Architecture of Enterprise Applications 11 JNLP and Availability

Haopeng Chen

***RE**liable, **IN**telligent and **S**calable Systems Group (**REINS**)*

Shanghai Jiao Tong University

Shanghai, China

http://reins.se.sjtu.edu.cn/~chenhp

e-mail: chen-hp@sjtu.edu.cn

Contents

- JNLP
- Availability
  - Source of stimulus
  - Stimulus
  - Environment
  - Artifact
  - Response
  - Response measure
  - Tactics

Java Web Start

- Java Web Start software provides the power to launch full-featured applications with a single click.
  - Users can download and launch applications, such as a complete spreadsheet program or an Internet chat client, without going through lengthy installation procedures.

- With Java Web Start software,
  - users can launch a Java application by clicking a link in a web page.
  - The link points to a Java Network Launch Protocol (JNLP) file, which instructs Java Web Start software to download, cache, and run the application.

Java Web Start

- Java Web Start software provides Java developers and users with many deployment advantages:
  - With Java Web Start software, you can place a single Java application on a web server for deployment to a wide variety of platforms, including Windows, Linux, and Solaris.
  - Java Web Start software supports multiple, simultaneous versions of the Java platform.
  - Users can create a desktop shortcut to launch a Java Web Start application outside a browser.
  - Java Web Start software takes advantage of the inherent security of the Java platform.
  - Applications launched with Java Web Start software are cached locally for improved performance.
  - Updates to a Java Web Start application are automatically downloaded when the application is run standalone from the user's desktop.

Java Web Start Application

- Software designed by using component-based architecture can easily be developed and deployed as a Java Web Start application.

Java Web Start Application

- WebStartJava.java

```
public class WebStartJava extends Applet {
  private MainJPanel mainPanel = new MainJPanel();

  public void init() {
    int r = 20;

    try {
       r = Integer.parseInt(getParameter("r"));
    } catch (Exception e) {
       e.printStackTrace();
    }

    mainPanel.setR(r);

    this.setLayout(null);
    this.setSize(800, 600);
    mainPanel.setBounds(0, 0, getWidth(), getHeight());
    this.add(mainPanel);
    this.addResizeLayoutListener();
    this.setVisible(true);
}
```

Java Web Start Application

- WebStartJava.java

```
  private void addResizeLayoutListener() {
    this.addComponentListener(new ComponentAdapter() {
      public void componentResized(ComponentEvent e) {
        mainPanel.setBounds(0, 0, getWidth(), getHeight());
      }
    });
  }
}

class MainJPanel extends JPanel {

  private int space = 10;
  private int x = space;
  private int y = space;
  private int r = 40;

  public MainJPanel() {
    this.setLayout(null);
    this.setBackground(Color.white);
    this.setVisible(true);
    new Move().start();
  }
```

Java Web Start Application

- WebStartJava.java

```
class Move extends Thread {
  private int dx = 1;
  private int dy = 1;
  public void run() {
    while (true) {
      try {
        if (x + dx <= space) {
          x = space;
          dx = -dx;
        } else if (x + dx >= getWidth() - 2 * space - r) {
          x = getWidth() - 2 * space - r;
          dx = -dx;
        } else {
          x += dx;
        }
        if (y + dy <= space) {
          y = space;
          dy = -dy;
        } else if (y + dy >= getHeight() - 2 * space - r) {
          y = getHeight() - 2 * space - r;
          dy = -dy;
        } else {
          y += dy;
        }
```

Java Web Start Application

- WebStartJava.java

```
            repaint();
            Thread.sleep(3);
          } catch (Exception e) {}
        }
      }
    }

  @Override
  public void paintComponent(Graphics g) {
    super.paintComponent(g);
    g.setColor(Color.red);
    g.fillOval(x, y, r, r);
    g.drawLine(space, space, getWidth() - 2 * space, space);
    g.drawLine(space, space, space, getHeight() - 2 * space);
    g.drawLine(getWidth() - 2 * space, space, getWidth() - 2 * space,
               getHeight() - 2 * space);
    g.drawLine(space, getHeight() - 2 * space, getWidth() - 2 * space,
               getHeight() - 2 * space);
  }

  public void setR(int r) {
    this.r = r;
  }
}
```

Java Web Start Application

- examples.jar

```
cd \WebStartJava\classes
jar cvfm examples.jar manifest.mf WebStartJava

jarsigner
    -keystore key.keystore
    -storepass 123
    -keypass 321
    example.jar
    example.cer
```

Java Web Start Application

- WebStartServlet.java

```
import sun.misc.BASE64Encoder;
public class MainServlet extends HttpServlet {
  ……
  public void doPost(HttpServletRequest request, HttpServletResponse response)
                      throws ServletException, IOException {
      response.setContentType("text/html;charset=UTF-8");
      PrintWriter out = response.getWriter();

      String rootPath = request.getSession().getServletContext().getRealPath("");
      int r = 20;
      try {
        File confFile = new File(rootPath + "/init.conf");
        BufferedReader br = new BufferedReader(new FileReader(confFile));
        String line;
        while ((line = br.readLine()) != null) {
          String s[] = line.split("=");
          if (s != null && s.length == 2 && s[0].equals("r")) {
            r = Integer.parseInt(s[1]);
            break;
          }
        }
      } catch (Exception e) {
      }
```

