Framework for managing large scale component-based distributed applications using JMX

HP Laboratories

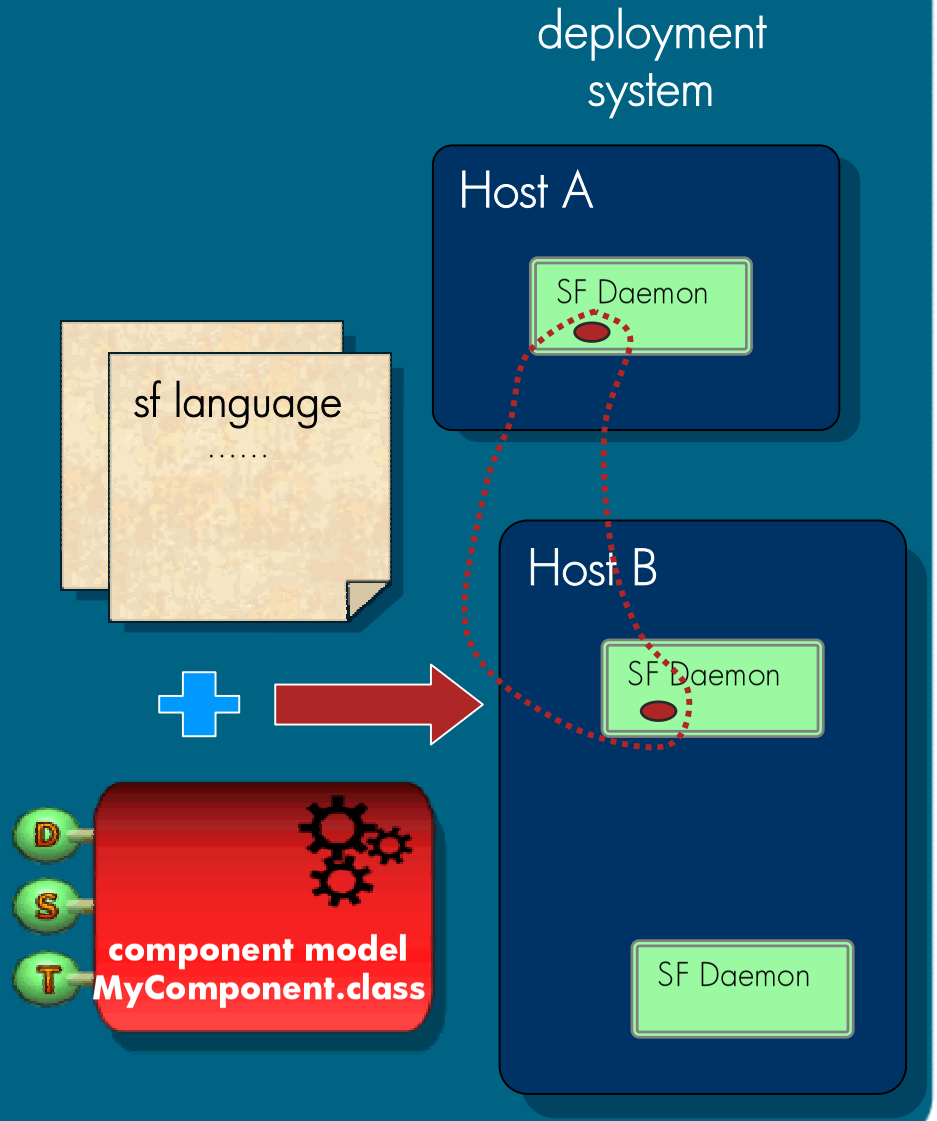
Julio Guijarro, Manuel Monjo, Patrick Goldsack

Contents

- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

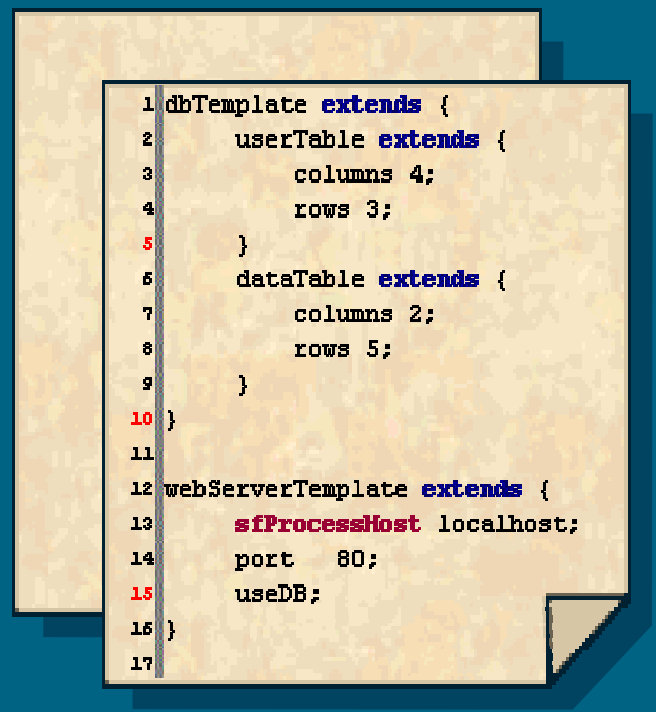
The smartfrog system

- smart framework for object groups
- a flexible framework for the construction of *configuration-driven systems*
- realized as:
  - a ***language*** for describing system configurations
  - a ***component model*** defining the interfaces that components provide to implement the **component lifecycle**
  - a ***deployment system*** for reifying the descriptions and managing the running systems through their lifecycles



Smartfrog Notation

- configuration data is organised as hierarchical *attribute:value* pairs
- supports inheritance (through *prototyping*)
- supports templates and parameterisation
- supports flexible variable linking and component binding
- supports a 'transformation framework' to extend the core language
  - functions, validations, ...
  - code written in Java, e.g. linking to databases, ...
  - defined as attributes, linked to inheritance

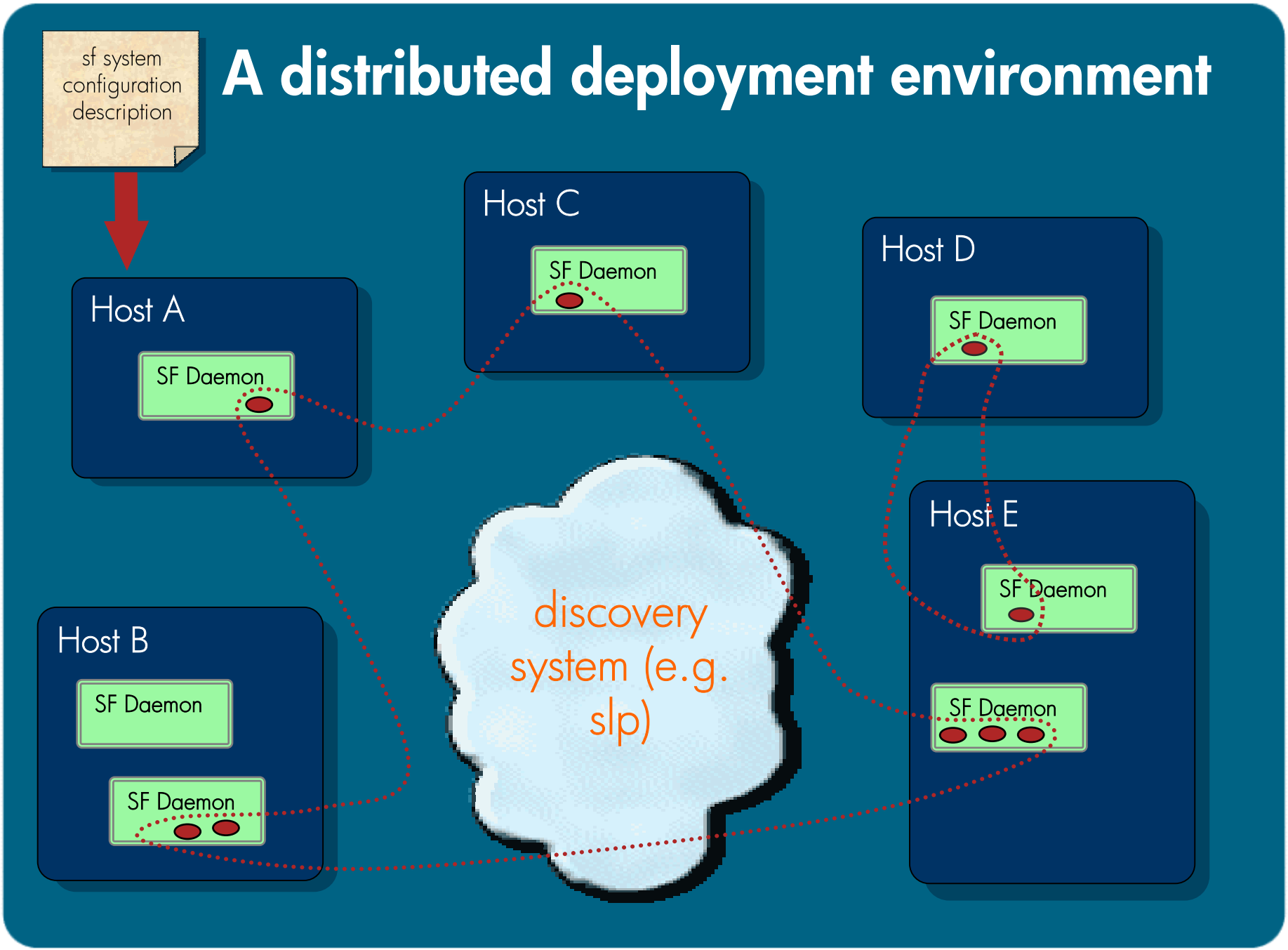


```
1  dbTemplate extends {
2       userTable extends {
3            columns 4;
4            rows 3;
5       }
6       dataTable extends {
7            columns 2;
8            rows 5;
9       }
10 }
11
12 webServerTemplate extends {
13      sfProcessHost localhost;
14      port   80;
15      useDB;
16 }
17
```

Smartfrog component model

- the notation can be used to describe “components” using attributes
  - their code
  - their location
  - their configuration data
- a component
  - has specified management interfaces
  - implements a pre-defined lifecycle
  - is written in Java (wrapping code in other languages)
  - has full access to all smartfrog APIs

