Technológie pre monitorovacie systémy

Andrej Lúčny

MicroStep-MIS

andy@microstep-mis.com

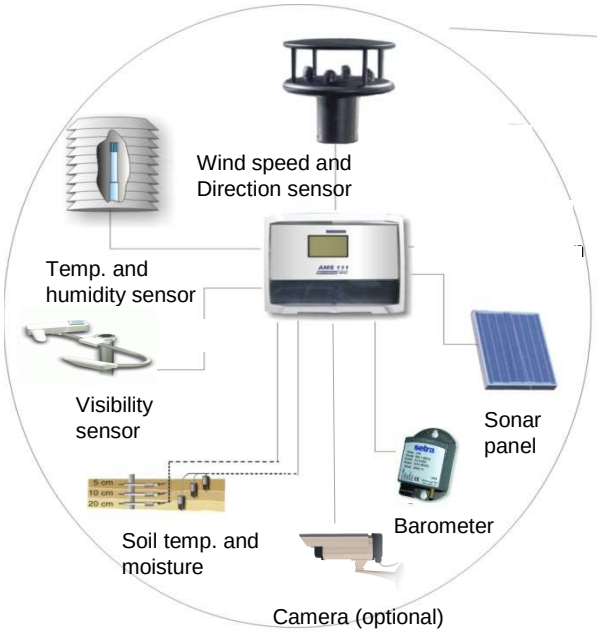
http://www.microstep-mis.com/~andy

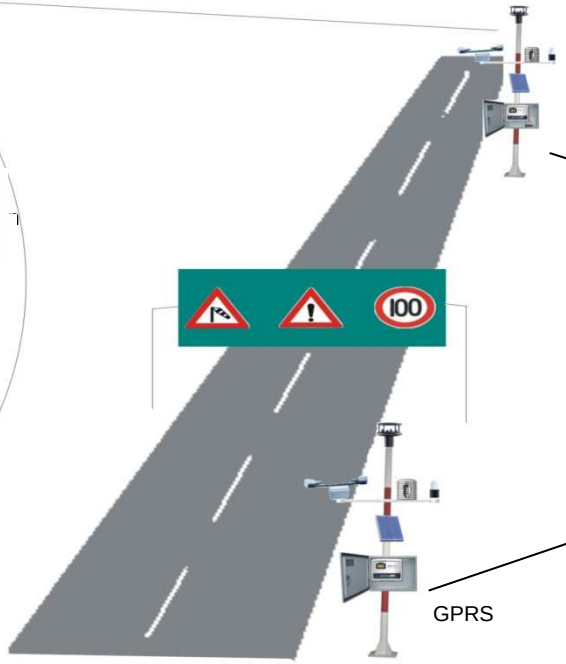
Typické produkty MicroStep - MIS

Monitorovacie a informačné systémy

(meteorológia, letecká doprava, seizmológia, krízový management)

Príklad:





GPRS

GPRS

Existing 5 x AWS
And 1 x buoy

UDCS/Database

Real time display

Model

Systém na tvorbu regionálnej predpovede hmly

Špecifické črty monitorovacích systémov

- Permanent operation
- Real-time operation
- Automatic operation
- Critical mission
- Complexity
- Ability to configure
- Incremental development

Technológie

Multiplatformnosť: Microsoft Windows ® or Linux ®.

Jazyky: Java, XML

Relačné SQL databázy:

