Deployment

Christian Perez
LIP/INRIA
2010-2011

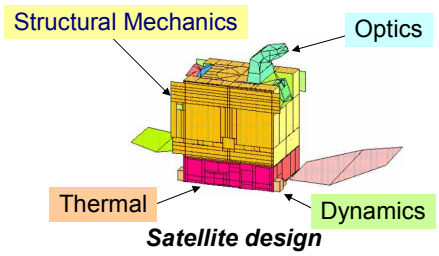
Content

- Deploying an application on a Grid
- CCM Deployment Model
- Tools for deploying
  - FDF
  - Tune
  - ADAGE

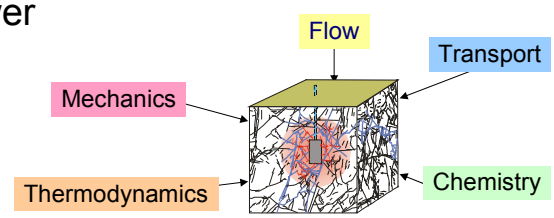
What is the problem?

Compute Intensive Applications

- Scientific computing
  - Code coupling
  - Complex multi-physics applications
- Computational power
  - Finer grain
  - Better accuracy



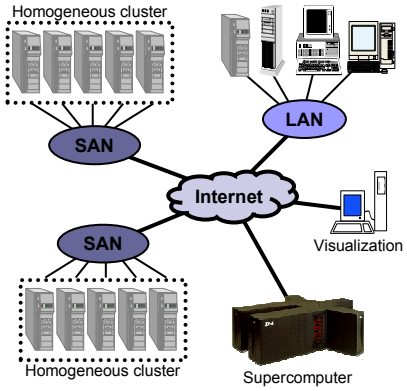
Satellite design



Simulation of fluid and solute transport

Computational Grids

- Compute and storage resources:
  - Geographically distributed
  - Interconnected over a WAN
  - Not dedicated to one application
- Network bandwidth increase
- Potentially huge computer power
- Issues: security, heterogeneity
  - Compute resources
  - Network technology, performance, and topology / hierarchy
- Complex environment



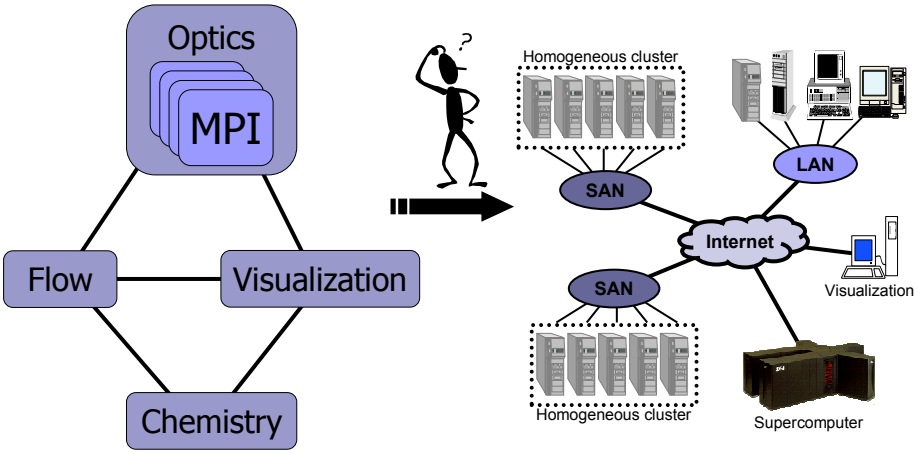
Grid Access Middleware

- Low-level services
  - Globus Toolkit, Unicore, Legion, Condor-G
- Resource management (GRAM)
  - Launch processes on remote resources
- Data management (GASS)
  - File transfer (GridFTP, RFT), replica management (RLS)
- Information service (MDS)
- Security (GSI)

What Programming Model?

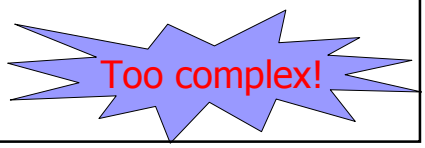
- Distributed
  - Component-based programming model
    - CCM, CCA
  - Workflow
- Parallel
  - MPI, OpenMP
- Combination of distributed and parallel
  - GridCCM
- Grids are not dedicated to **one** type of application

How to Deploy my Application on those Grid Resources?



Manual Deployment

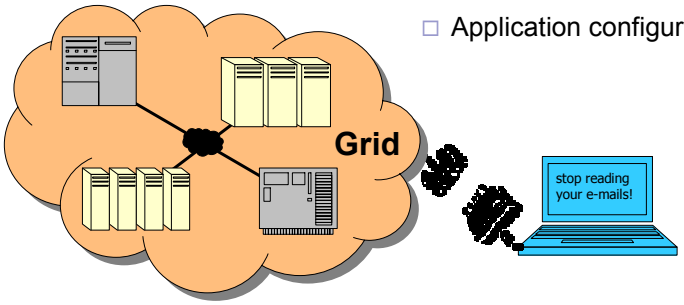
- Discover available grid resources
- Select grid resources for execution
  - OS, architecture compatibility
- Map the application onto the selected resources
  - MPI processes
  - Components
- Select compatible compiled executables
- Upload and install executables, stage input files in
- Launch processes on remote computers
- Set configuration parameters of the application
  - Components' attributes
  - Network topology information
- Retrieve result files



Automatic Deployment

- Hide application complexity
- Hide grid complexity

- Automatic
  - Resource discovery
  - Execution node selection
  - File installation
  - Process launch
  - Application configuration