Java Web Start Application

- WebStartServlet.java

```
    BASE64Encoder encoder = new BASE64Encoder();
    String base64bytes = new String(encoder.encode(createJNLP(r).
                                getBytes("UTF-8")));
    base64bytes = base64bytes.replaceAll("\r\n", "");
    out.print(base64bytes);
  }

  private String createJNLP(int r) {
    StringBuilder str = new StringBuilder("");
    str.append("<?xml version=\"1.0\" encoding=\"UTF-8\"?>\r\n");
    str.append("<jnlp >\r\n");

    str.append("<information>\r\n");
    str.append("<title>Java Web Start Example</title>\r\n");
    str.append("<vendor>L</vendor>\r\n");
    str.append("</information>\r\n");

    str.append("<resources>\r\n");
    str.append("<!-- Application Resources -->\r\n");
    str.append("<j2se version=\"1.6+\"/>\r\n");
    str.append("<jar href=\"example.jar\" main=\"true\" />\r\n");
    str.append("</resources>\r\n");
```

Java Web Start Application

- WebStartServlet.java

```
        str.append("<applet-desc\r\n");
        str.append("name=\"JavaWebStartExample\"\r\n");
        str.append("main-class=\"WebStartJava\"\r\n");
        str.append("width=\"800\"\r\n");
        str.append("height=\"600\">\r\n");

        str.append("<param name=\"r\" value=\"");
        str.append(r + "");
        str.append("\" />\r\n");

        str.append("</applet-desc>\r\n");

        str.append("<update check=\"background\"/>\r\n");
        str.append("</jnlp>");

        return str.toString();
    }
}
```

Java Web Start Application

- example.jnlp

```
<?xml version=\"1.0\" encoding=\"UTF-8\"?>
<jnlp codebase="http://127.0.0.1:8080/WebStartServer">
  <information>
    <title>Java Web Start Example</title>
      <vendor>L</vendor>
  </information>
  <resources>
    <!-- Application Resources -->
    <j2se version="1.6+"/>
    <jar href="example.jar" main="true" />
  </resources>
  <application-desc
        name="JavaWebStartExample"
        main-class="WebStartJava"
        width="80"
        height="600">
  </application-desc>
  <update check="background"/>
</jnlp>
```

Java Web Start Application

- index.html

```
<html>
  <head>
    <meta http-equiv="content-type" content="text/html" charset="UTF-8" />
        <script type="text/javascript" src="js/jquery-1.7.min.js"></script>
        <script type="text/javascript" src="js/deployjava.js"></script>
  </head>
  <body>
    <script>
      $.ajax({
              type: "POST",
              url: "ajax/main",
              contentType: "application/x-www-form-urlencoded; charset=utf-8",
              success: function(data){
                 console.log("OK");
                 console.log(data);
                 var attributes = {width:'100%', height:'98%', hspace:0, vspace:0 };
                 var parameters = {jnlp_href:'example.jnlp',jnlp_embedded : data,};
                 deployJava.runApplet(attributes, parameters, '1.6');
              },
              error: function(){ console.log("Error"); }
      });
    </script>
  </body>
</html>
```

