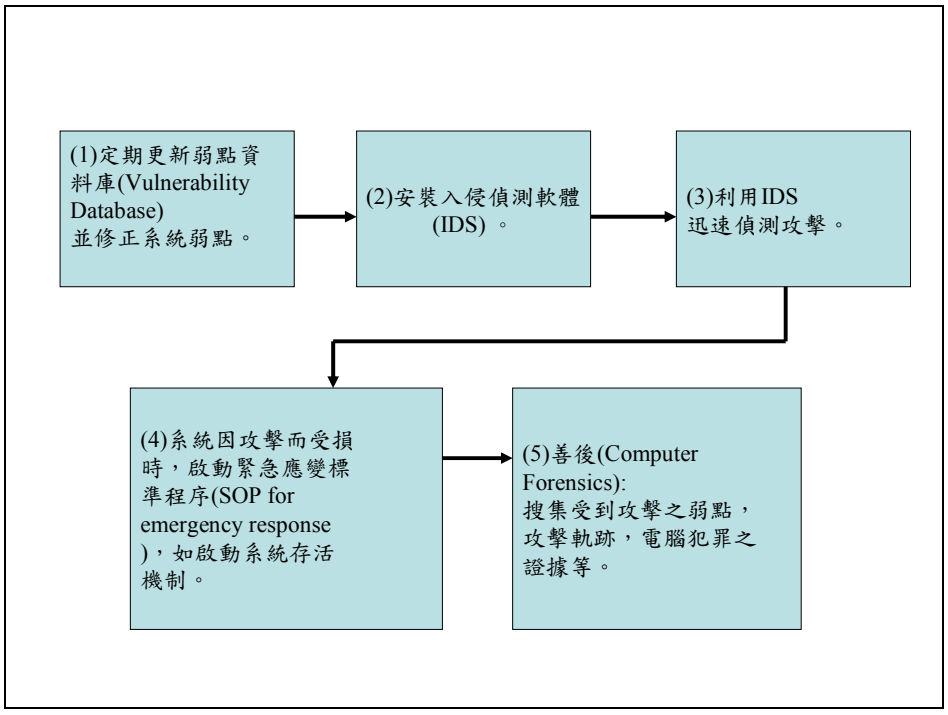
e、 因為 SOAP 依附於 HTTP，因此，它可應用一些 HTTP 的特色，如 Proxy，SSL 等， 以達到交易安全的要求。

f、 SOAP 利用 XML 標準語法作為資訊交換之標準，便於 B2B 交易伙伴間的供應鏈管理(supply-chain management)。

g、 SOAP 具備高度的延展性。

因此我們根據以上幾點原因判斷 SOAP 是最符合未來 EC 交易平台的 middleware。

此外根據 Internet Week 同樣一份的網路民調顯示，阻礙人們上網消費的原因中，排名第二的是 EC 交易平台的系統安全 (system security)。資訊安全專家相信網路上經常出現的網站受攻擊(internet attack)等負面事件是主要原因，資訊專家也同意要建構一套永遠不受駭客攻擊受傷的網站是近乎不可能達到的理想。一般相信維護 EC security 最有效的策略(policy)包含了 5 個步驟(phases)，如圖表 2 所示。



圖表 2 有關於 EC 網站系統安全管理機制的 5 個步驟

其中第(4)步驟主要就是要靠軟體容錯技術(Software Fault-Tolerance)而增加 EC 系統的存活能力[CHI01][NMN00]。

容錯技術 (fault tolerance) 在電腦世界中成為日漸重要的考量因素。從使用者的觀點來看，容錯的定義為應用程式的可使用性(availability)和資料的一致性(consistency)。應用程式的容錯技術牽涉到錯誤偵測(failure detection)，蒐集導致錯誤的資料，進而復原錯誤(failure recovery)。傳統的容錯技術大多由硬體層次上著手，雖然這些硬體的容錯技術對於大型及昂貴的系統處理相當有效率，但是仍存有兩大缺點：(1) 昂貴的成本，(2) 無法處理應用程式層次上的錯誤。較低成本的市場如個人電腦及工作站，要加入一套容錯硬體系統所增加的成本實在太高。而且一般用途的中介軟體如 CORBA 或 SOAP，並沒有真正提供完整的容錯服務。此外，大部份的錯誤發生在軟體應用程式上，這些錯誤可能導致應用程式毀損(crash)及待滯(hang)。所以要提升這些應用程式的效率及回復性，發展一套適用於 SOAP 架構的「軟體容錯」技術(software fault-tolerance technology)是項重要的課題。

為了以後討論方便，我們簡單介紹 SOAP 的架構與使用方法。我們首先介紹(1).SOAP Architecture，然後再介紹(2).主從架構(Client-Server) SOAP 程式設計，(3).SOAP 的訊息結構。

在(1).SOAP Architecture 部分，我們可以以圖表 3 為例，SOAP 在整個運作的大略分為 9 個步驟。

a、 程式產生一個需求(Request)動作。

b、 這個動作產生一個處理程序和需求介面。

c、 訊息被轉成 XML 格式且被送往 Web Server。

d、 XML 解析器檢查 XML 文件之一慣性(consistency)且將之經由 HTTP 送往接收端。

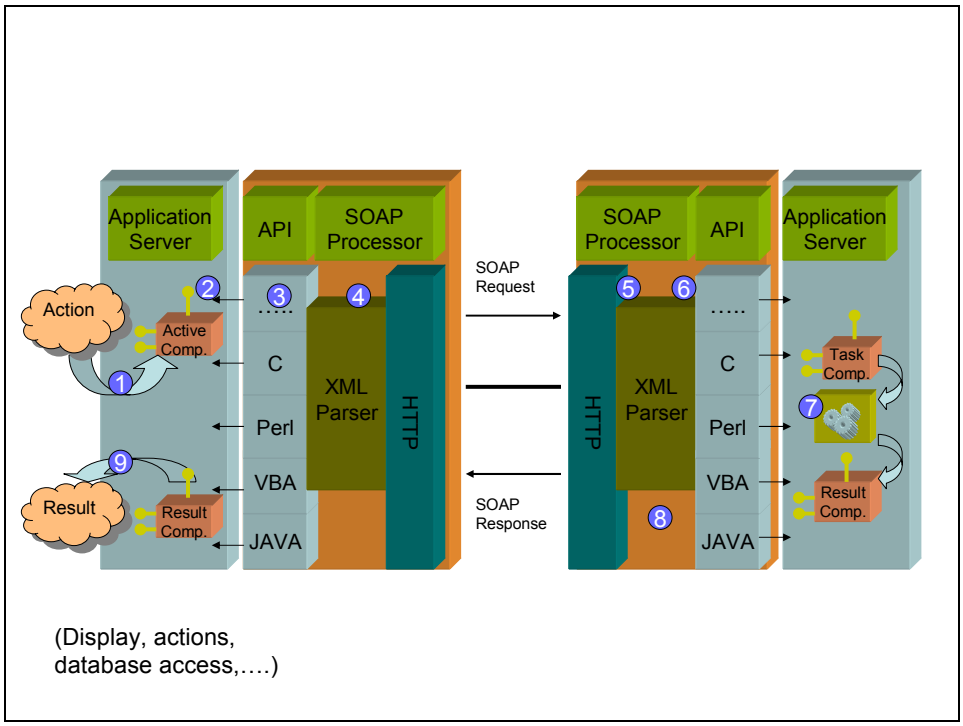
e、 接收端的 XML 解析器利用 HTTP 和 XML 的標頭資訊(TAG)檢查所接受到訊息的合法性。

f、 訊息轉送到適當之應用程式且將 XML 訊息反組譯成為一般 Code。

g、 應用程式根據訊息客戶之需求內容(Request)執行工作。

h、 訊息以原先需求端發送訊息之相同模式經由 HTTP 將訊息回傳。

i、 原始需求動作接收到回傳之結果，完成 Request。



圖表 3 SOAP Architecture

(2).接著我們介紹如何在符合 SOAP 之規範來實作 Client 以及 Server 端之應用程式。

Application Server 除了在 Server 端撰寫 Service 之外，還需要 Service 以 WSDL 的格式 Publish 到 Internet，將此 WSDL 的 URI 註冊到 UDDI(Universal Description, Discovery and Integration)上。WSDL、UDDI、XML Schema 分別是 W3C 提出之標準，詳細的 Specification 請見[W3C01a][UDDI01][W3C01b]。

為了介紹 SOAP 上的主從架構(Client-Server)程式如何撰寫，我們用一個簡單的範例。此範例提供了一個加法的功能，輸入兩個整數參數，以字串方式回傳兩數相加後之值。首先我們針對此服務提出一個 WSDL 的服務描述檔如圖表 4 所示，針對於該服務提供之介面、參數以及提供服務之 URI 利用 WSDL 詳加描述於該檔案之中。

```
<?xml version='1.0' encoding='UTF-8' ?>
 <!-- Generated 08/08/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.02.813.0 -->
<definitions    name ='StockTrasnsationSamplev1'     targetNamespace = 'http://tempuri.org/wsdl/'
   xmlns:wsdlns='http://tempuri.org/wsdl/'
   xmlns:typens='http://tempuri.org/type'
   xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
   xmlns:xsd='http://www.w3.org/2001/XMLSchema'
   xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
   xmlns='http://schemas.xmlsoap.org/wsdl/'>
   <types>
      <schema targetNamespace='http://tempuri.org/type'
         xmlns='http://www.w3.org/2001/XMLSchema'
         xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
         xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
         elementFormDefault='qualified'>
      </schema>
   </types>
   <message name='SampleServer.add'>
      <part name='n1' type='xsd:int'/>
      <part name='n2' type='xsd:int'/>
   </message>
   <message name='SampleServer.addResponse'>
      <part name='Result' type='xsd:String'/>
   </message>
   <portType name='SampleServerSoapPort'>
      <operation name='add' parameterOrder='n1 n2'>
         <input message='wsdlns:SampleServer.add' />
         <output message='wsdlns:SampleServer.addResponse' />
      </operation>
   </portType>
   <binding name='SampleServerSoapBinding' type='wsdlns:SampleServerSoapPort' >
      <stk:binding preferredEncoding='UTF-8'/>
      <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
      <operation name='add' >
         <soap:operation soapAction='http://tempuri.org/action/SampleServer.add' />
         <input>
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
             encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
```

```
            </input>
            <output>
               <soap:body use='encoded' namespace='http://tempuri.org/message/'
                encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
            </output>
         </operation>
      </binding>
      <service name='SampleServer' >
         <port name='SampleServerSoapPort' binding='wsdlns:SampleServerSoapBinding' >
            <soap:address location='http://localhost:8080/soap/servlet/rpcrouter' />
         </port>
      </service>
   </definitions>
```

圖表 4WSDL of SampleServer

緊接著便是 Client 與 Server 的程式設計部份，我們用 Java 的程式語言作為開發範例，用以下列四個階段來簡單說明：

首先，我們先開發 Server 端的 Service Component 如圖表 5 所示。

接著便是在 SOAP Server 端的開發端程式，如圖表 6 所示。

```
public class sampleServer
{
  public String add(int n1,int n2)
  {
  return "n1="+n1+",n2="+n2+",n1+n2="+(n1+n2);
  }
}
```

圖表 5 Server Component

```
import org.apache.soap.rpc.*;
import org.apache.soap.*;
import java.util.*;
import java.net.*;
import java.lang.*;
class SoapCaller
{
  String urlString = "http://140.121.197.102/soap/servlet/rpcrouter";
  String encodingStyleURI = Constants.NS_URI_SOAP_ENC;
```

```
String targetObjectURI="urn:sample";
public Object execCall (String methodName,Vector para)
{
        Call cobj = initCall(methodName,para);
        Response r = invokeCall(cobj);
        return (Object)((r.getReturnValue()).getValue());
}
private Response invokeCall(Call callobj)
{
        URL u = null;
        Response r = null;
        try
        {
                u = new URL(urlString);
                r = callobj.invoke(u,"");
        }
        catch(java.net.MalformedURLException e)
        {
                System.out.println("malformedurlexception");
        }
        catch(SOAPException e)
        {
                System.out.println("here comes the soapexception.");
        }
if(r.generatedFault())
{
        this.handleFault(r);
}
return r;
}
private Call initCall (String methodName,Vector para)
{
Call call = new Call();
call.setTargetObjectURI(targetObjectURI);
call.setMethodName(methodName);
call.setParams(para);
call.setEncodingStyleURI(encodingStyleURI);
return call;
```

```
}
private void handleFault(Response r)
{
  Fault fault = r.getFault();
  System.err.println("generated fault");
  System.err.println("fault code="+fault.getFaultCode());
  System.err.println("fault String= "+fault.getFaultString());
}
}
```

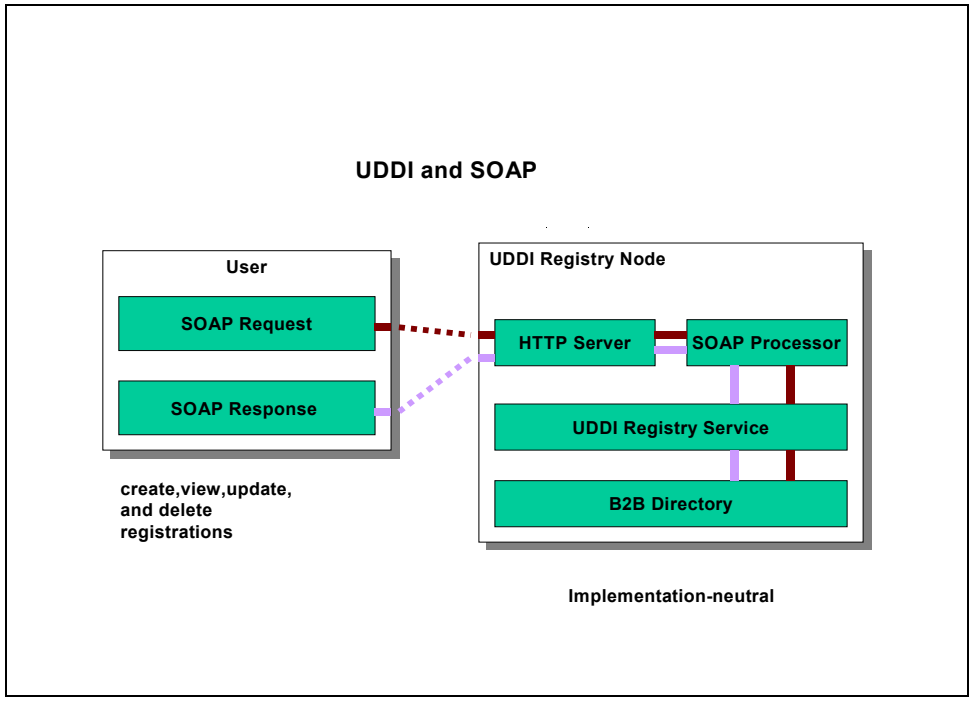
圖表 6 SOAP RPC Routing

在完成 WSDL 與 Server Side Development 階段後，便要將所 Implement 之 Web Service 註冊到網際網路 UDDI Server 上。 UDDI (Universal Description, Discovery and Integration)建立一個跨平台，開放式的架構級標準，UDDI 定義了 Web 服務的發布與發現的方法。UDDI 提供了一種基於分布式的商業註冊中心的方法，該商業註冊中心維護了一個企業和企業提供的 Web 服務的全球目錄,而且其中的信息描述格式是基於通用的 XML 格式的。 登錄資料包含兩大類：

a、 Businesses register public information about themselves:
   i. White Pages.
   ii. Yellow Pages.
   iii. Green Pages.
b、 Standards bodies, Programmers, Businesses register information about their Service Type:
   i. Service Type Registration.

譬如要將以上資料存入 UDDI 或取出可以利用 *Publisher::save_service(authOnfo,businessService)* …. 進行註冊，再利用 *Inquery::get_serviceDetail(serviceKey)* …. 取得註冊資料。

其他用以管理 UDDI 之使用請參考[MSFT01][UDDI01]。



圖表 7 UDDI 與 SOAP 的圖示.