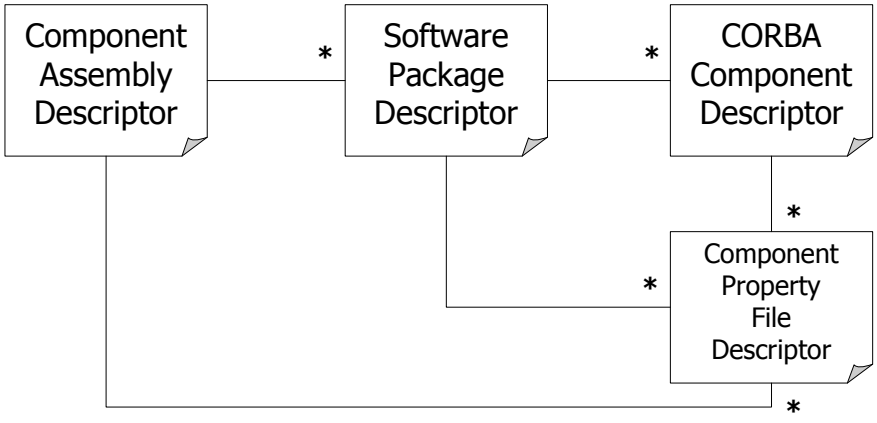


CCM Deployment Model

Deployment requirement

- Assembly package to deploy
  - Software descriptions and binaries
  - Assembly descriptor
- Target environment
  - Computers, networks, services

Relationship Between CCM XML Descriptors



Software Package Descriptor (.csd)

- Descriptive general elements
  - title, description, author, company, webpage, license
- Link to OMG IDL file
- Link to default property file
- Implementation(s)
  - Information about Implementation
    - Operating System, processor, language, compiler, ORB
    - Dependencies on other libraries and deployment requirements
    - Customized property and CORBA component descriptor
  - Link to implementation file
    - Shared library, Java class, executable
  - Entry point

Software Package Descriptor Example

```
<?xml version='1.0'?><!DOCTYPE softpkg>
<softpkg name="ObserverHome">
 <idl id="IDL:ObserverHome:1.0"><fileinarchive
   name="observer.idl"/></idl>
 <implementation id="observer_0x1">
  <os name="WinNT" /> <processor name="x86" /> <compiler
   name="VC++" />
  <programminglanguage name="C++" />
  <dependency type="DLL"><localfile name="jtc.dll"/></dependency>
  <dependency type="DLL"><localfile name="ob.dll"/></dependency>
  <descriptor type="CORBA Component">
   <fileinarchive name="observer.ccd" />
  </descriptor>
  <code type="DLL">
   <fileinarchive name="ObserverExecutors.dll"/>
   <entrypoint>create_ObserverHome</entrypoint>
  </code>
 </implementation>
</softpkg>
```

CORBA Component Descriptor (.ccd)

- Structural information generated by CIDL
  - Component / home types and features
  - Ports and supported interfaces
  - Component category and segments
- Container policies filled by the packager
  - Threading
  - Servant lifetime
  - Transactions
  - Security
  - Events
  - Persistence
  - Extended POA policies
- Link to component and home property files

CORBA Component Descriptor Example

```
<corbacomponent>
 <corbaversion>3.0</corbaversion>
 <componentrepid>IDL:DiningPhilosophers/Philosopher:1.0</componentrepid>
 <homerepid>IDL:DiningPhilosophers/PhilosopherHome:1.0</homerepid>
 <componentkind>
  <session><servant lifetime="component"/></session></componentkind>
 <threading policy="multithread"/>
 <configurationcomplete set="true"/>
 <homefeatures name="PhilosopherHome"
   repid="IDL:...PhilosopherHome:1.0"/>
 <componentfeatures name="Philosopher" repid="IDL:...Philosopher:1.0">
  <ports>
   <publishes publishesname="info" eventtype="IDL:StatusInfo:1.0">
    <eventpolicy policy="normal"/> </publishes>
   <uses usesname="left" repid="IDL:DiningPhilosophers/Fork:1.0"/>
   <uses usesname="right" repid="IDL:DiningPhilosophers/Fork:1.0"/>
  </ports>
 </componentfeatures>
</corbacomponent>
```

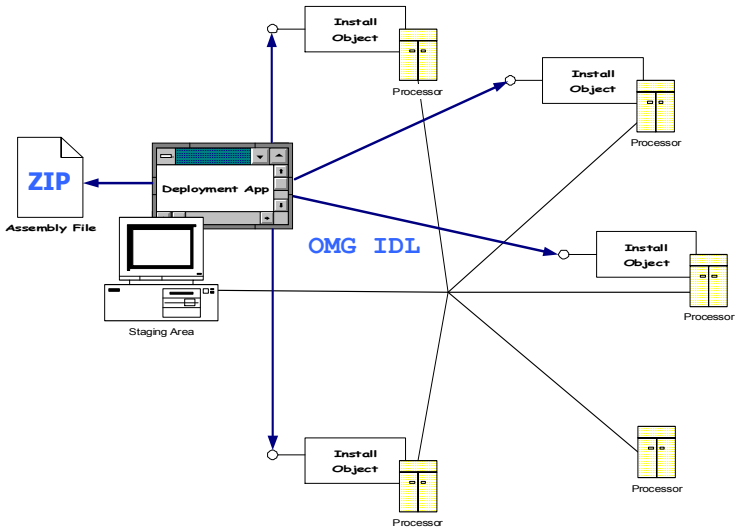
Deployment according to the OMG PIM

- Packaging
  - Pack all the elements required by a component into a package
- Installation
  - Installs the software package in a repository
  - Apply some policies before running, e.g. security authentication
- Configuration
  - Creates functional configurations for later execution
- Planning
  - Decides on which resources the elements of the software will be executed and which implementation will be run
- Preparation
  - Execute the plan by moving binaries to target resources
- Launch
  - Instantiate, connect, configure components and start execution