Java Web Start Application

- web.xml

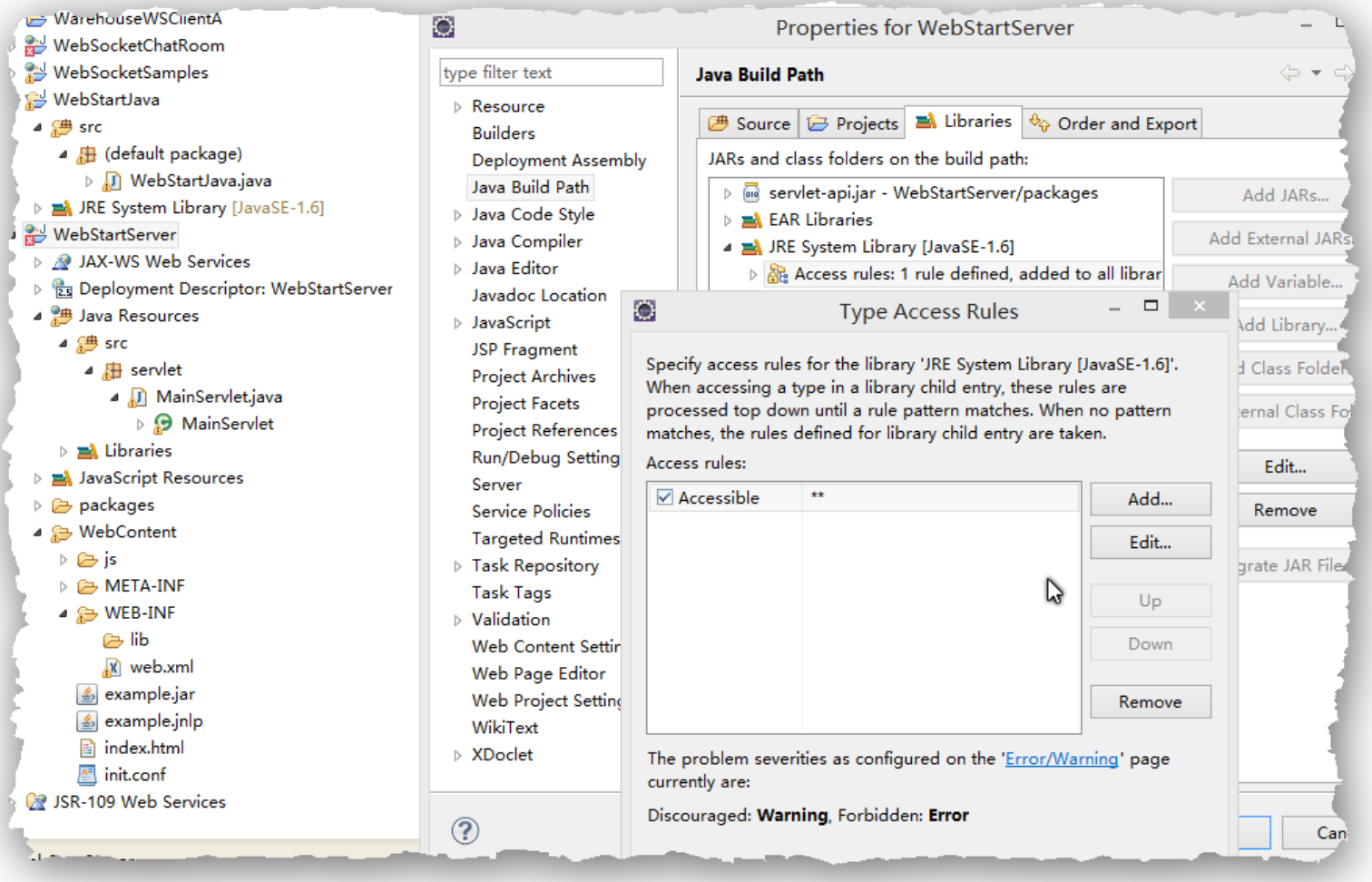
```
<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xmlns:web="http://xmlns.jcp.org/xml/ns/javaee"
         xsi:schemaLocation="http://xmlns.jcp.org/xml/ns/javaee
          http://java.sun.com/xml/ns/javaee/web-app_2_5.xsd">
  <servlet>
    <servlet-name>MainServlet</servlet-name>
    <servlet-class>servlet.MainServlet</servlet-class>
  </servlet>
  <servlet-mapping>
    <servlet-name>MainServlet</servlet-name>
    <url-pattern>/ajax/main</url-pattern>
  </servlet-mapping>
</web-app>
```