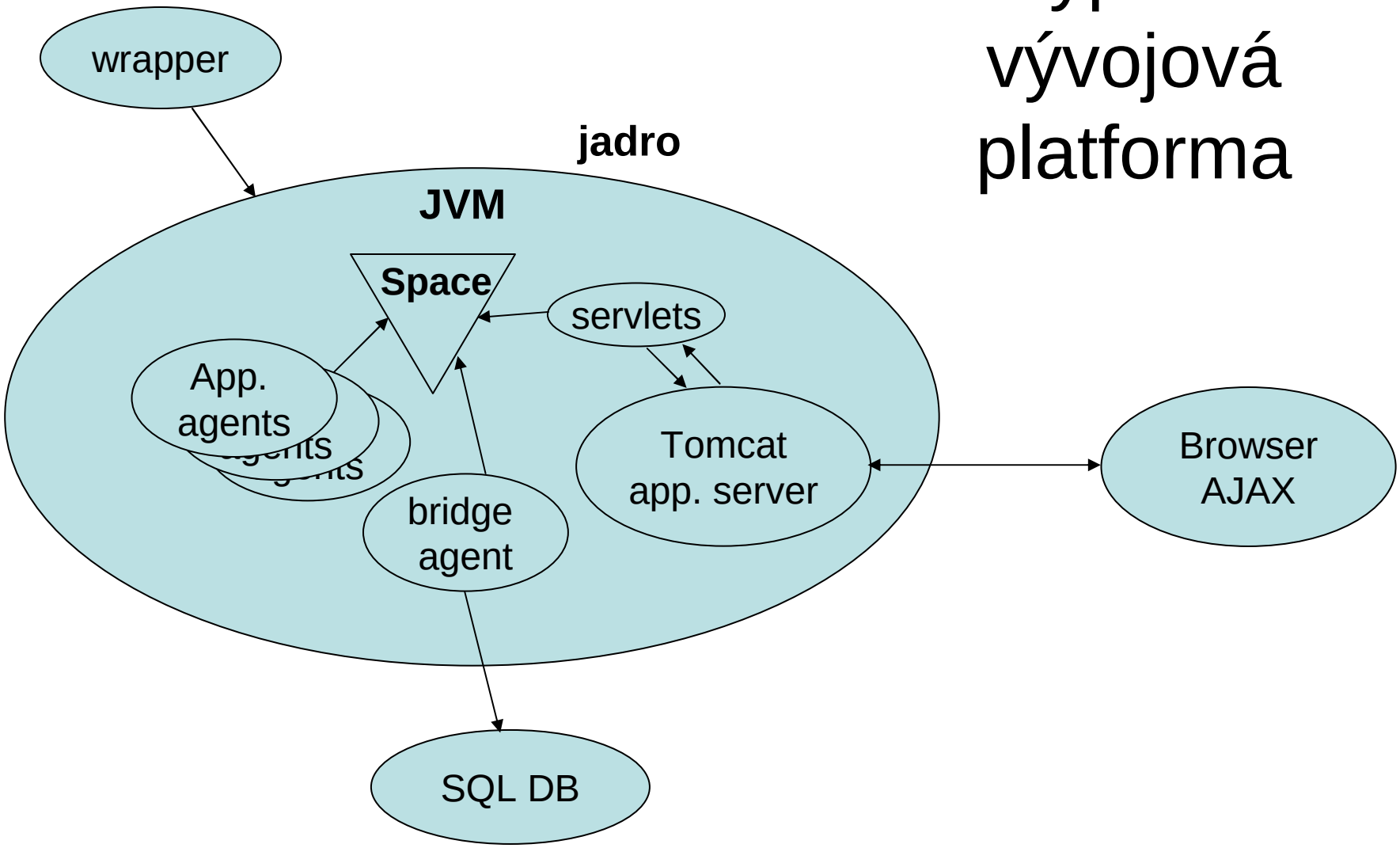
- PostgreSQL
- ORACLE

Spracovanie dát založené na XML
Archivácia dát založená na XML2SQL bridge.