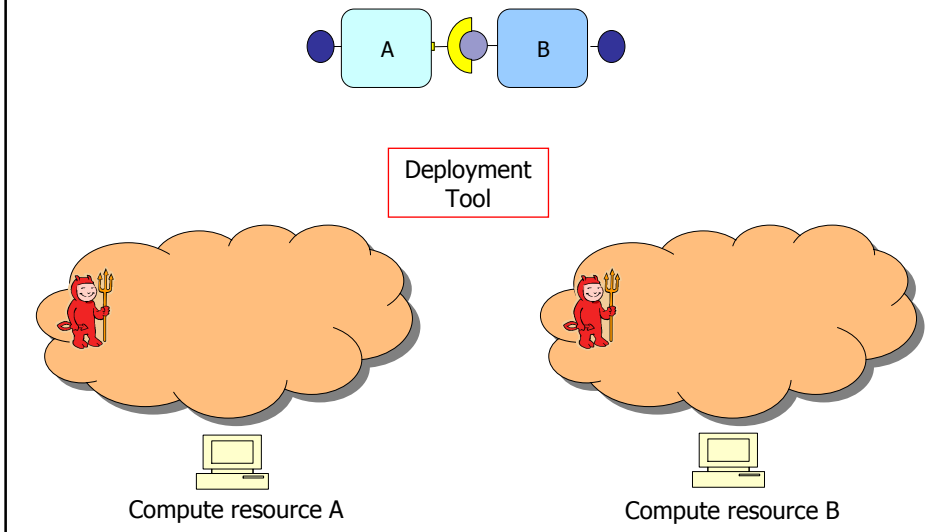
Deployment

- An Assembly Archive is deployed by a deployment tool
  - A component or a set of component
- The deployment tool might interact with the user to assign homes and components to hosts and processes
  - Finding available hosts/processes
  - Selecting some/all of them
  - Assigning components to them
- The deployment application interacts with installation objects on each host

The Component Deployment Process

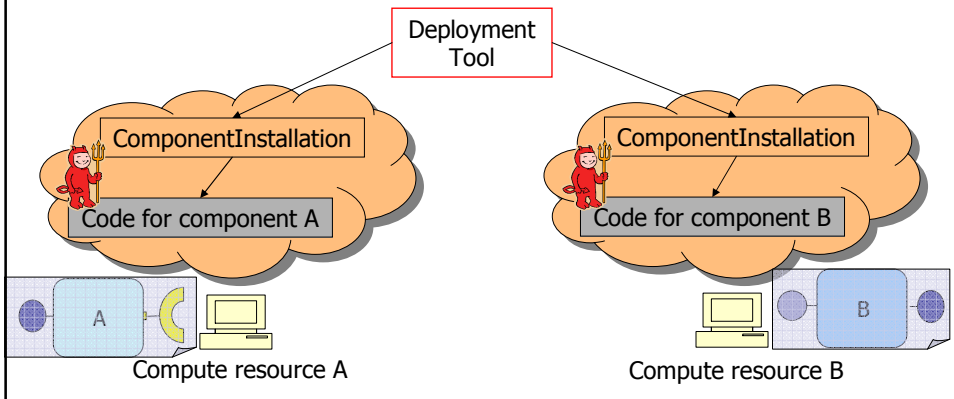


Deployment Model of CCM



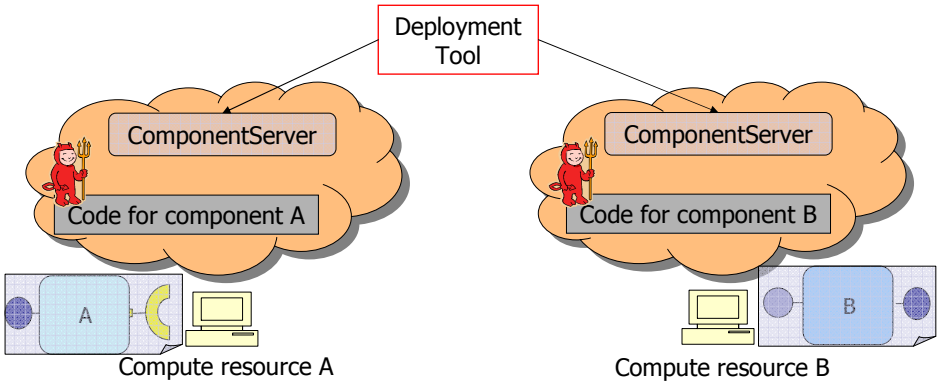
Deployment Model of CCM: Implementation UpLoading

- Install component implementation



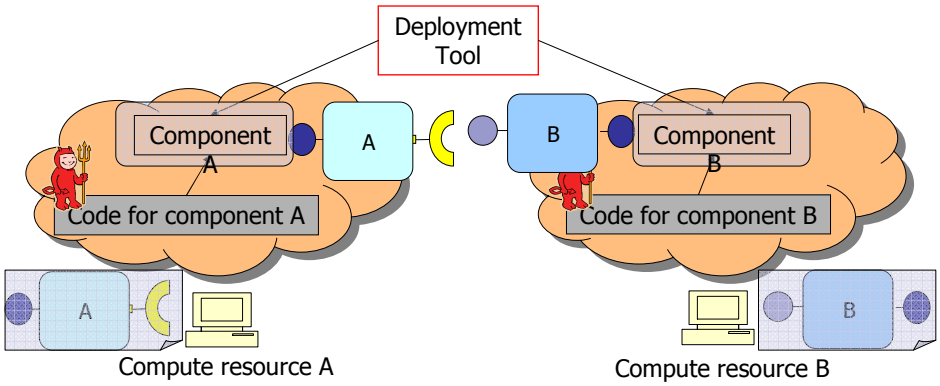
Deployment Model of CCM: ComponentServer Instantiation

- Create application processes



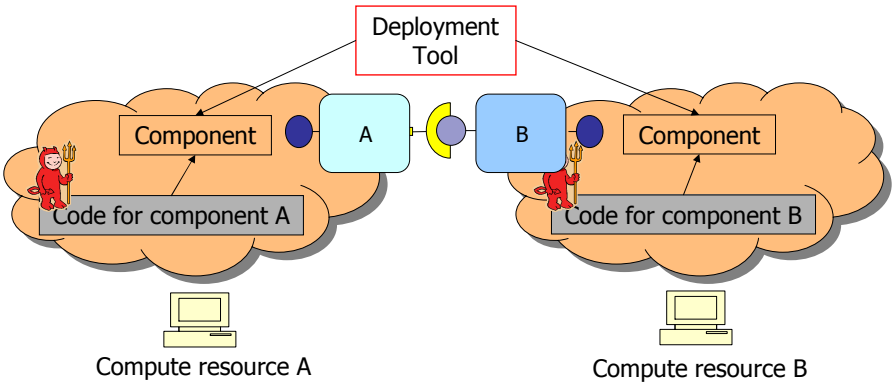
Deployment Model of CCM: Component Instantiation