A distributed deployment environment



Contents

- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

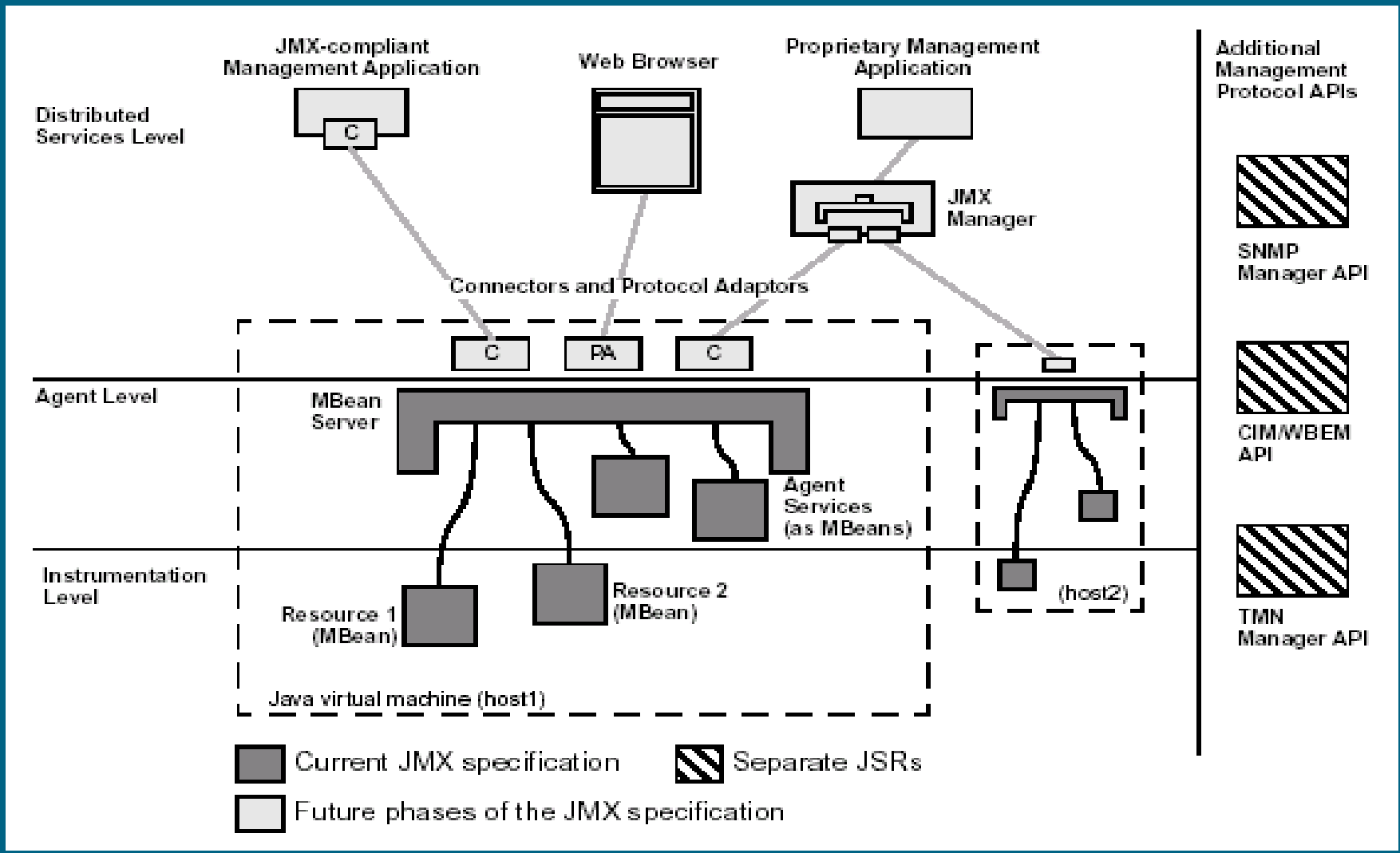
What is JMX?

- Javax Management eXtensions
- Define an architecture, the design patterns, the APIs, and the services for application and network management in the Java programming language.
- Aimed to be an unified framework to instrument heterogeneous Java code and integrate dispare existing management technologies

Benefits

- Enables Java applications to be managed without heavy investment
- Provides a scalable management architecture
- Integrates existing management solutions
- Can leverage other Java technology
- Can leverage other management concepts (e.g. lookup and discovery)
- Defines only interfaces necessary for management

Architecture overview of JMX

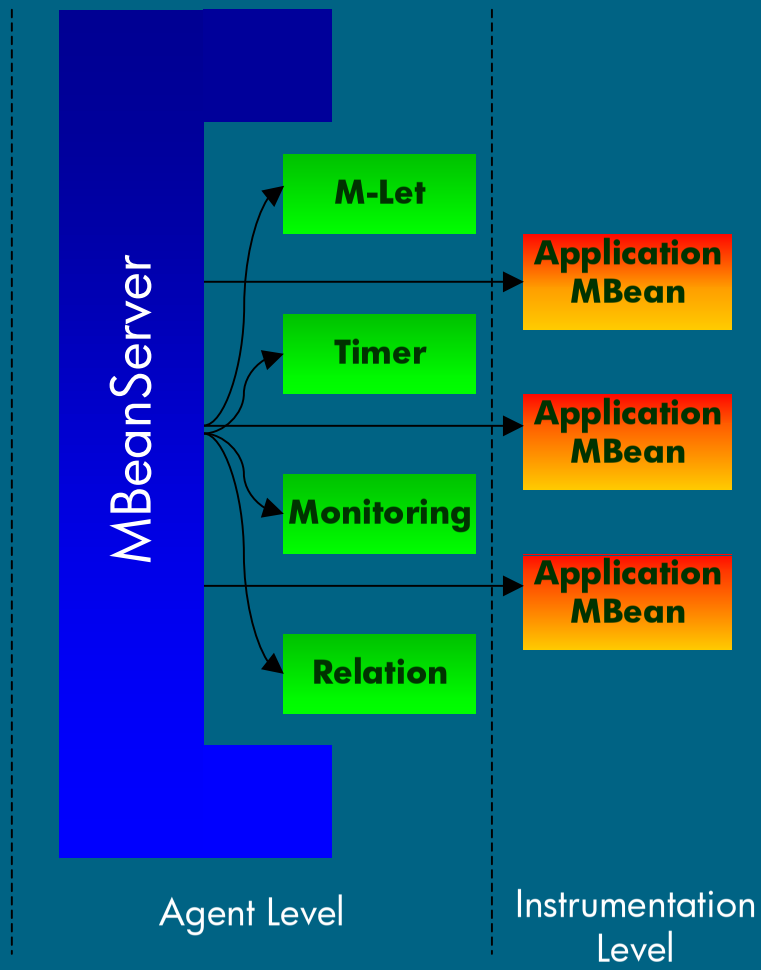


Instrumentation level

- Manageable Beans**:** components implementing a management interface
- Managed resources conform MBean patterns or are represented by Mbeans used as wrappers or proxies.
- Four types as for the way they expose their management interface:
  - **Standard Mbean:** implements a static Java interface
  - **Dynamic Mbean:** implements a generic interface to allow access a set of metadata classes . Special cases:
    - **OpenMBean:** restricts the set of objects types used in the management interface. Not fully specified and not required in specification v1.0
    - **ModelMBean:** additional descriptors specifying behavioural policies (persistence, caching, protocol mapping, transactions, security and so forth)

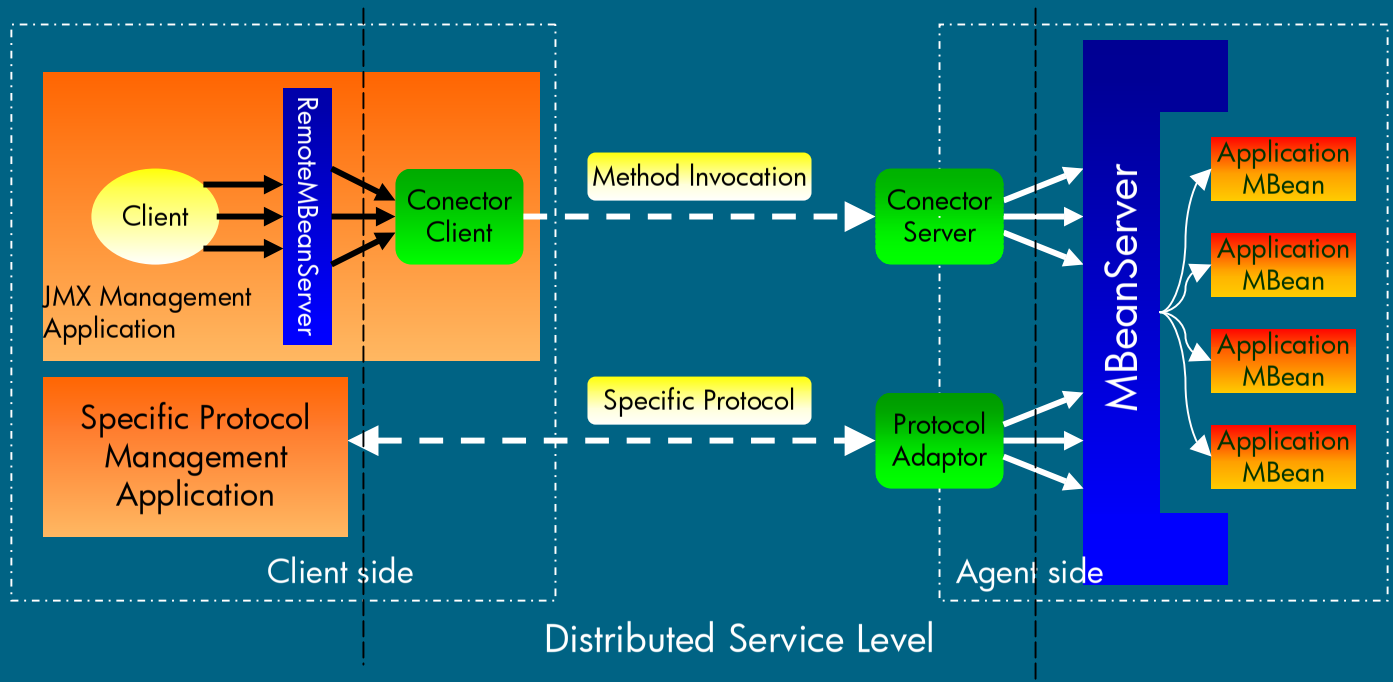
Agent level

- The JMX Agent is the access point to managed resources for management applications
- Architecture built upon Instrumentation level:
  - **MBeanServer:** corner stone acting as a repository for MBeans
  - **Set of services:** designed as MBeans.
    - **M-Let:** allows MBeans to be loaded from the network at runtime
    - **Timer:** scheduler that sends notifications at given times or at intervals
    - **Monitoring:** observes MBeans attributes and send notifications on changes
    - **Relation:** allows association between MBeans maintaining consistency
- Allows object to perform actions on MBeans using object names:
  - Getting/setting their attributes
  - Invoking operations
  - Create and register new MBeans
  - Get notifications emitted by any Mbean