- init.conf

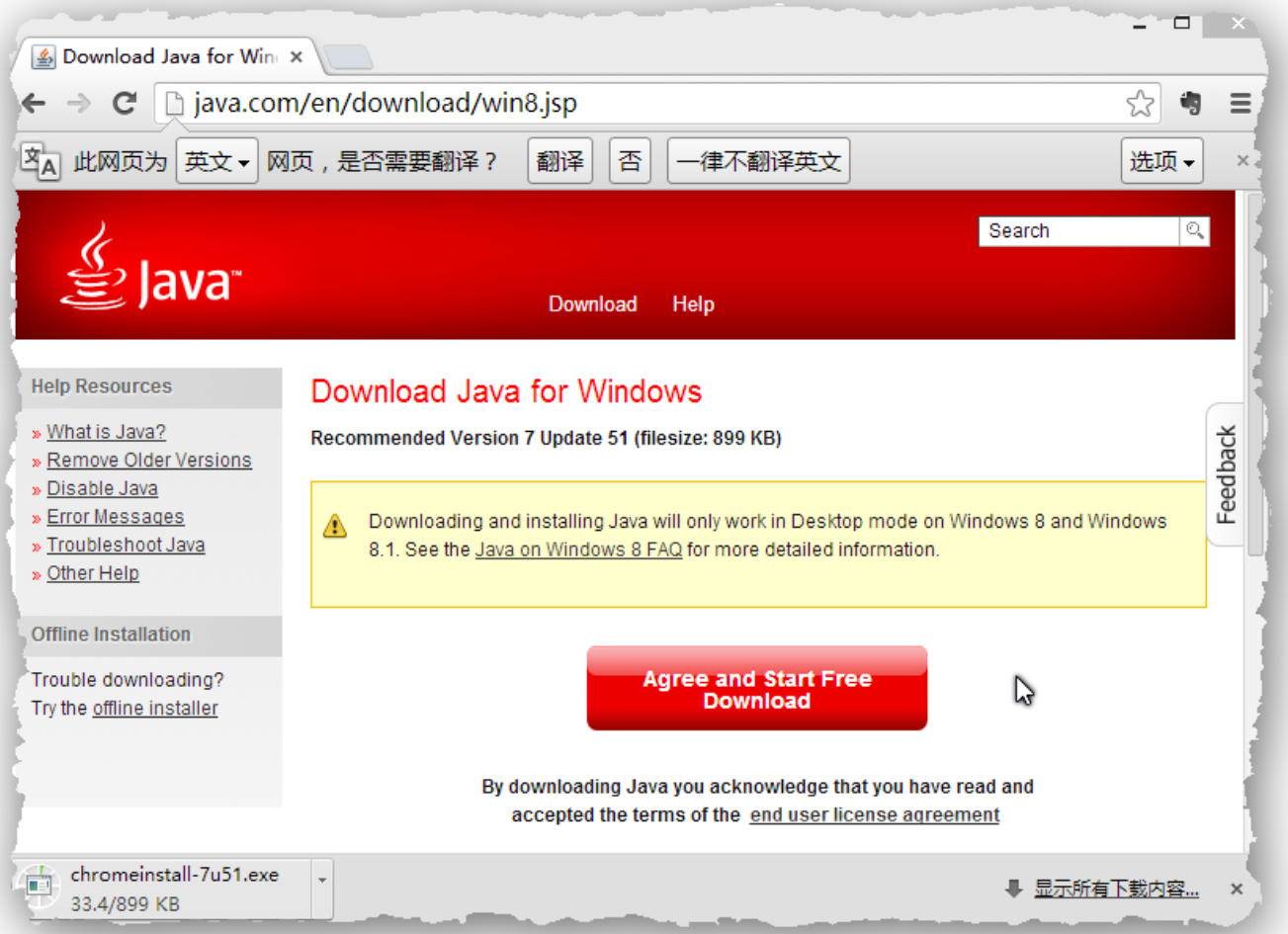
```
r=50
```

Java Web Start Application

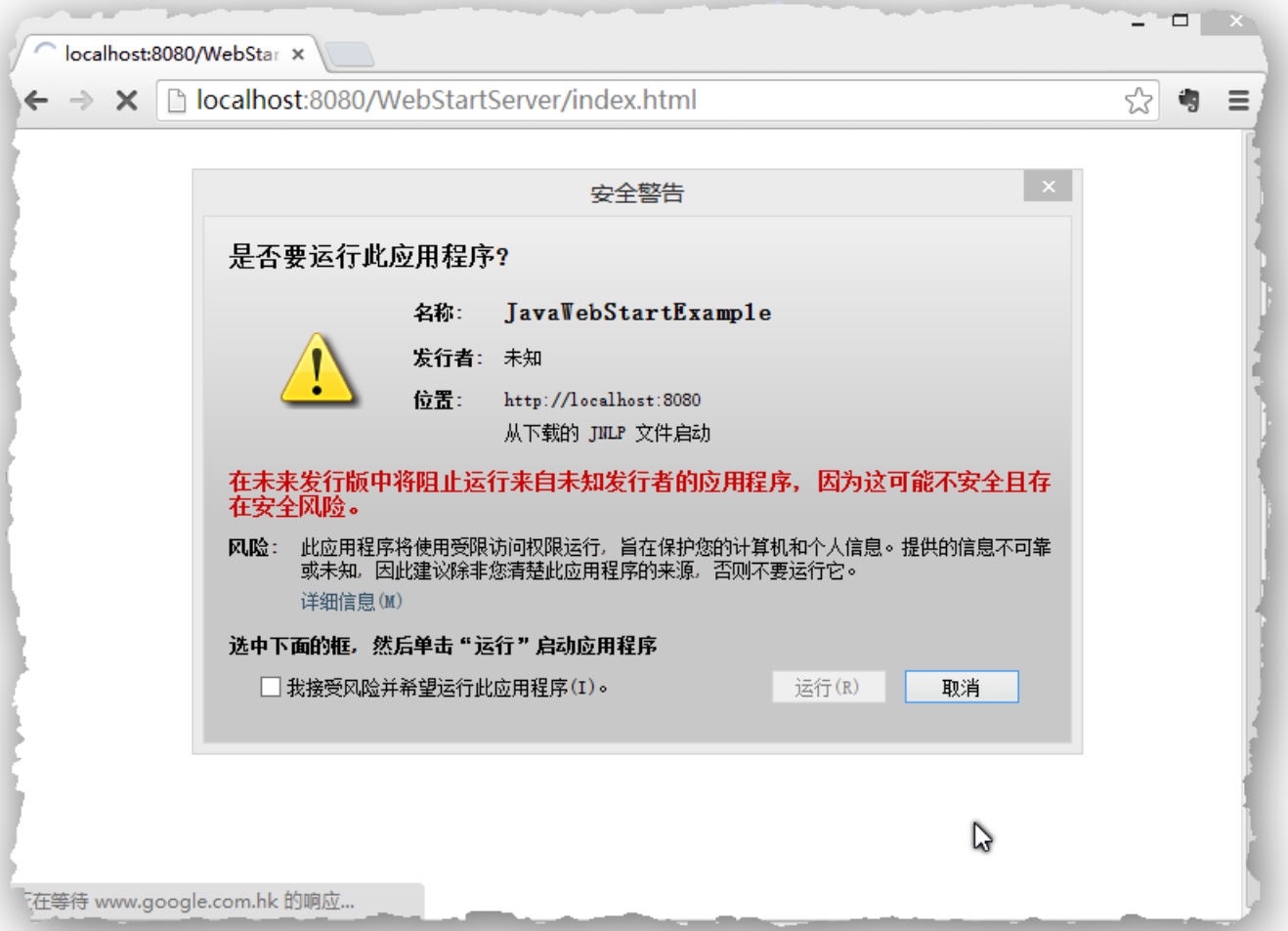
REliable, INtelligent & Scalable Systems



Java Web Start Application



Java Web Start Application



AVAILABILITY

- Availability is concerned with system failure and its associated consequences.
  - system failure occurs when the system no longer delivers a service consistent with its specification. Such a failure is observable by the system's users—either humans or other systems.