- Component creation (Corba operation)



Deployment Model of CCM: Component Configuration

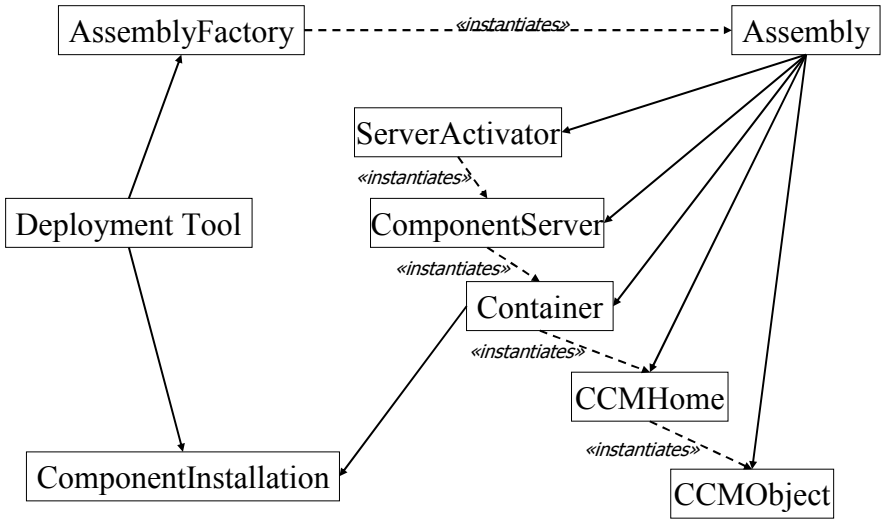
- Connection, configuration



Deployment Objects

- **`ComponentInstallation`**
  - Singleton by host, installs component implementations
- **`AssemblyFactory`**
  - Singleton, creates `Assembly` objects
- **`Assembly`**
  - Represents an assembly instantiation
  - Coordinates the creation and destruction of component assemblies and components
- **`ServerActivator`**
  - Singleton by host, creates `ComponentServer` objects
- **`ComponentServer`**
  - Creates `Container` objects
- **`Container`**
  - Installs `CCMHome` objects

The Component Deployment Process



Deployment API: overview

```
interface ComponentInstallation {
   void install(UUID, Location); // replace, remove, …
};
interface AssemblyFactory
   Cookie create(Location); // lookup(Cookie), destroy(Cookie);
};
interface Assembly {
   void build (); // tear_down()
};
interface ServerActivator {
   ComponentServer create_component_server(ConfigValues);
   // remove/get_component_server(…)
};
interface ComponentServer{  // void remove ();
   readonly attribute ConfigValues configuration;
   Container create_container(ConfigValues); // remove/get_container
};
interface Container {  // void remove();
   readonly attribute ConfigValues configuration;
   CCM_Home install_home (string, string, ConfigValues); // remove_home(…)
   CCM_Homes get_homes();
};
```

FDF

ADAM Project Team – LIFL – INRIA

FDF - Introduction

- Framework générique pour le déploiement de systèmes
  - Déploiement de différentes couches logicielles avec résolution des dépendances
  - Gestion du déploiement de systèmes distribués et/ou hétérogènes
- 1 langage spécifique au domaine du déploiement
  - Description de systèmes par administrateurs
  - Programmation de personnalités encapsulant les détails du déploiement
- 1 bibliothèque de composants primitifs encapsulant mécanismes bas niveau de déploiement
  - protocoles de transfert, de contrôle à distance, shells, etc.
- 1 interface graphique pour visualiser la configuration du système et déployer les logiciels sélectionnés
- Basé sur le modèle à composants Fractal

FDF - Langage de déploiement

- Fournit un langage haut niveau pour le déploiement
- Description de systèmes par les administrateurs
  - Déclaration de l'infrastructure système / les machines / les protocoles
  - Déclaration des logiciels à déployer
- Notion de personnalité logiciel
  - Programmation / scriptage des procédures de déploiement
    - Installation, configuration, démarrage, arrêt, déinstallation
  - Spécifique à chaque logiciel

FDF - Déclaration du système

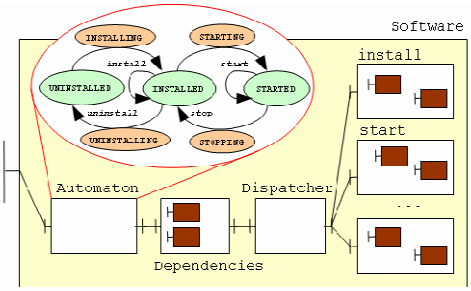
```
Hosts = INTERNET.NETWORK {
    horse = INTERNET.HOST {
      hostname = INTERNET.HOSTNAME(horse);
      user     =
        INTERNET.USER(dfournie,,/home/dfournie/.ssh/id_rsa.pub);
      transfer = TRANSFER.SCP;
      protocol = PROTOCOL.OpenSSH;
      shell    = SHELL.SH;
      software {
        java = JAVA.JRE {
          archive = JAVA.ARCHIVE(/media/sda5/fdf/lib/jdk-1.5.tgz);
           home   = JAVA.HOME(/home/dfournie/fdf/jdk1.5.0_09);
        }
      }
    }
}
```

FDF - Déclaration des logiciels

```
Tomcat-Servers {
  tomcat-horse = TOMCAT.SERVER {
    archive    =
         TOMCAT.ARCHIVE(/media/sda5/fdf/lib/apache-tomcat-5.5.23.tar.gz);
    home       = TOMCAT.HOME(/home/dfournie/fdf/apache-tomcat-5.5.23);
    host       = Hosts/horse;
    properties {
      http-port = HTTP.PORT(8080);
      http-user = HTTP.USER(admin);
      http-password = HTTP.PASSWORD(null);
    }
  }
}
Servlets {
  sca-print-ws  = TOMCAT.WAR.COMPONENT {
    archive = TOMCAT.WAR.ARCHIVE(/media/sda5/fdf/lib/print-ws.war);
    name    = TOMCAT.WAR.NAME(print-ws);
    tomcat  = Tomcat-Servers/tomcat-horse;
  }
}
```