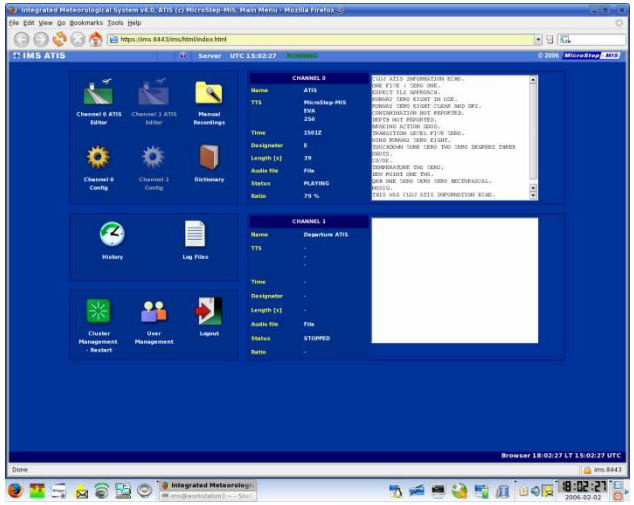
Konfigurácia systému založená na XML

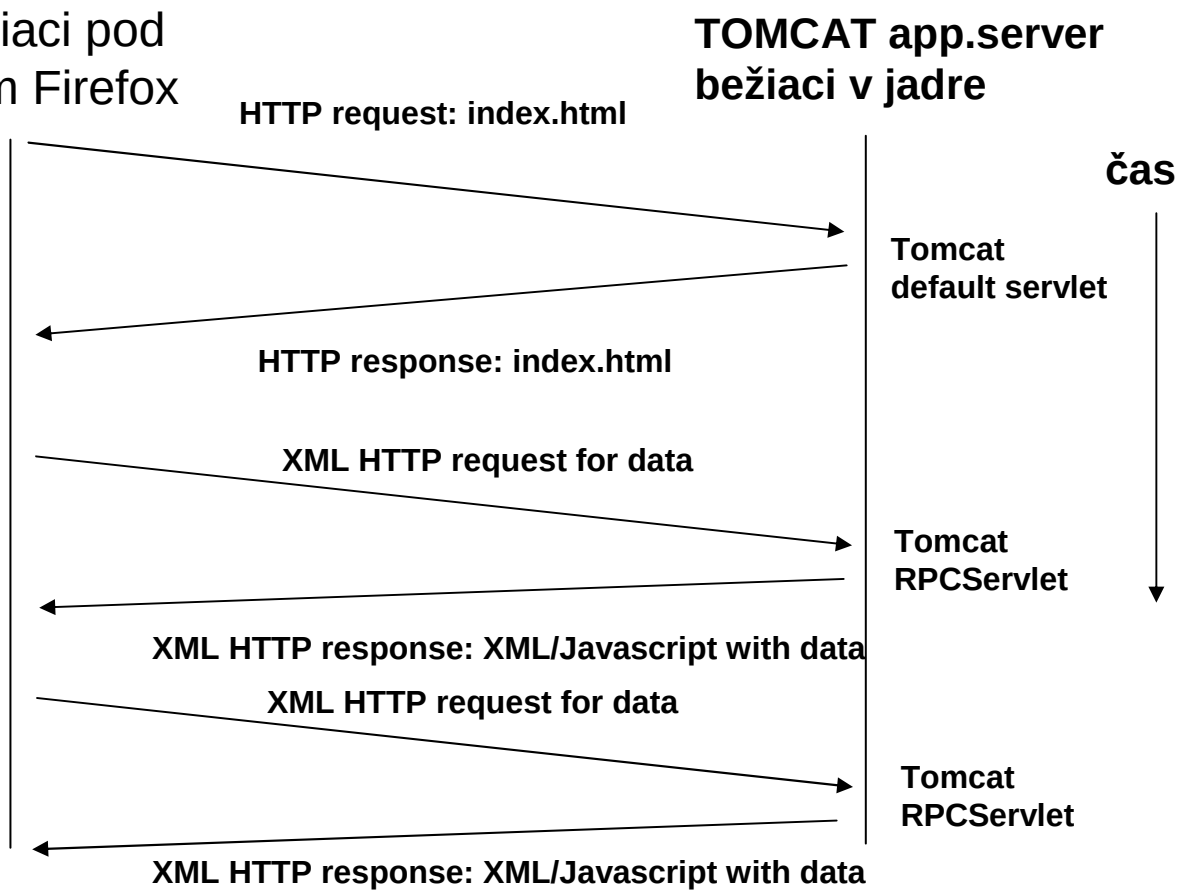
Užívateľské rozhranie založené na aplikačnom serveri:
Webové aplikácie (Tomcat, AJAX)