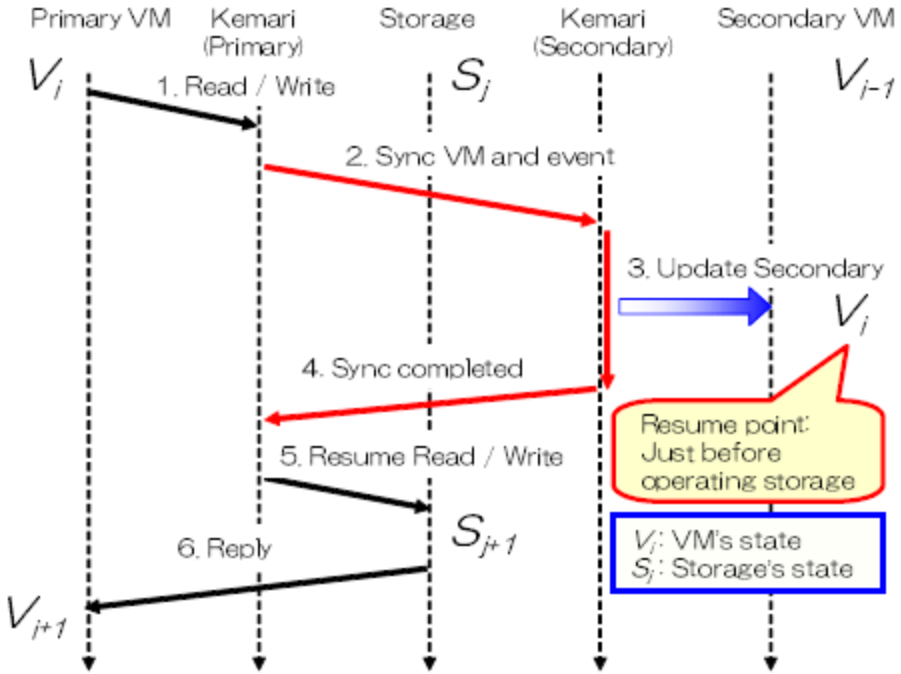
- Among the areas of concern are
  - how system failure is detected
  - how frequently system failure may occur
  - what happens when a failure occurs
  - how long a system is allowed to be out of operation
  - when failures may occur safely
  - how failures can be prevented
  - what kinds of notifications are required when a failure occurs

AVAILABILITY

- We need to differentiate between failures and faults.
  - A fault may become a failure if not corrected or masked. That is, a failure is observable by the system's user and a fault is not. When a fault does become observable, it becomes a failure.
  - For example, a fault can be choosing the wrong algorithm for a computation, resulting in a miscalculation that causes the system to fail.
- Once a system fails, an important related concept becomes the time it takes to repair it.
  - Since a system failure is observable by users, the time to repair is the time until the failure is no longer observable.
- The distinction between faults and failures allows discussion of automatic repair strategies.
  - That is, if code containing a fault is executed but the system is able to recover from the fault without it being observable, there is no failure.

AVAILABILITY

- For example：
  - Kemari
  - http://www.osrg.net/kemari/
  - Kemari: Virtual Machine Synchronization for Fault Tolerance
  - http://wiki.xensource.com/xenwiki/Open_Topics_For_Discussion?action=AttachFile&do=get&target=Kemari_08.pdf



AVAILABILITY

- Source of stimulus.
  - We differentiate between internal and external indications of faults or failure since the desired system response may be different.

- Stimulus. A fault of one of the following classes occurs.
  - omission. A component fails to respond to an input.
  - crash. The component repeatedly suffers omission faults.
  - timing. A component responds but the response is early or late.
  - response. A component responds with an incorrect value.

- Artifact.
  - This specifies the resource that is required to be highly available, such as a processor, communication channel, process, or storage.

- Environment.
  - The state of the system when the fault or failure occurs may also affect the desired system response. For example, if the system has already seen some faults and is operating in other than normal mode, it may be desirable to shut it down totally. However, if this is the first fault observed, some degradation of response time or function may be preferred.

AVAILABILITY

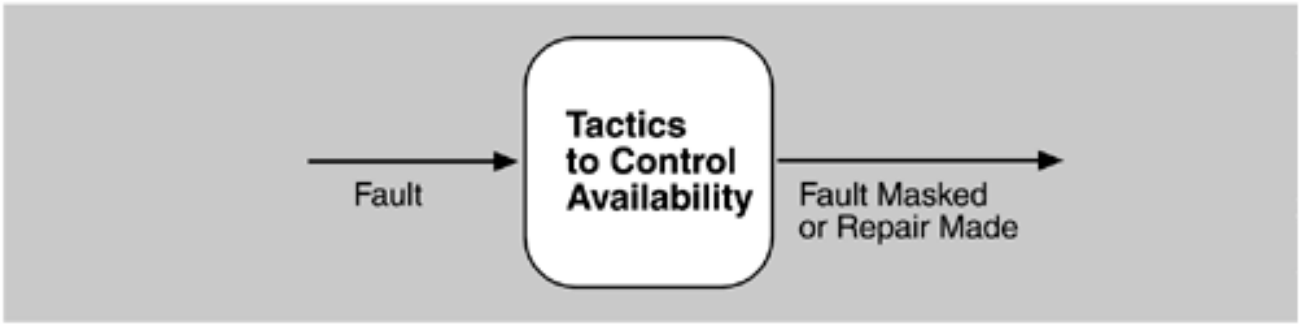
- Response.
  - There are a number of possible reactions to a system failure. These include
    - logging the failure
    - notifying selected users or other systems
    - switching to a degraded mode with either less capacity or less function
    - shutting down external systems
    - becoming unavailable during repair.