FDF - Personnalités

- Composant réifiant un logiciel à déployer
- Hérite du composite **`Software`** fourni par le framework



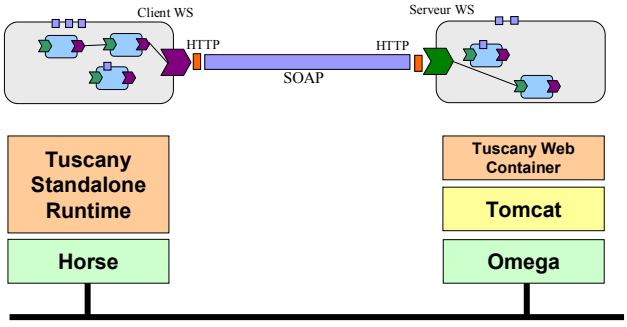
- De nombreuses personnalités existantes : Ant, J2EE, JOnAS, JAVA, Apache Tomcat, etc.

FDF - Composants primitifs

- Ensemble de composants exécutables ou bas niveau
- Composants exécutables fournissent les « fonctionnalités de bases » pour l'upload/download, exécution de commandes dans le shell, …
- Composants bas niveau représentent de manière abstraite les mécanismes d'accès aux machines hôtes

Déploiement de systèmes SCA distribués

- Utilisation de FDF pour déployer un système SCA



- Pré requis : JAVA, Tomcat (Web Service), ***Runtime Standalone Tuscany***

Déploiement de systèmes SCA distribués

Personnalité Tuscany - Utilisation

Description du Runtime SCA

```
tuscany-sca-horse = TUSCANY-SCA.SOFTWARE {
    archive = TUSCANY-SCA.ARCHIVE(/media/sda5/fdf/lib/tuscany-sca-1.0-
incubator-M2-bin.tgz);
    home = TUSCANY-SCA.HOME(/home/dfournie/fdf/tuscany-sca-1.0-incubator-M2-
bin);
    host = Hosts/horse;
}
```

Description du client

```
sca_client = TUSCANY-SCA.MODULE(/media/sda5/fdf/lib/sample-converter.jar,WS)
{       tuscany-sca = /tuscany-sca-horse;
        dependencies {
          start-when-install {
            Servlets/sca-print-ws;
            Servlets/sca-converter-ws;
            /tuscany-sca-horse;
          }
        }
}
```

Déploiement de systèmes SCA distribués

Personnalité Tuscany - Runtime

```
TUSCANY-SCA.SOFTWARE = software.Installable(tuscany-
   sca,TUSCANY SCA), JAVA.DependOn(tuscany-sca),
   software.Hosting(tuscany-sca)
{
  # archive is required.
  archive = TUSCANY-SCA.ARCHIVE(UNDEFINED);
  # home is required.
  home = TUSCANY-SCA.HOME(UNDEFINED);
  tuscany-sca-applications = TUSCANY-
   SCA.HOME(#[home]/applications,Applications);

  # The 'configure' procedure.
  configure {
       SHELL.MakeDirectory(#[tuscany-sca-applications]);
  }
  # The 'start' procedure.
  start {
  }
  # The 'stop' procedure.
  stop {
  }
}
```

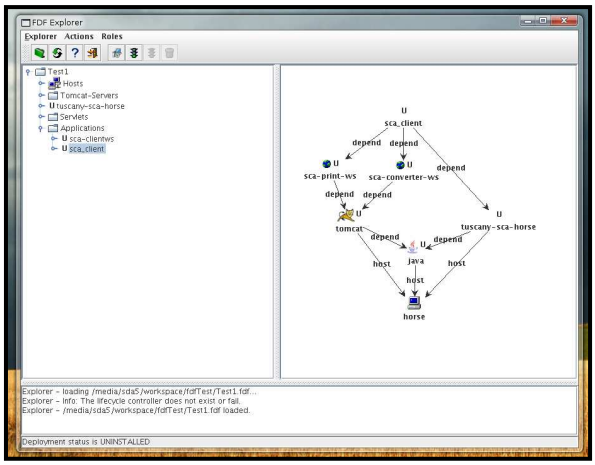
Déploiement de systèmes SCA distribués

Personnalité Tuscany – Standalone

```
TUSCANY-SCA.MODULE-SERVER(archive)= software.Hosted(module-
   sca,TUSCANY SCA ${execution-mode} Module (#[archive-
   filename]),tuscany-sca)
{
     archive = TUSCANY-SCA.ARCHIVE(${archive},${execution-
   mode} Module);
     # The 'install' procedure.
     install {
       TRANSFER.Upload(#[archive],#[tuscany-sca-
   applications]/#[archive-filename]);
     }

     # The 'start' procedure.
     start {  . . . }

     # The 'stop' procedure.
     stop { . . . }

     # The 'uninstall' procedure.
     uninstall {
       SHELL.RemoveFile(#[tuscany-sca-applications]/#[archive-
   filename]);
     }
}
```

Déploiement de systèmes SCA distribués

Vue globale du système



- **Visualisation :**
- Dépendances
- Etat des composants logiciels

- **Interaction :**
- Installation, exécution, désinstallation avec résolution des dépendances

TUNE

Toulouse

Tune: Définition des wrappers

- Langage de définition des wrappers
- Wrapper = composants Fractal permettant le contrôle d'un logiciel
- Permet d'exporter des interfaces de contrôle

42

Tune: Définition des wrappers

la.xml - Bloc-notes

Fichier Edition Format Affichage ?