在完成 Server 端的開發與建制以及將服務 Publish 到 InterNet 之後，便是 Client 端的程式開發如圖表 8 所示， Client 輸入兩個整數送至 Server 端，Server 端接收到 Request 後將此兩個整數做加法，然後將結果回傳回 Client 端顯示在螢幕上。

```
import org.apache.soap.rpc.*;
import org.apache.soap.*;
import java.util.*;
public class sampleClient
{
  public static void main(String arg[])
  {       System.out.println("call soap server....sampleServer.add(100,200)");
          SoapCaller sc =    new SoapCaller();
          Vector para = new Vector();
          para.add(new Parameter("n1",Integer.class,new Integer(arg[0]),null));
          para.add(new Parameter("n2",Integer.class,new Integer(arg[1]),null));
          String result=(String)sc.execCall("add",para);
          System.out.println("Result:"+result);
  }
}
```

圖表 8 SOAP Client 程式之範例

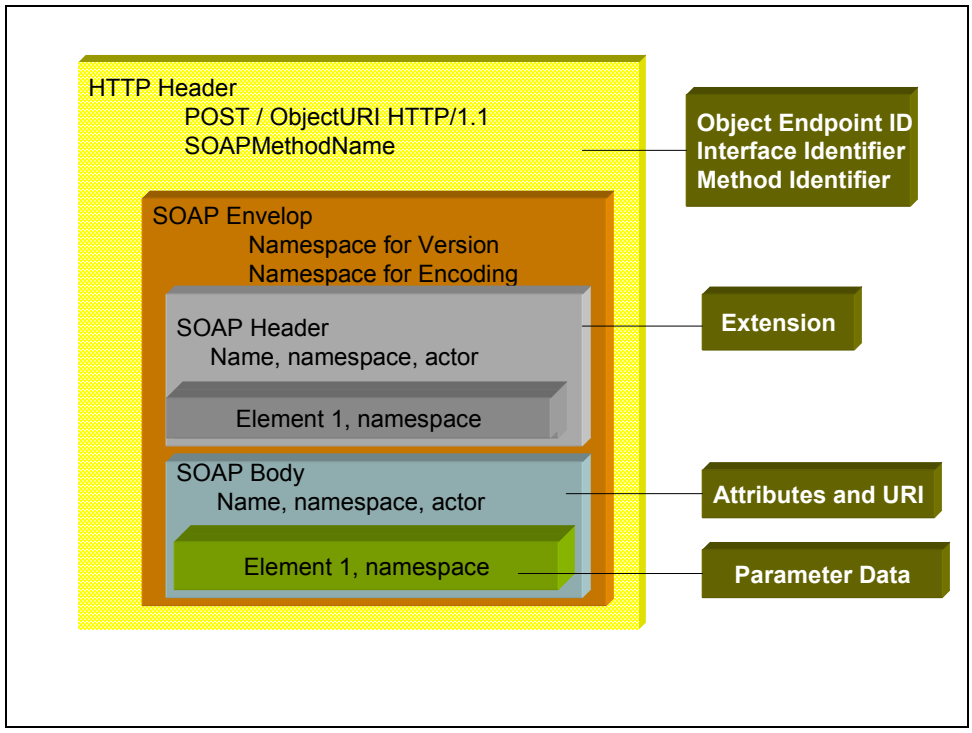
(3). 最後我們介紹 SOAP 的訊息結構, 也就是 SOAP 標準下的 “run time support”。

SOAP 的訊息結構如圖表 9 所示，SOAP 的 Message 主要區分為三個部分：

a、 SOAP Envelop.

b、 SOAP Encoding Rule.

c、 SOAP RPC Representation.



圖表 9 SOAP 的訊息結構

由於近年來 SOAP 在電子商務之應用中日益普遍，根據我們初步的研究，阻礙電子商務蓬勃發展的最主要之原因即為資訊安全相關議題。然而，現今 SOAP 架構中對資訊安全，特別是容錯問題並非完整之解決方案，因之，本研究計畫中，我們針對 SOAP 現有之架構擬提出一可行的 SOAP 軟體容錯建議標準 (Standard Recommendation)。因此本計畫的目的有二：

一. 定義一套符合 SOAP 架構之「軟體容錯物件服務」標準介面，簡稱 FT-SOAP:

在以下的 proposal 中， 我們將擬提之容錯「物件服務」稱為 Fault-Tolerance SOAP (或簡稱 FT-SOAP)。在 component technology 的觀念裡， 物件可利用 component interface 提供外界服務。相反的， 如果一組 interface 成為標準(standard specification)， 則任何宣稱支援此標準的軟體都應提供相同的服務。Standard specification 之好處，在此不再贅述[Bro95]。我們擬用 SOAP 支援的標準界面語言 WSDL 來描述 FT-SOAP 所提供的服務。

容錯系統的目的在於達到物件的持續可供應性。根據我們過去「軟體容錯物

件服務」的研究心得 [LFY99] 以及 Fault Tolerant CORBA[OMG98] 的經驗，我們認為 FT-SOAP 應當要能提供應用物件以下之服務:

a、 應制定出一套界面讓使用者能夠建構被動式群組(passive entity group)。

b、 應制定出一套界面讓使用者能夠建構主動式群組(active entity group)。

c、 應制定出一套界面，使外界必須能由這些界面來判斷群組內那些成員正處於正常運作狀態或者失誤狀態。

d、 應制定出一套界面讓使用者能夠做群組新增、設定、及刪除。

e、 應制定出一套界面不論在正常情況或在發生失誤及修護的狀況下都能維護這些 entity group 狀態的同步(state synchronization)。

f、 提供某種界面讓使用者能控制這些 entity 的復原情形。一旦應用程式發生錯誤，FT-SOAP 則應負責啟動 entity recovery 的工作，例如重新啟動應用程式並使其回復到發生錯誤前的狀態。

g、 制定界面讓群組之複製(entity redundancy)能 transparent to 客戶端，這包括讓客戶端沒有察覺群組內個別 entity 之失誤或修護情形。

h、 提出完整的解釋來說明各種 faults 和 resources 的定義及 entity 和 resource 之間如何結合。此外，FT-SOAP 亦需制定出一些界面來蒐集這些 failures 的統計資料並加以報告說明。

i、 提供錯誤偵測(failure detection) 的界面。

j、 提供錯誤照會(failure notification)的界面讓容錯管理者能夠執行所需的容錯管理。

k、 提供一組界面來避免客戶端獲得來自群組重複或多餘的回應(redundant responses)，或者讓使用者自行決定是否願意收到多重回應，或者是任其選擇所需的回應。

l、 提供一組界面來避免群組對其他伺服物件/群組發出重複多餘的服務須求。

我們將根據以上的需求，定義一套符合 SOAP 架構之「軟體容錯物件服務」標準介面 (standard interfaces)，以供 FT-SOAP object servers 以及所有應用物件(application objects)實作上有所依循。

二. 研發出一套支援 FT-SOAP 標準的原型服務系統 (prototype): 應用程式物件可以針對本身所需的錯誤偵測和復原方式向 FT-SOAP 要求服務，或者應用程式物件亦可扮演管理者的角色，替其他物件向 FT-SOAP 要求服務。藉由 FT-SOAP 所提供的容錯服務，應用程式中的物件將可提高其可靠性而不必另行發展其專屬的容錯軟體。

2. System Design

我們先利用 Use Case 之研究方法對 FT-SOAP 做系統分析，找出系統需求後，提出完整的 FT-SOAP 系統架構。

2.1 Use Cases Analysis

我們首先討論 8 個 Use Case，他們分別代表了：

Use Case 1:

(Infrastructure Control Scenario)應用程式(AP)想利用 FT-SOAP 內建之群組管理機制來建立一個容錯群組(Replication Group)，並告知 FT-SOAP 此群組如何組成，發生錯誤時，應通知哪些 Serves 以及後續之復原處理。

Use Case 2:

(Application Control Scenario)Use case 2 與 Use case 1 是提供相同功能的分析，只不過是以 Application Control 的方式來做分析。

Use Case 3:

(Fault management: Infrastructure control)當錯誤發生時，提供系統偵測、發佈與更新容錯系統內資訊的功能，然後由 RM 更新 UDDI 相關資訊。

Use Case 4:

(Fault management: Application control)與 Use Case 3 相似，提供相同之功能，不過在最後的 UDDI 更新資訊，是交由 application 處理。

Use Case 5:

(Centralized Logs)將 Log 紀錄集中紀錄在可靠的檔案系統，如此 FT-SOAP 便不需要 WSG 的資訊。

Use Case 6:

(Distributed Logs)將 Log 分散式的紀錄在每個 FT-SOAP 的伺服器中，再定期更新其內容。

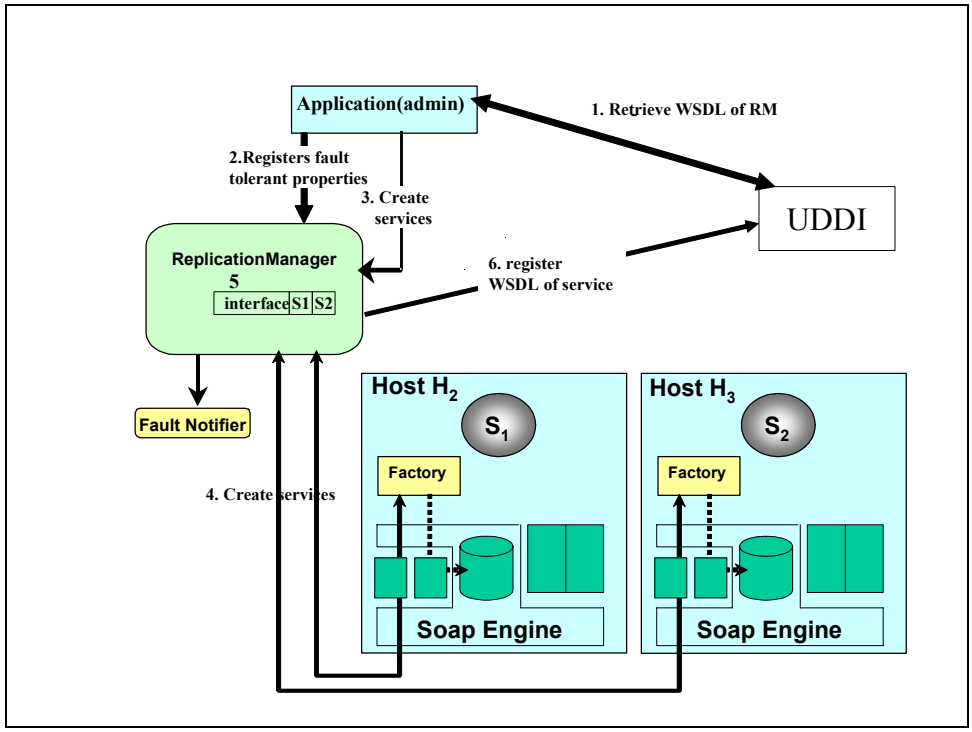
Use Case 7:

(Recovery)當主伺服器發生當機或其他不可預期之狀況無法繼續服務時，RM 選擇群組中的一個備援系統(Replica)，提升為主伺服器； 新的主伺服器會從紀錄檔(log)中取出資料復原系統。

Use Case 8:

(Fault Transparency)說明 FT-SOAP 如何使 Client 端絲毫感覺不到 Server 端發生錯誤，而繼續提供服務給 Client。

Case 1 Infrastructure Control Scenario:



圖表 10 Replication management: Infrastructure Control

Step 1:使用者應用程式(application)取得 RM 的 WSDL。

Step 2:向 RM 註冊相關的 Replication properties, 如 replication style, fault monitoring style。

Step 3:使用者應用程式向 RM 要求 create service。

Step 4: RM 根據應用程式註冊的 properties 對每個群組成員上的 factory 要求產生成員物件,即 factory 向 SOAP Engine 註冊 Service 相關資訊 factory 動作完成後回應該成員的 WSDL 的位置資訊 (URL) 給 RM。

Step 5:對每一個群組成員：

a. RM 將成員加到群組。

b. RM 依 replication Style 啟動成員。

c. RM 依據 replication style , fault monitoring style 決定是否啟始錯誤監測。

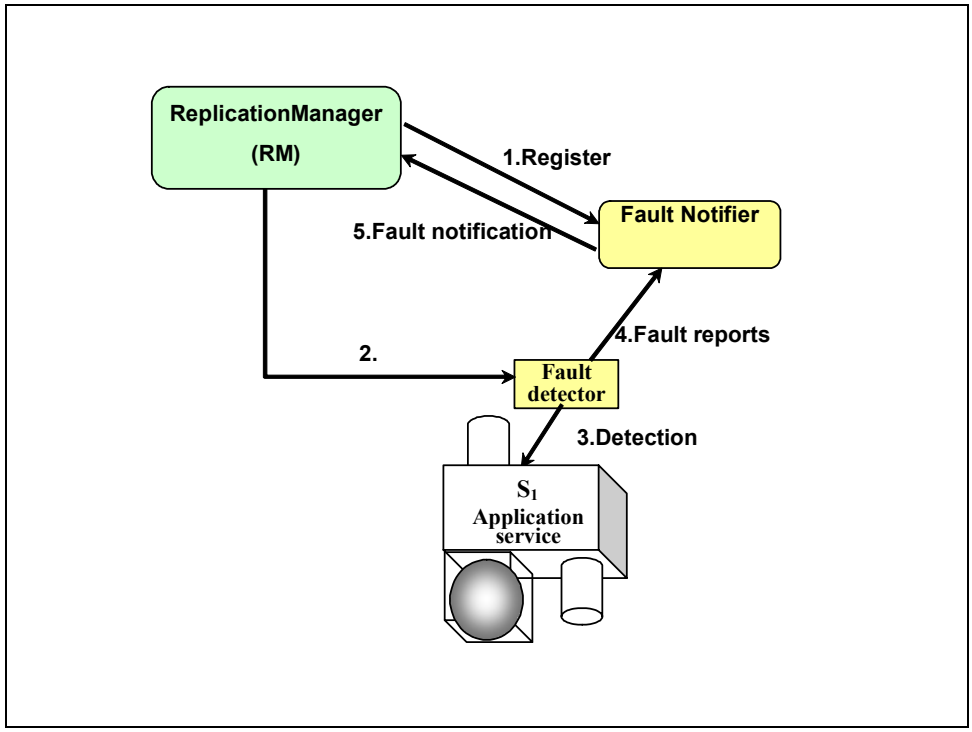
d. RM 向 fault notifier 註冊，以接收該成員的 fault notification。

e. (對 passive replication style)，RM 決定該群組的 primary member。

f. RM 建立 Group Service WSDL，並將設定為 primary 的成員 activate 起來，包括把記錄在 FT SOAP 的 state 改為 primary 以及由 FT SOAP 記錄 Group Service WSDL 的 URL。

Step 6: 由 RM 向 UDDI 註冊該 Service。

Fault Detection and Fault Notification



圖表 11 偵錯

Step 1: RM 取得 fault notifier 的 WSDL 後，RM 向 fault notifier 註冊(method of notification service)，以使 fault notifier 能在 service 發生錯誤時，向 RM 通報。