Typická vývojová platforma

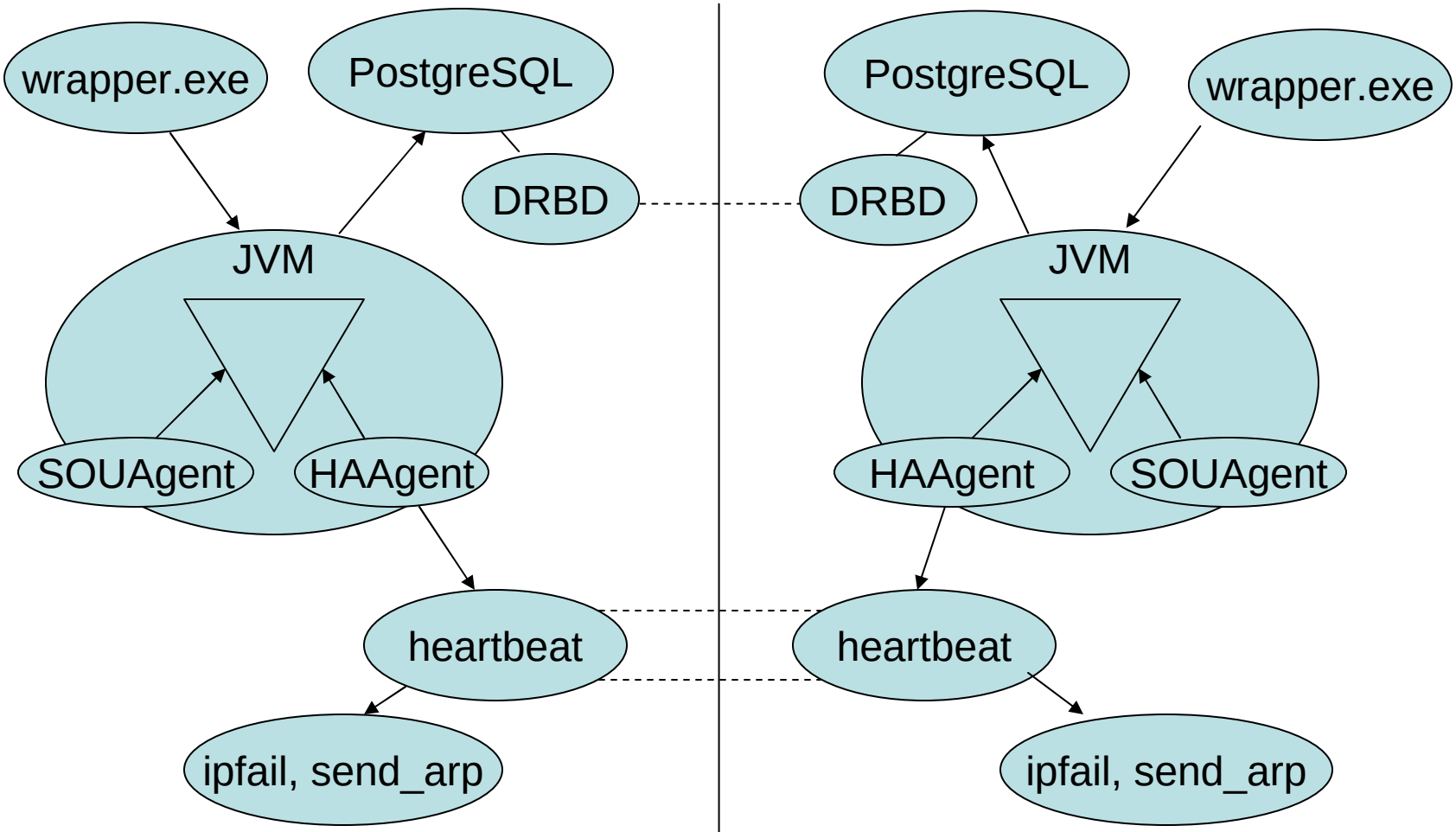


Front-end applications





High Availability Management

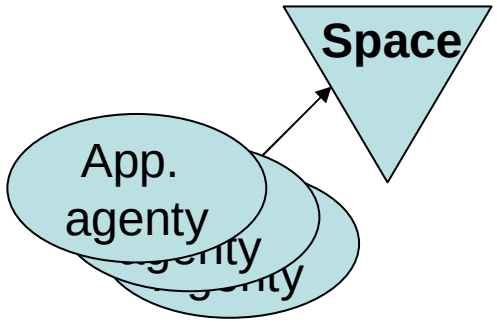


Architektúra Agent-Space

Agent: podtrieda com.microstepmis.agentspace.Agent:

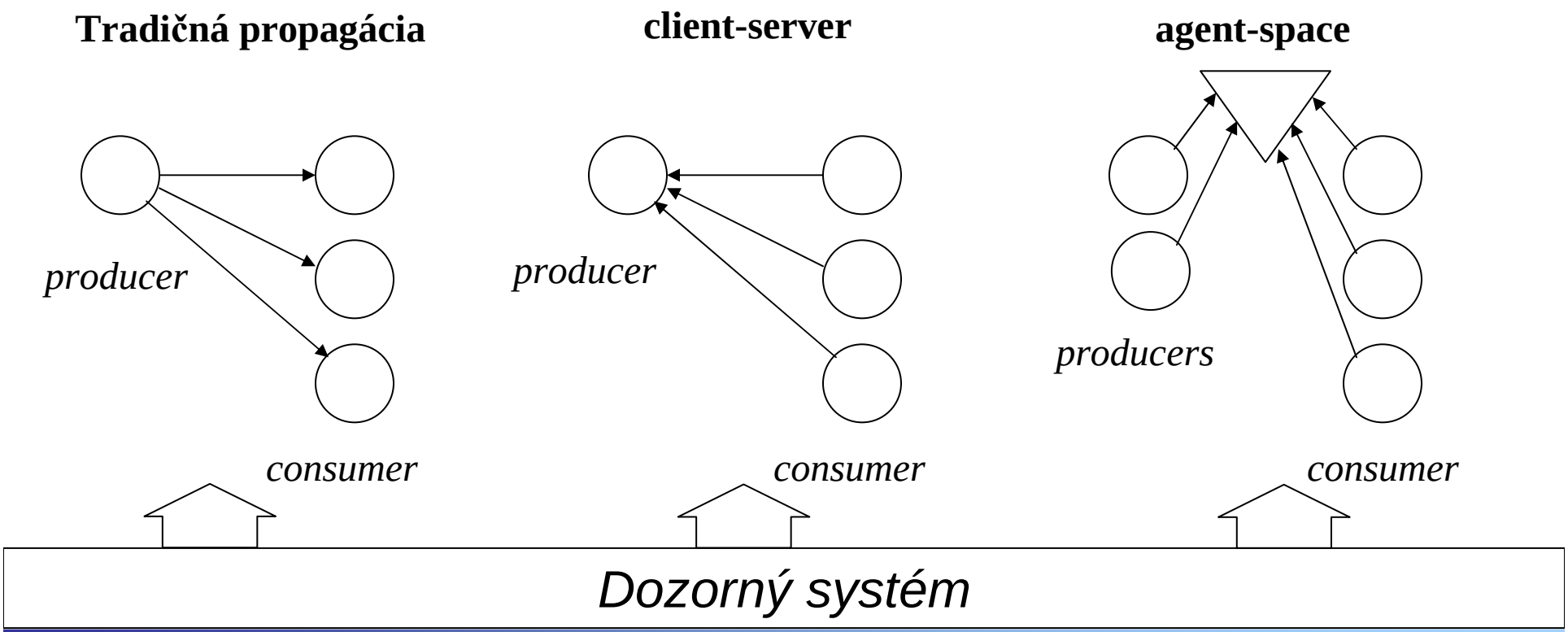
```
public void init(String[] args) {
        Proxy p1 = attachTrigger(“BLOK1”, Trigger.NORMAL );
        Proxy p2 = attachTrigger(“BLOK2”, Trigger.NORMAL );
}

public void senseSelectAct(Proxy proxy) {
        if( proxy.equals( p1 ) ) {
                // akcie pre BLOK1
                Object b1 = read( “BLOK1” );
                …
        } else if(proxy.equals( p2 ) ) {
                // akcie pre BLOK2
                Object b2 = read( “BLOK2” );
                …
                synchronized( space ) {
                        write(…);
                        write(…);
                }
        }
}
```



„Soft-crash landing“

- Automatické zotavenie sa z chýb



WatchDog-y

- Pokiaľ niečo nevieme automaticky zotaviť z chyby je potrebné to restartnúť
- Dozorný systém aplikácie
- Java wrapper
- Reboot počítača
- Vypnutie a zapnutie prúdu

Java/JavaScript Object – XML Mapping

```
<x xclass="com.microstepmis.xplatform.demo.TestCfg">
  <stations>
    <x id="st1" cccc="LZIB" longitude="40.2" latitude="52.2">
      <station id="st1_1" />
          </x>
    <x id="st2" cccc="LZPP" longitude="41.5" latitude="52.9">
      <station id="st1_2" />
    </x>
  </stations>
</x>
```

```
public class TestCfg {
  public ArrayList<Station> stations =
    new ArrayList<Station>();

  public TestCfg() {
}
```

```
static public class Station {
  public String id;
  public String cccc;

  @XCfg(label="Longitude",
    labelHint="Longitude in degrees,
    either number from <-180, 180)
    (negative for west) or in format
    'degrees minutes(W|E)', e.g. '140
    30.3W', '3 10E'",nullAllowed=true,
    restriction=XCfg.Restriction
    (minInclusive=-180,
     maxExclusive=180),
    display="longitude")
  public Double longitude;
  public Double latitude;
  public Double altitude;
}
```

MicroStep - MIS

Monitoring and Information Systems

Čavojského 1
841 04 Bratislava
Slovakia

Tel./Fax: +421 2 602 00 111/180
info@microstep-mis.com
www.microstep-mis.com

prestávka

Anotácie a mapovanie objektov za účelom perzistencie, konfigurácie a distribúcie

Pri súčasnom hardware vývojár zápasí s:

- **perzistenciou (trvácnosťou) informácie**
  - informácia je aktívne len v pamäti počítača, ale tam nie je perzistentná. Perzistentnou sa stáva na zariadeniach ako je pevný disk, a tam nie je aktívna. Súčasné operačné systémy nevedia zastrieť tento rozdiel a preto súčasné programovanie – aj vzhľadom na rôznu informačnú kapacitu externých zariadení a pamäte – pozostáva zo značnej časti z prehadzovania informácie medzi pamäťou a externými zariadeniami, s príslušnou zmenou formátu informácie.
- **konfigurovateľnosťou systému**
  - vzhľadom na customizáciu systémov, musíme vedieť systém vytvoriť tak, že beží berúc do úvahu množstvo parametrov, rádovo ich môžu byť tisíce
- **distribúciou informácií**
  - často, ba spravidla, je užívateľ pri inom počítači, než kde sú informácie, ktoré potrebuje

Tradičné riešenie

- Samozrejme, všetky tieto problémy ide riešiť jeden po druhom, každý špecifickým spôsobom, napríklad:
  - Perzistenciu natívnym marshallingom java.io.Serializable
  - Konfigurovateľnosť knižnicou na konfiguračné súbory java.util.properties
  - Distribúciu komunikačnou knižnicou a potenciálne iným marshallingom java.rmi + IIOP

Serializovateľný objekt

```
import java.io.*;

public class Gulka implements Serializable {

  public static final long serialVersionUID = 112132444L;

  float radius;
  public float x;
  public float y;

  public Gulka (float radius) {
    this.radius = radius;
  }

  public String toString () {
    return radius+"["+x+","+y+"]";
  }

}
```

Marshalling

```
import java.io.*;

public class Marshall {

  public static void main (String[] args) throws Exception {
    Gulka g = new Gulka(1.0f);
    g.x = 0.0f; g.y = 0.0f;
    ByteArrayOutputStream byteStream = new ByteArrayOutputStream();
    ObjectOutputStream marshaller = new ObjectOutputStream(byteStream);
    marshaller.writeObject(g);
    byte[] bytes = byteStream.toByteArray();
    System.out.println(bytes.length);
    for (int i=0; i<bytes.length; i++) System.out.print(bytes[i]+" ");
    System.out.println();
  }

}
```

Demarshalling

```
import java.io.*;

public class Demarshall {

  public static void main (String[] args) throws Exception {
    byte[] marshalledObject = {
      -84, -19, 0, 5, 115, 114, 0, 5, 71, 117, 108, 107, 97,
      0, 0, 0, 0, 6, -81, 1, 92, 2, 0, 3, 70, 0, 6, 114, 97,
      100, 105, 117, 115, 70, 0, 1, 120, 70, 0, 1, 121, 120,
      112, 63, -128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0
    };
    ByteArrayInputStream byteStream = new
   ByteArrayInputStream(marshalledObject);
    ObjectInputStream demarshaller = new
   ObjectInputStream(byteStream);
    Gulka g = (Gulka) demarshaller.readObject();
    System.out.println(g); // vypise 1.0[0.0,0.0]
  }

}
```

Reflekčný model

```
import java.lang.reflect.*;

public class RF {
  public static void main (String[] args) throws Exception {
    Gulka g = new Gulka(1.0f);
    Class cl = g.getClass();
    Field[] field = cl.getDeclaredFields();
    for (int i=0; i<field.length; i++) {
      Object obj = field[i].get(g);
      System.out.println(field[i].getName()+" = "+obj);
    }
  }
}
```

```
serialVersionUID = 112132444
radius = 1.0
x = 0.0
y = 0.0
```

Perzistencia

```
import java.io.*;
public class Perzistencia {

   public static void main (String[] args) throws Exception {
        Gulka g = new Gulka(1.0f);
        g.x = 0.0f; g.y = 0.0f;

        FileOutputStream out = new FileOutputStream(new File("g.obj"));
        ObjectOutputStream marshaller = new ObjectOutputStream(out);
        marshaller.writeObject(g);
        marshaller.close();

        FileInputStream in = new FileInputStream(new File("g.obj"));
        ObjectInputStream demarshaller = new ObjectInputStream(in);
        Gulka g2 = (Gulka) demarshaller.readObject();
        marshaller.close();

        System.out.println(g2); // vypise 1.0[0.0,0.0]
   }
}
```