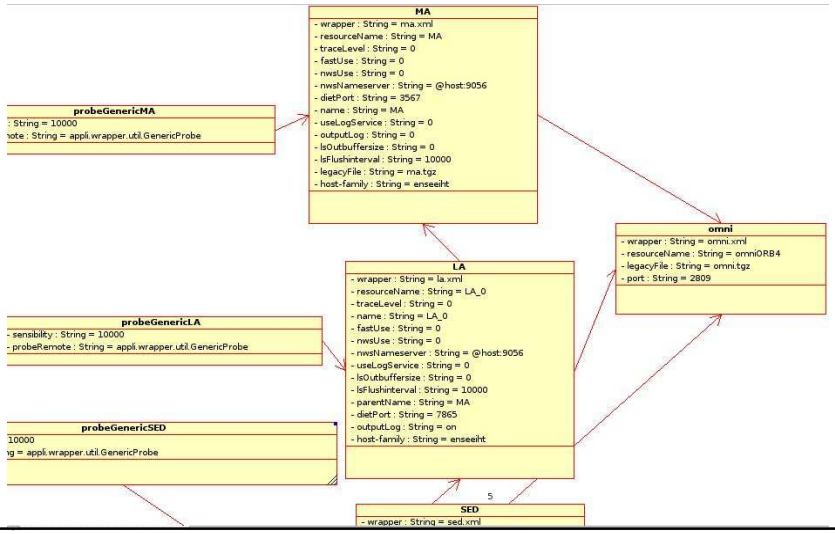
```
<?xml version='1.0' encoding='ISO-8859-1' ?>
<wrapper name='la'>

        <start>
                        <param value="dirLocal + / + dietAgent + __SPACE + dirLocal + / + resourceNa
                        <param value="LD_LIBRARY_PATH= + dirLocal + / "/>
                        <param value="OMNIORB_CONFIG= + dirLocal + /omniORB4.cfg" />
        </start>


        <stop/>

        <method name="myBinding" key="appli.wrapper.util.LiaisonPere" method="bindAgent" >
                <param value="type + __SPACE + DIET_LOCAL_AGENT + __SPACE + adresseOMNI + __SPACE
        + 2809 + __SPACE + name"/>
        </method>

        <linkfc-controler name="adresseOMNI" method="getAttribute"
        interface="attribute-controller" key="wrapperitf-client_omni_0">
                        <param value="nodeName"/>
        </linkfc-controler>


        <!-- modifs à faire dans fichier config avant démarrage appli -->
        <updateconfigfile path="dirLocal + / + resourceName + .cfg" type="plaintext"
         separator=" = ">
                <config-file-attribute name="traceLevel" linkattribute="traceLevel" />
                <config-file-attribute name="agentType" value="DIET_LOCAL_AGENT"/>
                <config-file-attribute name="parentName" linkattribute="parentName"/>
                <config-file-attribute name="name" linkattribute="name"/>
                <config-file-attribute name="dietHostname" linkattribute="nodeName"/>
                <config-file-attribute name="fastUse" linkattribute="fastUse"/>
                <config-file-attribute name="nwsUse" linkattribute="nwsUse"/>
                <config-file-attribute name="nwsNameserver" linkattribute="nwsNameserver"/>
                <config-file-attribute name="useLogService" linkattribute="useLogService"/>
                <config-file-attribute name="lsOutbuffersize" linkattribute="lsOutbuffersize"/>
                <config-file-attribute name="lsFlushinterval" linkattribute="lsFlushinterval"/>

        </updateconfigfile>

</wrapper>
```

Tune: Description d'une architecture

- ***Diagramme de classe dans un profil UML***
  - Plus intuitif
- ***Description par intension vs extension***
  - Décrit un patron d'architecture
  - Donne le nombre de serveurs lancés

44

Tune
Description d'une architecture



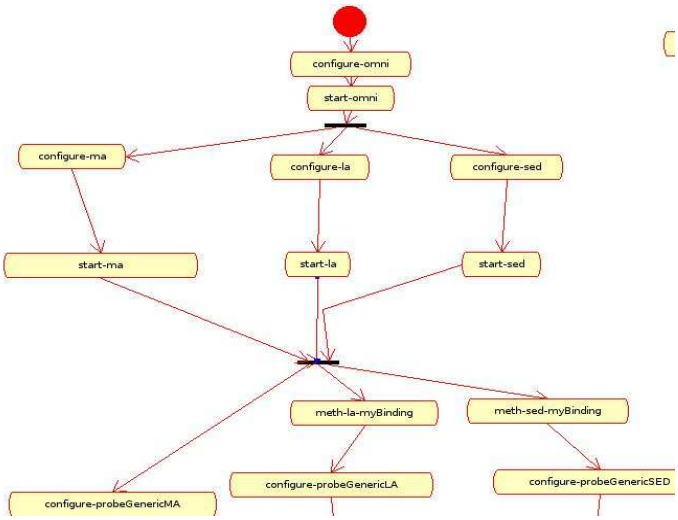
45

Tune: Description des reconfigurations

- ***Diagramme d'état dans un profil UML (automate)***
  - Respecte le patron (cohérence de l'architecture)
- ***Décrit un workflow d'opérations sur les élements du diagramme de classes***
  - ***Utilise les interfaces de contrôle des wrappers***
- Exemples simples
  - Workflow de démarrage de l'architecture
  - Réparation sur panne d'un serveur
- Travail sur la sémantique de ces diagrammes (de classe et d'état)

