APLICATII
PENTRU STRUCTURI GRID

APPLICATIONS
FOR GRID SYSTEMS

CAP. 1. Service Oriented Architecture (SOA)

Arhitectura orientată pe servicii

Acest capitol introduce arhitectura orientată pe servicii (SOA), ca fiind tehnologia ce stă la baza serviciilor grid, folosindu-se în primul rând de serviciile Web.

Pe lângă servicii Web, sunt prezentate și alte specificații complementare precum: **Web Service Description Language (WSDL),** pentru a descrie conținutul și utilizarea serviciilor Web; standardul **Simple Object Access Protocol (SOAP),** fiind protocolul folosit la schimbul de mesaje dintre serviciile Web; și specificația **Universal Description, Discovery and Integration (UDDI),** ce permite localizarea și publicarea serviciilor.

Descrierea conceptelor de bază și a elementelor lor principale este urmată de câteva exemple practice.

1.1 Ce este SOA?

Arhitectura orientată pe servicii (SOA) este o abordare arhitecturală prin care o aplicație este formată din componente *independete*, *distribuite* și care *interacționează* între ele, numite **servicii.**

O colecție de astfel de servicii compun o aplicație. Serviciile pot fi distribuite în interiorul sau în exteriorul limitelor fizice ale unei organizații, respectiv ale domeniilor de securitate. Mai mult decât atât, diverse componente ale unui serviciu pot exista pe platforme diferite și pot fi implementate folosind diverse limbaje de programare.

Conceptul cheie al SOA este faptul că funcțiile impementate de un anumit serviciu sunt puse la dispoziția utilizatorului prin intermediul unei interfețe standard. Astfel, detaliile implementării sunt ascunse utilizatorului serviciului. Utilizatorii apelează serviciul pe baza operațiilor puse la dispoziție de aceste interfețe.

1.2 Componente de bază ale SOA

Componentele de bază SOA sunt *elementele* și mesajele folosite pentru a efectua diverse *operații,* ce sunt inter-schimbate de către elemente.

Elemente. Există trei elemente cheie: **Service Provider** (Furnizorul de servicii), **Service Requestor** (Solicitantul de servicii) și **Service Registry** (Registrul de servicii). Cele trei elemente împreuna cu operațiile ce pot avea loc între ele sunt prezentate în Fig. 1.

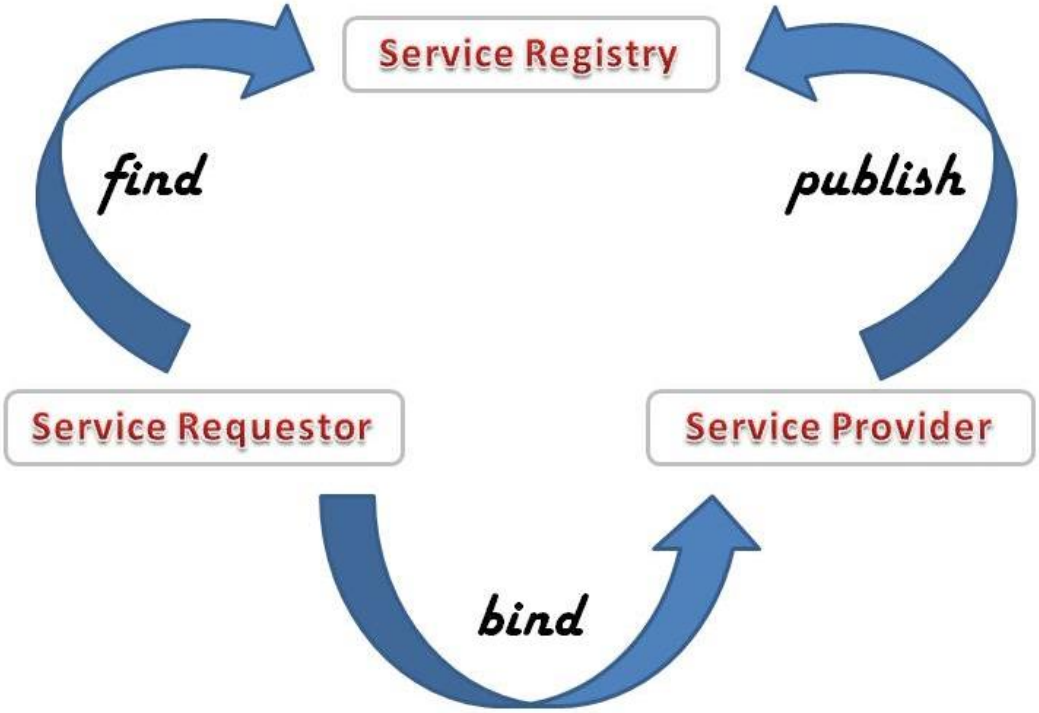


Figura 1 - Elementele SOA și operațiile dintre ele.

1. Service Provider-ul este responsabil pentru a crea descrierea serviciului, pentru a publica descrierea serviciului în unul sau mai mulți regiștri de servicii și pentru a primi mesaje de invocare de la unul sau mai mulți Service Requestor-i.

2. Service Requestor-ul trebuie să găsească descrierea serviciului publicată într-unul sau mai mulți regiștri de servicii și să folosească aceste descrieri pentru a face legătura cu sau a apela anumite servicii găzduite de Service Provider-i. Orice consumator de servicii poate fi considerat un Service Requestor.

3. Service Registry-ul are misiunea de a promova descrieriile serviciilor publicate într-unul sau mai mulți regiștri de servicii și de a permite Service Requestor-ilor să caute în colecția de descrieri dintr-un Service Registry. Astfel, un Service Registry acționează precum un mediator între Service Requestor și Service Provider. Odată ce s-a creat legătura între ultimii doi, Service Registry-ul nu mai este necesar.

O componentă a unei aplicații poate juca oricare dintre cele trei roluri descrise mai sus. De remarcat este faptul că o componentă poate juca mai multe roluri. De exemplu, o componentă poate, în același timp să fie și Service Provider, dar și Service Requestor.

Operații. Sunt definite de relațiile dintre elemente și sunt în număr de trei: **Publish, Find** și **Bind**.

1. Publish (publică). Exprimă relația dintre Service Provider și Service Registry. Service Provider-ul folosește operația „Publish" pentru a înregistra în Service Register interfețele serviciilor pe care le furnizează. Odată publicate în Service Register, serviciile sunt disponibile oricărui Service Requestor.

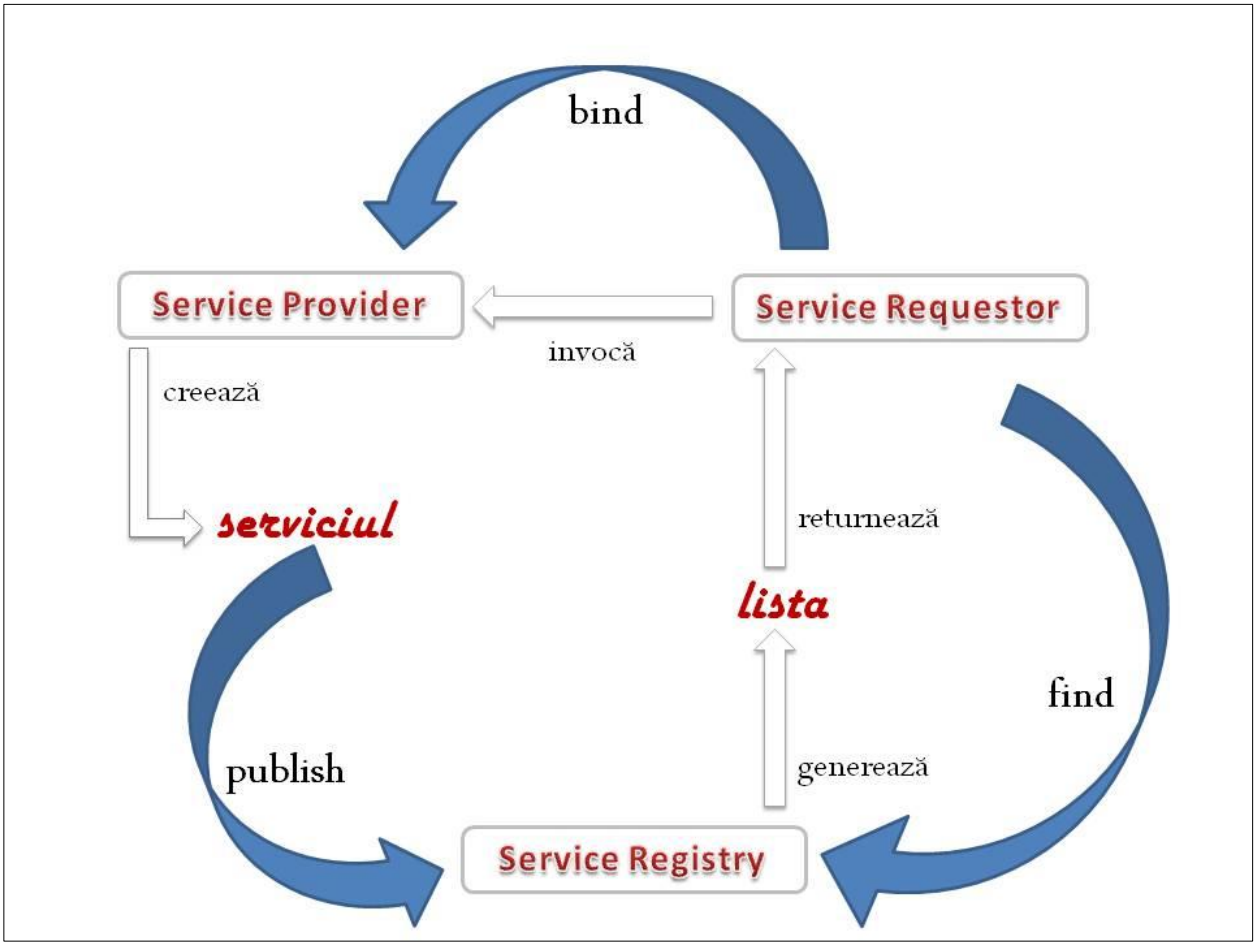


Figura 2 - Relațiile dintre componentele unui serviciu Web

2. Find (găsește). Este relația dintre Service Requestor și Service Registry. Service Requestor folosește operația „Find" pentru a extrage din Service Registry o listă cu toți Service Provider-i ce îi pot satisface cerințele. În operația „Find" se pot menționa anumite criterii de căutare. Service Registry-ul caută printre toți Service Provider-ii înregistrați și returnează rezultatele cele mai potrivite căutării.