Konfigurovateľnosť

```
#Gulka
#Mon Nov 08 06:40:10 CET 2010
radius=1.0
```

```
import java.util.*; import java.io.*;
public class Konfiguracia {
   public static void main (String[] args) throws Exception {
        Gulka g = new Gulka(1.0f);

        Properties properties = new Properties();
        properties.setProperty("radius",Float.toString(g.radius));
        FileOutputStream out = new FileOutputStream(new File("g.cfg"));
        properties.store(out,"Gulka");
        out.close();

        Properties properties2 = new Properties();
        FileInputStream in = new FileInputStream(new File("g.cfg"));
        properties2.load(in);
        in.close();
        String radiusString = properties2.getProperty("radius");
        System.out.println(Float.parseFloat(radiusString));
        // vypise 1.0
   }
}
```

Distribúcia

```
import java.io.*;
public class Distribucia { // client

  public static void main (String[] args) throws Exception {
    Gulka g = new Gulka(1.0f);
    InetAddress addr = InetAddress.getByName(„192.168.145.11");
    Socket socket = new Socket(addr,1234 /*port*/);
    out = new DataOutputStream(socket.getOutputStream());
    ObjectOutputStream marshaller = new ObjectOutputStream(out);
    marshaller.writeObject(g);
    marshaller.close();
}

// Server
// …
```

Iné formy: javax.rmi

Jednotný marshalling

- Je fakt, že by vo všetkých prípadoch ide o nejaký marshalling
- Tak prečo nepoužiť všade rovnaký ?
- Sun takúto filozofiu presadzuje a využíva natívny marshalling Javy, t.j. binárny formát

-84, -19, 0, 5, 115, 114, 0, 5, 71, 117, 108, 107, 97, 0, 0, 0, 0, 6, -81, 1, 92, 2, 0, 3, 70, 0, 6, 114, 97, 100, 105, 117, 115, 70, 0, 1, 120, 70, 0, 1, 121, 120, 112, 63, -128, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0

Alternatíva

- Binárna forma má jednu nevýhodu – užívateľ ju nemôže prečítať ani zmodifikovať bez aplikačne orientovaných prostriedkov
- Ako alternatíva k natívnemu marshallingu sa ponúka XML

XML

<tag attribute=value … > data </tag>	**element s dátami**
<tag attribute=value ... />	**prázdny element**
<? … ?>	**direktíva**
<!-- remark -->	**poznámka**
<![CDATA[…]]>	**raw data**
<ns:tag>	**namespace**

XML ako forma konfigurovateľnosti

```
public static void main(String[] args) {
  Document doc = parseXmlFile("g2.xml", false);
  NodeList list = doc.getElementsByTagName("gulka");
  for (int i=0; i<list.getLength(); i++) {
    Element element = (Element)list.item(i);
    if (element.getTagName().equals("gulka")) {
      NodeList chlist = element.getChildNodes();
      for (int j=0; j<chlist.getLength(); j++) {
        Node childNode = chlist.item(j);
        if (childNode.getNodeName().equals("radius")) {
          … radius = Float.parseFloat(childNode.getNodeValue); …
        }
      }
    }
  }
  writeXmlFile(doc, "g3.xml");
}
```

```
<gulka>
   <radius>1</radius>
</gulka>
```

(semištruktúrovaná forma)

```
<gulka radius="1"/>
```

(plneštruktúrovaná forma)

XML ako forma persistencie

```
public static void main(String[] args) {
  Document doc = parseXmlFile("g2.xml", false);
  NodeList list = doc.getElementsByTagName("gulka");
  for (int i=0; i<list.getLength(); i++) {
    Element element = (Element)list.item(i);
    if (element.getTagName().equals("gulka")) {
      NodeList chlist = element.getChildNodes();
      for (int j=0; j<chlist.getLength(); j++) {
        Node childNode = chlist.item(j);
        if (childNode.getNodeName().equals("radius")) {
          … childNode.getNodeValue(); … childNode.setNodeValue(“2"); …
        }
      }
    }
  }
  writeXmlFile(doc, "g3.xml");
}
```

```
<gulka>
  <radius>1</radius>
  <x>1</x>
  <y>1</y>
</gulka>
```