Distributed services level

- Allows remote applications to access the JMX Agent. Not specified yet.
- Remoteness achieved by the following types of components:
  - **Connectors:** split in two components specific to a certain protocol
    - **Connector Server**: MBean able to receive remote method invokations
    - **Connector Client**: provides a remote view of the MBeanServer and invoke operations transparently to the user
  - **Protocol Adaptors:** adapt operations of the MBeanServer into a representation of a given protocol and possibly into a different information model (e.g. SNMP, CIM)



JMX Limitations

- Most implementations sharing JVM with applications
- Present specification does not scale.
- No specified way to describe and deploy large JMX instantiations
- No discovery mechanisms for JMX Agents or Applications
- Smartfrog hierarchical data types need to be adapted to JMX

Contents

- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

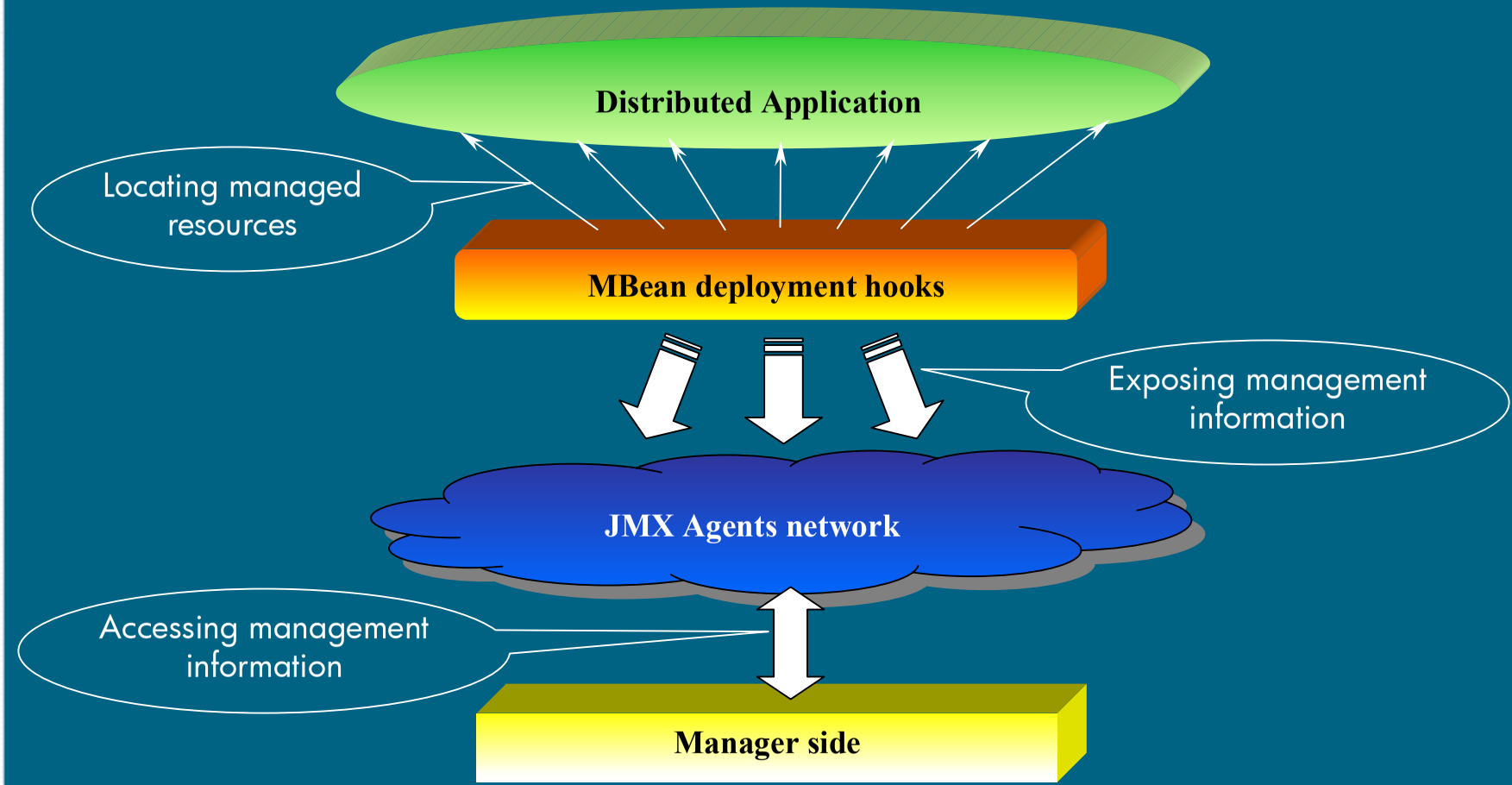
Requirements

- Main motivation:
  - Be able to deploy a infrastructure for managing large scale applications
- With a minimum of desired characteristics:
  - **Ease of use**: exposing an application to be managed should be as transparent as possible for the application designer
  - **Scalability**: management infrastructure should be able to grow according to our needs without impacting performance and ease of management
  - **Flexibility**: should be fully configurable to fit the management needs of the applications

Contents

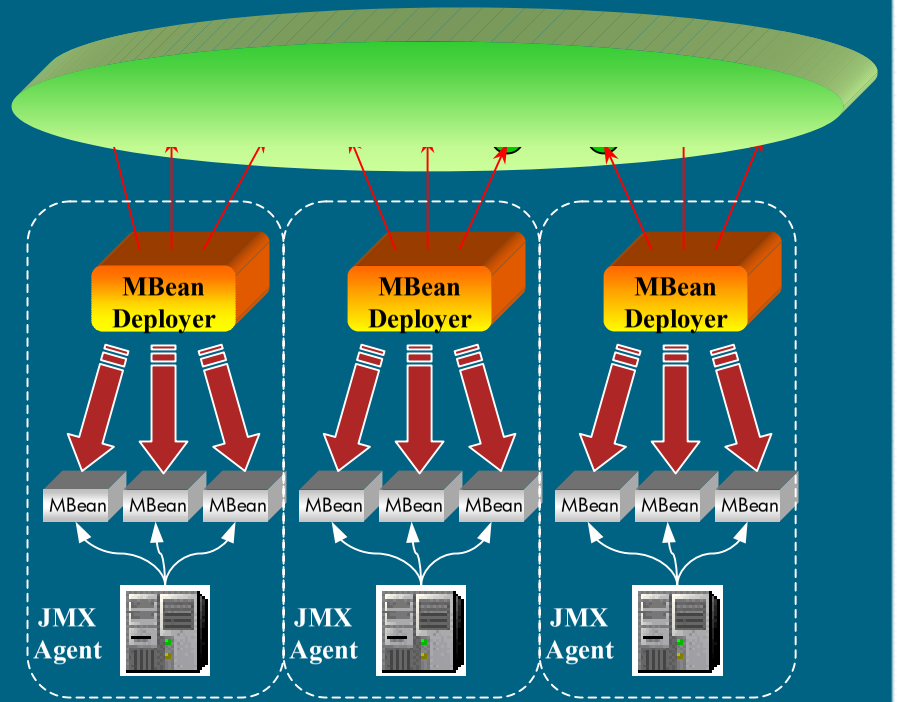
- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

Management framework overview



MBean deployment

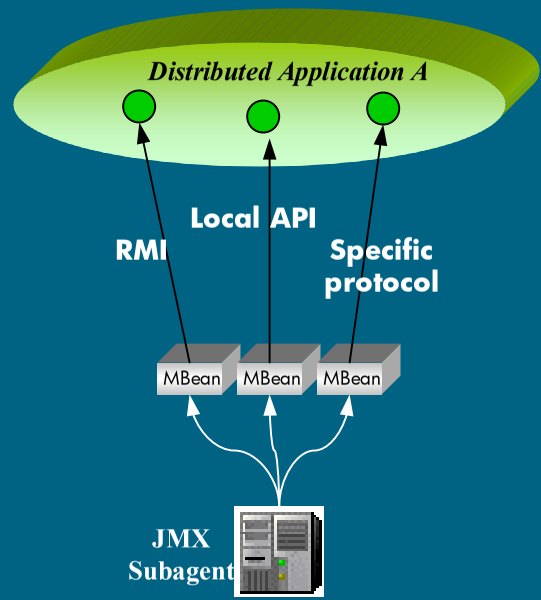
- Prevent our Java code from being aware of the JMX infrastructure
- It is performed by a set of MBean deployers
- MBean deployment phases:
  1. Look up a JMX Agent and binds to it
  2. Get a managed resource (component)
  3. Build an appropriate MBean
  4. Register MBean within the JMX Agent
- Deployers produce 3 types of MBeans:
  - Standard MBeans
  - ModelMBeans
  - PrimDynamicMBeans
- Flexibility given by SmartFrog configuration




```
MBeanDeployer extends Compound {
  sfClass "com.hp.sfServices.jmx.deployer.MBeanDeployer";
  sfAgentAddress LAZY HOST localhost:rootProcess:sfJMXAgent;
  sfMBeans extends Compound {
    mbean LAZY component;
  }
  modelMBeans extends LAZY {
    modelmbean extends modelMBeanDesc;
  }
  primDynamicMBeans extends Compound {
    primmbean LAZY primComponent;
  }
}
```