3. Bind (leagă). Această operație face legătura între Service Requestor, cel ce inițiază o cerere, și Service Provider, cel ce urmează să îndeplinească cerința. Operația îi permite

Service Requestor-ului să se conecteze la Service Provider, înainte de a transmite cererea. De asemenea, îi permite Service Requestor-ului să utilizeze proxy-ul de la nivelul clientului pentru serviciul furnizat de Service Provider. Legătura poate fi dinamică sau statică. În cazul unei legături dinamice, Service Requestor-ul generează proxy-ul la nivelul clientului pe baza descrierii serviciului obținută de la Service Registry în momentul în care serviciul este invocat. Cu o legătura statică, Service Requestor-ul crează proxy-ul în timpul dezvoltării aplicației.

1.3 Serviciile Web ca implementare a SOA

Serviciile Web sunt tehnologii în continua dezvoltare predominant folosite pentru a implementa arhitecturiile orientate pe servicii. Ele implică un model bazat pe comunicația între programe construit pe standardul XML pentru specificarea datelor. Serviciile Web nu impun un anumit protocol pentru comunicație; totuși orice protocol de comunicație precum HTTP sau JMS poate fi folosit în schimbul de mesaje.

În ceea ce privește serviciile grid, serviciile Web folosesc **Web Services Description Language (WSDL)** pentru a descrie conținutul și utilizarea, standardul **Simple Object Access Protocol (SOAP)** ca protocol în schimbul de mesaje dintre doua servicii Web și specificația **Universal Description, Discovery and Integration (UDDI)** pentru a permite provider-ilor de Web să-și înregistreze serviciile și solicitanților să localizeze cei mai potriviți provider-i de servicii.

1.3.1 Web Service Description Language (WSDL)

Web Service Description Language (WSDL) este un standard bazat pe XML al consorțiului World Wide Web (W3C), folosit pentru a descrie interfețelor pentru servicii Web.

Standardul WSDL este folosit de către provider-ii de servicii pentru a descrie caracteristicile specifice serviciilor publicate în regiștrii de servicii prin intermediul specificației UDDI. Astfel, solicitanții de servicii pot căuta provider-ul de servicii adecvat și apela serviciul dorit pe baza informației exprimate prin WSDL.

Un document WSDL conține **trei categorii de informații** despre un serviciu Web: *Interfața serviciului* (Service Interface), *Conexiunile serviciului* (Service Bindings) și *Implementarea serviciului* (Service Implementation).

Interfața serviciului definește structura datelor transmise și semnătura operațiilor furnizate de un serviciu, într-o maniera independentă de limbaj, platformă și protocol de comunicare. Conexiunea serviciului specifică protocoalele de transport și regulile de codificare ce urmează să fie folosite la accesarea operațiilor publice furnizate de serviciu, iar

Implementarea serviciului menționează detaliile implementării pentru toate operațiile serviciului. Cele trei categorii sunt în continuare detaliate cu ajutorul câtorva exemple.

1. Interfața serviciului

Elementele de bază ale interfeței unui serviciu sunt *Tipurile, Mesajele* și *Operațiile.*


```
<?xml version="1.0" encoding="UTF-8"?>
<wsdl:definitions targetNamespace="http://DefaultNamespace"
   xmlns:impl=http://DefaultNamespace
   xmlns:intf="http://DefaultNamespace"
   xmlns:apachesoap="http://xml.apache.org/xml-soap"
   xmlns:wsdlsoap="http://schemas.xmlsoap.org/wsdl/soap/"
   xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema"
   xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
   xmlns="http://schemas.xmlsoap.org/wsdl/">

   <wsdl:types>
      <schema xmlns="http://www.w3.org/2001/XMLSchema"
         targetNamespace="http://DefaultNamespace">
        <import namespace="http://schemas.xmlsoap.org/soap/encoding/"/>
        <complexType name="ArrayOf_xsd_string">
          <complexContent>
            <restriction base="soapenc:Array">
              <attribute ref="soapenc:arrayType" wsdl:arrayType="xsd:string[]"/>
            </restriction>
          </complexContent>
        </complexType>
      </schema>
   </wsdl:types>

   <wsdl:message name="getMOTDResponse">
      <wsdl:part name="getMOTDReturn" type="xsd:string"/>
   </wsdl:message>

   <wsdl:message name="getMOTDRequest">
      <wsdl:part name="in0" type="impl:ArrayOf_xsd_string"/>
   </wsdl:message>

   <wsdl:portType name="MOTD1">
      <wsdl:operation name="getMOTD" parameterOrder="in0">
         <wsdl:input name="getMOTDRequest" message="impl:getMOTDRequest"/>
         <wsdl:output name="getMOTDResponse" message="impl:getMOTDResponse"/>
      </wsdl:operation>
   </wsdl:portType>
```

Figura 3 - Definirea interfeței unui serviciu Web

- ✓ **Tipurile.** Definițiile tipurilor de date folosite de mesajele schimbate între Service Requestor și Service Provider sunt menționate în secțiunea Tipuri.

În Figura 3 este prezentat un exemplu de definire a interfeței unui serviciu Web. Tipul complex „Arrayof_xsd_string” descrie un vector de șiruri de caractere. Acest tip va fi folosit la descrierea mesajelor.

- ✓ **Mesajele.** Un mesaj reprezintă o singură interacțiune dintre un Service Requestor și un Service Provider.

  Dacă o operație este o „Remote Procedure Call (RPC)”, ce solicită o valoare să fie returnată, atunci interacțiunea este bidirecțională și necesită să fie definită prin două mesaje. În exemplul din Figura 3 sunt definite doua mesaje: getMOTDRequest și getMOTDResponse. Ambele sunt de tipul „Arrayof_xsd_string”, menționat mai devreme la definirea tipurilor. Primul mesaj este trimis de la Service Requestor către Service Provider. Service Provider-ul răspunde trimițând înapoi mesajul getMOTDResponse. Aceste mesaje vor fi folosite la definirea operațiilor.

- ✓ **Operațiile.** O operație reprezintă descrierea unei acțiuni suportate de un serviciu Web. Operațiile pot fi de patru feluri; ele reprezintă și tipuri de modele folosite la schimbul de mesaje (Message Exchange Patterns - MEP):
  1. *One-Way* (unidirecțională). Folosind modelul One-Way, Service Requestor-ului îi este permis să trimită un mesaj pentru a activa o operație într-un singur sens, fără a primi un răspuns imediat de la Service Provider.
  2. *Request-response* (cerere-răspuns). Cu Request-response, Service Provider-ul răspunde trimițând înapoi rezultatul operației.
  3. *Solicit-response* (solicitare-răspuns). Modelul Solicit-response este folosit în cazul în care Service Provider-ul solicită un răspuns de la Service Requestor, iar Service Requestor-ul îi trimite un mesaj de răspuns.
  4. *Notification* (notificare). Prin intermediul modelului Notification, Service Provider-ul poate trimite Service Requestor-ului un mesaj unidirecțional de notificare.

Cele patru modele sunt prezentate schematic și în Tabelul 1.

În exemplul de mai sus, operația getMOTD este de tipul Request-response. Mesajul getMOTDRequest reprezintă input-ul operației, iar getMOTDResponse este răspunsul operației.

- ✓ **Tipul portului.** Tipul portului este o colecție de operații ce pot fi suportate de un serviciu Web. În exemplul din Figura 3 există un singur tip de port numit MOTD1 ce conține o singura operație, getMOTD.
- ✓

#	MEP	Ordinea mesajelor	Descriere
1	**One-way** (unidirecțional)	Mesaj input	*Punctul de final primește un mesaj.*
2	**Request-response** (cerere-răspuns)	Mesaj input Mesaj output	*Punctul de final primește un mesaj și trimite înapoi un mesaj.*
3	**Solicit-response** (solicitare-răspuns)	Mesaj output Mesaj input	*Punctul de final trimite un mesaj și primește înapoi un mesaj.*
4	**Notification** (notificare)	Mesaj output	*Punctul de final trimite un mesaj.*

Tabelul 1 - Message Exchange Patterns (MEP) - modele de schimb de mesaje ale unei operații.

2. Conexiunea serviciului

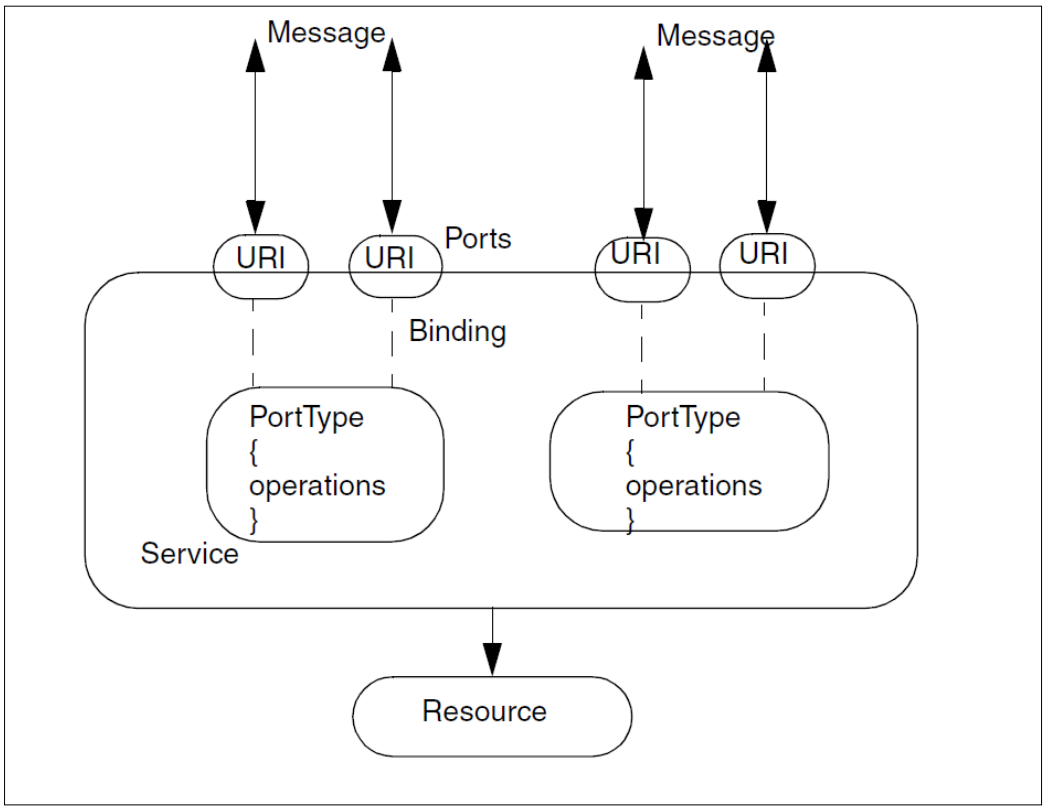


Figura 4- Relațiile dintre elementele WSDL

Conexiunea unui servicii oferă detalii despre cum poate fi folosit un protocol de transport la schimbul de mesaje dintre Service Requestor și Service Provider pentru un anumit tip de port. Un serviciu poate accepta mai mult conexiunii la un tip de port. Fiecare conexiune este identificată printr-un URI (Uniform Resource Identifier). Relațiile dintre un serviciu, conexiuniile acestuia, tipuri de port, URI-uri și mesajele transmise sunt prezentate în Figura 4.