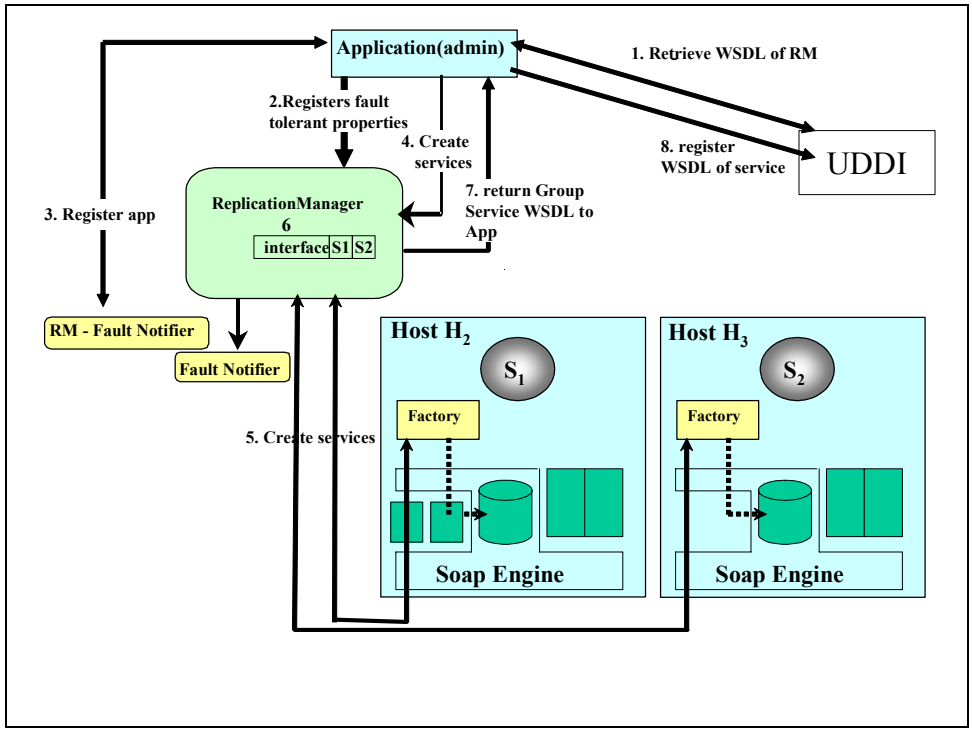
Step 2: RM 取得 fault detector 的 WSDL 後，RM 通知 fault detector 開始偵測運行中受保護的 service。

Step 3: fault detector 取得 service 的 WSDL 後，開始週期性地呼叫受測 service，以確認其存在，在此機制下，受測 service 須實作特定介面，以便 fault detector 能呼叫受測 service 上的特定偵測功能。

Step 4: fault detector 取得 fault notifier 的 WSDL 後，當受偵測的 service 發生錯誤時，fault detector 察覺後便向 fault notifier 報告錯誤發生 (notification service)。

Step 5: fault notifier 取得 RM 的 WSDL 後，fault notifier 接到 fault detector 報告，隨即向已註冊的 RM 通報。

Case 2 Application Control Scenario:



圖表 12 Replication management: Application Control

Step 1: 使用者應用程式(application)藉由 UDDI 取得 RM 的 WSDL。

Step 2: 使用者應用程式向 RM 註冊相關的 Replication properties，如 replication style, fault monitoring style 等。

Step 3: application 向 RM 要 RM-fault notifier 的 WSDL 後，對 RM-fault notifier 進行註冊成為 consumer 。

Step 4: 使用者應用程式向 RM 要求 create service。

Step 5: RM 根據應用程式註冊的 properties 對每個群組成員上的 factory 要求產生 Service，即 factory 向 SOAP Engine 註冊 Service 相關資訊, factory 動作完成後，回應該 Service 的 WSDL 的位置資訊 (URL) 給 RM。

Step 6: 對每一個群組成員：

a. RM 將 Service 加到群組。

b. RM 依 replication Style 啟動成員。

c. RM 依據 replication style , fault monitoring style 決定是否啟始錯誤監測。

d. RM 向 fault notifier 註冊，以接收該成員的 fault notification。

e. (對 passive replication style)，RM 決定該群組的 primary member。

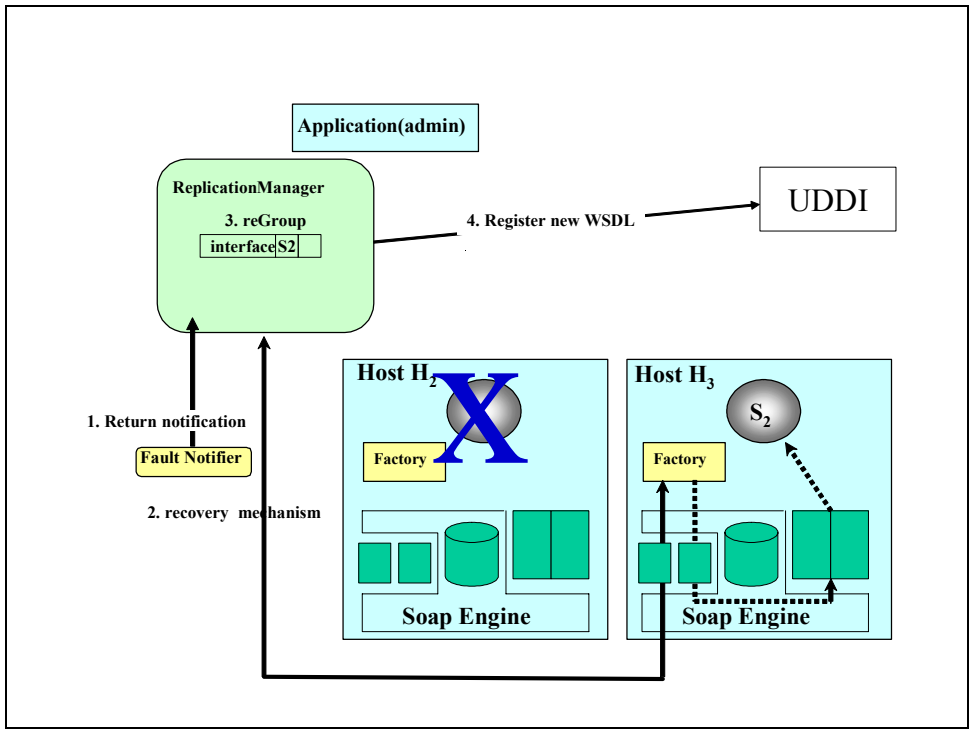
f. RM 建立 Group Service WSDL，並將設定為 primary 的成員 activate 起來，包括把記錄在 FT SOAP 的 state 改為 primary、由 SOAP Engine 記錄 Group Service WSDL 的 URL，以及由 recovery 機制去復原狀

態，但此時無 state 存在，所以不影響任何狀態。

Step 7: RM 返回 Group Service WSDL 的 URL 給應用程式。

Step 8: 由 application 向 UDDI 註冊該 Service。

Case 3 Fault management: Infrastructure control



圖表 13 Fault management: Infrastructure Control

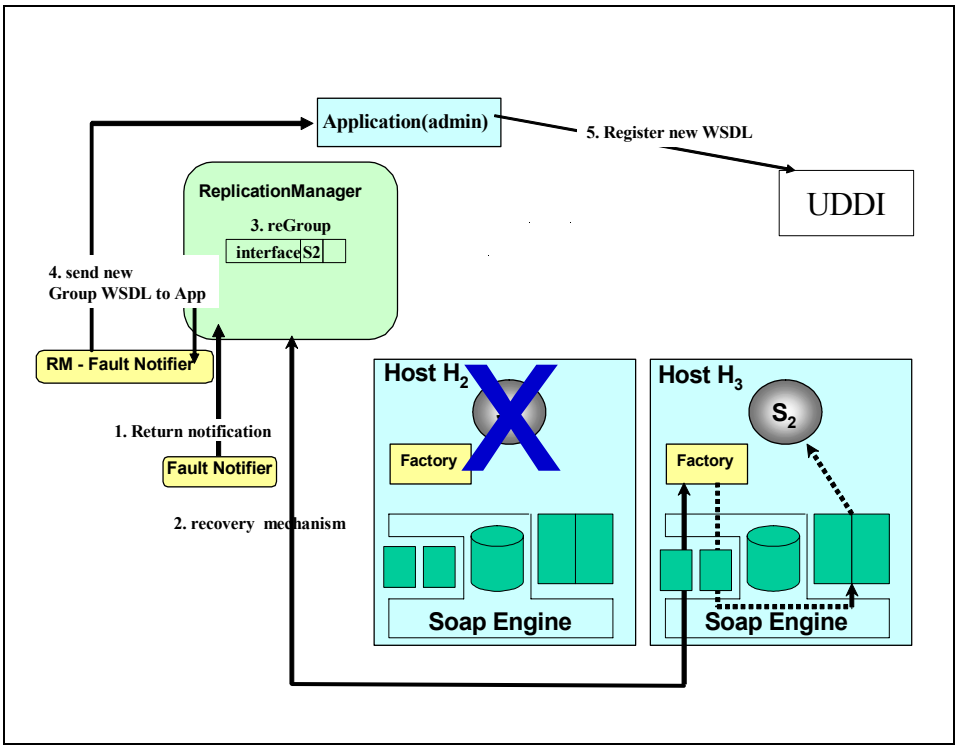
Step 1: 當錯誤發生時，fault notifier 接到 fault detector 錯誤報告，隨即向已註冊的 RM 發出錯誤訊息。

Step 2: RM 執行 recovery 機制。

Step 3: RM 產生新的 Group Service WSDL。

Step 4: RM 向 UDDI 註冊新的 Group Service WSDL。

Case 4 Fault management: application control



圖表 14 Fault management: Application control

Step1: 當錯誤發生時，fault notifier 接到 fault detector 錯誤報告，隨即向已註冊的 RM 發出錯誤訊息。

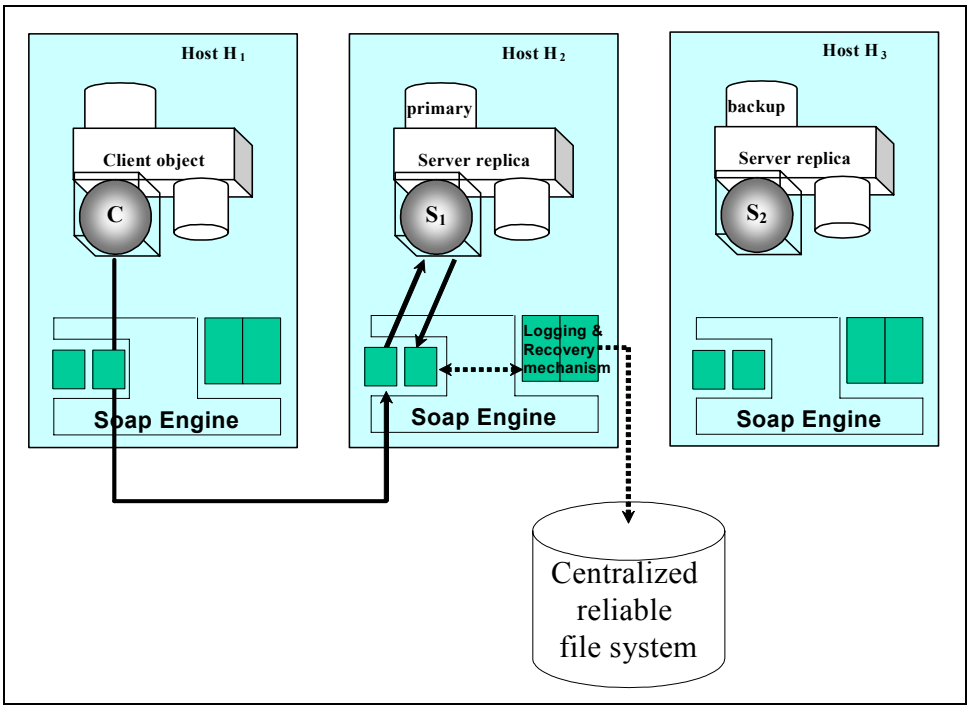
Step 2: RM 執行 recovery 機制。

Step 3: RM 產生新的 Group Service WSDL。

Step 4: 由 RM 將新的 Group Service WSDL 送交給 RM-fault notifer，再由 RM-fault notifer 交給最初執行註冊的 application。

Step 5: application 向 UDDI 註冊新的 Group Service WSDL。

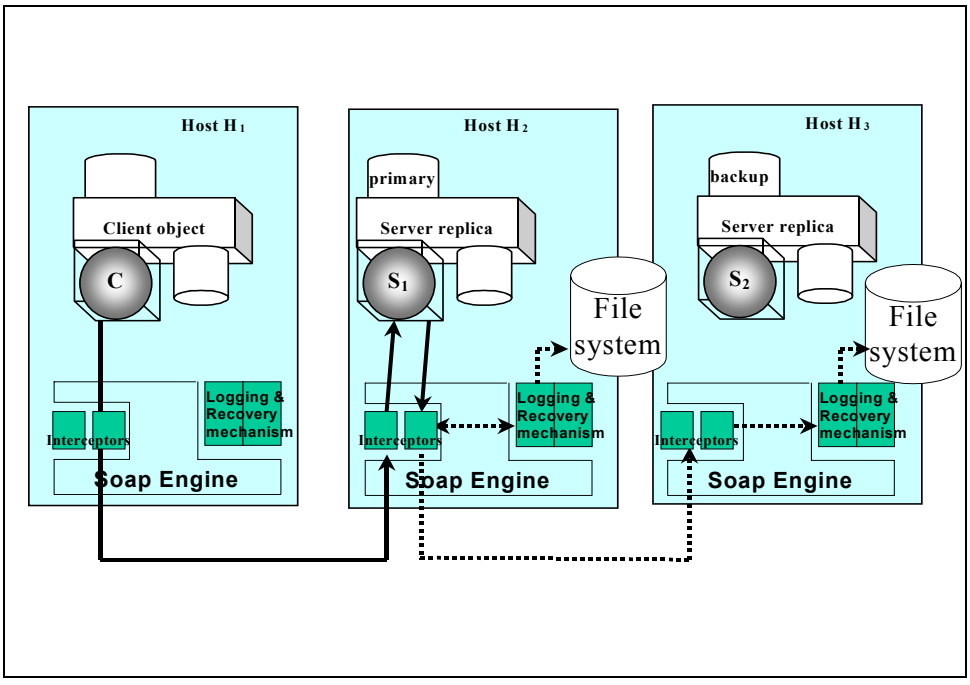
Case 5 Centralized Logs:



圖表 15 Logging: Centralized

將 log 記到 centralized reliable file system, 若採用此方法, 則 FT SOAP 不需要 WSG 的資訊。

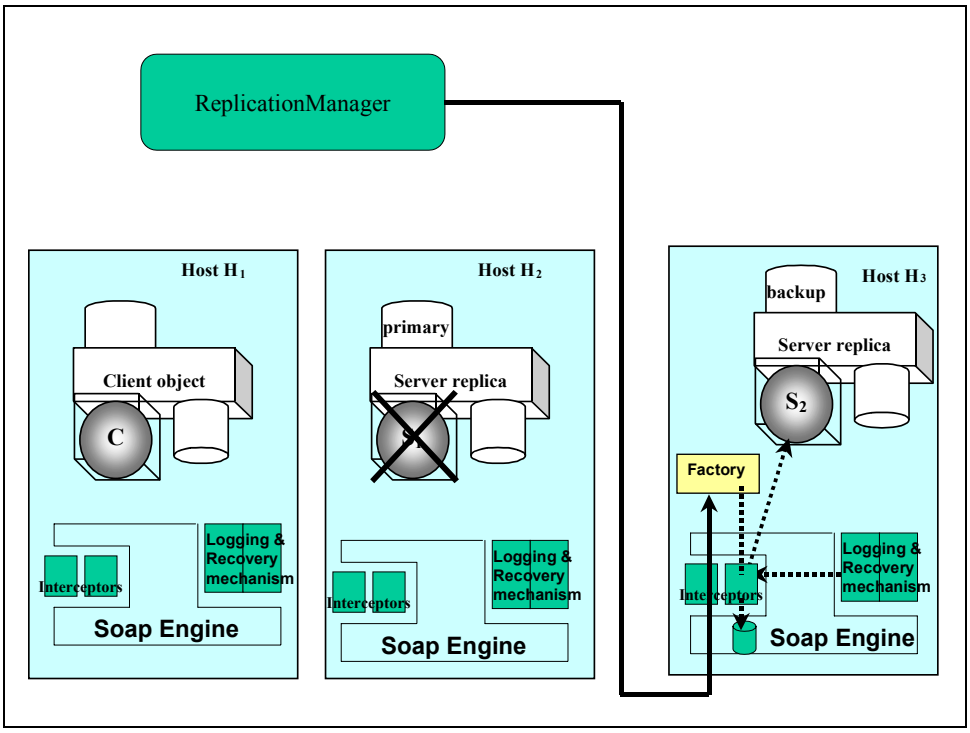
Case 6 Distributed Logs:



圖表 16 Logging: distributed

FT SOAP 必須有正確的 WSG 的資訊 (RM 在向 factory 做 activate 時, 會將 WSDL (WSG)的 URI 當做參數傳給 factory, factory 再傳給 FT SOAP 做記錄)，由 primary 藉由 checkpoint interval 定期記錄 replica 的狀態。

Case 7 Recovery:

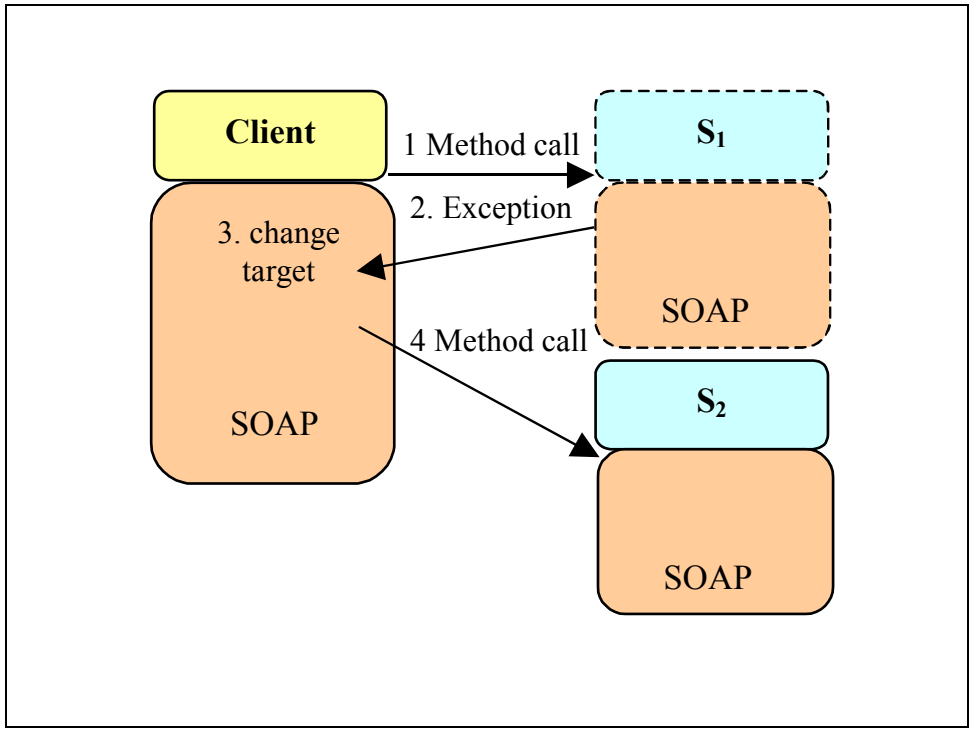


圖表 17 復原機制

RM 選定 replica service 作為 primary service 後，對該 replica service 的 factory 下達 *activate_service()*。接著 Factory 啟動 Recovery Mechanism，從記錄檔取出資料復原 service 的狀況，並將成為 primary 的狀況更新至 soap engine 中，以接收 client 的資訊。

若群組成員個數小於設定值時，RM 會通知其他可用主機新增 servic 加入群組。

Case 8 Fault transparency to client:



圖表 18 復原機制

Client :

Step 1: send request to primary service.

Step 2: primary 錯誤發生，傳回 exception， 由 client 的 FT-SOAP 接收處理。

Step 3: Client 端的 FT-SOAP 依群組的參照資料(WSG)，取得 replica 的 location。

Step 4: 依序對 replica 重新 request，直到回傳正確的執行結果。

Step 5: 若所有 replica 都 failure (皆傳回 exception)， 則使用者端程式必須做例外處理(取得 WSG 的 URI 重新下載一次新的 WSG)，根據新的 WSG 再重覆步驟一。

Server :

Interceptor 根據 FT SOAP 的所保有的資訊決定該 request 要不要處理， 也就是說若 replica 不是 primary， 則 FT SOAP 會傳回錯誤訊息(message handling)給 Client 端。

2.2 FT-SOAP 系統須求分析及架構

根據以上 8 個 Use Cases 的分析，我們的到以下幾點心得:

FT-SOAP 應當提供以下 1~12 項服務。

1. Passive Group。
2. Active Group。
3. Query Replicas。

4. Group Management。
5. Fault Classification。
6. Fault Detection。
7. Fault Notification。
8. State Synchronization。
9. Recovery。
10. Failure Transparency。
11. Suppression of Redundant Responses。
12. Suppression of Redundant Invocations。

這些功能，大致可以分成 4 組主要功能類別(Classes)

a、 Replication Management.
b、 Fault Management.
c、 Logging and Recovery Mechanism.
d、 Fault-tolerance Infrastructure.

我們將 FT-SOAP 的主要功能歸納分類於。

項次	須求	功能分類
1	Passive Group	***Replication Management***
2	Active Group	
3	Query Replicas	
4	Group management	
5	Fault Classification	Fault Management
6	Fault Detection	
7	Fault Notification	
8	State synchronization	Logging & Recovery Mechanism
9	Recovery	
10	Failure Transparency	Fault Tolerance Infrastructure
11	Suppression of Redundant Responses	
12	Suppression of Redundant Invocations	

表格 1 FT-SOAP 須求分析以及功能分類。

同時根據我們初步的系統分析，FT-SOAP 的系統架構應該至少具備有以下

幾個重要的功能組件(components)，用以支持以上的所有功能:

a、 Replication Manager (RM).

b、 Fault Manager，包含了 fault detector，fault-notifier… 等等；

c、 Logging & Recovery mechanism.

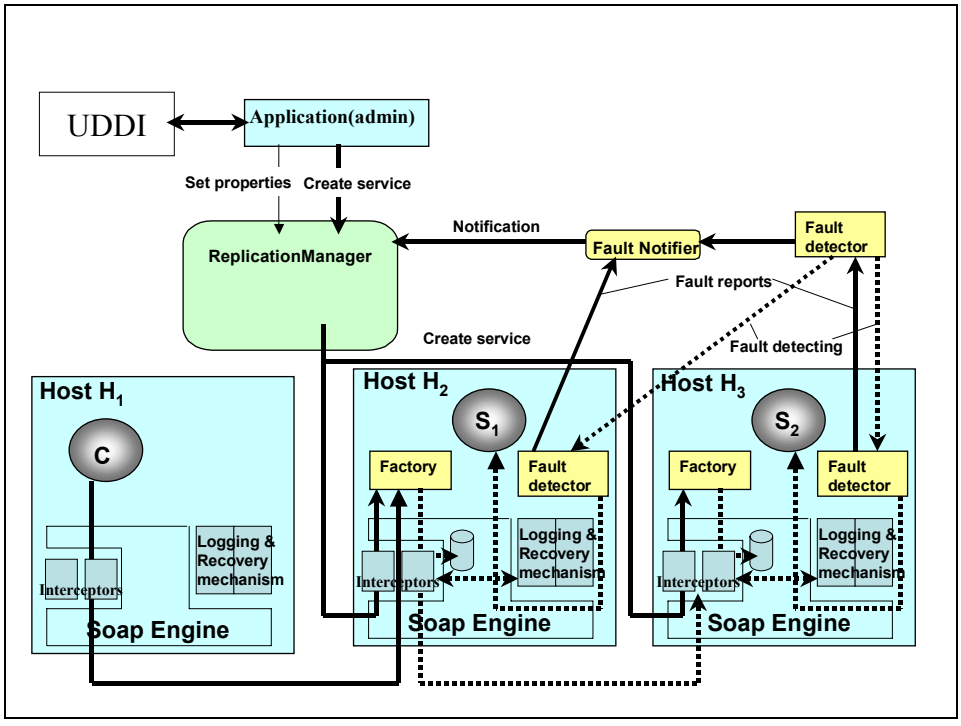
以上新增的功能元件，在現今的 SOAP 架構下都不存在，需另行設計，這牽涉到了新的 interface (WSDL) 以及語意定義(Schema)等重要定義。除了以上新增的元件之外， 根據我們初步的探討，在 SOAP 之架構(architecture) 至少增加以下三項功能，堪能支援 FT-SOAP 之基本需求：

a、 擴充 WSDL 之語意，期能包含群組服務之描述(Group Service Description)，我們因之 propose 一個新概念：WSG (web service group)。

b、 SOAP Engine 必須增加 Interceptor[CORBA95]之功能。

c、 SOAP Engine 內部用以描述 SOAP object 的資訊必須增加。

根據以上系統分析(system analysis)的結果，我們大致可以設計出 FT-SOAP 的大略的系統架構如下圖表 19。



圖表 19 FT-SOAP 容錯系統架構圖

首先、主要的工作為 WSG 之設計以及 RM interface 的制定。我們首先介紹 WSG。 根據我們初步的分析，建議在 WSDL 內新增一個 Tag，<WSG/>，用以描述群組服務(service group)中，每一個 replica 扮演角色，如 “primary” or

“secondary”等。Tag <WSG/> 之定義如圖表 20 以及圖表 21。

```
<WSG>
<PRIMARY version=”xx.xx” location=”host : port” service_id=””/>
<REPLICA version=”xx.xx” location=”host : port” service_id=”” />
<REPLICA version=”xx.xx” location=”host : port” service_id=”” />
</WSG>
```

圖表 20 The extension to the WSDL: the service group tag <WSG/>.

```
<?xml version=”1.0”?>
<xsd:schema xmlns:xsd=”http://www.w3.org/2000/10/XMLSchema”
  targetNamespace=”http://www.ft-soap.org”
  xmlns=”http://www.ft-soap.org”
  elementFormDefault=”qualified”>
<xsd:element name=”WSG”>
  <xsd:complexType>
    <xsd:sequence>
          <xsd:element ref=”PRIMARY” minOccurs=”0” maxOccurs=”1”/>
          <xsd:element ref=”REPLICA” minOccurs=”0” maxOccurs=unbounded”/>
    </xsd:sequence>
  </xsd:complexType>
    </xsd:element>

<xsd:element name=”PRIMARY”>
  <xsd:attribute name=”version” type=”xsd:string”/>
  <xsd:attribute name=”location” type=”xsd:string”/>
  <xsd:attribute name=”service_id” type=”xsd:string”/>
    </xsd:element>

<xsd:element name=”REPLICA”>
  <xsd:attribute name=”version” type=”xsd:string”/>
  <xsd:attribute name=”location” type=”xsd:string”/>
  <xsd:attribute name=”service_id” type=”xsd:string”/>
    </xsd:element>
</xsd:schema>
```

圖表 21 The WSG Schema definition.

第二項重要的工作是RM interface的制定以及分析用以建構描述容錯群組所需之屬性(attributes)。如 use case 1 所述，群組管理提供了以下的功能：1.create passive group，2,set property，3 membership 管理，4 詢問 group member，和 5. 註冊 fault notifier 。根據我們初步的分析， 這些功能可大致分成以下 4 組 Interfaces：*PropertyManager*(), *GenericFactory*(){}, *ObjectGroupManager*(){}, and *ReplicationManager*(){}. 他們的大致內容分別列於圖表 24~圖表 27 值得注意的是圖表 22 是用標準的 WSDL 語言所撰寫，其功能等同於圖表 24。**錯誤! 找不到參照來源。**(用 Java IDL 所撰寫).為得簡潔起見，在以下的章節中我們用 Java IDL 來取代 WSDL。

Interface PropertyManager WSDL Descriptions file:

```
<?xml version='1.0' encoding='UTF-8' ?>
<!-- Generated 08/08/01 by Microsoft SOAP Toolkit WSDL File Generator, Version 1.02.813.0 -->
<definitions    name ='StockTrasnsationSamplev1'    targetNamespace = 'http://tempuri.org/wsdl/'
        xmlns:wsdlns='http://tempuri.org/wsdl/'
        xmlns:typens='http://tempuri.org/type'
        xmlns:soap='http://schemas.xmlsoap.org/wsdl/soap/'
  xmlns:xsd='http://www.w3.org/2001/XMLSchema'
        xmlns:stk='http://schemas.microsoft.com/soap-toolkit/wsdl-extension'
        xmlns='http://schemas.xmlsoap.org/wsdl/'>
<types>
            <schema targetNamespace='http://tempuri.org/type'
            xmlns='http://www.w3.org/2001/XMLSchema'
                xmlns:SOAP-ENC='http://schemas.xmlsoap.org/soap/encoding/'
            xmlns:wsdl='http://schemas.xmlsoap.org/wsdl/'
            elementFormDefault='qualified'>

    <xsd:element name="Properties">
            <xsd:complexType >
                    <xsd:sequence>
                    <xsd:element ref="Property" minOccurs="1" maxOccurs="unbounded" />
                    </xsd:sequence>
                </xsd:complexType >
    </xsd:element >
    <xsd:complexType name="Property">
            <xsd:sequence >
                    <xsd:element name="name" type="xsd:string" />
                    <xsd:element name="value" type="xsd:anyType" />
```

```
                </xsd:sequence >
  </xsd:complexType >
      <xsd:element name="TypeId">
                <xsd:complexType >
                        <xsd:sequence>
                        <xsd:element ref=" TypeId " minOccurs="1" maxOccurs="unbounded" />
                        </xsd:sequence>
                    </xsd:complexType >
      </xsd:element >
      <xsd:complexType name=" TypeId ">
                <xsd:sequence >
                        <xsd:element name="name" type="xsd:string" />
                        <xsd:element name="value" type="xsd:anyType" />
                </xsd:sequence >
  </xsd:complexType >
                </schema>
          </types>

<message name='PropertyManager.set_default_properties'>
          <part name='props'    type='Properties' />
</message>
<message name='PropertyManager.get_default_properties' > </message>
<message name='PropertyManager.get_default_propertiesResponse' >
        <part name='Result'    type='Properties' />
</message>
<message name='PropertyManager.remove_default_properties'>
  <part name='props'    type='Properties' />
</message>
<message name='PropertyManager.set_type_properties'>
  <part name='type_id'    type='TypeId' />
  <part name='overrides'    type='Properties' />
</message>
<message name='PropertyManager.get_type_properties'>
  <part name='type_id'    type='' />
</message>
<message name='PropertyManager.get_type_propertiesResponse'>
  <part name='Result'    type='Properties' />
</message>
```

```
<message name=’PropertyManager.remove_type_properties’>
  <part name=’type_id’    type=’TypeId’ />
  <part name=’props’    type=’Properties’ />
</message>
<portType name=' PropertyManagerSoapPort'>
      <operation name=' set_default_properties ' parameterOrder='props'>
         <input message='wsdlns:PropertyManager.set_default_properties' />
         <output message='wsdlns:PropertyManager.set_default_propertiesResponse' />
      </operation>
      <operation name=' get_default_properties' parameterOrder=''>
         <input message='wsdlns: PropertyManager.get_default_properties' />
         <output message='wsdlns: PropertyManager.get_default_propertiesResponse' />
      </operation>
      <operation name=' remove_default_properties' parameterOrder='props'>
         <input message='wsdlns: PropertyManager.remove_default_properties' />
         <output message='wsdlns: PropertyManager.remove_default_propertiesResponse' />
      </operation>
      <operation name=' set_type_properties ' parameterOrder=’’type_id overrides '>
         <input message='wsdlns: PropertyManager. set_type_properties ' />
         <output message='wsdlns: PropertyManager. set_type_propertiesResponse' />
      </operation>
      <operation name=' get_type_properties ' parameterOrder='type_id'>
         <input message='wsdlns: PropertyManager.get_type_properties ' />
         <output message='wsdlns: PropertyManager.get_type_propertiesResponse' />
      </operation>
      <operation name=' remove_type_properties ' parameterOrder='type_id props '>
         <input message='wsdlns: PropertyManager.remove_type_properties ' />
         <output message='wsdlns: PropertyManager.remove_type_propertiesResponse' />
      </operation>
</portType>
<binding name=' PropertyManagerSoapBinding' type='wsdlns: PropertyManagerSoapPort' >
      <stk:binding preferredEncoding='UTF-8'/>
      <soap:binding style='rpc' transport='http://schemas.xmlsoap.org/soap/http' />
      <operation name='set_default_properties' >
         <soap:operation soapAction='http://tempuri.org/action/ PropertyManager.set_default_properties' />
         <input>
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
             encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
```

```
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='get_default_properties' >
    <soap:operation soapAction='http://tempuri.org/action/ PropertyManager.get_default_properties' />
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='remove_default_properties' >
  <soap:operation soapAction='http://tempuri.org/action/ PropertyManager.remove_default_properties ' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name='set_type_properties' >
    <soap:operation soapAction='http://tempuri.org/action/’PropertyManager.set_type_properties ' />
    <input>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </input>
    <output>
      <soap:body use='encoded' namespace='http://tempuri.org/message/'
       encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
    </output>
  </operation>
  <operation name=' get_type_properties ' >
    <soap:operation soapAction='http://tempuri.org/action/PropertyManager.remove_type_properties' />
    <input>
```

```
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
          </input>
          <output>
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
          </output>
        </operation>
        <operation name=' remove_type_properties ' >
          <soap:operation soapAction='http://tempuri.org/action/PropertyManager.remove_type_properties' />
          <input>
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
          </input>
          <output>
            <soap:body use='encoded' namespace='http://tempuri.org/message/'
              encodingStyle='http://schemas.xmlsoap.org/soap/encoding/' />
          </output>
        </operation>
      </binding>

      <service name=' PropertyManager' >
        <port name=' PropertyManagerSoapPort' binding='wsdlns: PropertyManager SoapBinding' >
          <soap:address location='http://localhost:8080/soap/servlet/rpcrouter' />
        </port>
      </service>
    </definitions>
```