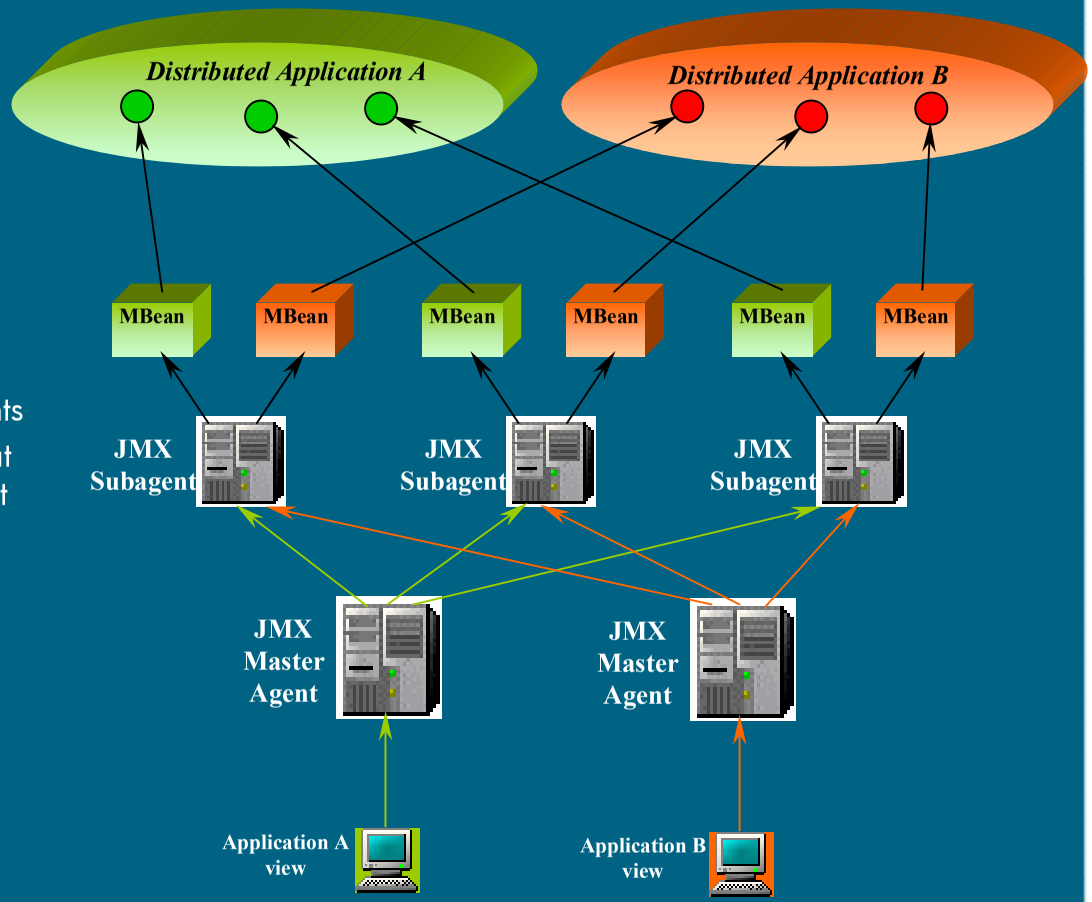
JMX Network topology (I)

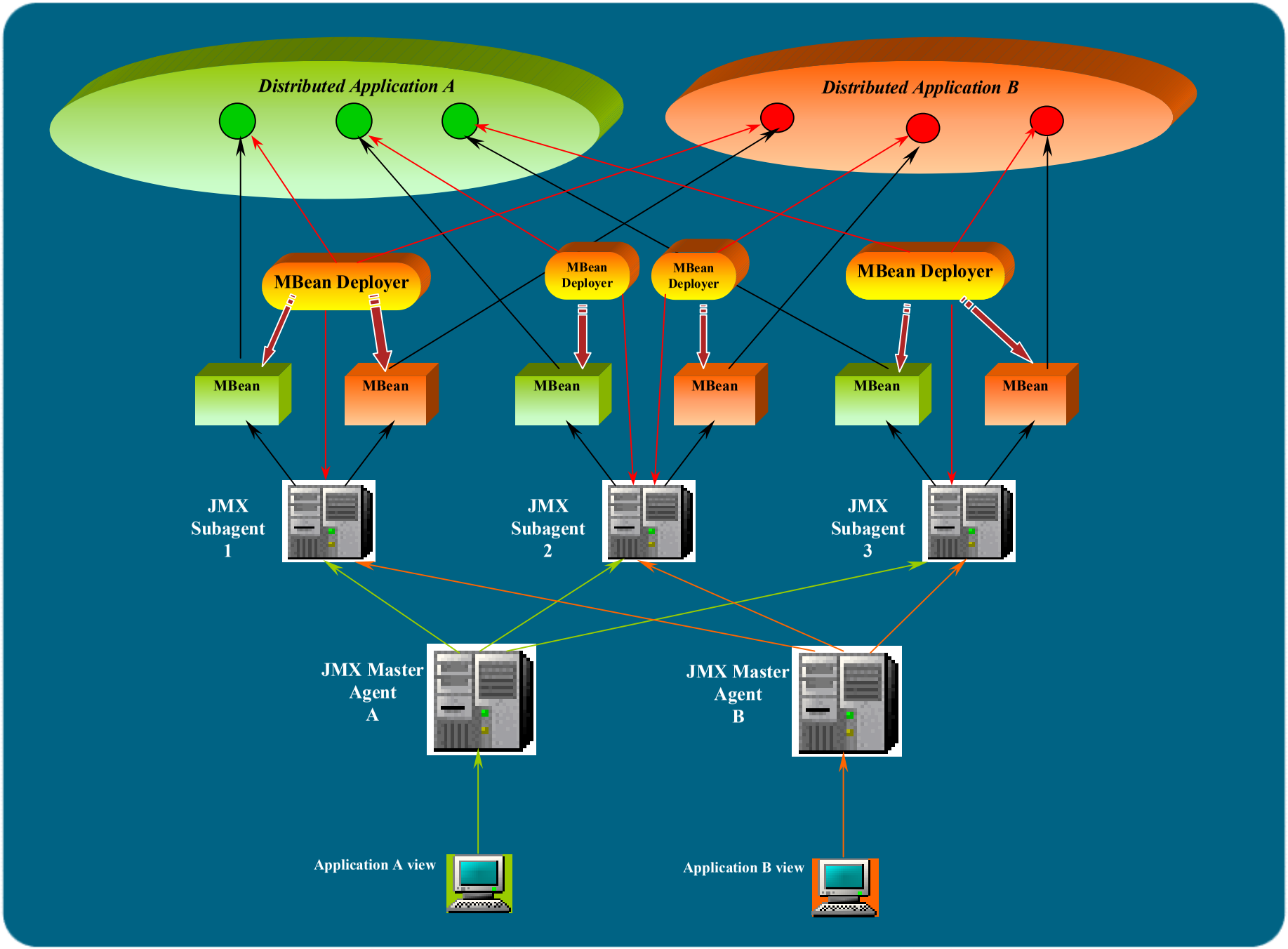
- Single JMX Agent topology:
  - Distributed application and centralized JMX Agent
  - Simple solution
  - Small scalability



JMX Network topology (II)

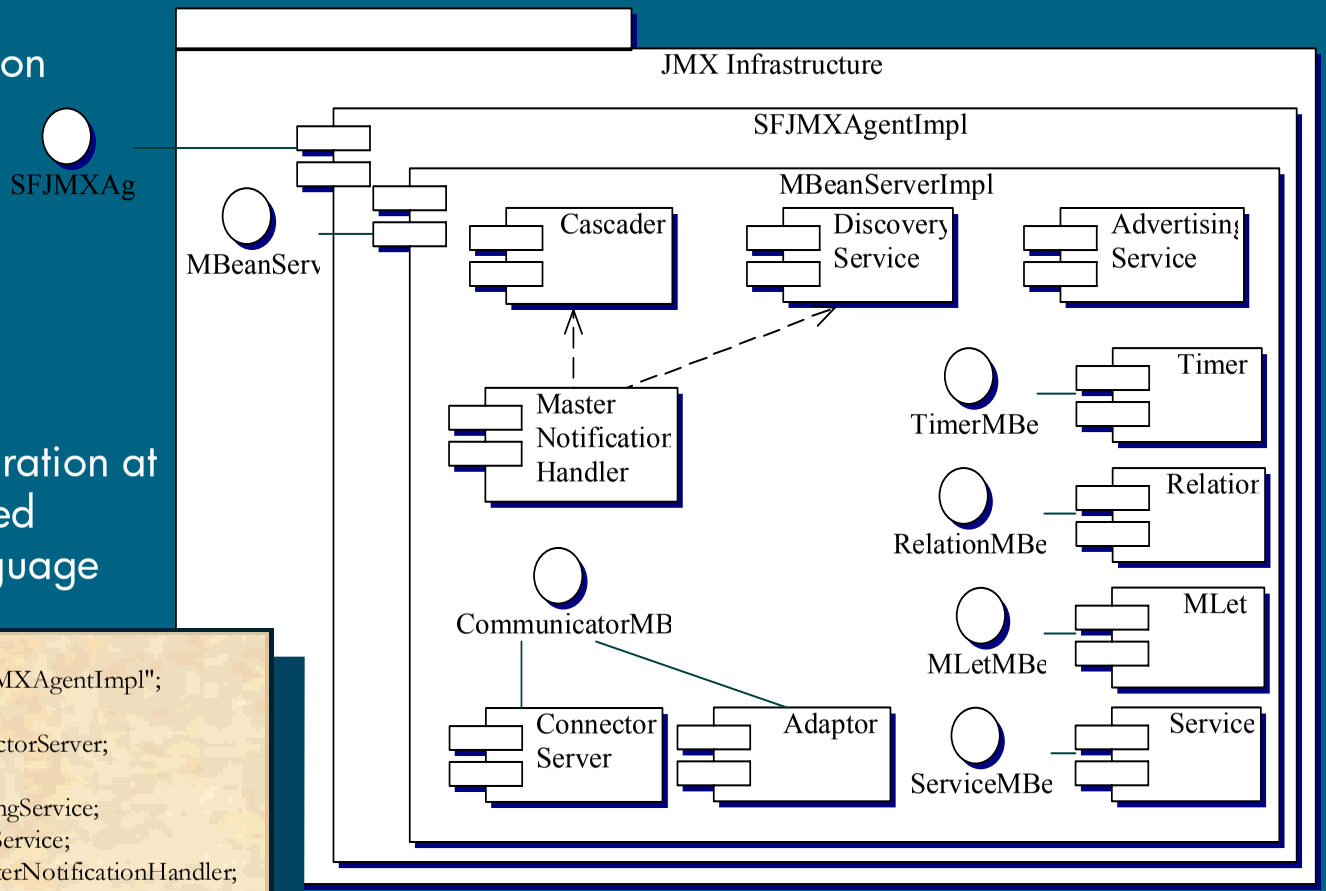
- Distributed JMX Agent topology:
  - Deployment of JMX Agent farms (for instance, one for every SFDaemon)
  - Management information fully distributed across the network
  - Master – Subagent architecture:
    - Cascading different levels of Agents
    - MBean proxy based approach that forwards invokations to final Agent
    - Autodiscovery and mastering at start-up time
  - Federation of JMX Agents
  - Hierarchies of JMX Agents