După cum se poate observa, un serviciu facilitează accesul la o anumită resursă și poate conține tipuri de port multiple. Fiecare tip de port definește una sau mai multe operații și permite mai mult de o singură conexiune.

Într-un document WSDL, definirea unei conexiunii arată ca în Figura 5.

```
<wsdl:binding name="MOTDSoapBinding" type="impl:MOTD1">
  <wsdlsoap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
  <wsdl:operation name="getMOTD">
    <wsdlsoap:operation soapAction=""/>
    <wsdl:input name="getMOTDRequest">
      <wsdlsoap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace"/>
    </wsdl:input>
    <wsdl:output name="getMOTDResponse">
      <wsdlsoap:body use="encoded"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"
        namespace="http://DefaultNamespace"/>
    </wsdl:output>
  </wsdl:operation>
</wsdl:binding>
```

Figura 5 - Definirea unei conexiuni, parte a definirii unui serviciu Web

O conexiune este menționata în secțiunea numita „binding” și conține următoarele elemente:

- ✓ **Nume și tip.** Conexiunea este specificată pentru un anumit tip de port. În exemplul din Figura 5, numele conexiunii este MOTDSoapBinding, iar tipul portului este MOTD1.
- ✓ **Stil.** Stilul conexiunii definește tipul comunicării folosite de către protocolul de transport atunci când invocă operațiile tipului de port. Specificația WSDL definește doua tipuri de stiluri de conexiune pentru protocolul SOAP (Simple Object Access Protocol):
    1. **Document Style.** Folosind stilul „Document”, toată informația este încapsulată într-un document XML și este definită într-o schemă XML.
    2. **Remote Procedure Call (RPC) Style.** Cu stilul „RPC”, corpul unui mesaj SOAP este folosit pentru a apela o funcție iar elementele sunt reprezentate ca

parametrii funcției. În exemplul din Figura 5 este folosit stilul de comunicare RPC.

- ✓ **Transport.** Transportul specifică protocolul folosit pentru comunicare. Detaliile transportului menționate în conexiune sunt valabile pentru toate operațiile definite în cadrul unui tip de port.
- ✓ **Stilul codificării.** În plus față de stilul conexiunii, WSDL prezintă opțiunea de a specifica stilul de codificare al unui set de mesaje. Acesta poate fi de două feluri:
    1. **Encoded** (codificat). Stilul „Encoded" folosește regulile de codificare SOAP pentru a transpune tipurile de date abstracte în date concrete.
    2. **Literal** (literal). Pe de altă parte, folosind stilul de codificare „Literal", tipul de date abstract produce datele concrete.

Combinând stilul conexiunii cu stilul codificării, se obțin patru modele de comunicare diferite, și anume:

1. RPC/Encoded;
2. RPC/Literal;
3. Document/Encoded și
4. Document/Literal.

În Figura 5, operația getMOTD cu mesajele asociate: getMOTDRequest și getMOTDResponse folosește stilul de comunicare RPC/Encoded. Regulile de codificare sunt precizate la adresa: http://schemas.xmlsoap.org/soap/encoding/, iar tipurile ce urmează să fie folosite sunt definite în namespace-ul http://DefaultNamespace.

3. Implementarea serviciului

Implementarea serviciului furnizează detalii specifice implementării de care Service Requestor-ul se poate folosi pentru a cere diversele operații oferite de un serviciu Web. Un exemplu este prezentat în Figura 6.

```
<wsdl:service name="MOTD1Service">
  <wsdl:port name="MOTD" binding="impl:MOTDSoapBinding">
    <wsdlsoap:address location="http://localhost:8080/services/MOTD"/>
  </wsdl:port>
</wsdl:service>
```

Figura 6 – Definirea implementării unui serviciu, parte a definirii unui serviciu Web.

Elementele cheie specifice implementării unui serviciu sunt următoarele:

- ✓ **Serviciul.** Numele serviciului definește numele serviciului Web ce furnizează operațiile specificate în tipurile de port. În Figura 6, numele serviciului Web este MOTD1Service.
- ✓ **Portul.** Un port este în mare punctul de legătură ce oferă serviciul. Service Requestor-ul se conectează la acesta pentru a accesa serviciul. Numele portului, conexiunile și adresa portului sunt menționate în documentul WSDL. În exemplul din Figura 6, numele portului este MOTD, folosește conexiunea MOTDSoapBinding, iar adresa portului este http://localhost:8080/services/MOTD .

1.3.2 Simple Object Access Protocol (SOAP)

SOAP este un protocol folosit în schimbul de mesaje, bazat pe XML. Specificația SOAP definește un mecanism pentru schimbul de informație structurată într-un mediu decentralizat și distribuit. În ceea ce privește serviciile Web, SOAP este folosit ca protocol de comunicare între cele trei elemente cheie ale SOA: Service Provider, Service Requestor și Service Registry.

SOAP este un protocol independent de limbaj și platformă, de aceea poate fi cu eficiență folosit între servicii Web implementate în diferite limbaje și între o varietate de platforme. De asemenea, este independent de protocolul de transport și poate fi folosit împreuna cu diverse protocoale de transport, chiar daca cel mai comun protocol de transport pentru serviciile Web este protocolul HTTP.

Pentru a ilustra modul în care o astfel de comunicare are loc, figurile 7 și 8 prezintă un exemplu de mesaj SOAP transmis de la un Service Requestor la un Service Provider, solicitând apelarea unei operații specificată de către serviciul Web și mesajul de răspuns corespunzător. Ambele mesaje folosesc drept protocol de transport HTTP și sunt încorporate într-o cerere și într-un răspuns HTTP.

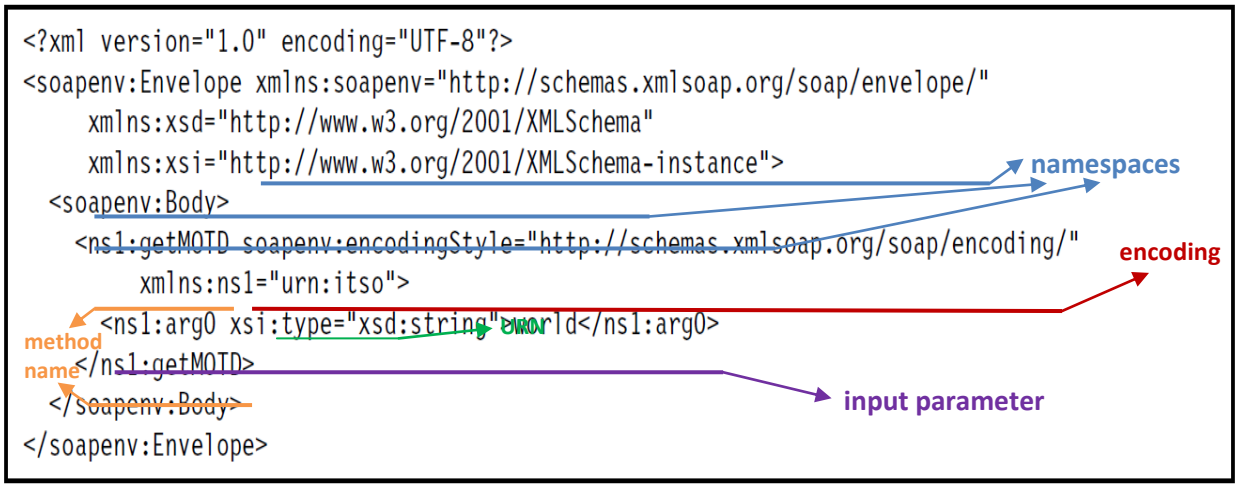


Figura 7 – Mesaj SOAP transmis de la Service Requestor către Service Provider

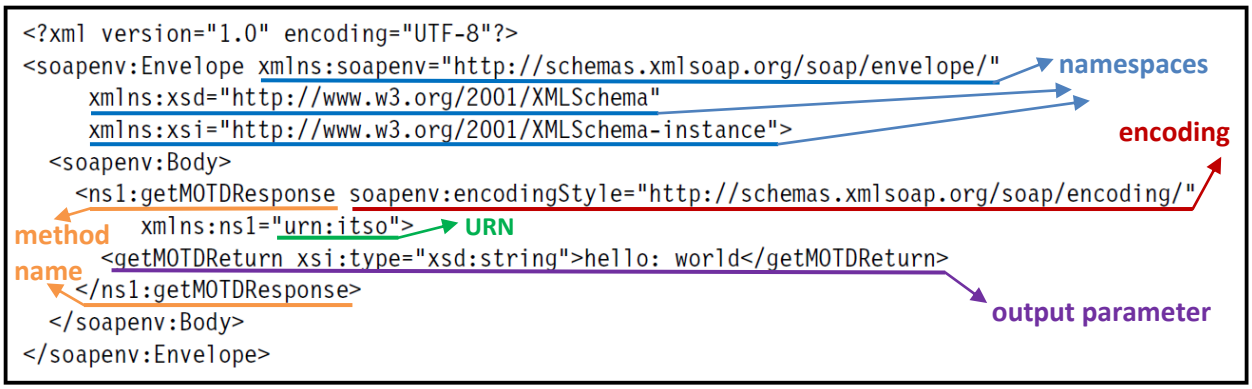


Figura 8 – Mesaj SOAP transmis de la Service Provider către Service Requestor

De remarcat sunt următoarele elemente SOAP:

- ✓ **Namespaces.** Pentru a permite interoperabilitatea între diferite limbaje de programare ce pot implementa un serviciu Web, diferite namespace-uri independente de limbaj sunt definite și specificate în mesajul SOAP.
  În exemplele de mai sus, namespace-urile ce definesc elementele folosite în mesajul SOAP sunt: xmlns:soapenv = http://schemas.xmlsoap.org/soap/envelope/, xmlns:xsd = http://www.w3.org/2001/XMLSchema și xmlns:xsi = http://www.w3.org/2001/XMLSchemainstance.