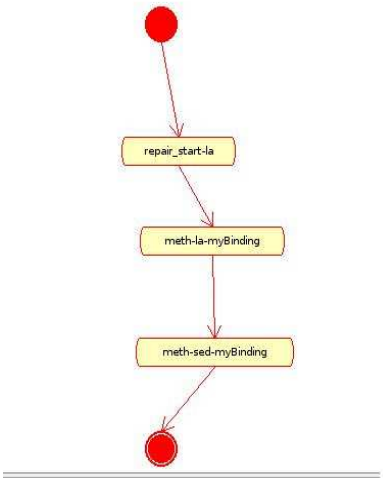
46

Tune: Workflow de démarrage



47

Tune: Workflow de réparation

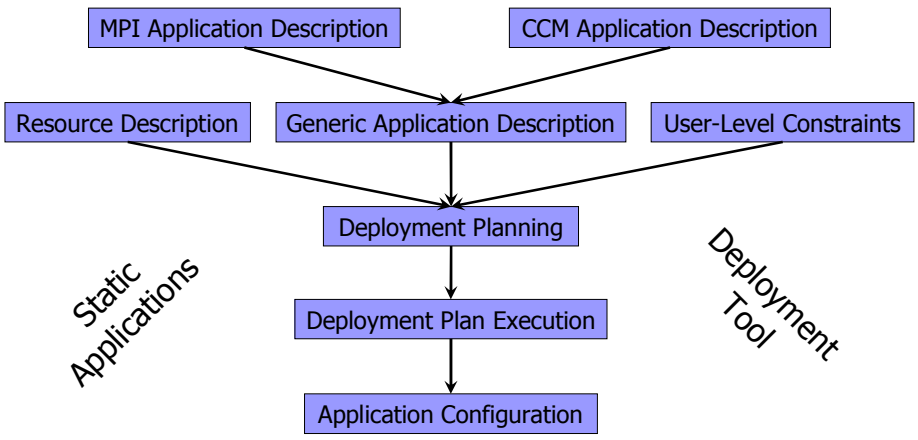


48

ADAGE

PARIS & GRAAL INRIA Project Team

Identification of the Steps of Automatic Deployment



Deployment Planning

- Heart of automatic deployment process
  - Makes every single decision
- Information needed in input
  - Description of available resources
  - Description of the application
  - Optional user-level constraints
- Output
  - Deployment plan
  - List of all actions to perform

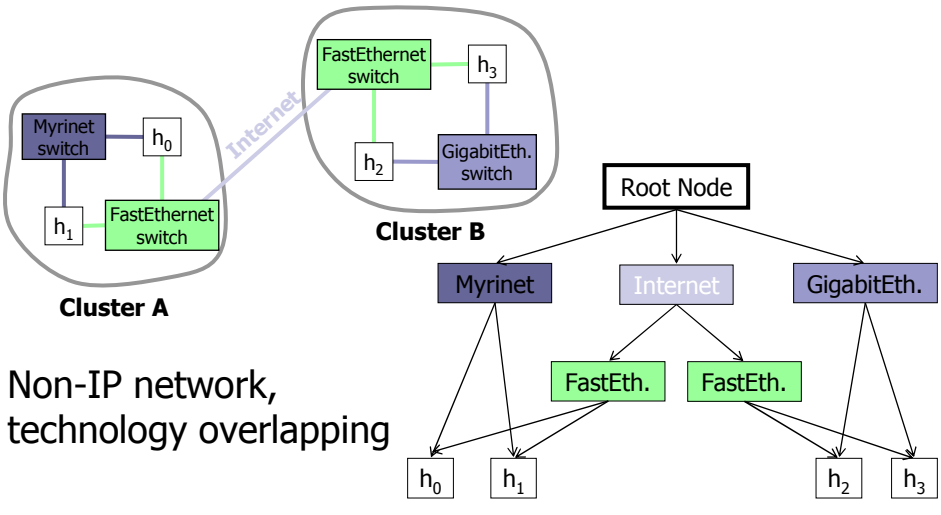
Grid Resource Description

- Compute & storage node description
  - Well mastered: Globus MDS, etc.
- Processes distributed over a grid do communicate with each other
  - More or less intensively, critically
- Grid network description
  - Performance characteristics (NMWG, GGF)
  - Network topology **between** nodes? Hum...

Network Topology Description

- Latency and bandwidth hierarchy
  - Clusters (SAN)
  - Sites (LAN)
  - National WAN
  - Intercontinental WAN
- NAT, non-IP networks, etc.

Example of Network Description

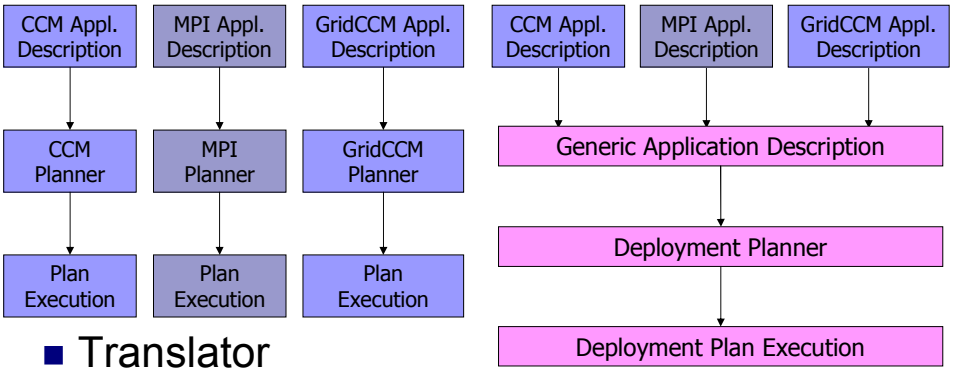


Non-IP network, technology overlapping

Generic Application Description

- How to deploy a parallel component?
  - Using a parallel technology (MPI)
  - Using a distributed technology (CCM)
- Deployment planner: complex to write
  - Implement it once and re-use it

Generic Application Description

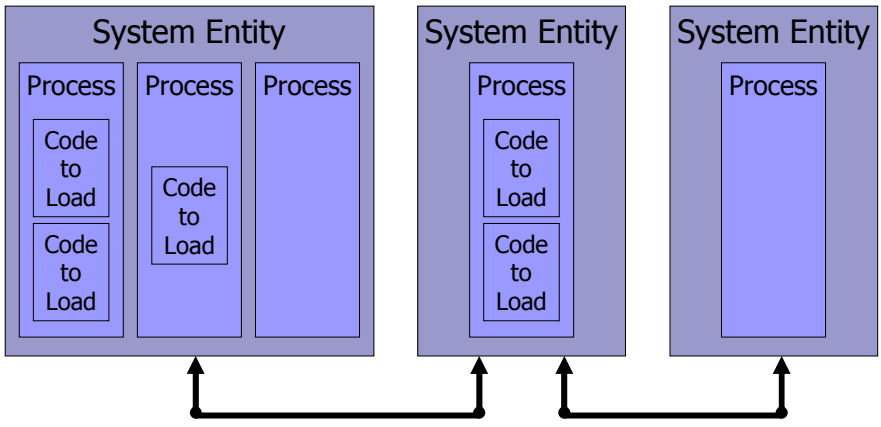


- Translator
  - From specific to generic application description
  - Straightforward to write