Distributed Application A
Distributed Application B
MBean Deployer
MBean Deployer
MBean Deployer
MBean Deployer
MBean
MBean
MBean
MBean
MBean
MBean
JMX Subagent 1
JMX Subagent 2
JMX Subagent 3
JMX Master Agent A
JMX Master Agent B
Application A view
Application B view

JMX Agent arquitecture

- Architecture based upon JMX MBeanServer



- Flexibility and configuration at deployment time reached through SmartFrog language

```
SFJMXAgent extends Compound {
  sfClass "com.hp.sfServices.jmx.agent.SFJMXAgentImpl";
  componentMBeans extends Compound {
    rmiConnectorServer extends RmiConnectorServer;
    htmlAdaptor extends HtmlAdaptor;
    advertisingService extends SLPAdvertisingService;
    discoveryService extends SLPDiscoveryService;
    masterNotificationHandler extends MasterNotificationHandler;
  }
  descriptionMBeans extends LAZY {
    xsltProccessor extends XSLTProcessor;
    relation extends RelationService;
    timer extends Timer;
    mlet extends MLet;
  }
}
```

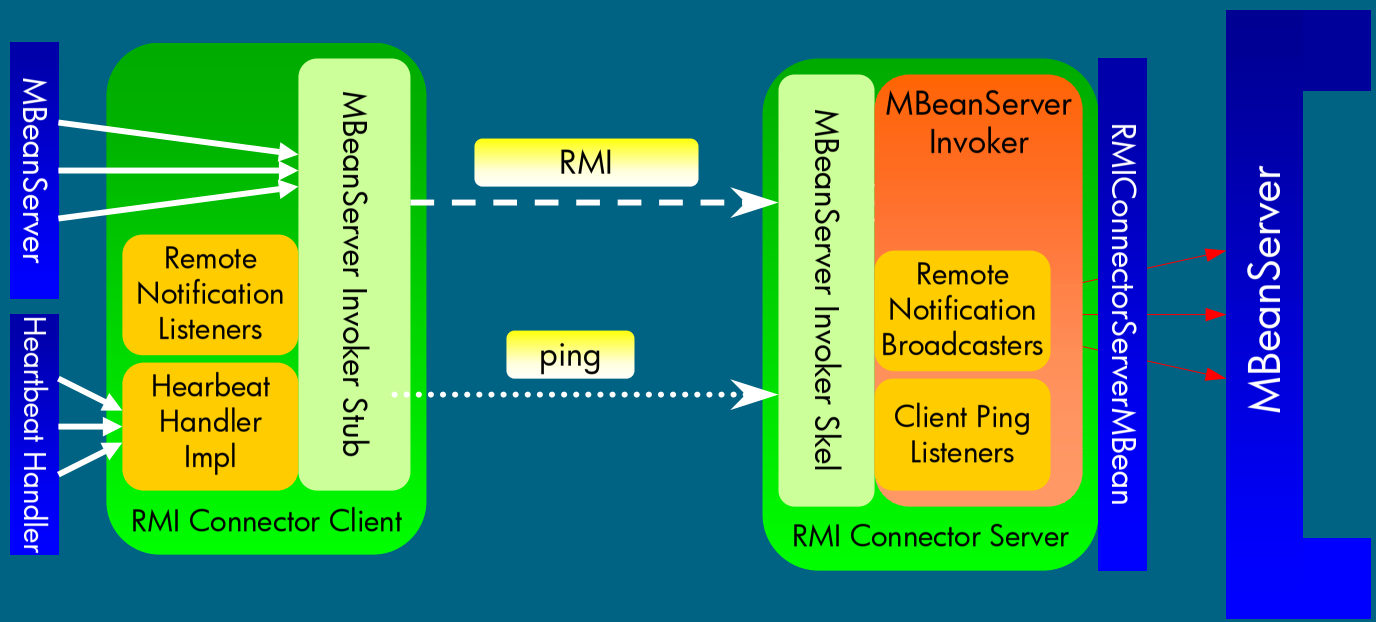
Contents

- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

RMI Connector

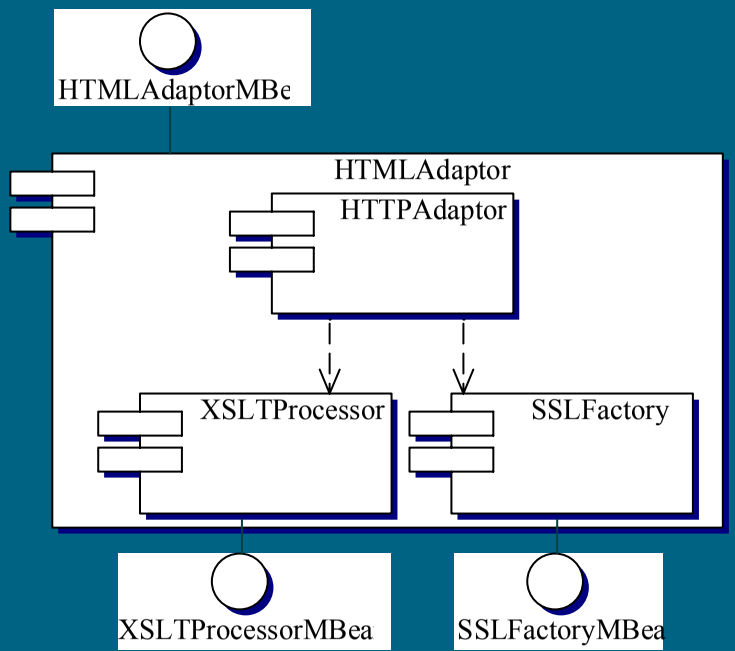
- Constitutes part of the Distributed Service Level enabling remote access to the MBeanServer
- Hides remoteness by offering the same local MBeanServer interface
- Heartbeat mechanism emits Notifications informing about the current state of the connection
- A Remote Notification System allows users to receive remote notifications transparently

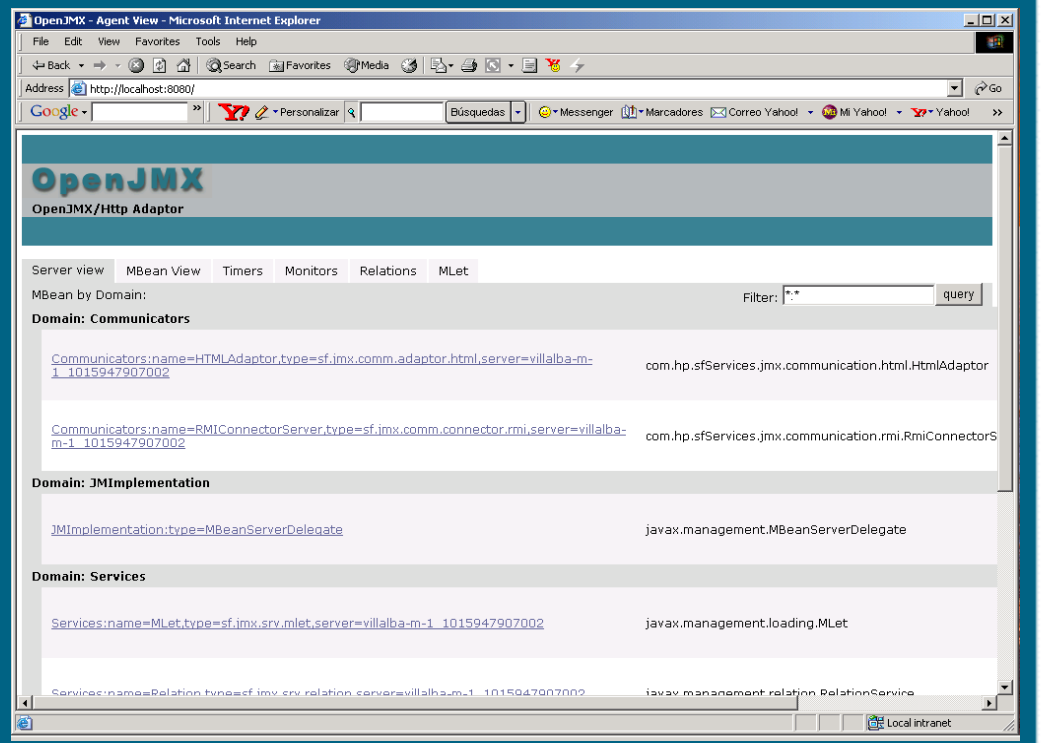
```
RmiConnectorServer extends Communicator {
  sfClass "com.hp.sfServices.jmx.communication.rmi.RmiConnectorServer";
  description "Provides access to the MBeanServer using the RMI protocol";
  domain "Communicator";
  properties:name "RMIConnectorServer";
  properties:type "sf.jmx.comm.connector.rmi";
  port ATTRIB ProcessCompound:sfRootLocatorPort;
  name ATTRIB properties:name;
}
```



HTML Adaptor

- Based on the OpenJMX project
- Fully configurable through SmartFrog
- Features:
  - Authentication (none, basic, diggest)
  - SSL