- ✓ **Encodings.** Codificările definesc modul în care valorile datelor definite în aplicație pot fi traduse în și din formatul protocolului. Acești pași ai translației sunt cunoscuți ca *serializare* și *deserializare.* Astfel, codificarea SOAP presupune traducerea structurilor de date dintr-un anumit limbaj de programare în SOAP XML și viceversa.
  „soapenv:encodingStyle=http://schemas.xmlsoap.org/soap/encoding/”, prezent în corpul mesajului SOAP, specifică tipul de codficare ce urmează să fie folosită la serializarea și deserializarea elementelor menționate în mesajul SOAP.

- ✓ **Uniform Resource Name (URN).** URN-ul specifică Service Requestor-ului, serviciul Web. Fiecare serviciu Web are un URN unic.
  În exemplul nostru, URN-ul este „itso” (urn:itso). Acest URN este unic în cadrul serverului SOAP „127.0.0.1/axis/services/MOTD”, menționat în header-ul HTTP.
- ✓ **Method name.** Reprezintă numele metodei sau al operației apelată în serviciul Web. Numele metodei este definită în corpul mesajului SOAP.
  De exemplu, în Figura 8 „ns1:getMOTD” este specificație metodei getMOTD din serviciul Web.
- ✓ **Input parameter.** O metoda poate accepta zero sau mai multi parametrii de intrare. Acești parametri sunt definiți în corpul mesajului SOAP, imediat dupa definirea numelui metodei.
  În Figura 7, metoda getMOTD suportă un parametru de intrare de tip string și valoare „world”, specificat prin „ns1:arg0”.
- ✓ **Output result.** Metoda poate returna o valoare de ieșire ca răspuns al apelării respectivei metode sau operații. Această valoare este returnată de la Service Provider către Service Requestor într-un mesaj SOAP de răspuns.

Figura 8 reprezintă chiar mesajul de răspuns, iar valoare returnată este „hello world”, de tip string. Aceasta este specificată în tag-ul getMOTDReturn.

1.3.3 Universal Description, Discovery and Integration (UDDI)

Universal Description, Discovey and Integration (UDDI) este o specificație care definește *mecanismele de depozitare* și *căutare* a serviciilor Web definite în documentele WSDL.

Service Provider-ul semnalează serviciile Web pe care le oferă registrându-le definițiile WSDL în registrul UDDI. De aici, Service Requestor-ul poate obține detalii despre un anumit serviciu pe perioada „build time” sau în timpul execuției. Pe piață există diverse unelte ce îi permit Service Requestor-ului să creeze proxy-uri independente de limbaj, platformă sau protocol, bazându-se pe informațiile serviciului furnizate în documentul WSDL. Astfel, la „build time”, Service Requestor-ul poate căuta cel mai potrivit WSDL, genera și încorpora proxy-ul într-o aplicație pentru a accesa serviciul oferit de serviciul Web.

Pe lângă aceste unelte, există anumite librării specifice interacțiunii cu registrul UDDI. De exemplu, UDDI4J acceptat de IBM este o librărie Java pentru interacția cu registrul UDDI. În mod similar, există pachete ce interpretează în mod dinamic documentele WSDL și apelează serviciul Web. Folosind aceste librării, în timpul execuției, Service Provider-ul poate obține descrieri ale serviciului de la registrul UDDI în mod dinamic și apoi poate invoca respectivul serviciu.

CAP. 2. Proiectarea aplicațiilor grid

Acest capitol tratează aspectele ce trebuie considerate în construirea unei aplicații grid, precum: analiza problemei, arihitectura aplicației și design-ul aplicației. Unele dintre aceste aspecte nu se aplică pentru fiecare proiect. Câteva aspecte sunt familiare din dezvoltarea altor tipuri de aplicații și nu necesită detaliere. Altele - noi datorită naturii aplicației grid, sunt examinate în detaliu.

2.1 Construim de la început sau re-folosim cod?

Dezvoltarea unei aplicații grid începe prin examinarea problemei și prima întrebare adresată este: Unde începe proiectul? Primul pas este o analiză de calificare a sistemului pentru a vedea dacă are sens să fie încadrat în grid. Dacă sistemul rezultă satisfăcator, se merge mai departe, trecându-se prin fazele standard de dezvoltare ale unui software de stabilire a cerințelor, design, implementare, testare, lansare și întreținere.

2.1.1 Construirea unei aplicații grid de la început

Un sistem grid nou îi oferă dezvoltatorului o putere mare de control exprimată prin:

- ✓ Libertatea de a alege limnajul de programare și uneltele;
- ✓ Libertatea de a alege cele mai bune technologii și set de unelte;
- ✓ O foaie albă de hârtie pentru design-ul tuturor aspectelor.

Dezvoltatorilor unui proiect nou le va fi ușor să urmărească indicațiile din acest curs, cu precizarea că mereu vor exista condiții asupra design-ului aplicației, independente de natura sa de aplicație grid. Astfel de condiții îl constrâng pe dezvoltator să:

- ✓ Considere diversele conexiuni la baze de date, mesaje aflate în așteptare, network sockets;
- ✓ Citească și scrie formate de date existente;
- ✓ Fie în conformitate cu politica de securitate locală.

2.1.2 Construirea unei aplicații grid prin re-utilizarea de cod existent

Un task mai greu de realizat pentru dezvoltatori este să încadreze un sistem deja existent într-un sistem grid. Aceasta acțiune este în general numită împachetare (wrappering) și poate întâlni următoarele dificultăți, iar sistemul are o șansă mare să eșueze dacă acestea nu sunt depășite.

1. **Codul existent este scris în Java.**

Atunci când sistemul existent este codat în Java, integrarea este ușor de făcut pentru că necesită scrierea unei noi clase Java ce prezintă una sau mai multe metode publice.

2. **Codul existent nu este scris în Java.**
   Dacă sistemul existent nu este scris în Java, atunci se pot folosi tehnicile Java Native Interface (JNI), dar există o restricție cu privire la sistemul de operare pe care serviciul grid poate fi lansat. De exemplu, un serviciu ce apelează direct Windows DLL poate fi lansat doar în Windows. Aceasta nu este o problemă majoră dacă grid-ul este omogen, dar cele mai multe grid-uri sunt de natură heterogenă ce limitează potențialul de dezvoltare al unui serviciu grid.
3. **Codul existent este mic și înglobat.**
   Aceasta este cea mai bună situație deoarece este posibilă construirea unui serviciu bine-definit, bazat pe o singură sarcină. Acest serviciu se folosește intens de procesor, acceptă un input, prelucrează și întoarce un rezultat.
4. **Codul existent este mare și/sau prezintă multiple conexiuni**
   Dacă sistemul existent prezintă aceste caracteristici, acestea îl transformă într-un candidat suspicios pentru un serviciu grid. Nodurile grid-ului trebuie să fie foarte puternice pentru a putea face față sarcinilor mari și complexe. Dacă prin codul existent se fac multiple conexiuni cu alte sisteme și/sau o cantitate mare de date este transferată este foarte probabilă crearea de blocaje.

2.2 Calificarea unei aplicații

Este foarte important ca o aplicație să fie întai analizată pentru a vedea dacă este bună pentru a rula într-un grid. Nu toate aplicațiile se dovedesc a fi de succes sau eficiente din punctul de vedere al costruilor. O aplicație este analizată din punctul de vedere a mai multor factori. În următoarele, o listă de criterii de considerat în analiza unei aplicații potențială pentru un sistem grid, este prezentată. Lista include atribute ale unei aplicații sau cerințe ce pot împiedica o aplicație de a deveni un bun candidat pentru a fi executată într-un mediu grid. Lista poate să nu fie completă, acest lucru depinzând de circumstanțele locale ale resurselor și de infrastructură. O schema de calificare a unei aplicații grid poate fi găsită la: http://developerWorks®.ibm.com/.

Lista următoare cuprinde cele mai critice aspecte ce pot dăuna sau chiar exclude o aplicație din a fi folosită într-un grid:

- ✓ Comunicare între procesele unui job ridicată, fără a avea o viteză mare de schimbare a conexiunii (de exemplu, MPI); în general aplicațiile multi-threaded necesită să le fie verificată nevoia de comunicare între procese.

- ✓ Cerințe stricte de planificare a job-urilor (job scheduling) ce depind de o furnizare necontrolată de date.
- ✓ Obstacole nerezolvate în stabilirea unei lățimi de bandă suficiente în rețea.
- ✓ Dependențe ce limitează mult mediul sistemului.
- ✓ Cerințe pentru tranzacții de tip business (commit și roll-back) sigure prin sisteme de tip grid. În momentul de față nu există standard pentru prelucrarea tranzacțiilor în grid.
- ✓ Inter-dependențe mari între job-uri, ce pot cauza o comunicare între procese crescută.
- ✓ Protocoale de rețea nerecunoscute folosite de diverse job-uri pot duce la interzicerea executării task-urile datorită regulilor firewall-ului.

2.3 Înțelegerea cerințelor

Odată ce punctul de pornire al unui proiect a fost stabilit, următorul pas este examinarea cerințelor unui proiect. Cerințele sunt cheia unui design de succes, a implementării și a lansării în execuție a unui sistem. Orice programator este conștient de faptul ca fără un set detaliat de cerințe, nici un sistem nu va putea fi gata la timp și cu totală funcționalitate.

În ziua de astăzi există zeci de metodologii aplicate în definirea cerințelor. Ideea este de a convinge dezvoltatori să folosească o metodologie, fără a face o comparație a celor mai bune metodologii. Majoritatea proiectelor necesită ca 50% din efortul total să fie depus în analiza cerințelor, arhitectură și design, înainte ca prima linie de cod să fie scrisă. De aceea este foarte important a se folosi o metodologie în definirea cerințelor, indiferent care.

Una dintre metode, care a fost folosita cu succes în trecut presupune definirea tuturor cerințelor și a use-case-urilor corespunzătoare. Apoi, fiecare parte participantă la proiect semnează o scrisoare de acceptare. Odată ce fiecare membru a semnat, se poate asuma că cerințele întrunesc obiectivele finale ale sistemului și nu vor apărea surprize sau diferențe de scopuri. Orice schimbare substanțială adusă în timpul dezvoltării este vazută ca Change Request (cerere de schimbare) și este fie realizată prin adaos de timp sau buget, fie întârziată până la dezvoltarea următoarei versiuni.