Generic Application Description

- All applications amount to
  - Processes
    - Location constraints (host collocation)
    - Launch mechanism (plain exec., JavaVM, MPI)
  - Codes to load into processes
    - Location constraints (process collocation)
    - Load method
  - Connections
    - Between communicating processes

Example of Generic Application Description



GADe Extension

- Enable to describe a general graph between process
- Enable recursive node
  - To support DIET architecture

User-Level Constraints

- Keep a certain level of control on the automatic deployment process
- User's comfort
  - Minimize execution time (select fastest CPUs)
  - Run the application close to a visualization site
  - Do not split the application too much
  - Specify input data/parameters

Deployment Planner

- Select resources
  - Computers and launch methods
  - Networks
- Place application processes on selected resources
- Select application's compiled executables
  - Check OS/architecture compatibility

Deployment Plan Execution

- Every single decision made by the planner
- Upload and install files
  - Compiled executable selected by the planner
  - Input data requested by the user
- Launch processes remotely
  - Various job submission methods

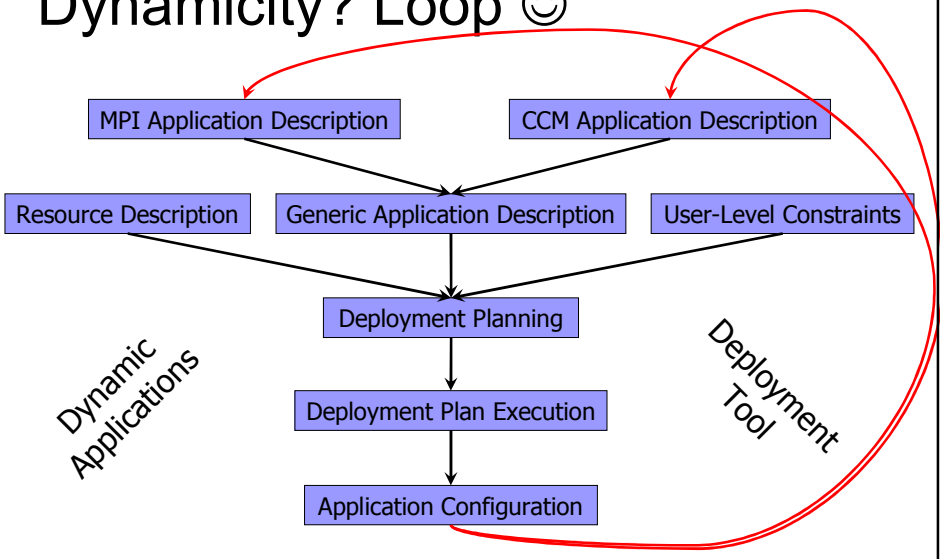
Application Configuration

- Once processes have been launched
  - Set parameters of components, interconnect components
    - OpenCCM can do that (Jacquard, LIFL, Lille)
  - Provide network topology information
    - Topology-aware MPI or DSM implementations
    - Minimize communications on slow network links
    - Optimize collective operations
    - Improve synchronization operations

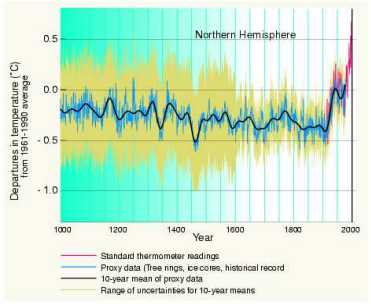
How Easy is it, in Practice?

- As simple as A-B-C
  - adage_grid_deploy –resource http://host/file.xml
    –application my_appl.zip
    –usr_lvl my_usr_lvl_contraints
- Grid resource description
  - Written once for all by grid admins
- Optional user-level constraints
  - Keep control on the deployment process

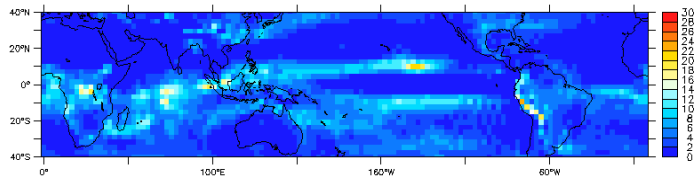
Dynamicity? Loop ☺



Deploying a Meteorological Application (CERFACS)



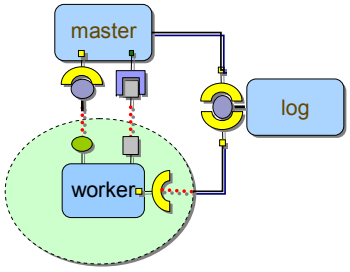
- Parametric application of workflow of a couple simulation

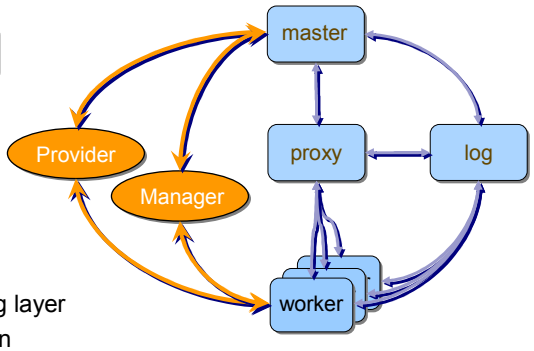


Deploying a Meteorological Application (CERFACS)

	OASIS	ARPEGE 4 processeurs	OPA	TRIP	Total
CPU (sec)	200	1500/p	1500	200	7900
Elapsed (min)	3	25	25	3	25
Max memory (Mo)	350	200/p	600	20	600
Static memory (Mo)	2.8	153	562	1.5	720
Inputs (Mo)	170	2	190	1	360
Outputs (Mo)	3	190	130	1	320
MPI comm. (Mo)	240	62	136	40	240

Deploying a Meteorological Application (CERFACS)





- "Simple" scenario
  - JUXMEM as data sharing layer
  - CCM Proxy based pattern

Deploying a Meteorological Application (CERFACS)

- More complex scenario
  - JUXMEM
    - as data sharing layer
  - DIET based pattern