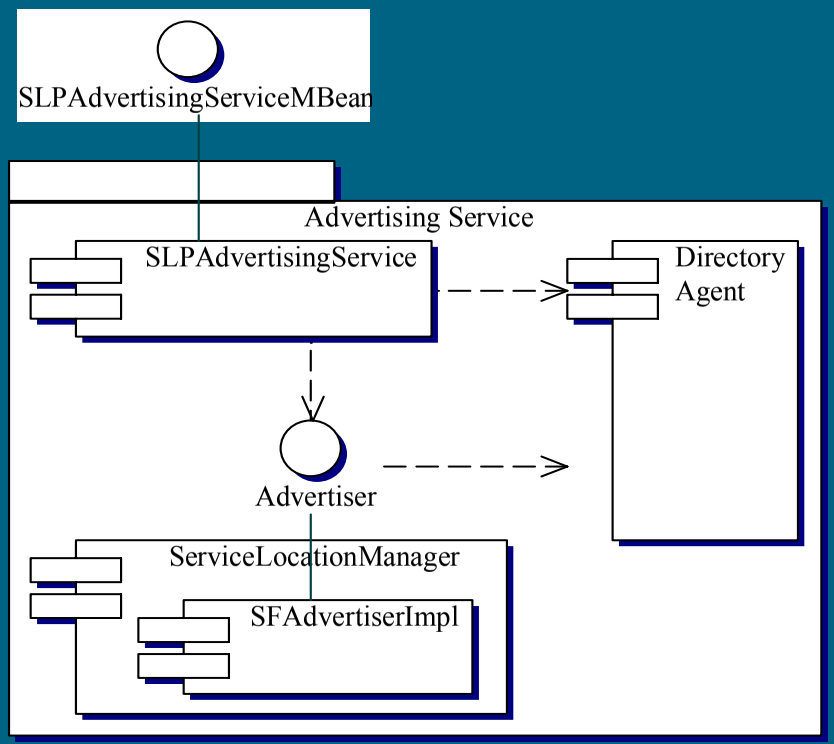


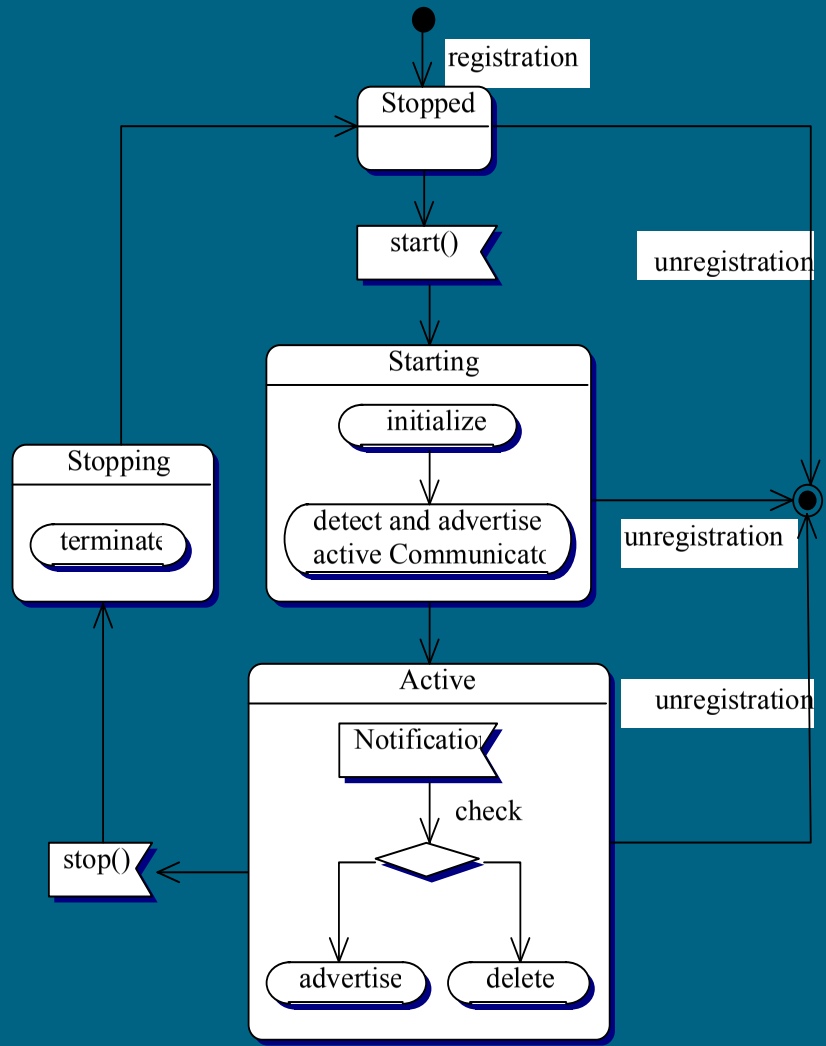


- Components:
  - HTTPAdaptor is a HTTP server that interacts with the MBeanServer
  - XSLTProcessor transform the XML code produced by HTTPAdaptor into HTML
  - SSLFactory provides SSL sockets

Advertising Service

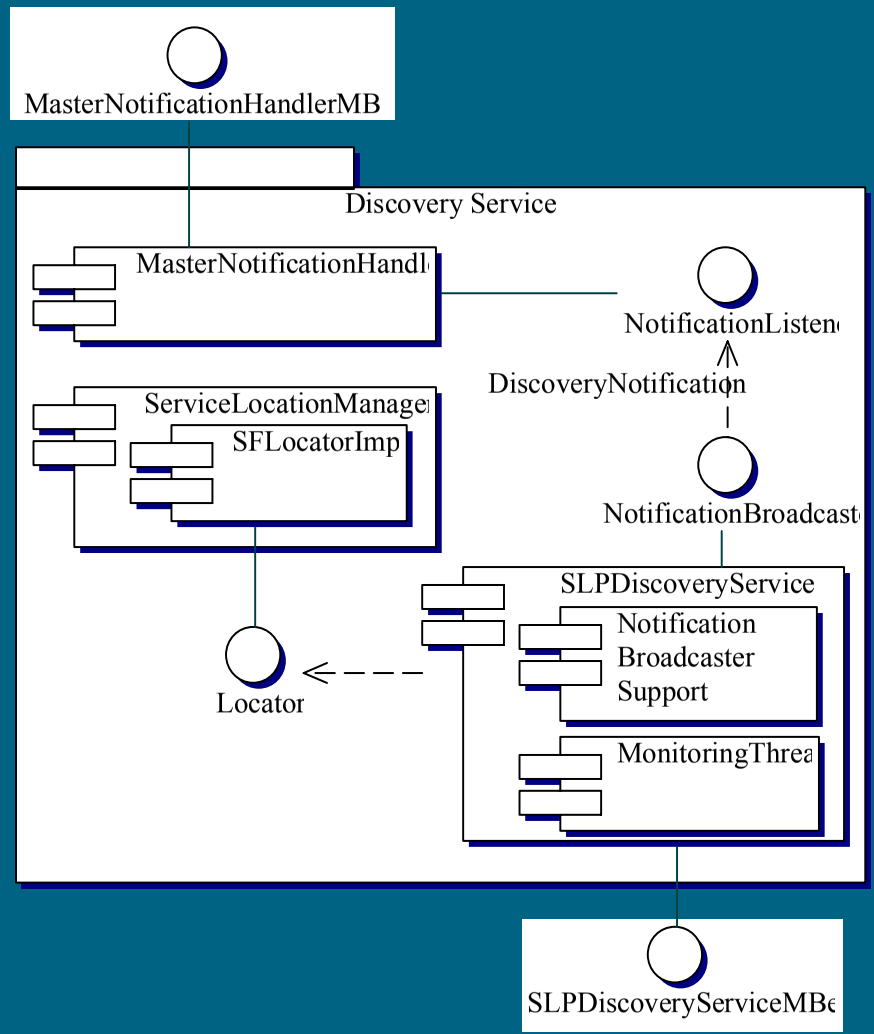
- Based on SLP: DA and SA (Advertiser)
- Designed as MBean
- Detects dynamically all available services
- Keeps a consistent set of service advertisements
- Allows other MBeans to advertise services through its JMX interface





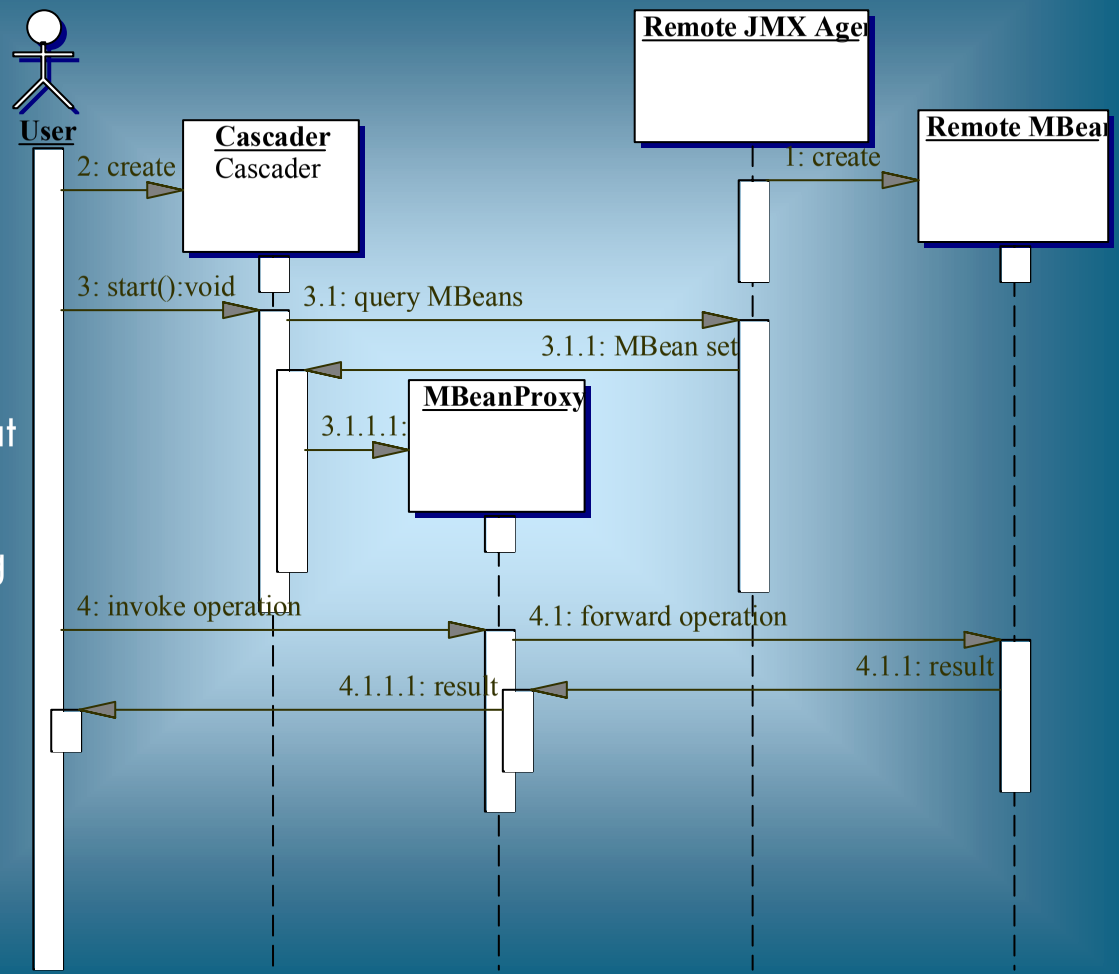
Discovery service

- Based on SLP: UA (Locator)
- Designed as MBean too (of course)
- Allows other MBeans to perform a discovery of JMX Agents
- The Monitoring Thread sends a Discovery Notification whenever an advertisement is registered or deleted in the DA
- Master Notification Handler MBean uses Discovery service to discover new JMX Agents and master them