2.3.1 Cerințe funcționale

Cerințele funcționale captează comportamentul ce se intenționeză ca sistemul sa-l manifeste, privind serviciile pe care sistemul intenționează să le ofere. Câteva exemple sunt:

- ✓ User login
- ✓ User logoff
- ✓ Customer Data Entry and Validation

- ✓ Transaction
- ✓ Print Monthly Report
- ✓ Overnight Batch Process
- ✓ Pulling Data from Business Partner

Un mijloc foarte comun de a determina cât de complete și corecte sunt cerințele este construirea de „use-case-uri". „Use-case"-urile sunt un set de descrieri pas cu pas ale proceselor unui sistem cu referință la toate componentele interne și externe afectate. Use-case-urile pot fi folosite de echipa de testare pentru a verifica dacă cerințele au fost îndeplinite.

Use-case-urile sunt în mod general descrise în UML (Unified Modeling Language). Pe lângă use-case-uri, UML prezintă și alte tipuri de design orientat pe obiecte, precum:

- › **Diagrama de tip Clasă:** definiția claselor și a relațiilor dintre clase; majoritatea uneltelor UML pot genera cod Java din aceste diagrame;
- › **Diagrama de Interacțiune:** secvența obiectelor unui sistem în timp;
- › **Diagrama de Stare:** secvențe de stări prin care un obiect trece în ciclul sau de viață;
- › **Diagrama de Activitate:** o clasă specială de Diagrama de Stare.

UML a devenitUML a devenit o metodologie de modelare predominantă deoarece există multe produse ce le permit arhitecților și programatorilor să construiască un model întreg al sistemului, apoi să genereze cod Java direct din model. Un exemplu de familie de astfel de produse este IBM Rational Rose®.

2.3.2 Cerințe non-funcționale

Cerințe non-funcționale	Descriere
Scalabilitate	Cu cât utilizarea resurselor crește cu atât sistemul nu mai poate suporta sarcini adiționale. Astfel, un sistem poate ceda din mai multe motive. Obiectivul unui sistem bine modelat este capabilitatea de a adăuga mai multe resurse să rezolve sarciniile adiționale, asigurând astfel o performanță aproape liniară și nevoia de a nu remodela sistemul.
Factori pentru o mai bună interfață a utilizatorului	Programatorii tind sa neglijeze anumiți factori atunci când construiesc interfețele utilizatorilor, precum: › *Distribuția obiectelor într-o fereastră.* O bară de meniuri în partea de sus, o bară de butoane sub, o bară de parcurgere în partea dreaptă a ecranului și uneori și în partea de jos, iar în centru conținutul aplicației este

	standardul fiecarui sistem de operare modern, în ceea ce privește așezarea în pagină. › *Meniurile*. Structura clasică File-Edit-View-...-Help este bine înțeleasă de către utilizatori. Mai mult, aceștia sunt obișnuiți cu butoanele OK, Cancel și Help. › *Taste de accelerare*. Combinațiile de taste permit efectuarea de operații fără a folosi mouse-ul, preferate de mulți și deloc costisitoare pentru dezvoltatori. › *Wizard-uri de multi-dialog.* Constau într-o secvență de ferestre de dialog cu scopul de a colecta informații complexe de la utilizatori. Un button Previous este esențial pentru corectarea erorilor. › *Help.* Majoritatea ferestrelor și dialogurilor prezintă pop-up-uri de ajutor, dar este necesar să se asigure că ajutorul furnizat este într-adevăr de ajutor, permițându-i utilizatorului să înțeleagă un concept, să rezolve o problemă sau să fie condus către o sursă externă. › *Internaționalizare.* O aplicație va reuși sa-și satisfacă utilizatorii și să câștige unii noi, daca aceștia vor putea folosi aplicația în limba lor maternă. › *Accesibilitate.* Utilizatorilor cu dizabilități trebuie să li se permită să folosească aplicația în mod eficient. Scopul general este ca utilizatorii, de orice fel, să poată folosi o aplicație cât mai repede și mai eficient.
Tratarea erorilor	Atunci când o eroare apare și nu este rezolvată cu succes de către aplicație, este una dintre cele mai frustrante situații pentru utilzator.
Securitate	Securitatea este un domeniu complex și de aceea de cele mai multe ori nu este tratată suficient în contextul unei aplicații. Dezvoltatorii trebuie să considere următoarele: › *Autentificarea utilizatorului;* › *Autorizarea utilizatorului;* › *Criptarea datelor;* › *Înregistrarea și alertarea evenimentelor.*

Flexibilitatea aplicației	Un sistem nu este niciodată scris, lansat și neatins ulterior. Sistemul este restructurat și rescris de fiecare dată când apar probleme sistemul trebuie consolidat. Pentru o consolidare mai ușoară a sistemului există câteva aspecte de luat în cosiderare: › *Separarea codului și a datelor.* Obiectele de tip dată nu trebuie să fie prea strâns legate de cod. Structurile de date ale unei aplicații trebuie să poată fi modificate fără a schimba codul. › *Definirea interfețelor de tip published.* Codul unei aplicații poate fi modificat fără a crea niciun efect asupra modulelor externe. › *Design Plug-in.* Sistemul trebuie definit ca un set de module independente ce se conectează la o bază și nu afectează codul acesteia.
Platforma server și client	Oricine implicat în sistem trebuie să înțeleagă minimul platformelor hardware precum și o listă de pre-recuzite software pentru a evita apariția surpizelor în timpul dezvoltării, testării și executării. De exemplu comun este un sistem Web, ce așteaptă ca utilizatorul să folosească un browser Web.
Conexiuni externe	Aproape orice aplicație necesită conexiuni externe pentru a citi și/sau scrie date. Înainte ca dezvoltarea să înceapă este crtic să fie listate toate sistemele externe împreună cu metodele de comunicare, format-ele pachetelor de date, autentificare, autorizare, metode de conectare, proceduri de recuperare, performanțe așteptate și numele persoanelor de contact.
Performanță	Când intenționăm să executăm o aplicație într-un mediu grid, trebuie să considerăm performanța grid-ului precum și cerințele de performanță ale aplicației. Obiectivele performanței pot fi discutate din mai multe perspective. › *Din perspectiva furnizorului de servicii,* obiectivul performanței este ca infrastructura grid să atingă utilizarea maximă a diverselor resurse din grid. › *Din perspectiva solicitantului de servicii,* timpul de răspuns al unei aplicații executate în grid este important. Timpul de răspuns este afectat de o multitudine de factori

	precum: întârzieri în comunicare, întârzieri în accesul datelor, deficiențe în optimizarea aplicației la resursele grid, fiabilitatea rețelei.
Fiabilitate	Fiabilitatea este mereu o problemă în calcul și nici calculul grid nu este o excepție. Cea mai bună metodă de a trata această problemă este de a anticipa toate eșecurile posibile și de a furniza un mijloc de a le rezolva. › *Checkpoint-restart.* Când un job este executat, imagini de tip checkpoint sunt înregistrate la intervale regulate de timp. Dacă sistemul cedează în timpul execuției unui job, acel job este reluat de la cel mai recent checkpoint. › *Persistant storage.* Starea relevantă a fiecarui job este memorată persistent de un grid manager. › *Heartbeat monitoring.* Un mesaj de probă este trimis unui proces și procesul răspunde. Dacă procesul nu răspunde, un proces alternativ poate fi probat. Acesta poate determina statusul primului proces și îl poate chiar restart-a.
Managementul sistemului	Orice design necesită un set de unelte de bază pentru management pentru a determina disponibilitatea și performanța în grid.
Topologie	› *Topologia rețelei* poate lua diferite forme. Responsabilitatea rețelei este să furnizeze o lățime de bandă adecvată pentru oricare dintre sistemele grid. › *Topologia datelor.* De preferat este ca job-urile ce trebuie executate să fie alocate mașinilor cele mai apropiate din punct de vedere fizic de datele de care este nevoie pentru execuție. În așa fel, traficul în rețea ar fi redus precum și limitele scalanilității.
Medii cu platforme diverse	Un sistem grid constă într-o colecție de host-uri heterogene cu sisteme de operare și software-uri variate. Pentru a rula o aplicație, infrastructura grid are nevoie să găsească host-ul din grid corespunzător aplicației. Pentru a face ca o aplicație să fie executată, infrastructura grid trebuie să considere anumiți factori despre: › *Timpul execuției.* Cerințele despre timpul de execuție al unei aplicații trebuie să corespundă mediilor de

	execuție ale host-urilor unui grid. › *Portabilitate.* Executabilele anumitor aplicații sunt specifice platformei. De exemplu, o aplicație scrisă în C sau C++ trebuie recompilată pe o platformă înainte să fie executată pe platforma respectivă › *Sistemul de operare.* Un grid este o colecție de resurse de calcul heterogene. Dacă aplicația are anumite dependențe sau cerințe specifice sistemului de operare, aplicație trebuie să verifice dacă mediul corect este disponibil și să rezolve problemele diferitelor medii.
Format-ele fișierelor	Atunci când rezultatul unei aplicații ce rulează pe un grid este vvesat de o aplicație executată pe un grid diferit, este nevoie de cunoștiințe despre format-ul fișierelor din sistem. Cele două host-uri grid pot avea platforme diferite. XML poate fi format-ul folosit în schimbul datelor.
Licență software	Există multe produse și soluții de management al licențelor software: › *Licențe software comerciale.* Un număr insuficient de licențe pot afecta grav expansiunea sau pot chiar exclude anumite programe sau aplicații din a fi folosite în sistemul grid. Gama de modele pentru software-urile comerciale se extinde de la restrictiv la permisiv. Între aceste două extreme există numeroase modele. Licențele software sunt plătite o singură dată sau pe baza unei taxe lunare. Pot include update-uri sau pot solicita cumpărarea de noi licențe. Toate aceste aspecte variază în funcție de vendor, de situația clientului, depinzând de acorduri individuale sau de alte criterii. Licențele pot permite migrarea software-ului de la un server la altul sau pot fi legate de un singur calculator. Câteva dintre modelele de licențe sunt enumerate mai jos: › *Licența provider-ului de servicii:* Subscriber Access Licenses (SALs); › *Open source licensing;* › *FreeBSD, MIT, Apache (licențe permisive):* http://www.freebsd.org/, http://www.opensource.org/licenses/bsd-license.php,

	http://www.opensource.org/licenses/apachepl.php; › *LGPL (licența persistentă):* Lesser General Public License: http://www.opensource.org/licenses/lgpl-license.php › *Licențele publice GNU General Public License și IBM Public License (licențe persistente și virale):* http://www.gnu.org/copyleft/gpl.html http://opensource.org/licenses/gpl-license.php http://www.opensource.org/licenses/ibmpl.php Unelte pentru management-ul licențelor: FLEXIm (http://www.globetrotter.com/flexlm/flexlm.shtm), IBM Tivoli License Manager (http://www.ibm.com/software/tivoli/products/license-mgr/), IBM License Use Management (http://www.ibm.com/software/is/lum/), Platform Global License Broker (http://www.platform.com/products/wm/glb/index.asp)

2.4 Dezvoltarea unui design de nivel înalt

Odată cu stabilirea cerințelor, design-ul la nivel înalt poate fi finalizat. Prima parte a design-ului este definirea interfeței serviciilor grid-ului.