XML ako forma distribúcie

- SOAP, RDF, WSDL

```
String endpoint =
  "http://semcows.microstep-mis.com:8443/axis/DigitalElevationModel.jws";
Service  service = new Service();
Call call = (Call) service.createCall();
QName qn1 = new QName( "http://model.microstepmis.com","Area" );
call.registerTypeMapping(Area.class, qn1,
  new org.apache.axis.encoding.ser.BeanSerializerFactory(Area.class,qn1),
  new org.apache.axis.encoding.ser.BeanDeserializerFactory(Area.class,qn1)
);
call.setTargetEndpointAddress( new java.net.URL(endpoint) );
call.setOperationName(new QName("http://soapinterop.org/", "getDEMData"));
Object[] params=new Object[1];
params[0]= new Area( 8D, 31D, 40D, 55D );
String retstr = (String) call.invoke( params );
System.out.println("returned URL:"+retstr);
DEMData ret = (DEMData) PackUtil.unzipObjectFromURL(retstr);
```

Mapovanie XML na Objekty - JAXB

XML Schema

```
<xsd:complexType name="Address">
 <xsd:sequence>
   <xsd:element name="name" type="xsd:string"/>
   <xsd:element name="street" type="xsd:string"/>
</xsd:sequence>
</xsd:complexType>
```

XML

```
<Address>
   <name> Fero Frtala </name>
   <street> Sturova ul.
</street>
</Address>
```

instance of

compilation

validation

processing

```
public  interface Address {
    String getStreet();
    void setStreet(String value);
    String getName();
    void setName(String value);
 }
```

Java interface (.po)

```
JAXBContext j  = JAXBContext.newInstance( "interface.po" );
Unmarshaller u = j.CreateUnmarshaller();
Address po =  (Address) u.unmarshal(new
                    FileInputStream("xml.xml"));
System.out.println( po.getName() + " " + po.getStreet()  );
```

Java code

Mapovanie objektov na XML

- Proprietárne formy, napr. X2O

```
import com.microstepmis.xplatform.*;
import java.io.*;

public class Mapping {
   public static void main (String[] args) throws Exception {
        Gulka g = new Gulka(1.0f);
        X2O.mapToXML(new File("g.xml"),g);
        Gulka g2 = (Gulka) X2O.mapFromXML(new File("g.xml"));
        System.out.println(g2.radius); // vypise 1.0
   }
}
```

```
<xplatform radius="1.0" x="0.0" xclass="Gulka" y="0.0"/>
```

Ďalšie manipulácie

- Zatiaľ sme pri manipulácii s objektom vystačili s reflekčným modelom (napr. typ atribútu)
- Čo však keď v definícii objektu nemáme potrebné údaje, (napr. minimálnu a maximálnu hodnotu potrebnú napr pre XML editor, ktorým konfigurujeme systém) ?

Potreba anotácií

```
@XCfg(label="Contact data configuration", labelFunc="(isNull('label')
   ? (isNull('value') ? '' : get('value')) : (isNull('value') ?
   get('label') : get('label') + ': ' + get('value')) )")

public class Contact {
   @XCfg(order=1, label="Label", restriction = @XCfg.Restriction(
   maxLength=30))
   public String label;

   @XCfg(order=2, label="Value", restriction = @XCfg.Restriction(
   maxLength=60))
   public String value;

   @XCfg(order=3, label="Type",
                restriction = @XCfg.Restriction( enumeration={
                @XCfg.Enum(key="text", value="Text"),
                @XCfg.Enum(key="URL", value="URL"),
                @XCfg.Enum(key="e-mail", value="e-mail")
   }) )
   public String type = "text";
}
```

Túto informáciu vieme pridať pomocou tzv. anotácii.

Všeobecnosť a znovuvyužiteľnosť

- Pomocou anotácii, je možné obohacovať definíciu objektu tak, aby sa dala všetka potrebná manipulácia s ním vykonať všeobecnými – aplikačne nezávislými nástrojmi

Je dobré všetko robiť všeobecne?

- Je to dobré z hľadiska znovu-použiteľnosti
- Nie je to dobré z hľadiska priateľskosti systému k užívateľovi
- Preto je dobré použiť túto stratégiu na činnosti, ktoré nevykonáva užívateľ (napr. automatické testy), alebo ich vykonáva zriedkavo (konfigurácia), ale nie ako hlavné užívateľské rozhranie

Ďakujem za pozornosť !

MicroStep - MIS

Monitoring and Information Systems

Čavojského 1
841 04 Bratislava
Slovakia

Tel./Fax: +421 2 602 00 111/180
info@microstep-mis.com
www.microstep-mis.com