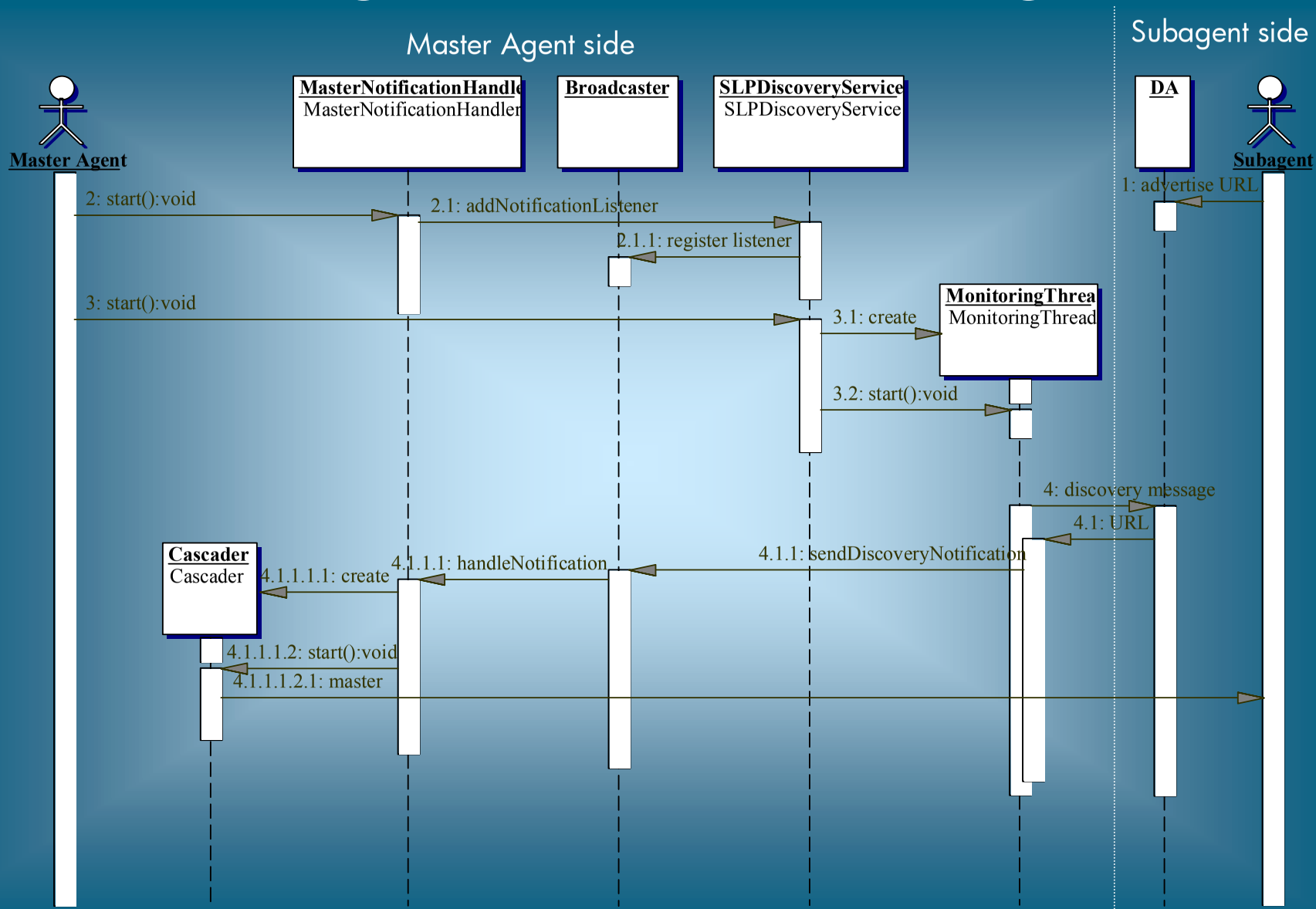


Cascader

- Implements Master Agent - Subagent arquitecture
- Connects to a the Subagent through a Connector
- For every remote MBean, it creates a local MBeanProxy that forwards all invokations
- Synchronizes both Agents using the Notification System



Mastering discovered JMX Subagents



Predesigned MBeans

- **SFModelMBean** basically extends RequiredModelMBean with following improvements:
  - Fully configurable through SmartFrog language by using SFModelMBeanInfoBuilder
  - Extends the managed resource type to RMI references
  - Enables notification broadcasting from the managed resource
  - Fixes several bugs
- **PrimDynamicMBean**
  - Designed specifically for SmartFrog components
  - Introspects and exposes all SmartFrog attributes of the target component at runtime
  - Notifies attribute changes made through its interface
- **MBeanProxy**
  - Dynamic MBean created in a Master JMX Agent to represent an MBean of a remote Subagent
  - Forwards all the invokation to represented MBean

```
ModelMBean extends MBean {
  class "com.hp.sfServices.jmx.modelmbean.SFModelMBean";
  managedResource LAZY targetObject;
  properties:name "SFModelMBean";
  properties:type "sf.jmx.modelmbean";

  fields extends {
    name ATTRIB properties:name;
    displayName ATTRIB name;
    descriptorType "MBean";
    log "True";
    logfile "jmxNotification.log";
    currencyTimeLimit 0;
    persistPolicy "Never";
    persistPeriod 0;
    persistLocation "jmxRepository";
    persistName ATTRIB name;
    export true;
    visibility 1;
    //presentationString "";
  }

  attributes extends { }

  operations extends { }

  notifications extends {
    generic extends Generic;
    change extends Change;
    load extends Load;
    store extends Store;
  }
}
```

Tools (I)

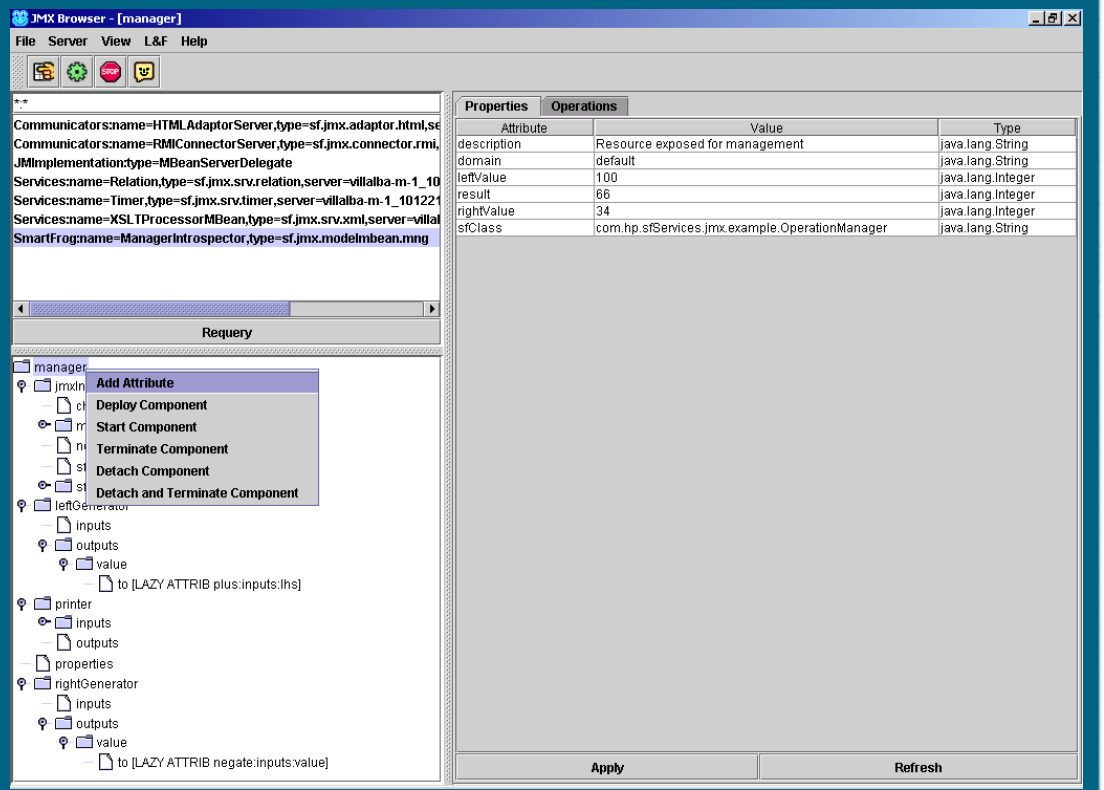
- **Deployment Analyser**:
  - Designed as Mbean
  - Exposes deployment related JMX attributes
  - Exposes JMX operations to go throw the SF deployment tree
  - Only shows component attributes/methods specified as Manageable Metadata
  - Allows modifying attributes and invoking methods on components

```
DeploymentAnalyzer extends Compound {
  sfClass "com.hp.sfServices.jmx.deployment.DeploymentAnalyzer";
  rootTarget LAZY PARENT;
  sfManageable extends LAZY ManageableMetadata;
}

DeploymentModelMBean extends ModelMBean {
  properties:name "DeploymentModelMBean";
  properties:type "sf.jmx.modelmbean.deployment";
  managedResource LAZY DeploymentAnalyzer;
  attributes extends {
    rootPath extends RootPath;
    deployedHost extends DeployedHost;
    processName extends SfProcessName;
  }
  operations extends {
    sfGetAttribute extends SfGetAttribute;
    sfGetRoot extends SfGetRoot;
    sfGetAttributes extends SfGetAttributes;
    sfSetAttribute extends SfSetAttribute;
    sfAddAttribute extends SfAddAttribute;
    sfRemoveAttribute extends SfRemoveAttribute;
    sfGetMethods extends SfGetMethods;
    sfInvokeMethod extends SfInvokeMethod;
    sfChangeAccess extends SfChangeAccess;
  }
  notifications extends {
    generic extends Generic;
    change extends Change;
    deploy extends Deploy;
    start extends Start;
    terminate extends Terminate;
  }
}
```