2.4.1 Definirea interfețelor

Interfața serviciului grid privește toate aspectele publice furnizate precum metode (împreună cu parametrii și rezultatele lor), datele serviciului și strategia de notificare. Definirea unui serviciu grid este similară cu definirea unui serviciu Web, dar dezvoltatorii de servicii grid au ceva mai multe caracteristici de considerat.

2.4.2 Definirea tipurilor parametrilor și a rezultatelor unei metode

Fiecare serviciu grid cuprinde una sau mai multe metode. În codul serviciului pentru client, prin intermediul unui număr mic de clase ajutătoare Globus, un obiect proxy local este creat. Acesta reprezintă serviciul grid rulând altundeva în rețea. Metodele definite ca publice sunt disponibile în acest obiect proxy local.

În plus față de lista de nume a metodelor, un designer trebuie să definească în întregime metoda, incluzând orice parametru de intrare și valori returnate.

Programatorii pot defini interfața unui serviciu grid în două feluri:

- ✓ Printr-o abordare **top-down (de sus în jos):** definiția descrierii unui serviciu grid este scrisă într-un limbaj XML, numit GWSDL, care este o extensie a limbajului WSDL pentru servicii Web; sau,
- ✓ **De jos în sus (bottom-up),** prin crearea unei clase de interfață de tip Java, ce definește metoda în Java.

Avantajele și dezavantajele fiecărei dintre cele două abordări vor fi discutate mai târziu.

2.4.3 Definirea datelor unui serviciu și a strategiei de notificare

Serviciile grid sunt scrise ca extensii ale standardelor serviciilor Web. Două dintre extensiile majore pe care serviciile grid le oferă serviciilor Web sunt datele serviciului și strategia de notificare.

Datele serviciului

SDE (Service Data Elements) este o colecție de obiecte ce pot fi direct accesate de clienți. Acest tip de acces al datelor diferă de cel bazat pe metode deoarece obiectul în sine este public și orice client îl poate accesa simplu, după nume. Interfața serviciului poate plasa restricții la datele serviciului, precum definirea de atribute precum minimul sau maximul de instanțe ale SDE, scrierea în obiecte (în plus față de accesul clienților la citire) și existența obligatorie sau nulitatea unui obiect.

Notificarea

Datele serviciului și notificarea sunt strâns legate. Primele poti fi folosite pentru a implementa procesarea asincronă, dar cum va ști un client dacă operația este completă și dacă datele în așteptare sunt SDE? Răspunsul este notificarea. Notificarea acționează ca un model de subscriere: când clientul dorește notificare asupra unui SDE, folosește metode de ajutor GT3 pentru a subscrie, dat fiind numele simbolic al SDE. Când serviciul grid plasează un rezultat într-un SDE, folosește metode GT3 pentru a trimite notificări către toți clienții subscriși. Notificarea poate fi de două feluri:

- ✓ **Push:** SDE-ul este trimis către client odată cu notificarea;
- ✓ **Pull:** un SDE provizoriu este trimis către client odată cu notificarea, iar clientul trebuie să trimită o cerere înapoi către SDE pentru a primi valoarea actualizată.

Designer-ii aleg stilul de notificare în funcție de nevoile aplicației.

Pentru a rezuma, datele serviciului și notificarea, din punctul de vedere al designer-ului:

- ✓ Determină ce date se vor a fi expuse într-o manieră ce nu solicită apelarea unei metode. Metoda datelor unui serviciu este în general folosită pentru a expune starea

unui serviciu, calcule intermediare etc. Acestea pot fi expuse și prin apelarea unei metode, dar în anumite aplicații SDE este mai potrivit.

- ✓ Determină dacă vor fi furnizate notificări despre unul sau mai multe SDE-uri, și ce stil de notificare va fi folosit. Notificarea este în general folosită pentru a furniza notificare asincronă a schimbării de stare a SDE. Nu există nici un alt fel de a furniza acest tip de funcționalitate.

2.4.4 Definirea life cycle-ului (ciclului de viață)

Ciclul de viață al unui serviciu grid începe cu instanțierea, continuă cu execuția și se oprește odată cu terminarea acestuia.

- ✓ **Instanțierea unui serviciu grid.** Există mai multe feluri de a instanția un serviciu grid:
  - › Unelte pentru linii de comandă furnizate de GT3: comenzi ajutătoare precum **ogsi-create-service** pot fi folosite pentru a iniția un serviciu grid. Aceasta este un mijloc de inițializare potrivit în timpul dezvoltării și al testării dar în producție este recomandat ca inițializarea să se producă în mod automat.
  - › Începerea automată de către un container de servicii grid. Un serviciu grid poate fi marcat pentru inițiere automată atunci când container-ul de servicii grid este pornit.
  - › Crearea programatică folosind clasele ajutătoare GT3. Cea mai comună metodă este folosirea claselor GT3 pentru a crea instanțe.
- ✓ **Execuția.** În timpul execuției, o instanță a unui serviciu răspunde la apelarea de metode, actualizează SDE-urile și trimite notificări când este nevoie.
- ✓ **Terminarea unui serviciu grid.** La fel ca în cazul pornirii unui serviciu, pentru oprirea acestuia există mai multe metode:
  - › Unelte pentru linii de comandă furnizate de GT3: comenzi ajutătoare precum **ogsi-destroy-service** pot fi folosite pentru a termina un serviciu grid. Aceasta este un mijloc de finalizare potrivit în timpul dezvoltării și al testării dar în producție este recomandat ca terminarea să se producă în mod automat.
  - › Terminarea automată, de către un container de servicii grid. O instanță grid are o durată de viață ce tinde la infinit, dar îi poate fi atribuită o durată mai scurtă în timpul creării. Când acea perioadă de timp expiră, container-ul serviciului grid automat oprește instanța. Oricărui client ce va încerca să acceseze serviciul după ce acesta a fost terminat, îi vor fi returnate excepții.
  - › Distrugerea programatică folosind clasele GT3 este cea mai comună metodă de a distruge instanțele care nu mai sunt folositoare.

2.4.5 Definirea securității

Securitatea este un subiect foarte răspândit, dar la acest punct în design nu este nevoie decât sa fie identificate acele componente ce prezintă o sensibilitate la siguranță. Această sensibilitate poate fi exprimată prin nevoia de criptare a datelor, de autentificare a utilizatorilor sau de verificare a autorizațiilor, etc.

2.4.6 Rularea de scenarii pentru a asigura îndeplinirea cerințelor

Arhitectura trebuie verificată din punctul de vedere al cerințelor funcționale și non-funcționale pentru a asigura lipsa problemelor.

2.5 Dezvoltarea unui design de nivel jos

La acest punct, design-ul de nivel înalt este detaliat până în momentul în care poate fi dat mai departe dezvoltatorilor. Această fază poate dura puțin mai mult deoarece trebuie asigurat că toate detaliile design-ului sunt suficient definite.

2.5.1 Cursul unei aplicații în grid

Întâi este nevoi de a defini câțiva termeni ce vor fi folosiți în număr repetat în acest sub-capitol:

- ✓ **Aplicație grid** = o colecție de unități de lucru ce folosesc o infrastructură grid pentru a rezolva o anumită problemă sau a atinge un rezultat dorit. O aplicație grid poate să consiste într-un număr de job-uri care împreună îndeplinesc un anumit obiectiv.
- ✓ **Job =** este considerat o singură unitate de lucru dintr-o aplicație grid.
- ✓ **Producător și consumator de date =** job-uri ce produc date se numesc producători, iar cele ce primesc date ca input-uri se numesc consumatori de date.

O aplicație grid constă în executarea mai multor job-uri. Avantajul unei aplicații grid este acela de a folosi mai mute resurse în mod concurent. Pentru aceasta, un aspect ce nu trebuie neglijat este procesarea datelor: în paralel sau în serie.

Cursul unei aplicații vs. cursul unui job

Înțelegem cursul unei aplicații ca fiind cursul lucrului între job-urile ce compun aplicația grid. Cursul intern al lucrului în interiorul unui job se numește cursul unui job. Există trei tipuri de bază ale cursului unei aplicații:

- › **În paralel.** Dacă o aplicație constă în mai multe job-uri ce pot fi toate executate în paralel, un grid poate fi foarte adecvat pentru executarea eficientă în anumite noduri, în special în cazul în care nu este solicitat (deloc sau în cantitate foarte limitată) schimb de date între sarcini.

  Începând cu o sarcină inițială, un anumit număr de job-uri sunt lansate execuției în noduri pre-selectate și dinamic alocate ale grid-ului. Rezultatul fiecărui job este colectat de către un job final și salvat.

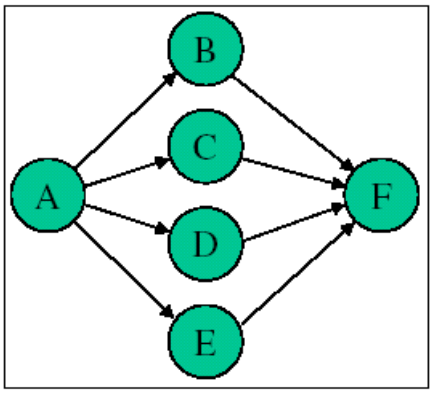


Figura 9 – Cursul paralel al unei aplicații

Pentru o anumită problemă sau aplicație, poate fi necesar spargerea acesteia în mai multe unități independente. Pentru a profita de execuția în paralel în grid, este important ca respectiva aplicație sau problemă să fie analizată spre a vedea dacă poate fi ruptă în unități de lucru independente ce pot fi executate precum job-uri individuale.