圖表 22 Interface PropertyManager WSDL

```
Properties Schema:
<xsd:element name="Properties">
  <xsd:complexType >
        <xsd:sequence>
              <xsd:element ref="Property" minOccurs="1" maxOccurs="unbounded" />
        </xsd:sequence>
  </xsd:complexType>
</xsd:element>
<xsd:complexType name="Property">
```

```
  <xsd:sequence>
        <xsd:element name="name" type="xsd:string" />
        <xsd:element name="value" type="xsd:anyType" />
  </xsd:sequence>
</xsd:complexType>
```

圖表 23 Properties Schema

```
interface PropertyManager {
void set_default_properties(in Properties props);
Properties get_default_properties();
void remove_default_properties(in Properties props);
void set_type_peroperties(in TypeID type_id, in Properties overrides);
Properties get_type_properties(in TypeId type_id);
void remove_type_properties(in TypeId type_id, in Properties props)
};
```

圖表 24 interface PropertyManager

```
interface GenericFactory {
  URL create_service(in TypeID type_id, in Criteria the_criteria, out FactoryCreationId);
        Boolean activate_service(in FactoryCreationId factory_creation_id);
        Boolean passive_service(…);
        Boolean destroy_service(…);
}
```

圖表 25 interface GenericFactory

```
interface ObjectGroupManager {
}
```

圖表 26 interface ObjectGroupManager

```
interface Replication Manager : PropertyManager, ObjectGroupManager, GenericFactory {
  URL get_rm_fault_notifier();
  URL get_fault_notifier();
}
```

圖表 27 interface Replication Manager

Fault tolerant property

在 PropertyManager interface 之中，需要應用程式(application)設定各式的屬性, attributes, 我們在此稱為 properties，每一群組都必須有他們自己的特性設定。我們建議有以下數種 properties: ReplicationStyle, MembershipStyle, ConsistencyStyle, FaultMonitoringStyle, InitialNumberReplicas, MinimumNumberReplicas, FaultMonitoringIntervalandTimeOut, and CheckpointInterval 等八種。 現在就 PropertyManager 介面會用到的 Properties，我們一一說明如下：

ReplicationStyle

Name	ftsoap.ft.ReplicationStyle
Value	FT::STATELESS FT::COLD_PASSIVE FT::WARM_PASSIVE FT::ACTIVE FT::ACTIVE_WITH_VOTING

FT::STATELESS 通常用於 read-only 的伺服系統。

FT::COLD_PASSIVE 群組主有一 primary 成員提供服務，錯誤發生後才以其 log 來進行復原。

FT::WARM_PASSIVE 群組主有一 primary 成員提供服務，其所有行動都會送到備援(backup)紀錄，錯誤發生後才以備援的 log 來進行復原。

FT::ACTIVE 群組所有成員都同時執行服務動作，以獲的最佳復原效果。

FT::ACTIVE_WITH_VOTING 同 FT::ACTIVE，但其回應結果必須經由投票來節定。

MembershipStyle

Name	ftsoap.ft.MembershipStyle
Value	FT::MEMB_APP_CTRL FT::MEMB_INF_CTRL

FT::MEMB_APP_CTRL 群組成員增刪都由使用者本身來指定。

FT::MEMB_INF_CTRL 群組管理由系統來完成。

ConsistencyStyle

Name	ftsoap.ft.ConsistencyStyle
Value	FT::CONS_APP_CTRL

	FT::CONS_INF_CTRL

FT::CONS_APP_CTRL 使用者自行進行 checkpinting,logging 的動作

FT::CONS_INF_CTRL 進行 checkpinting,logging 的動作.

FaultMonitoringStyle

Name	ftsoap.ft.FaultMonitoringStyle
Value	FT::PULL FT::PUSH FT::NOT_MONITORED

FT::PULL 系統偵錯器會定期呼叫受偵測的物件以確定它是活的。

FT::PUSH 受偵測的物件定期呼叫系統偵錯器會以確定它是活的。

FT::NOT_MONITORED 不做偵錯動作。

Factories

Name	ftsoap.ft.Factories
Value	FactoryInfos

FactoryInfos 整組所有相關 Factory 的資訊。

InitialNumberReplicas

Name	ftsoap.ft.InitialNumberReplicas
Value	An unsigned short

群組起始時的啟始 replica 數目。

MinimumNumberReplicas

Name	ftsoap.ft.MinimumNumberReplicas
Value	An unsigned short

群組最少 replica 數目。

FaultMonitoringIntervalAndTimeout

Name	ftsoap.ft.FaultMonitoringIntervalAndTimeout
Value	TimeBase::TimeT

TimeBase::TimeT 偵錯系統間格時間。

CheckpointInterval

Name	ftsoap.ft.CheckpointInterval
Value	TimeBase::TimeT

TimeBase::TimeT 系統作 checkpoint 間隔時間。

圖表 28 描述有關於群組各項 Properties 的 XML Schema 定義。

```
<xsd:element name="Properties">
        <xsd:complexType >
        <xsd:sequence>
                    <xsd:element ref="Property" minOccurs="1" maxOccurs="unbounded" />
              </xsd:sequence>
        </xsd:complexType>
</xsd:element>

<xsd:complexType name="Property">
        <xsd:sequence>
              <xsd:element name="name" type="xsd:string" />
              <xsd:element name="value" type="xsd:anyType" />
        </xsd:sequence>
</xsd:complexType>
```

圖表 28 (a) Fault Tolerance Properties 之 XML Schema 定義

```
<xsd:simpleType name=”ReplicationStyleValue”>
  <xsd:restriction base=”xsd:integer”>
        <xsd:minInclusive value=”0” />
        <xsd:maxInclusive value=”4” />
  </xsd:restriction>
</xsd:simpleType>
```

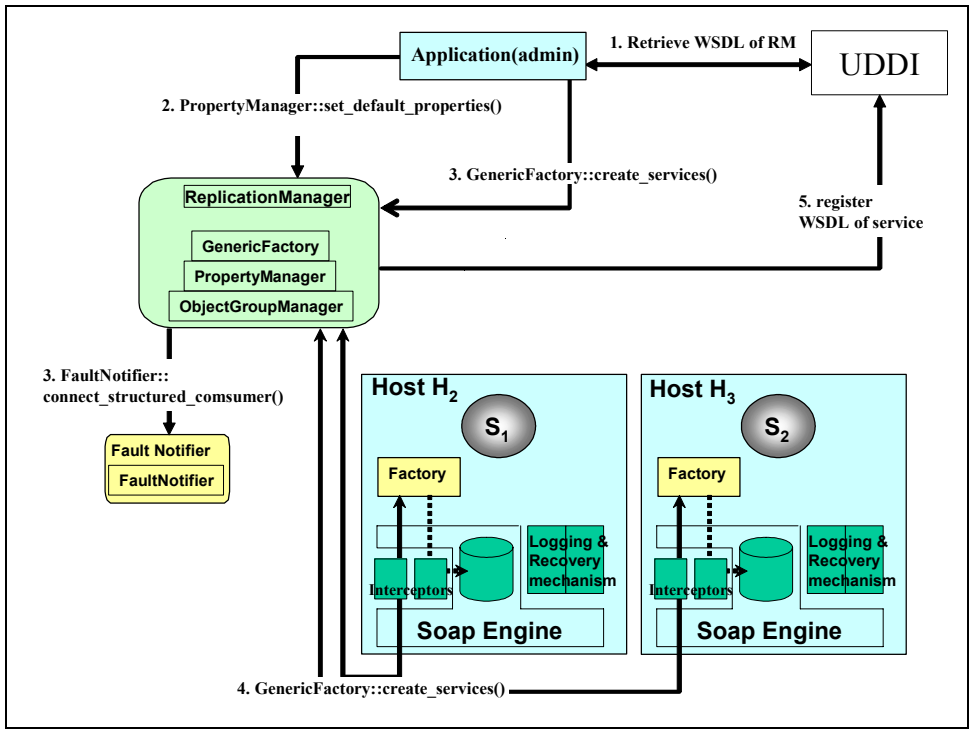
圖表 28(b) Fault Tolerance Properties 有關值(Value)之 XML Schema 定義

```
<xsd:simpleType name=”ReplicationStyleValue”>
  <xsd:restriction base=”xsd:integer”>
        <xsd:enumeration value=”0” />
        <xsd:enumeration value=”1” />

  </xsd:restriction>
</xsd:simpleType>
```

圖表 28(c) Fault Tolerance Properties 有關值(Value)之 XML Schema 定義，列舉方式

我們利用 Use Case 1 的例子來說明 replication management 相關的 interfaces 以及 properties 等之用法如圖表 29。



圖表 29 Replication management

Step 1: 首先由 Application Admin 由 UDDI 取回 Replication Manager 的 WSDL。

Step 2: 使用者應用程式欲組成 replication group 前，先以 RM 中的 PropertyManager 介面提供的容錯特性設定功能設定使用者期望的容錯功能。利用這介面可設定 ReplicationStyle，MembershipStyle，InitialNumberReplicas，MinimumNumberReplicas 等等，我們將在 Fault tolerant property 小節表列說明。當然這裡須借用 CORBA 已經有的 Property 物件定義了。在此，*PropertyManager::set_default_properties()* 用來設定系統內定的特性、*PropertyManager::get_default_properties()* 可查詢目前系統的內定設定、*PropertyManager::remove_default_properties()*；另外此介面還提供強制性設定以隨時取代(override)目前系統的內定設定，當然都包含設定、清除、及查詢 property 的功能：*PropertyManager::set_type_properties()*、*PropertyManager ::remove_type_properties()*、和 *PropertyManager::get_type_properties()*。

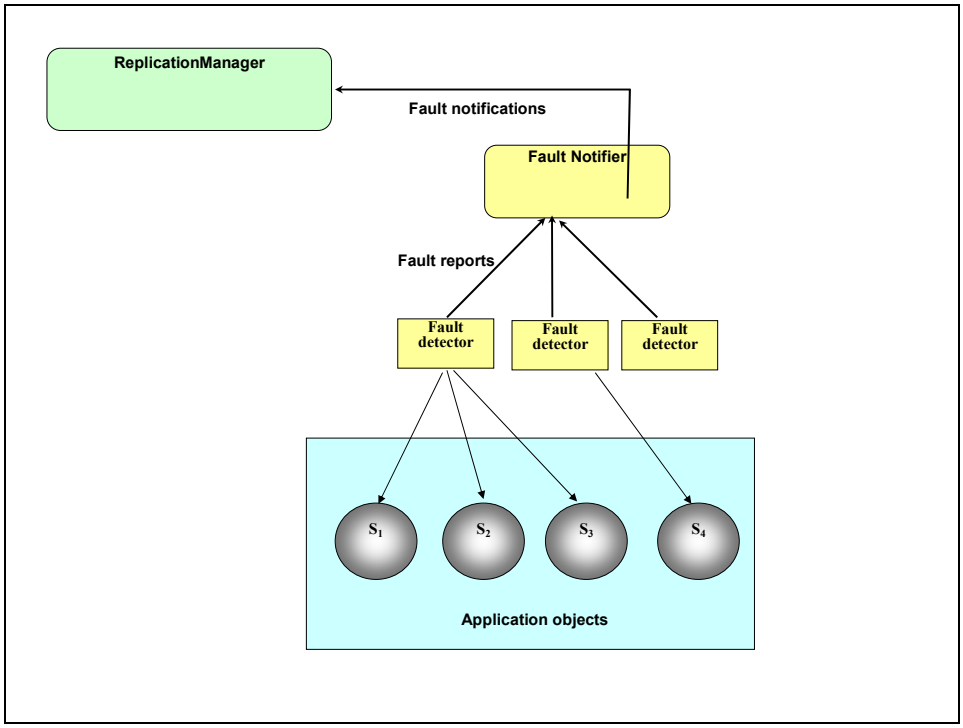
Step 3: GenericFactory 介面提供 Infrastructure-controlled 的 membership style 管理。對使用者來說是一個簡單方便全自動的管理方式，它們只要先以 PropertyManager 設定它們期望的系統特性，然後只要使用

*GenericFactory::create_service()*和 *GenericFactory::delete_service()*就可達到設定群組。

Step 4: Replication Manager 呼叫在數個不同主機的 *factory::create_service()* 來 deploy service，並回傳 service 的 WSDL 給 Replication Manager。

Step 5: Replication Manager 依據 replication style, fault monitoring style 決定是否啓始錯誤監測，並藉由 *FaultNotifier::connect_structured_comsumer()* 向 fault notifier 註冊，以接收成員的 fault notification。若為 passive replication style，Replication Manager 須決定群組的 primary service，藉由 *GenericFactory::activate_service()*向 primary service 告知其為 primary 可以接受 client 的要求進行服務。最後 Replication Manager 將群組間的 WSDL 組合成 WSDL(內含 WSG)，將該 WSDL 的內容註冊至 UDDI。

第三項工作主要有兩部分：分別是分析設計「錯誤管理」(or fault management)相關的功能，以及 "logging and recovery" 機制的設計。錯誤管理應包含 Fault Detection，Fault Notification，和 Fault Classification，在此我們不討論 Fault Classification。如圖表 30 中之系統架構，我們分別用 Fault Detector, Fault Notifier 來達成以上功能。不管使用何種態的 Fault Detector 來偵測主機和網路錯誤，Fault Detector 都得向 Fault Notifier 報告錯誤發生，以便轉告 Replication Manager 和任何向它登記要獲知該訊息的物件。圖表 30 說明一簡單的系統中 Fault Detector，Fault Notifier 和 Replication Manager 之間的對應作。圖中的 fault detector 可能須同時偵測數個物件(如圖中的 S_1,和 S_2)或者僅偵測一個物件(如圖中的 S_3)，而 fault detector 會向 fault notifier 報告發生的錯誤事件。