Tools (II)

- **Mbean Browser**:
  - Connects to a specified JMX Agent and browses all registered Mbeans
  - Main features:
    - Displays Mbeans along with their attributes and operations
    - Allows modifying attributes values
    - Allows invoking operations
    - Unregisters/creates Mbeans
    - Creates Monitors on attributes
    - Display all notifications
    - Uses Connector Client Heartbeat
  - Enhanced features:
    - Displays the deployment tree shown by a Deployment Analyser Mbean
    - Allows modifying SF attributes ands invoking methods on components



Contents

- Introduction to SmartFrog
- Introduction to JMX
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

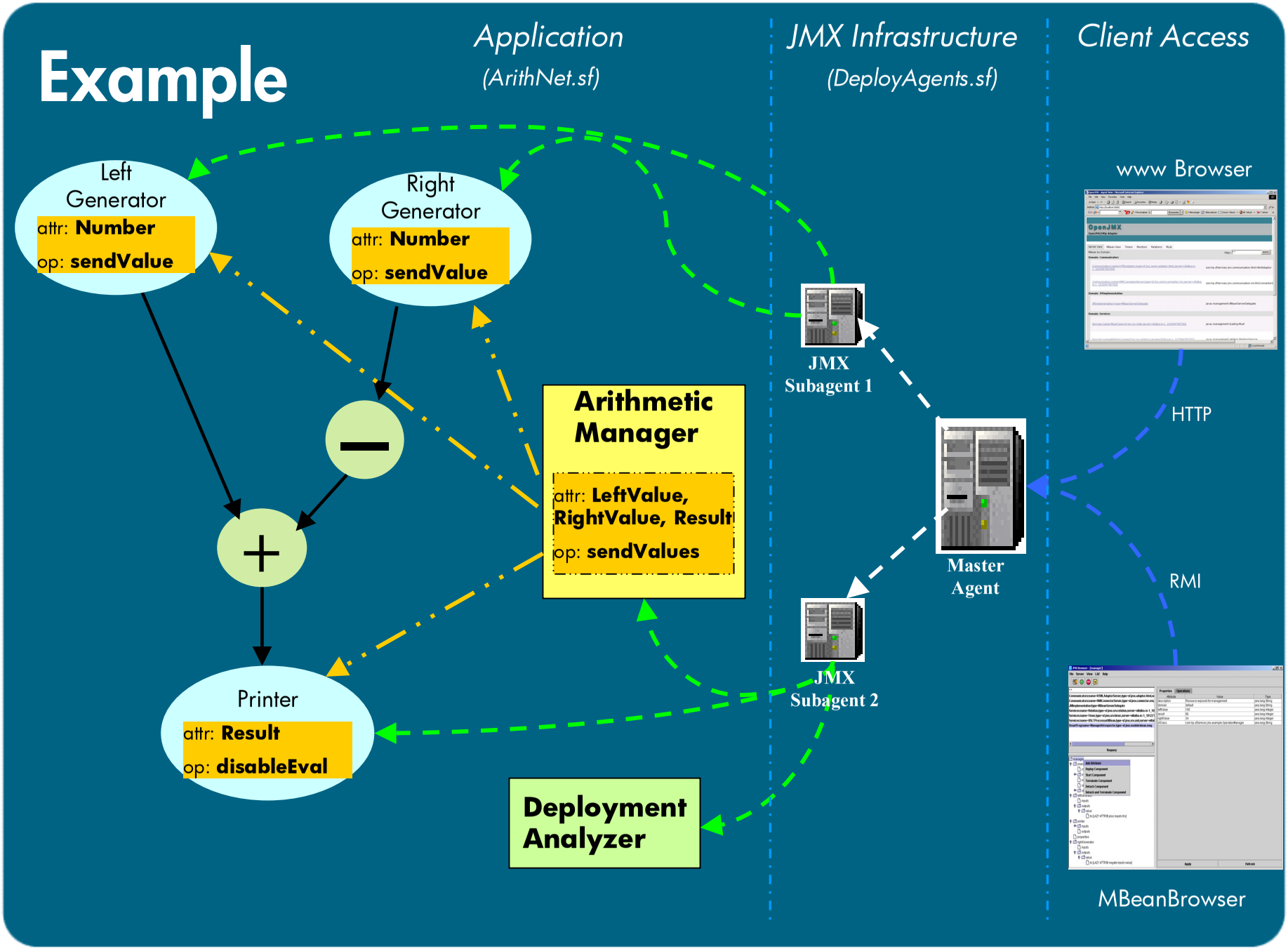
Conclusions

- Large scale application can host a great deal of different technologies making management complex. JMX can resolve this problem.
- JMX does not address large distributed systems. Introduced solutions:
  - SmartFrog Notation for description and deployment of complex systems
  - Transparent connectivity
  - Distributed MBean deployment (from anywhere to anywhere)
  - Hierarchy and federation
  - Service advertising, lookup and discovery
- Future work:
  - Designing a security model for accessing the MBeanServer and JMX in general
  - Improving the access by building new Connectors (HTTP, HTTPS, IIOP, SOAP, …) and new Adaptors (SNMP, CORBA, …)

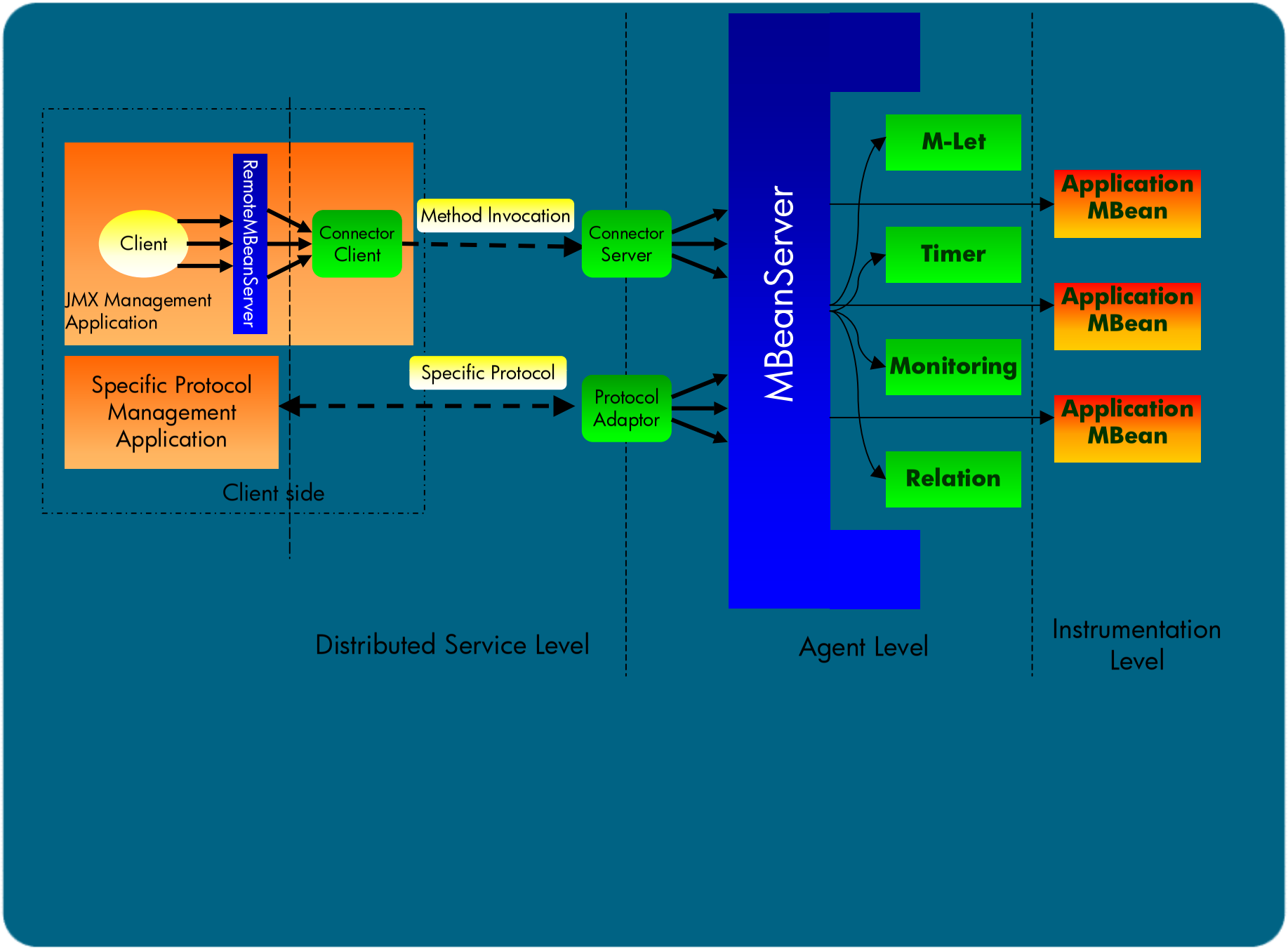
Contents

- Introduction to JMX
- Introduction to SmartFrog
- Proposed requirements
- Framework design
- Building blocks
- Conclusions

Example


Application
(ArithNet.sf)
JMX Infrastructure
(DeployAgents.sf)
Client Access
Left Generator
attr: Number
op: sendValue
Right Generator
attr: Number
op: sendValue
Arithmetic Manager
attr: LeftValue, RightValue, Result
op: sendValues
Printer
attr: Result
op: disableEval
Deployment Analyzer
JMX Subagent 1
JMX Subagent 2
Master Agent
www Browser
HTTP
RMI
MBeanBrowser

THANK YOU


Client
RemoteMBeanServer
JMX Management Application
Connector Client
Method Invocation
Connector Server
Specific Protocol Management Application
Specific Protocol
Protocol Adaptor
Client side
MBeanServer
M-Let
Timer
Monitoring
Relation
Application MBean
Application MBean
Application MBean
Distributed Service Level
Agent Level
Instrumentation Level