- Response measure.
  - The response measure can specify an availability percentage, or it can specify a time to repair, times during which the system must be available, or the duration for which the system must be available.

Availability Tactics

- A failure occurs when the system no longer delivers a service that is consistent with its specification; this failure is observable by the system's users.
- A fault (or combination of faults) has the potential to cause a failure.
- Recovery or repair is an important aspect of availability.
- The tactics we discuss in this section will keep faults from becoming failures or at least bound the effects of the fault and make repair possible.

Goal of availability tactics



Availability Tactics

- Many of the tactics we discuss are available within standard execution environments such as operating systems, application servers, and database management systems.
- It is still important to understand the tactics used so that the effects of using a particular one can be considered during design and evaluation.
- All approaches to maintaining availability involve some type of redundancy, some type of health monitoring to detect a failure, and some type of recovery when a failure is detected.
  - In some cases, the monitoring or recovery is automatic and in others it is manual.
- We first consider fault detection. We then consider fault recovery and finally, briefly, fault prevention.

Availability Tactics-fault detection

- Three widely used tactics for recognizing faults are ping/echo, heartbeat, and exceptions.
  - Ping/echo.
    - One component issues a ping and expects to receive back an echo, within a predefined time, from the component under scrutiny.
    - This can be used within a group of components mutually responsible for one task.
    - It can also used be used by clients to ensure that a server object and the communication path to the server are operating within the expected performance bounds.
    - "Ping/echo" fault detectors can be organized in a hierarchy, in which a lowest-level detector pings the software processes with which it shares a processor, and the higher-level fault detectors ping lower-level ones.
    - This uses less communications bandwidth than a remote fault detector that pings all processes.

Availability Tactics-fault detection

- Heartbeat (dead man timer).
  - In this case one component emits a heartbeat message periodically and another component listens for it.
  - If the heartbeat fails, the originating component is assumed to have failed and a fault correction component is notified.
  - The heartbeat can also carry data. For example, an automated teller machine can periodically send the log of the last transaction to a server. This message not only acts as a heartbeat but also carries data to be processed.
- Exceptions.
  - One method for recognizing faults is to encounter an exception, which is raised when one of the fault classes is recognized.
  - The exception handler typically executes in the same process that introduced the exception.

- The ping/echo and heartbeat tactics operate among distinct processes, and the exception tactic operates within a single process.

- The exception handler will usually perform a semantic transformation of the fault into a form that can be processed.

Availability Tactics-fault detection

- Suppose we want to add fault detection of DBMS into e-Book in order to detect the connectivity error of DBMS
  - Now that the workload of DB server is high, the influence of added fault detection on performance shall be as light as possible.
  - Meanwhile, since connectivity error is a serious error, we hope we can detect it as soon as possible.

- Consequently, heartbeat is chosen as the tactic for fault detection.
  - To add a service in the system which creates connection to DBMS periodically and sends the result of operation to other components.

Availability Tactics-fault detection

- In an embedded safety-critical system: aircraft

- Self check
  - Power-on self check
  - Periodical self check
  - Manual self check
  - Command triggered self check

  - To display the error code with LED

Availability Tactics-fault recovery

- Fault recovery consists of preparing for recovery and making the system repair. Some preparation and repair tactics follow.

- Active redundancy (hot restart).
  - All redundant components respond to events in parallel. Consequently, they are all in the same state.
  - The response from only one component is used (usually the first to respond), and the rest are discarded.
  - When a fault occurs, the downtime of systems using this tactic is usually milliseconds since the backup is current and the only time to recover is the switching time.
  - Active redundancy is often used in a client/server configuration, such as database management systems, where quick responses are necessary even when a fault occurs. In a highly available distributed system, the redundancy may be in the communication paths.