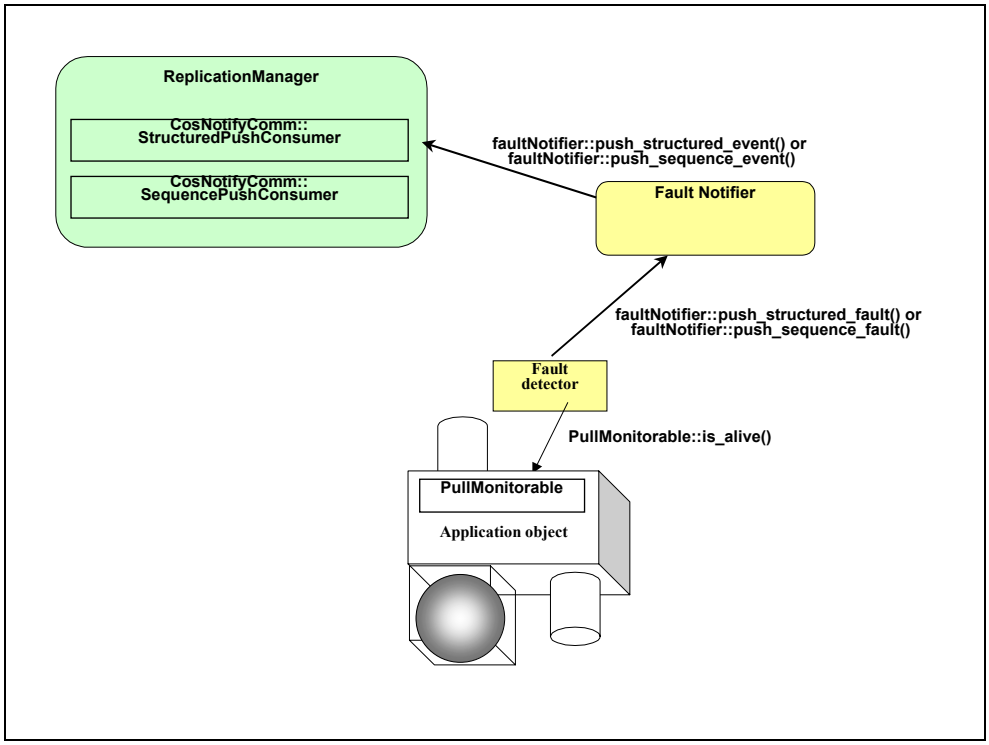


圖表 30 Fault management

Fault Detection

我們就圖表 31 的分析。我們定義一組介面來提供偵錯的功能。而且一般系統具有數個偵錯機制以偵測主機當機，資源用盡等。我們在第一年計劃中定義且說明 FaultMonitoringStyle 有 PULL 和 PUSH 兩種。我們擬先定義一簡單的 PullMonitorable 介面來提供 pull style fault detection。該介面由應用程式來繼承。PullMonitorable 介面有 *is_alive()*讓 Fault Detector 來呼叫。因效率考量，Fault Detector 通常是偵測同一機的物件，而各主機 Fault Detector 再受整體 Fault Detector 偵測保護。因 push 態因應用特性而有所不同，因此我們將不定義這種型態。

```
interface PullMonitorable{
  boolean is_alive(;
};
```



圖表 31 Fault detection

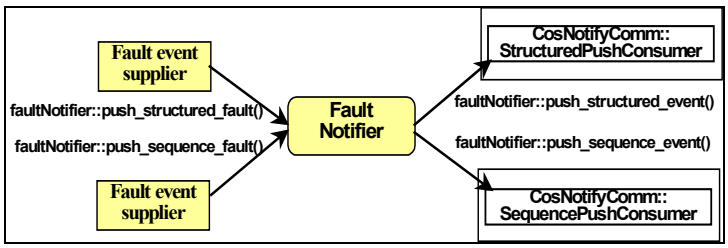
Fault Notification

```
interface faultNotifier{
   void push_structured_fault(in StructuredEvent event);
void push_sequence_fault(in EventBatch events);
ConsumerId connect_structured_fault_consumer(in URL push_consumer, in filter filter);
ConsumerId connect_sequence_fault_consumer(in URL push_consumer, in Filter filter);
…
};
```

Step 1 : FaultNotifier 介面的 *faultNotifer::push_structured_fault()*和 *faultNotifier::push_sequence_fault()*使偵錯器將錯誤報告推給 FaultNotifier。

Step 2 : 另外還有 *connect_structured_fault_consumer()*和 *connect_sequence_fault_consumer()*讓 Replication Manager 或其他應用程式來登記以獲得錯誤通報。

Step 3 :圖表 32 說明 fault detector 測知錯誤發生時，呼叫 *faultNotifer::push_structured_fault()*或 *faultNotifier::push_sequence_fault()* 而 fault detector 接收到通知後，呼叫 RM 繼承的 CosNotifyComm 介面中的 *CosNotifyComm::StructuredPushConsumer()*以知會 RM 發生錯誤。



圖表 32 Fault notification

Logging & Recovery management

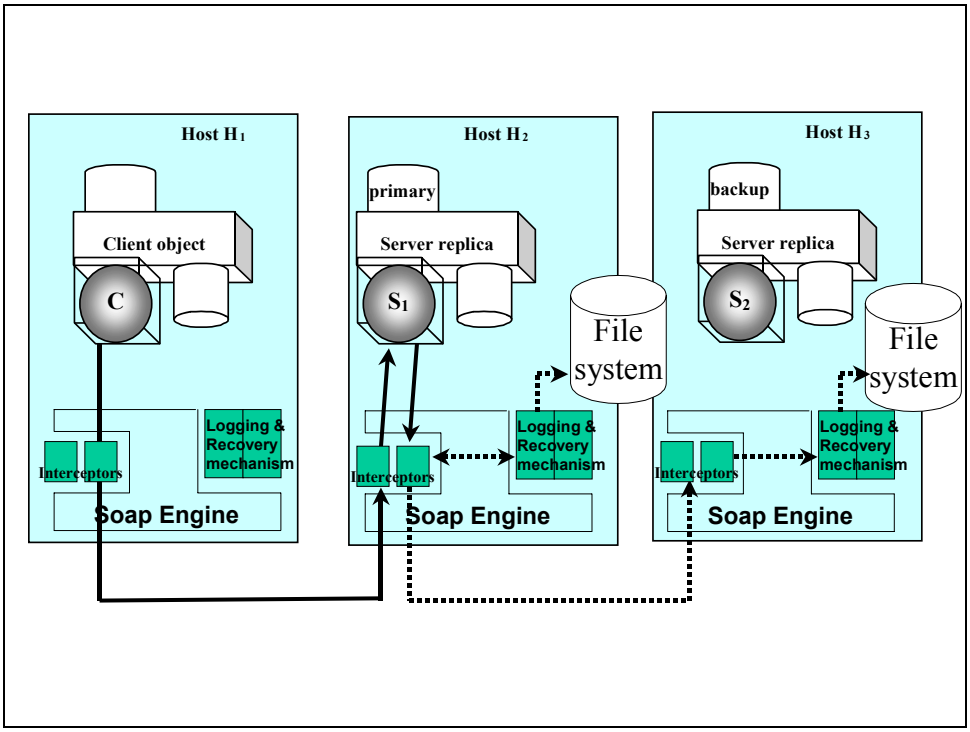
```
interface Checkpointable{
State get_state();
void set_state(in State s)
};
```

```
interface Updatable:Checkpointable{
   State get_update();
Void set_update(in State s)
};
```

容錯基礎系統含記錄及復原機制。在正常運作時，記錄機制記錄主要成員的狀態及動作。錯誤發生後，回復機制取出記錄並以此回復備援物件的狀態，以復它能繼續原主要成員死後的各項服務。記錄及復原機制也被用來啟動新加入的備援物件。我們將不定義介面，因本機制並不會為應用程式呼叫到。

任何應用物件要有記錄機制必須繼承 *Checkpointable* 介面。另外，它可繼承 *updateable* 介面，使應用程式能遞增式記錄狀態。

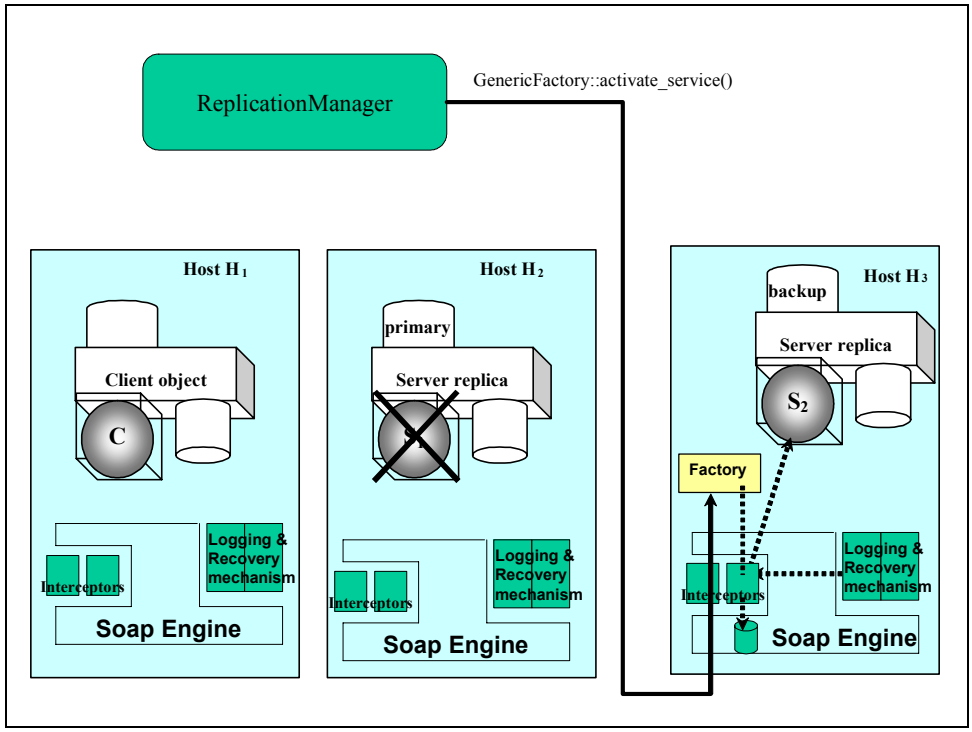
Logging mechanism



圖表 33 Logging mechanism

Logging 機制由 PropertyManager 得知 CheckpointInterval 便週期性呼叫 *checkpointable::get_state()*或 *updaptable::get_update()*。一般來說，記錄格式不須特別指定。通常是記錄呼叫及回應訊息。以及狀態更新的活動，如上圖所示。記錄檔必須保持訊息的次序，以便回覆(recovery)能正確重作(replay)。讀取及更新狀態也須指向呼叫該兩動作的時間點。讀取及更新狀態都應包括其呼叫及回應訊息。

Recovery mechanism

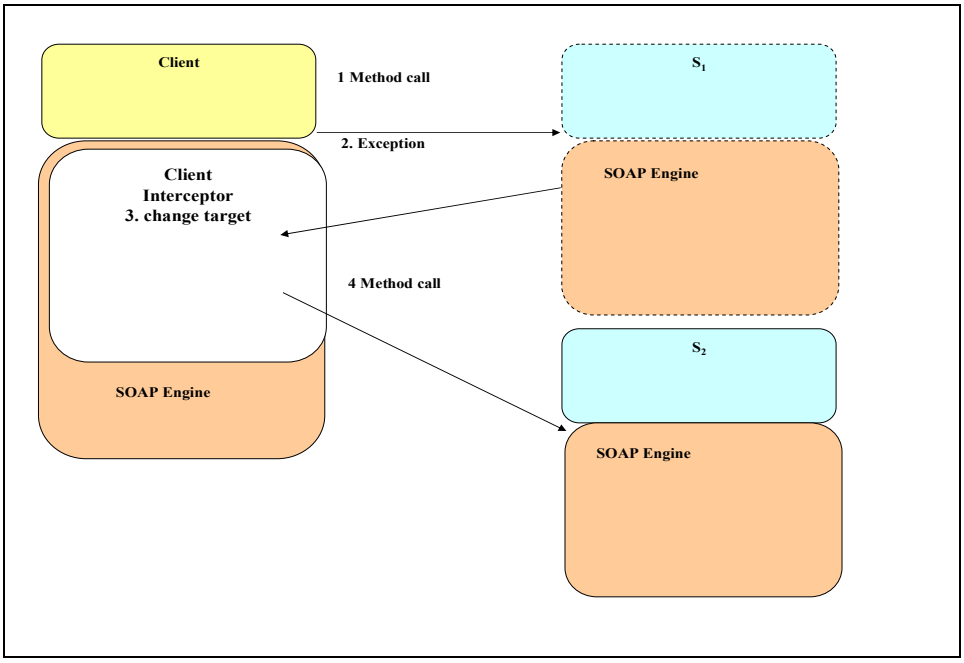


圖表 34 Recovery mechanism

錯誤發生後，RM 利用 GenericFactory:: activate_service()啓始復原機制，復原機制呼叫 checkpointable::set_state()或 updaptable::set_update()設定備援主要成員的狀態及新加入群組的狀態。復原機制以記錄檔的資訊來更新其狀態成正確值以開始正常運作，如上圖所示。

第四項工作 Fault Tolerance Infrastructure，主要區分成四個部分，分別為 1.客端的容錯透通性(failure transparency)、2.有效(Active)成員群組管理、3.抑制多餘需求(suppression of redundant invocations)機制，以及 4.抑制多餘回應(suppression of redundant responses)機制。

首先，為了減少客端(client)不必要的困擾，提供 1.客端的容錯透通性(Redundancy and Failure transparency to clients)部分。



圖表 35 Client transparency

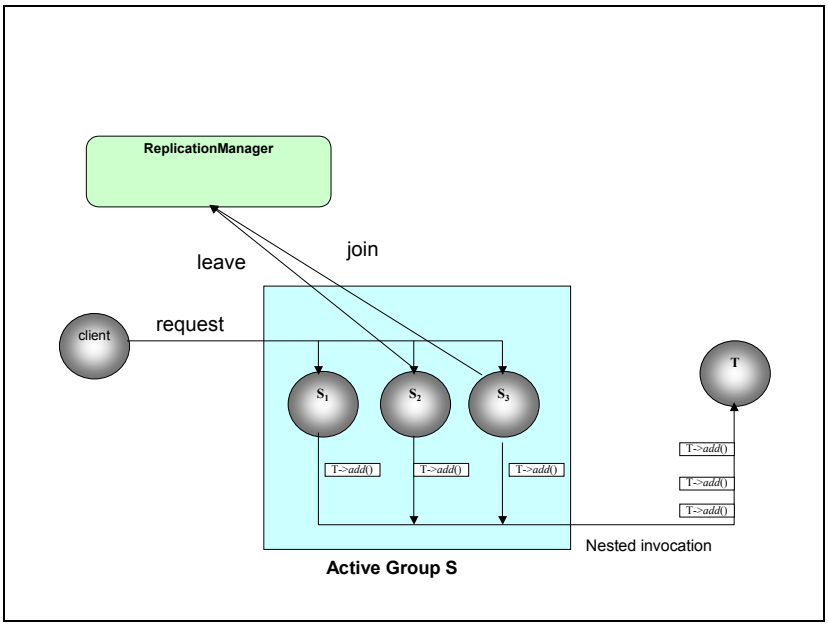
我們計劃在 client 端使用 FT-SOAP interceptor 的機制來達成容錯的透通性，其基本概念如圖表 35 所示。圖中 Replicated Servers S_1 和 S_2 為同一群組，client 取得其 group reference 後對其要求服務，對應動作如下：

Step 1: 正常呼叫傳到 S_1。

Step 2: 錯誤發生，並傳回客端的 ORB，被 Interceptor 所攔截。

Step 3: Interceptor 依群組的參照資料轉換呼叫新對象 S_2。

Step 4: 重新對新 S_2 呼叫。



圖表 36 Active group management and the nested invocation problem.