Acest tip de curs poate fi folosit cu succes atunci când nici unul dintre job-uri nu necesită ca input rezultatul altui job.

› **În serie.** În cursul paralel, fiecare job, cu excepția celui inițial, trebuie să aștepte ca predecesorul său să-și termine execuția pentru a putea returna un rezultat ce servește ca intrare pentru următorul job. În alte cuvinte, fiecare job este consumatorul predecesorului sau – producatoru de date.

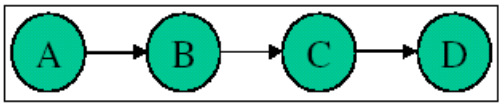


Figura 10 - Cursul serial al unei aplicații

În acest caz, avantajul executării într-un sistem grid nu este acela de a accesa sisteme multiple în paralel, ci abilitatea de a folosi oricare dintre resursele potrivite și disponibile. Această proprietate de a rula job-urile pe oricare dintre resurse contribuie la creșterea disponibilității și fiabilității aplicației. În plus, face o aplicație să fie mult mai scalabilă prin faptul ca în orice moment pot fi folosite resurse mai mari și mai rapide.

În orice caz, de preferat este întotdeauna să se verifice dacă job-urile sunt dependente unul de celălalt, sau dacă, prin natura lor, pot fi împărțite sau nu în unități executabile în paralel. Acest procedeu se numește *paralelizare.*

› **În rețea.** Aceasta este poate cea mai comună situație, dar și cea mai complexă. Anumite job-uri într-o aplicație sunt executate în paralel, dar totuși există inter-dependențe între ele.

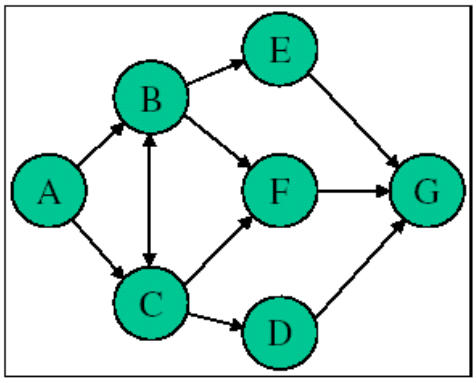


Figura 11 - Cursul job-urilor în rețea

Decuplarea. Într-un grid, decuplarea job-urilor solicită serviciului de management al cursului job-urilor să asigure sincronizarea rezultatelor individuale. Pentru a decupla, trebuie întâi realizată o analiză într-u a decide cum este cel mai bine ca o aplicație să fie despărțită în așa fel încât paralelismul să fie maxim. Asta înseamnă a a adăuga mai multe dependențe serviciilor infrastructurii grid, dar odată ce infrastructură este aranjată, aplicația poate beneficia de flexibilitatea și utilizarea unui mediu de calcul virtual.

Job-uri și sub-job-uri. O altă abordare este a ușura management-ul job-urilor într-o aplicație grid prin a introduce un sistem ierarhic de sub-job-uri. Un job se poate folosi de serviciile mediului grid pentru a lansa unul sau mai multe sub-job-uri. Pentru un astfel de mediu, o aplicație ar fi partiționată și creată în așa fel încât job-urile de nivel înalt ar include logică de a obține resurse și de a lansa sub-job-uri. Aceasta ar putea aduce beneficii pentru aplicațiile de mărime foarte mare.

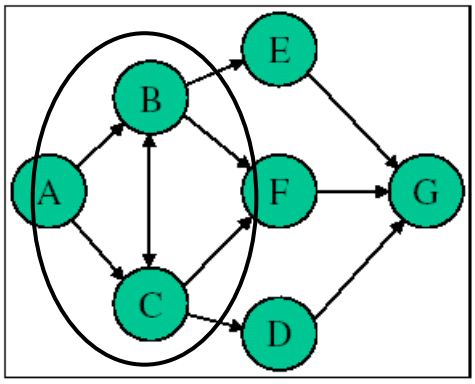


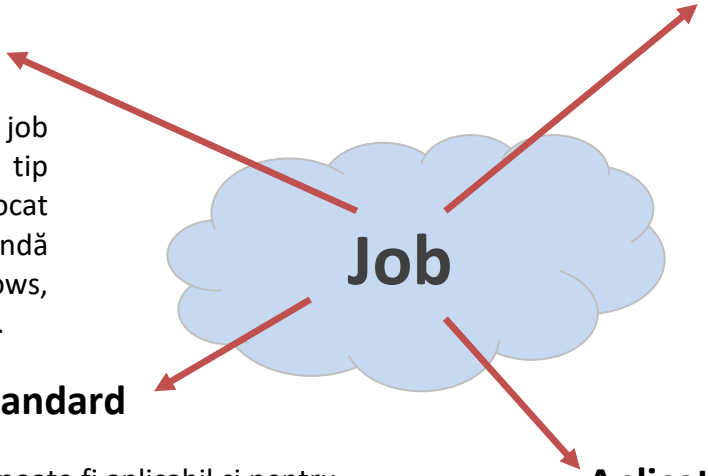
Figura 12 - Job-uri și sub-job-uri

2.5.2 Criterii de clasificare a job-urilor

Un job, ca parte a unei aplicații grid, teoretic poate să fie: de tip batch (lot), aplicație standard, aplicație paralelă sau/și interactivă.

Batch

Poate fi un job tradițional de tip batch invocat printr-o comandă din Windows, Linux sau Unix.



Aplicație paralelă

Sunt aplicațiile care au deja un curs paralel de execuție. Dar pentru a le folosi la maxim, server-ul grid trebuie să identifice și să acceseze funcțiile/job-urile interne ale acestora. Altfel, aplicațiile paralele vor fi văzute ca un întreg și tratate precum cele standard.

Aplicație standard

Un mediu grid poate fi aplicabil și pentru o aplicație standard, de exemplu pentru foi de calcul sau sisteme de redare video. O aplicație standard necesită o procedură de instalare specială și nu poate fi executată în rețea precum un job de tip batch.

Aplicație interactivă

Interacțiunea cu o aplicație grid este de cele mai multe ori facută prin intermediul unui portal grid sau a unei interfețe de server grid. Aceasta implică nici o interacțiune să aibă loc între utilizator și job, cu excepția celei de lansare a unui job.

2.5.3 Considerații despre limbajul de programare

Când o aplicație este dezvoltată apare întrebarea limbajului de programare. Mediul grid poate aduce lucruri în plus de luat în vedere în alegerea limbajului de programare.

Job-uri făcute pentru calculul de înaltă performanță sunt de obicei scrise în limbaje de programare precum C sau Fortran. Acele job-uri ale căror timp de execuție individual nu joacă cel mai important rol în aplicație, dar ale căror conținut și activități sunt mai importante, pot fi scrise în alte limbaje precum Java sau Perl.

În funcție de cerințele pentru job-urile individuale și resursele disponibile, într-o aplicație grid este posibil ca diferite părți să fie scrise în limbaje diferite. Câteva lucruri cheie de considerat în alegerea limbajului, însă sunt:

- ✓ **Portabilitatea pe o varietate de platforme.** Portabilitatea include și compatibilitatea binară, unde limbaje precum Java constituie un avantaj. Perl tinde să fie și el portabil, deoarece permite executarea aplicației independent de platformă. De asemenea, portabilitatea codului sursă trebuie considerată.
- ✓ **Librăriile/modulele din timpul execuției.** În funcție de limbaj și de cum porgramul este conectat, poate să apară cerința ca librăriile sau alte module să fie disponibile.

- ✓ **Interfețele către infrastructura grid.** Dacă un job trebuie să comunice cu infrastructura grid-ului atunci alegerea limbajului depinde de legăturile disponibile. De exemplu, Globus Toolkit 2.2 include legaturi pentru C.

2.5.4 Dependențele job-ului de mediul sistemului

Pentru orice job dintr-o aplicație grid, există o serie de factori de mediu ce îi pot afecta execuția. În design-ul unei aplicații acești factori trebuie considerați, iar aplicația trebuie construită pe cât mai independentă de acești factori pe cât posibil.

- ✓ Trebuie luate în considerare versiunea sistemului de operare, nivelul de serviciu și setările parametrilor sistemului de operare necesare pentru execuția unui job. Trebuie văzut dacă o aplicație grid va fi capabilă să-și ruleze job-urile pe orice nod cu sisteme de operare diferite sau dacă execuția va fi restricționată unui singur sistem de operare.
- ✓ Mărimea memoriei solicitată de un job poate limita nodurile posibile pe care acest job poate rula. Capacitatea memoriei disponibilă depinde nu numai de prezența ei fizică la un nod ci și de cat de multă memorie este sistemul de operare dispus să asigure la momentul execuției.
- ✓ DLL-uri ce urmează să fie conectate pentru execuție trebuie să fie disponibile pe resursa țintă sau pot fi transferate pe resursă înainte ca job –ul să fie executat.
- ✓ Setările compiler-ului influențează deoarece flag-urile și locațiile compiler-ului pot fi diferite. De exemplu, mici diferențe precum ordinea bits-ilor și numărul de bytes folosiți pentru numerele reale și întregi pot cauza căderi atunci când un job este executat pe un nod sau sistem de operare diferit.
- ✓ Mediul de execuție trebuie să fie gata să primească un job pentru execuție.
- ✓ Trebuie luate în vedere standardul și versiunea server-ului aplicației, precum și capacitatea acestuia și cerințele de accesare și serviciile folosite.
- ✓ Pentru anumite job-uri poate fi nevoie de diverse dispozitive hardware. De exemplu, cerințe de depozitare, dispozitive de măsurare și alte periferice trebuie considerate în construcția aplicației și în planificarea arhitecturii grid.

2.5.5 Punctul de verificare(checkpoint) și capabilitatea de a restarta

În cazul în care un job al unei aplicații grid eșuează, acel job poate fi lansat în execuție pentru a doua oară, dacă nu a schimbat nici o dată de natură persistentă înainte de a da eroare. Acest proces poate fi repetat pâna la finalizarea cu succes a job-ului respectiv. În orice caz, o modalitate mai sofisticată prin care server-ul grid să trateze erorile ar fi construirea unui punct de verificare (checkpoint). Un astfel de punct are caracteristica de a restart-a un job de la momentul în care acesta a eșuat, chiar și pe un nod diferit.

2.5.6 Topologia job-ului

Există numeroase aspecte de luat în vedere în ceea ce privește topologia unei aplicații grid. O parte dintre aceste aspecte includ cerințe de tip arhitectural pentru topologie și date.