Availability Tactics-fault recovery

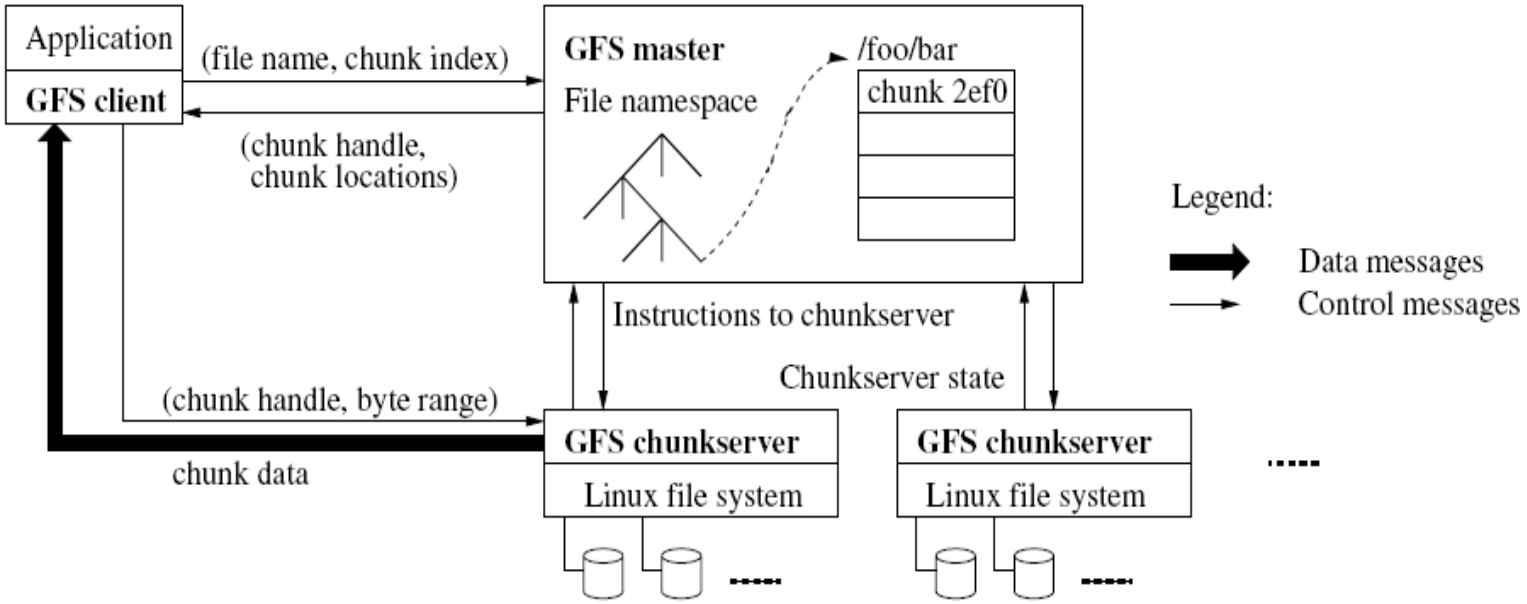
- Passive redundancy (warm restart/dual redundancy/triple redundancy).
  - One component (the primary) responds to events and informs the other components (the standbys) of state updates they must make. When a fault occurs, the system must first ensure that the backup state is sufficiently fresh before resuming services.
  - This approach is also used in control systems, often when the inputs come over communication channels or from sensors and have to be switched from the primary to the backup on failure.
  - This tactic depends on the standby components taking over reliably. Forcing switchovers periodically—for example, once a day or once a week—increases the availability of the system.
  - Synchronization is the responsibility of the primary component, which may use atomic broadcasts to the secondaries to guarantee synchronization.

Availability Tactics-fault recovery

- Spare.
  - A standby spare computing platform is configured to replace many different failed components.
  - It must be rebooted to the appropriate software configuration and have its state initialized when a failure occurs.
  - Making a checkpoint of the system state to a persistent device periodically and logging all state changes to a persistent device allows for the spare to be set to the appropriate state.
  - This is often used as the standby client workstation, where the user can move when a failure occurs.
  - The downtime for this tactic is usually minutes.

Availability Tactics-fault recovery

- GFS



Availability Tactics-fault recovery

- Suppose we want to add backup mechanism to the DB in order to replace main server with backup server when the former has some faults.
  - Now that e-Book is not a critical system, its availability is not necessary to be very high, we allow some sessions of clients to be lost.
  - Meanwhile, once a user generates new data, it should be persistently stored into DB

- Consequently, we use passive redundancy as the backup mechanism of e-Book

Availability Tactics-fault recovery

- In an embedded safety-critical system: aircraft

- Redundancy
  - Storage
  - Binary code
  - Communication linkage

Availability Tactics-fault repair

- There are tactics for repair that rely on component reintroduction.
  - When a redundant component fails, it may be reintroduced after it has been corrected. Such tactics are shadow operation, state resynchronization, and rollback.

- Shadow operation.
  - A previously failed component may be run in "shadow mode" for a short time to make sure that it mimics the behavior of the working components before restoring it to service.
- Checkpoint/rollback.
  - A checkpoint is a recording of a consistent state created either periodically or in response to specific events.
  - Sometimes a system fails in an unusual manner, with a detectably inconsistent state. In this case, the system should be restored using a previous checkpoint of a consistent state and a log of the transactions that occurred since the snapshot was taken.