接著，加上 2.有效成員群組管理的部分(Active Group Management)，提高整體系統在群組管理上機能。

圖表 36 說明了一 Active group 的組合及其組員加入離開群組的運作情形，圖中的 S2 對 RM 發出離開(leave)群組的呼叫以脫離群組的運作。而 S2 對 RM 發出加入(join)群組的呼叫以參加群組的運作。

然後，為了減少伺服端(Server)多餘的負載，提供 3.抑制多餘需求(suppression of redundant invocations)機制。圖表 36 也說明了一 Active group 可能引起的 Redundant nested invocation 問題，圖中的 S1,S2,及 S3 都執行 client 的服務呼叫(request)。如果，該服務會引發對另外的伺服程式 T 提出 nested invocation T->add()，如果讓 S1,S2,及 S3 nested invocation 都送到伺服程式 T，則因執行三次 T->add()有可能破壞伺服程式 T 的正確性。因此必須有一個機制來去除多餘的 nested invocations (redundant netsted invocations)。除此之外，還需要將可能的返回結果分散到每一個成員。如此，整個群組的運作才能正確無誤。

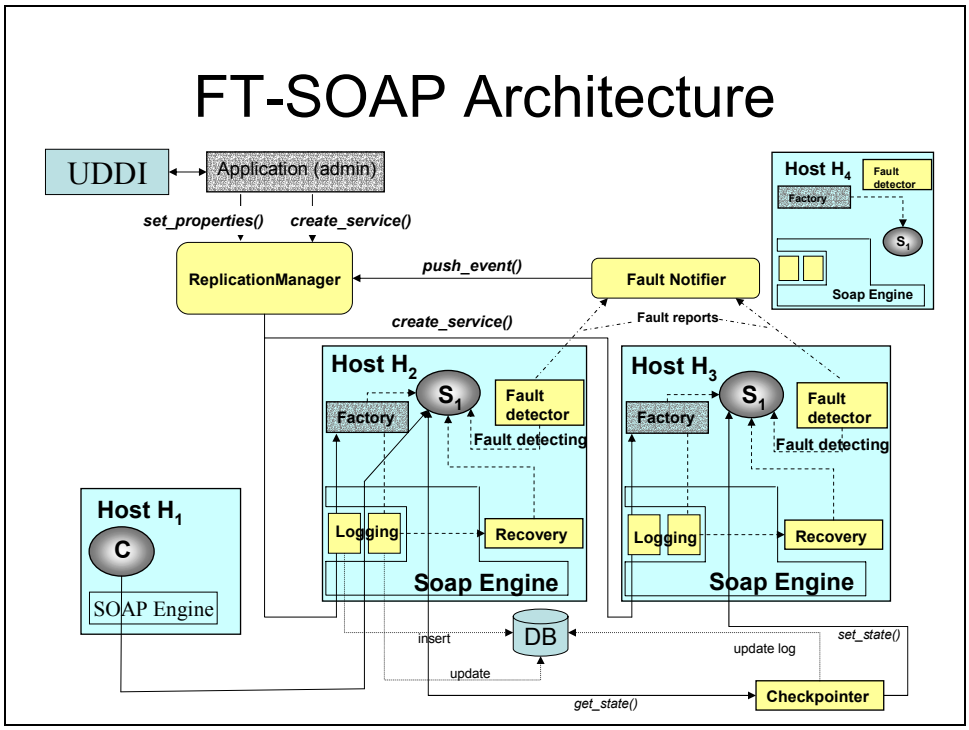
最後，為了降低系統的資源浪費，再提供 4.抑制多餘回應(suppression of redundant responses)機制，提升系統資源利用率。 圖表 36 除了 Redundant nested invocation 問題還須有一個機制來去除 S1,S2,及 S3 可能都返回的多餘回應結果。否則，會讓 client 得到多餘的執行返回值(return value)，進而造成錯誤。

3. Experiment Results

根據 FT-SOAP 所設計的整體架構，我們實做了一個 FT-SOAP 的雛型，經由對原始的 SOAP 系統架構與 FT-SOAP 的架構比對分析結果發現，將原有之 SOAP 系統提昇為具備容錯(Fault Tolerance)機制的 FT-SOAP，將會增加系統的整體負荷(overhead)，而這些多餘的負荷主要歸咎在提供容錯機制時所增加的三個部分功能，分別為：Logging、Checkpointing 與 Fault Detection 等三項機制，因此，我們便針對此三項機制做一系列的測試。實驗如下：

我們之實驗主要是針對原始 SOAP 系統在增加容錯機制後，對原始系統的影響程度所做的效能影響評估。藉此以評估建置 FT-SOAP 系統的實質效益。

根據系統架構圖表 37 所示，FT-SOAP 系統對原始的系統的環境影響與服務(Service)的反應時間影響因子中，佔絕大部分的系統元件一共可分 Logging、Checkpointig 以及 Detector 三項。因此，根據此三項重要因子，我們特別設計三項實驗，根據實驗結果以評估 FT-SOAP 是否具備實做之實值效益。



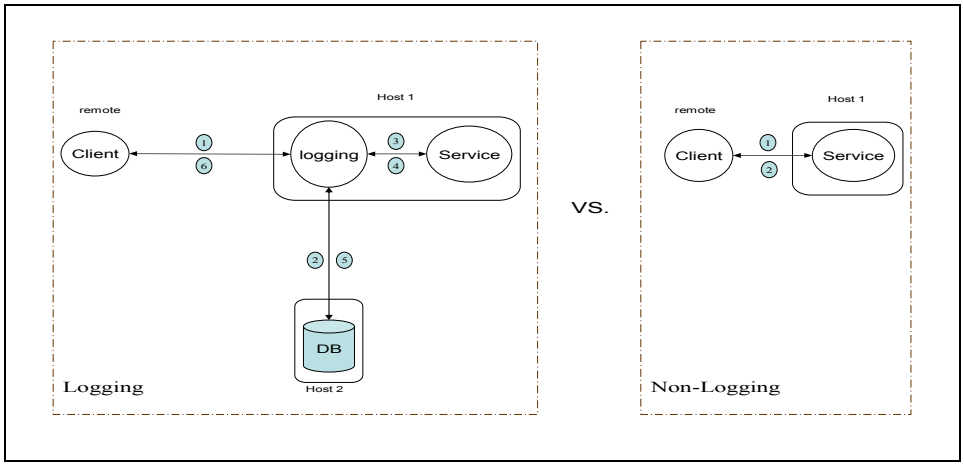
圖表 37 FT-SOAP 雛型

3.1 Logging Experiment

Logging 實驗的設計原因是因為，每個服務(Service)在接收到客端的需求前，都需要先由 Logging 機制，將需求的相關資訊紀錄到資料庫中，服務完成後，將訊息回應給要求服務之客端，在回應前，必須將該需求之紀錄參數狀態值更新為完整需求。以便提供給 Recovery 機制在進行系統 Recovery 時使用。

但由於 Logging 對原始服務的 Overhead 影響相當高，因此，我們認為需針對該項機制做效能上的評估，以判斷該項機制是否能夠在實做 FT-SOAP 後，符合整體系統的容忍範圍。評斷方式是以客端發出需求訊息後，Web Service 的回應時間來做平均值計算。

Logging 實驗的設計如圖表 38 中所示，此實驗的設計分為實驗組與對照組，加上 Logging 機制的為實驗組；未加上 Logging 機制的則為對照組。實驗環境：伺服器端的設備(Web Service 與 Database)皆處於 Local LAN 的環境由 10/100 Switch Hub 所連接，客端與伺服器端的距離則位於跨過兩個 Router 的 Internet 上，藉此使實驗環境更接近實際的使用環境。

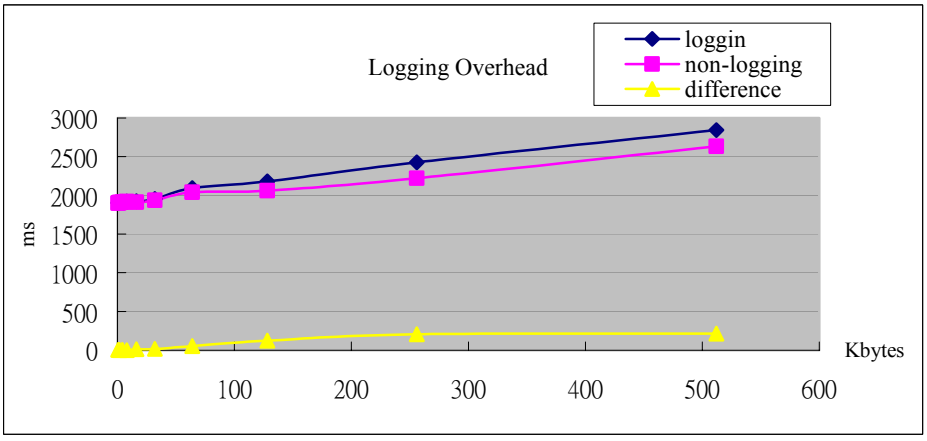


圖表 38 Logging Experiment Scenario

實驗數據如表格 2 所示，實驗結果的數值是經過 25 次的測試後，所得之平均值，經由表格 2 的數值，畫出圖表 39 之 X,Y 分佈圖。

Kbytes / Resp.(ms)	0.5	1	2	4	8	16	32	64	128	256	512
Logging	**1904**	**1909**	**1909**	**1916**	**1922**	**1923**	**1954**	**2092**	**2182**	**2425**	**2844**
Non-logging	**1899**	**1906**	**1907**	**1913**	**1921**	**1912**	**1939**	**2938**	**2059**	**2221**	**2631**
difference	**5**	**3**	**2**	**3**	**1**	**11**	**15**	**53**	**123**	**204**	**213**

表格 2 Logging Experiment Result



圖表 39 Logging Experiment X,Y 分佈圖

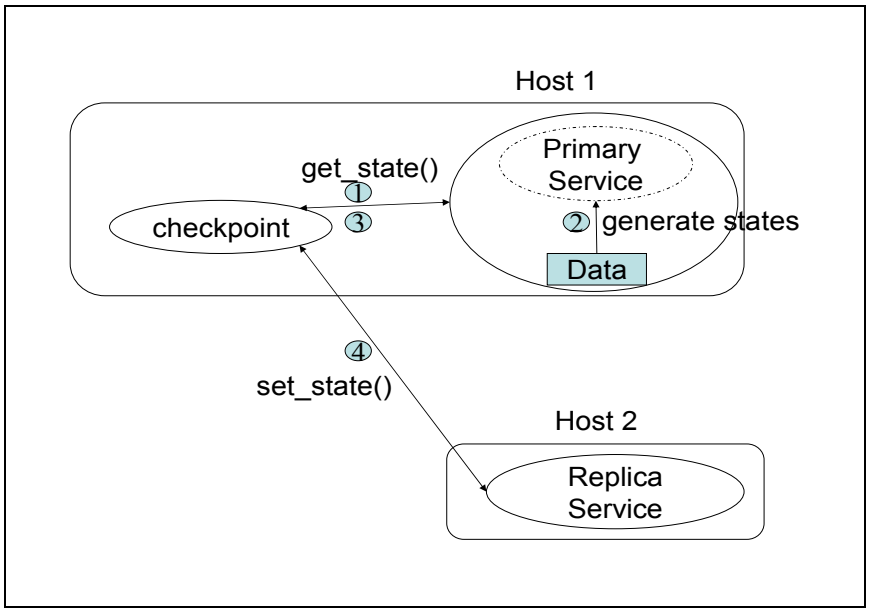
根據實驗結果的數據顯示，事實上，在系統上加上 Logging 機制後，在傳輸之資料量小於 16kbytes 時，系統回應時間約在 5ms 之內，根據景文技術學院的計算機中心所提供的 Proxy Log 數據，針對一般 Web 的需求(Request)所發出的資料封包大小平均在 5kbyes 左右，相對之下，FT-SOAP 所使用的 Logging 機制，對整個系統的影響程度並不大，使用者對這 5ms 的 overhead 的感覺並不會很明顯，即使是在資料封包為 512Kbytes 大小，overhead 平均為 213ms 時，使用者對這樣短暫的時間亦不會有太大的影響才是。

3.2 Checkpointing Experiment

Checkpointing 實驗的設計主要原因是，因為 Recovery 機制是在新的主要服務(Primary Service)上 replay Logging 機制所紀錄下來的有效需求(validate request logs)，這些紀錄會與先前 Crash 的主要服務的生命週期與服務的次數成正比，假使，一個 Crash 的主要服務在其正常情況下提供了一萬次完整的服務，則 Logging 機制將會紀錄一萬筆有效需求。

結論是，新的主要服務若要使其記憶體內所儲存的狀態(States)回復到 Crash 的服務，在服務完那一萬次完整服務後的狀態時，新的主要服務便需要依序 replay Logging 機制所紀錄的一萬次需求，這是相當耗時的。為了縮短 Recovery 的時間，Checkpointing 的設計便因應而生，依據系統的需求，週期性的將主要服務儲存於記憶體中的狀態與備援服務(Replica Service)做同步的動作。

然而，同步時所需花費的時間是必須經過評估的，藉以考量此一機制是否符合適用性。實驗如圖表 40 所示。

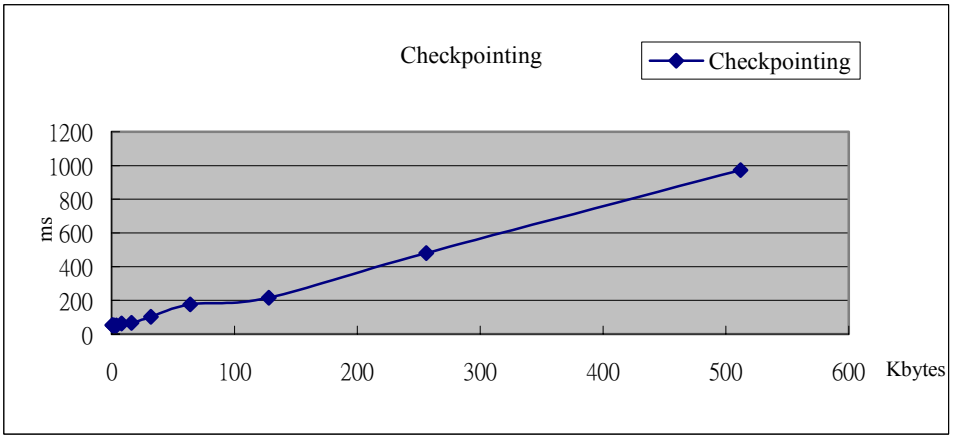


圖表 40 Checkpointing Scenaior

由於，狀態的同步與狀態資料量的大小關係相當密切，因此，我們利用資料量的大小變動對執行時間影響多寡來做實驗，實驗數據如表格 3 所示，實驗數據是以相同資料量模擬記憶體中的狀態值大小進行 Checkpointing 動作，連續執行 105 次，取其中的 100 次後計算其平均時間，將所得的數據畫成 X,Y 分佈圖，如圖表 41 所示。

Kbytes / Resp.(ms)	0.5	1	2	4	8	16	32	64	128	256	512
checkpointing	54	52	52	51	63	67	102	176	215	480	972

表格 3 Checkpointing Experiment Result



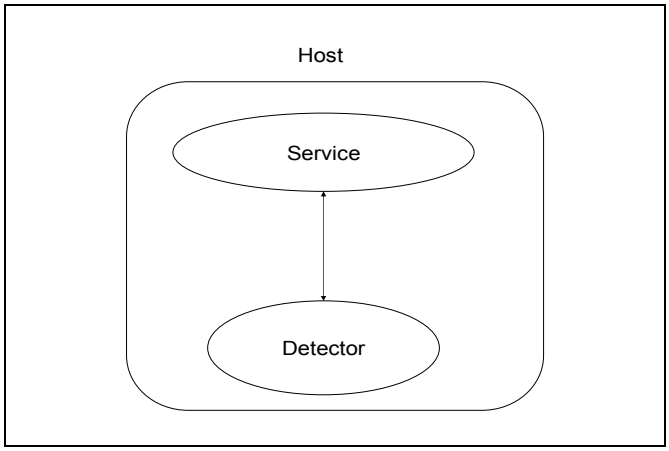
圖表 41 Checkpointing Experiment X,Y 分佈圖

根據實驗數據顯示，Checkpointing 機制所需花費的時間確實與該服務所需同步的狀態值資料量成正比，因此，服務提供商(Service Provider)可參考此項實驗數據，根據此項數據對其所開發之服務，依據狀況減少或增加其儲存於服務記憶統中的狀態資料量。