La construirea arhitecturii unei aplicații grid, următoarele lucruri trebuie verificate:

- ✓ Unde trebuie și unde pot să ruleze job-urile;
- ✓ Cum să fie distribuite acestea și executate în rețea;
- ✓ Cum să fie grupate cu datele esențiale;
- ✓ Unde în rețea să fie salvate executabilele;
- ✓ Cum să determini un anumit nod să execute job-uri individuale.

Următorii factori trebuie considerați în tratarea aspectelor enumerate mai sus:

- ✓ Locația datelor și condițiile de acces ale job-ului;
- ✓ Cantitatea de date ce urmează să fie procesată de un job;
- ✓ Interfețele necesare pentru a interacționa cu orice fel de dispozitiv;
- ✓ Comunicarea între procese necesară unui job pentru a finaliza;
- ✓ Valorile de disponibilitate și performanță ale fiecărui nod la timpul execuției;
- ✓ Mărimea executabilei job-ului și abilitatea sa de a fi mutată în rețea.

2.5.7 Plasarea de date de intrare/ieșire

Orice job într-o aplicație grid plasează date de intrare și de ieșire devenind astfel în același timp producător și consumator de date, după cum a fost definit mai devreme.

Există numeroase moduri în care se realizează această plasare de date, ce trebuie considerate în timpul arhitecturii și al design-ului aplicației:

- ✓ ***Interfața linie de comandă*** este un mod natural de a primi date pentru job-urile de tip batch și aplicații standard. În acest caz, datele de intrare nu vor fi de natură complexă, însă vor fi compuse din anumiți parametrii de control al cursului intern al job-ului. Transferul de date către job are loc imediat, la momentul lansării job-ului, iar cantitatea de date intrate este în mod normal mică. Pentru cantități mai mari, pot exista argumente ce specifică fișierul sau sursa datelor.
- ✓ ***Date stocate,*** de orice fel pot servi ca intrare precum și ca ieșire. Transferul datelor de intrare se poate face oricând înainte de execuție, iar cel al datelor de ieșire oricând după ce job-ul a terminat.
- ✓ ***Cozi de mesaje,*** precum cele furnizate de WebSphere MQSeries®, sunt potrivite pentru activități asincrone dintr-o aplicație grid, în special atunci când livrarea de date furnizate job-ului și generate de către acesta este de o deosebită importanță. Un job poate accesa datele dintr-o coadă în mai multe feluri, în mod normal folosind API-uri specifice pentru a plasa și extrage datele.

- ✓ ***Valoarea returnată de către sistem (System return value)*** este un caz ce corespunde interfeței linie de comandă și în mod normal mijlocul prin care un job de tip batch sau orice program invocat prin interfața linie de comandă va returna date, sau cel puțin status-ul job-ului terminat. Astfel îi este indicat server-ului grid status-ul fiecărui job. Datele rezultate pot fi plasate unui depozit de date sau cozi de mesaje spre a fi procesate sau prezentate.
- ✓ ***Alte API-uri.*** Comunicând cu serviciile Web, orice sistem extern trebuie să îndeplinească anumite condiții pentru plasarea și preluarea de date. În astfel de cazuri, pot fi folosite HTTP, HTML, XML, SOAP sau alte protocoale sau API-uri de nivel înalt.

Pentru o aplicație grid este posibil să nu existe numai un mod în care datele sunt plasate pentru un job. Combinații de mecanismele enumerate mai sus pot fi folosite. Soluția optimă depinde însă de mediul și de cerințele propuse în faza de arhitectură și design a aplicației.

2.5.8 Tranzacții

Abordarea tranzacțiilor într-o aplicație grid devine destul de complexă și necesită deosebit de multă atenție, întrucât beneficiile de care se bucură o aplicație grid pot fi mascate de complexitatea implementării tranzacțiilor. Viitorul dezvoltării OGSA va putea include tratarea tranzacțiilor ca un serviciu, însă momentan nu există un astfel de suport.

2.5.9 Criterii de date

Orice aplicație are în centru procesarea datelor. De aceea, datele folosite pentru și înăuntrul unei aplicații grid trebuie privite în detaliu.

Datele influențează multe aspecte ale design-ului unei aplicații și determină dacă o aplicație grid planificată poate asigura beneficiile așteptate mai bine decât oricare altă soluție.

2.5.10 Criterii de utilizabilitate

Cerințe de utilizabilitate tradiționale

Se referă la caracteristicile ce facilitează ușurința de a folosi sistemul precum: interacțiune, afișare și atribute afective ce îl determină pe utilizator să se folosească de sistem într-un mod cât mai eficient, prompt și satisfăcător. Prin urmare, aceste cerințe trebuie considerate în dezvoltarea unei soluții de tip grid. Ele sunt folosite pentru a:

- ✓ Furniza o ghidare de bază a dezvoltatorilor în design-ul interfețelor utilizatorului;
- ✓ Stabili standarde de performanță pentru evaluarea utilizabilității;
- ✓ Defini scenarii de test pentru teste de utilizabilitate.

Câteva cerințe de utilizabilitate tipice privesc:

- ✓ Abilitatea de a personaliza. Sunt cerințele existente pentru utilizator în a-și adapta interfața și componentele sale întru a susține optimizarea.

- ✓ Eficiența: cerințele ce privesc minimizarea pașilor activităților, simplificarea operațiilor și permiterea cerințelor utilizatorilor de a fi rapid îndeplinite.

Cerințe de utilizabilitate pentru soluții grid

Soluțiile grid trebuie să adreseze cerințe de utilizare cu privire la o gamă variată de categorii de utilizatori, ce include:

- ✓ Utilizatori ce doresc să se înregistreze în grid, să execute aplicații în grid și să vizualizeze rezultatele;
- ✓ Proprietari/utilizatori de masini de calcul;
- ✓ Administratori și operatori ai grid-ului.

2.5.11 Instalarea

Soluțiile grid ar trebui să permită instalarea rapidă și automatică de către o persoană non-tehnică decât de către expert. Procesul instalării ar trebui să fie la fel de simplu pentru host, management și nodurile clienților indiferent de natura potențial heterogenă a sistemului de operare sau a configurației.

2.5.12 Criterii de obstrucție

Transparența și ușurința în utilizare sunt esențiale pentru un bun design de grid. Privind aceste aspecte, următoarele trebuie considerate:

- ✓ Folosirea grid-ului trebuie să fie transparentă utilizatorului. Portalul grid trebuie să izoleze utilizatorul de nevoia de a înțelege construcția grid-ului.
- ✓ Documentație trebuie să fie disponibilă pentru orice categorie de utilizator, în caz de necesitate. Ea trebuie să includă exemple de utilizare și demo-uri.
- ✓ Ușurința de a înrolare a resurselor la grid.
- ✓ Ușurința de a executa un job trebuie să mascheze nevoie utilizatorului de a întelege construcția grid-ului.

2.5.13 Aspecte informative și predictibile

Status-ul grid-ului trebuie să fie continuu disponibil. Asta ar putea include indicatori de încărcare sau utilizare a grid-ului, al numărului de job-uri aflate în execuție, al resurselor disponibile, al celor rezervate și posibil, al punctelor slabe din grid.

Deoarece construcția grid-ului se poate schimba în mod dinamic, estimarea timpului de răspuns poate deveni dificilă.

2.5.14 Flexibilitatea și fiabilitatea

Câteva aspecte de flexibilitate și fiabilitate ale aplicațiilor grid au fost deja acoperite. În această secțiune, ele sunt subliniate din punctul de vedere al utilizatorului grid-ului:

- ✓ O deosebită atenție trebuie acordată cerințelor pentru tratarea erorilor. Acestea pot fi rezolvate cu succes, dacă natura aplicației este bine înțeleasă pentru a oferi un tratament corect eșecurilor și posibilitatea de repornire automată. Notificări către utilizator trebuie trimise, deoarece acesta nu se află în permanență înregistrat la grid. Ca o consecință, mecanisme de feedback trebuie încorporate.
- ✓ Natura aplicațiilor potrivite pentru a fi executate în grid pot permite un anumit nivel de toleranță la erori, lucru neîntâlnit la aplicațiile tradiționale.
- ✓ Aplicațiile trebuie integrate cu unelte de management al sistemului pentru a raporta status-ul și căderile. În plus, cerințe trebuie stabilite pentru felul în care această informație va fi comunicată utilizatorului, indicând status-ul job-urilor.
- ✓ Ar trebui considerat și faptul de a furniza rezultate intermediare utilizatorului în cazul în care rezultate valide pot fi obținute.

2.6 Implementarea design-ului

În acest moment începe procesul de scriere. Acesta cuprinde următorii pași:

- ✓ **Scrierea interfeței.** Interfața trebuie scrisă într-unul dintre cele două moduri:

  1. o clasă de interfață Java, sau
  2. un fișier Grid Web Services Description Language(GWSDL).

  Dacă serviciul grid constă într-o listă de metode publice atunci mai potrivită pentru implementare este o interfață Java. Dacă serviciul este complex, atunci automat se va folosi un fișier GWSDL.

  Toate interfețele includ:

  - › Toate metodele publice;
  - › Parametrii metodei;
  - › Tipurile returnate de către metodă;
  - › Definițiile elementelor de date ale serviciului.

- ✓ **Scrierea implementării.** Cu interfața serviciului complet definită, tot ce rămâne de făcut este implementarea codului Java ce execută funcția solicitată atunci când metodele serviciului sunt apelate.
- ✓ **Scrierea părților non-Java.** Aceste părți non-Java se încadrează în următoarele categorii, de descriere a:
  - › **Tipului datelor.** Dacă serviciul folosește tipuri de date complexe, acestea trebuie definite. Descrierea datelor este facută în XML și primește extensia .xsd.
  - › **Interfeței serviciului.** Acesta este fișierul ce definește serviciul grid, foarte similar cu fișierul WSDL ce definește un serviciu Web. Formatul Grid

WSDL (GWSDL) a fost inventat deoarece fișierele WSDL nu prezintă anumite caracteristici necesare pentru definirea completă a unui serviciu grid.

- **Lansarea serviciului.** Prin acest fișier sunt furnizate instrucțiunile de lansare către container-ul serviciului grid și este similar stilului folosit pentru serviciile Web.

- **Scrierea clienților.** Cu întregul serviciu grid implementat, trebuie completată acțiunea finală, aceea de a scrie clienții ce pot accesa serviciul grid.