Availability Tactics-fault repair

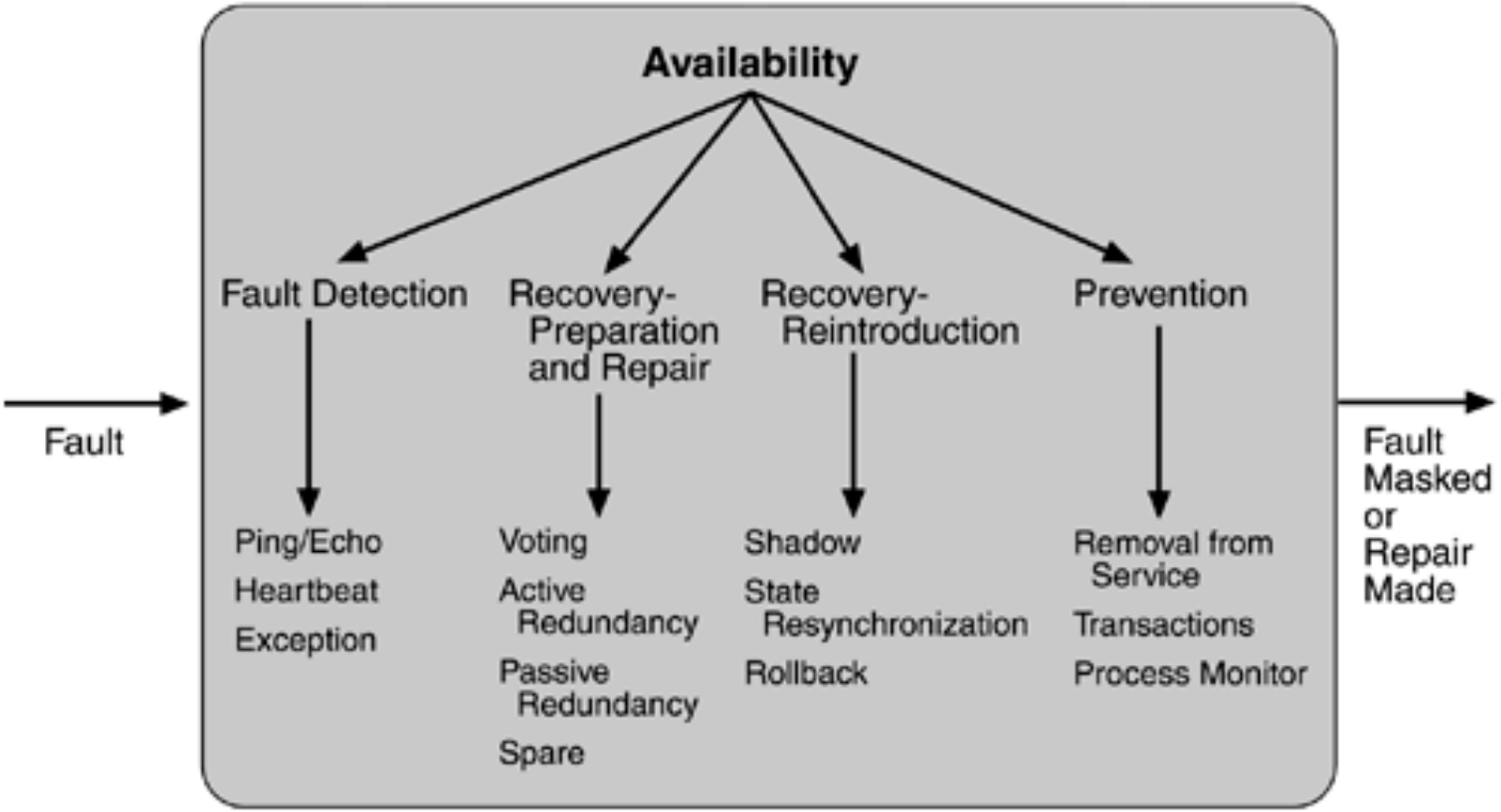
- For e-Book,
  - we adopt checkpoints to do fault repair in order to keep the performance of system.

- For instruments of aircraft
  - Watchdog
  - Software interrupts
  - Pluggable components

Availability Tactics-fault prevention

- The following are some fault prevention tactics.
- Removal from service.
  - This tactic removes a component of the system from operation to undergo some activities to prevent anticipated failures.
  - One example is rebooting a component to prevent memory leaks from causing a failure.
  - If this removal from service is automatic, an architectural strategy can be designed to support it. If it is manual, the system must be designed to support it.
- Transactions.
  - A transaction is the bundling of several sequential steps such that the entire bundle can be undone at once.
  - Transactions are used to prevent any data from being affected if one step in a process fails and also to prevent collisions among several simultaneous threads accessing the same data.
- Process monitor.
  - Once a fault in a process has been detected, a monitoring process can delete the nonperforming process and create a new instance of it, initialized to some appropriate state as in the spare tactic.

Availability Tactics-Summary



Project

- 2nd Iteration Requirement
  - Applying transaction control on order processing
  - Utilizing JMS(MDB) to process orders.
  - Utilizing Ajax to implement book browsing page. When user clicks on a book, the details of this book will be retrieved from server with Ajax and displayed on this page.
  - Utilizing WebSocket to implement an on-line chat room for customer group.
  - JSON processing in Browser and Server. In the ajax-based book list page, the book details should be transferred as JSONs.
- Deadline 5.1 0:00

References

- Java Web Start
  - http://docs.oracle.com/javase/tutorial/deployment/webstart/index.html
- The Java EE 7 Tutorial
  - http://docs.oracle.com/javaee/7/tutorial/doc/javaeetutorial7.pdf
- Software Architecture in Practice, Second Edition
  - By Len Bass, Paul Clements, Rick Kazman
  - Publisher : Addison Wesley

Thank You!