3.3 Detector Experiment

Detector 實驗的設計主要原因是，由於每個接受 FT-SOAP 系統保護的服務都需要接受 Detector 週期性的監控存活狀況，而這項機制的使用需要佔用 CPU 的處理時間，因此，我們依據這個狀況設計了這項實驗。

實驗方式是設計一個 Detector 偵測一個服務，經由監測週期時間的改變來觀察 Detector 對 CPU 的使用率，測試時間為 30 分鐘後的平均值。此項實驗是在 Linux/Unix 類型之作業系統所做的測試，主因是唯有在該類型之作業系統才能夠針對每項處裡程序(Process)做 CPU 使用率的監測。實驗如圖表 42 所示。

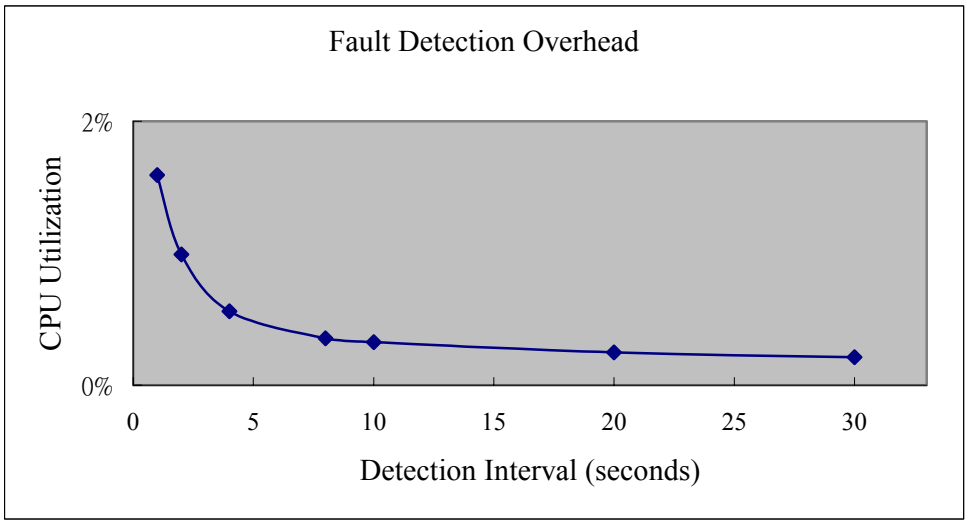


圖表 42 Fault Detection Scenario

實驗數據如表格 4 所示，依據實驗數據所畫出的 X,Y 分佈圖如圖表 43 所示。

Poll Interval(sec.)	1	2	4	8	10	20	30
CPU Util. (%)	1.59	0.99	0.56	0.35	0.33	0.25	0.21

表格 4 Fault Detection Experiment Result



圖表 43 Fault Detection Overhead X,Y 分佈圖

根據實驗所得之數據可以發現，Detector 的 CPU 使用率並不高。這是因為，

一般 Detector 的監測週期大都以 30 秒鐘為基本單位，依據需求往上增加，根據測試所得到的數據，以每 30 秒鐘做一次監測動作的情況下，只需 0.21%的 CPU 時間。而且，一般而言，在同一部伺服器上，並不會提供太多的服務的，以台新銀行的電話理財中心而言，其提供主機與 PC 間資料交易連線 Data Center 伺服器上只提供八個相同服務，因此，眾合比較起來，Detector 所造成的 Overhead 仍為一般使用者所接受。

經由以上三項實驗結果，我們所設計與實做之 FT-SOAP 系統，其整體效能並不會因為增加容錯機制後，而使系統的效能降低太大，而這些因容錯機制所需付出的代價亦可讓一般使用者所接受。因此，我們所設計 FT-SOAP 的容錯機制應是值得推廣的。

4. Summary

本研究在前文中提到，以目前 SOAP 的標準上並沒有提供任何容錯的功能，因此，我們藉由此計畫提供學術界一個 SOAP 的容錯機制的標準。經由我們實做研究雛形(prototype)得經驗來驗證此標準界面之可行性(concept-proving)。並藉此提供業學界一個後續研究法展的平台。

近年來由於電子商務的盛行，由於 SOAP 的特性(可穿透各類型之防火牆、XML 與 HPPT Base)，SOAP 更是提供電子商務的最佳理想平台，但是由於目前 SOAP 並未提供容錯的機制，造成許多欲利用 SOAP 為電子商務(EC)的基礎開發平台的廠商因此停滯不前； 因此，藉由本計畫的實行，為 SOAP 提供容錯機制，而那些因為 SOAP 未提供容錯機制而停滯的相關行業之廠商將會因為這項計畫的完成，將能夠依照原先既定計畫以 SOAP 為其發展 EC 平台的方向進展。更由於目前世界各大廠商(如:Compaq)都積極發展與運用 SOAP 技術於 B2B 或 B2C 的電子商務平台在其製造或銷售流程上，若要能夠獲得各大廠訂單，必須能夠與其 EC 系統能夠互通，如此一來更可以增加我們產業之競爭力。

5. Reference

[APM93] Architecture Projects Management, ANSAware Version 4.1 Manual Set, Castle Park, Cambridge UK, March 1993.

[Ari85] A. Arizienis, "The N-version approach to fault-tolerant software," IEEE Trans. on Software Engineering, Vol. SE-11, No. 12, pp. 1491-1501, December 1985.

[Bac93] J. Bacon, Concurrent systems, Addison-Wesley, New York, 1993.

[BHG87] P. A. Bernstein, V. Hadzilacos, and N. Goodman, Concurrency Control and Recovery in Database Systems, Addison-Welsey, NJ, 1987.

[Boo91] G. Booch, Object-Oriented Design with Applications, The Benjamin and Cummings Publishing, Redwood City, Calif., 1991.

[BR94] K. P. Birman and R. van Renesse, ed., Reliable Distributed Computing with the Isis Toolkit, IEEE CS Press, Los Alamitos, CA., 1994.

[Bro95] K. Brockschmidt, Inside OLE, 2nd ed., Microsoft Press, Washington, 1995.

[CHI01] John Chirillo, Hack attacks denied - a complete guide to network lockdown, 2001.

[CHY98] E. Chung, Y. Huang, S. Yajnik, Deron Liang, C. Shih, C.-Y. Wang, and Y.-M. Wang, "DCOM and CORBA Side by Side, Step by Step, and Layer by Layer", C++ Report, Vol. 10, No. 1, pp. 18-30, Jan. 1998.

[CPM98] Charlie Scott, Paul Wolfe & Mike Erwin,Virtual Private Network 2nd Ed., O'REILLY

[Free98] L. Freeman, “Net Drives B-to-B to New Highs World-Wide”, NetMarketing，January 1998 (www.netb2b.com).

[HK93] Y. Huang and C. Kintala, “Software Implemented Fault Tolerance,” in Proc. of 22nd Fault-tolerance Computing Symposium, pp. 2-10, 1993.

[IBM01] IBM, MQSeries Integrator: A Technical Overview, http://www-4.ibm.com/software/ts/mqseries/library/whitepapers/eai-tech/

[Inte98] Internet week, August 31,1998.

[KT87] R. Koo and S. Toueg, "Check-pointing and rollback-recovery for distributed systems," IEEE Trans. on Software Engineering, Vol. SE-13, No 1, pp. 23-31, January 1987.

[LFY99] Deron Liang, Cheng-Liang Fang and S.-M. Yuan, "A Fault-Tolerant Object Service on CORBA," Journal of Systems and Software, Vol. 48, pp. 197-211, 1999.

[LLG92] R. Ladin, B. Liskov, and S. Ghemawat, "Replication," ACM Transaction on Computer Systems, Vol. 10, No. 4 , pp. 360-391, November 1992.

[MSFT98] Microsoft, DCOM Technical documents, http://msdn.microsoft.com/library/

[MSFT01] Microsoft, UDDI technical documents, http://uddi.microsoft.com/developer/default.aspx

[NMN00] Stephen Northcutt, Donald McLachlan, Judy Novak, Network Intrusion Detection: An Analyst's Handbook, September, 2000.

[OMG98] OMG, “Fault Tolerant CORBA using entity redundancy – Request for Proposal”, OMG document orbos/98-04-01.

[OMG00] OMG, Fault Tolerant CORBA Specification,V1.0, OMG document orbos/2000-04-04.

[PBS89] L. Peterson, N. Buchholz and R. Schlichting, "Preserving and using context information in inter-process communication," ACM Transactions on Computer Systems 7, Vol. 3, pp. 217-246, August 1989.

[RBM96] R. van Renesse, K. P. Birman, and S. Maffeis, “Horus: A Flexible Group Communications System”, Communications of the ACM, Vol. 39, No. 4, pp. 76-83, 1996.

[RC98] T. Retter and M. Calyniuk, Technology Forecast:1998,(Price Waterhouse, March 1998).

[Rog97] D. Rogerson, Inside COM, Microsoft Press, Washington, 1997.

[Rym93] J. R. Rymer, “IBM’s System Object Model”, Distributed Computing Monitor, Vol. 8, No. 3, page 1-24, March 1993.

[SDP88] S. K. Shrivastava, G. N. Dixon, and G. D. Parrington, "An Overview of the Arjuna Distributed Programming System," Computing Laboratory, University of Newcastle upon Tyne, UK, 1988.

[SS83] R. D. Schlichting and F. B. Schneider, "Fail-Stop Processors: An approach to designing fault-tolerant computing systems," ACM Transactions on Computer Systems, Vol. 1, No. 3, pp. 222-238, August 1983.

[UDDI01] UDDI White Paper, http://uddi.org

[W3C00] Simple Object Access Protocol (SOAP) 1.1 , http://www.w3.org/TR/SOAP/

[W3C01a] Web Services Description Language (WSDL) 1.1 ,W3C Note 15 March 2001, http://www.w3.org/TR/wsdl

[W3C01b] XML Schema Part 0: Primer , http://www.w3.org/TR/xmlschema-0/

[WHF93] Y. M. Wang, Y. Huang and W. K. Fucks, "Progressive Retry for Software Error Recovery in Distributed Systems," in Proc. of 22nd Fault-tolerance Computing Symposium, pp. 138-144, 1993.

[Zwas96] V. Zwass, “Electric Commerce :Structures and Issues”, International Journal of Electric Commerce (Fall 1996).

Appendix: Java IDL Interface for FT-SOAP

```
#ifndef _FTSOAP_IDL_
#define _FTSOAP_IDL_

#pragma prefix “ftsoap.org”
module FTSOAP {
typedef int ReplicationStyleValue;
const ReplicationStyleValue STATELESS = 0;
const ReplicationStyleValue COLD_PASSIVE = 1;
const ReplicationStyleValue WARM_PASSIVE = 2;
const ReplicationStyleValue ACTIVE = 3;
const ReplicationStyleValue ACTIVE_WITH_VOTING = 4;

typedef int MembershipStyleValue;
```

```
const MembershipStyleValue MEM_APP_CTRL = 0;
const MembershipStyleValue MEM_INF_CTRL = 1;

typedef int ConsistencyStyleValue;
const ConsistencyStyleValue CONS_APP_CTRL = 0;
const ConsistencyStyleValue CONS_INF_CTRL = 1;

typedef int FaultMonitoringStyleValue;
const FaultMonitoringStyleValue PULL = 0;
const FaultMonitoringStyleValue PUSH = 1;
const FaultMonitoringStyleValue NOT_MONITORED = 2;

typedef int FaultMonitoringGranularityValue;
const FaultMonitoringGranularityValue MEMB = 0;
const FaultMonitoringGranularityValue LOC = 1;
const FaultMonitoringGranularityValue LOC_AND_TYPE = 2;

typedef int InitialNumberReplicasValue;
typedef int MinimumNumberReplicasValue;
typedef int Degree;
typedef int Threshold;
typedef long CheckpointInterval;
typedef string Location;
typedef string Service; //WSDL
typedef string typeId;
typedef Service ServiceGroup; //WSG
typedef Service FaultNotifier; //WSDL
typedef sequence<Location>Locations;

struct Property {
   string name;
   string value;
}
typedef sequence<Property> Criteria;

struct FactoryInfo {
   GenericFactory the_factory; //Factory Name.
   Location the_location;
   Criteria the_criteria;
```

```
}

typedef sequence<FactoryInfo> FactoryInfos;
typedef FactoryInfos FactoriesValue;

struct Properties {
   Property[] props;
   FactoryInfos factoryInfos ;
}

// Exception Declare
exception InterfaceNotFound {};
exception ServiceGroupNotFound {};
exception MemberNotFound {};
exception ServiceNotFound {};
//exception MemberAlreadyPresent {};
//exception BadReplicationStyle {};
exception ServiceNotDeployed {};
exception ServiceNotAdded {};
//exception PrimaryNotSet {};
exception UnsupportedProperty {
   Property prop;
   //string name;
   //string value;
};
exception InvalidProperty {
   Property prop;
   //string name;
   //string value;
};

exception NoFactory {
   Location the_location;
   typeId type_id;
};
exception InvalidCriteria {
   Criteria invalid_criteria;
};
exception CannotMeetCriteria {
```

```
    Criteria unmet_criteria;
};
interface PropertyManager {
    void set_default_proerties(in Properties props)
            raises (InvalidProperty,UnsupportedProperty);
    Properties get_default_properties()
            raises (InvalidProperty,UnsupportedProperty);
    void remove_default_properties(in Properties props)
            raises (InvalidProperty,UnsupportedProperty);
    void set_type_properties(in typeId the_typeId, in Properties props)
            raises (InvalidProperty,UnsupportedProperty);
    Properties get_type_properties(in typeId the_typeId)
            raises (ServiceNotFound);
            //raises (InvalidProperty,UnsupportedProperty);
    void remove_type_properties(in typeId the_typeId, in Properties props)
            raises (ServiceNotFound,InvalidProperty,UnsupportedProperty);
            //raises (InvalidProperty,UnsupportedProperty);
};

interface GenericFactory {
    Service create_service(in typeId the_typeId)
            raises(ServiceGroupNotFound,ServiceNotFound,ServiceNotAdded);
    void destroy_service(in Service service)
            raises(ServiceNotFound);
    string activate_service(in typeId the_typeId)
            raises(ServiceNotFound,ServiceNotAdded);
};

interface ServiceGroupManager {
    ServiceGroup create_member(in typeId    the_typeId, in Location the_location)
            raises(ServiceGroupNotFound,MemberNotFound);
    ServiceGroup add_member(in ServiceGroup srvGrp, in Service ServiceMember)
            raises(ServiceGroupNotFound);
    ServiceGroup remove_member(in ServiceGroup srvGrp, in Location the_location)
            raises(ServiceGroupNotFound,MemberNotFound);
    ServiceGroup set_promary_member(in ServiceGroup srvGrp, in Location the_location)
            raises(ServiceGroupNotFound);
    Locations locations_of_member(in ServiceGroup srvGrp)
            raises(ServiceGroupNotFound);
```

```
};

interface NS_consumer {
   void push_event(in Event faultEvent);
};

interface FaultNotifier:NS_consumer {
   void subscribe(in Service rm, in string filter);
   void unsubscribe (in Service rm, in string filter);
};
interface FaultDetector {
   void register(in Service fault_notifier, in Service detectionTarget);
};

interface PullMonitorable {
   boolean is_alive();
};

interface ReplicationManager:PropertyManager,GenericFactory,ServiceGroupManager,NS_consumer {
   void register_fault_notifier(in FaultNotifier fault_notifier);
   FaultNotifier get_fault_notifier()
            raises (InterfaceNotFound);
};

typedef String State;

//Exception Declare
exception NoStateAvailable {};
exception InvalidState {};
exception NoUpdateAvailable {};
exception InvalidUpdate {};

interface Checkpointable {
   State get_state()
            raises(NoStateAvailable);
   void set_state(in State s)
            raises(InvalidState);
};
```

```
interface Updateable:Checkpointable {
   State get_update()
            raises(NoUpdateAvailable);
   void set_update(in State s);
            raises(InvalidUpdate);
};

interface RecoveryMechanism {
   string doRecovery(in typeId the_typeId);
};
};
#endif
```