Why Components?

What is the motive for producing, distributing, buying, or using software components? What are the benefits of component software? The simplest answer is:

Components are the way to go because all other engineering disciplines introduced components as they become mature - and still use them.

Szyperski 1999
Components

- Components provide a service without regard to where the component is executing or its programming language
  - A component is an independent executable entity that can be made up of one or more executable objects;
  - The component interface is published and all interactions are through the published interface;

Component definitions

- Councill and Heinmann:
  - A software component is a software element that conforms to a component model and can be independently deployed and composed without modification according to a composition standard.

- Szyperski:
  - A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third-parties.
Component as a service provider

- The component is an independent, executable entity. It does not have to be compiled before it is used with other components.
- The services offered by a component are made available through an interface and all component interactions take place through that interface.

Component characteristics 1

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Standardised</td>
<td>Component standardisation means that a component that is used in a CBSE process has to conform to some standardised component model. This model may define component interfaces, component meta-data, documentation, composition and deployment.</td>
</tr>
<tr>
<td>Independent</td>
<td>A component should be independent – it should be possible to compose and deploy it without having to use other specific components. In situations where the component needs externally provided services, these should be explicitly set out in a ‘requires’ interface specification.</td>
</tr>
<tr>
<td>Composable</td>
<td>For a component to be composable, all external interactions must take place through publicly defined interfaces. In addition, it must provide external access to information about itself such as its methods and attributes.</td>
</tr>
</tbody>
</table>
Component characteristics 2

Deployable  
To be deployable, a component has to be self-contained and must be able to operate as a stand-alone entity on some component platform that implements the component model. This usually means that the component is a binary component that does not have to be compiled before it is deployed.

Documented  
Components have to be fully documented so that potential users of the component can decide whether or not they meet their needs. The syntax and, ideally, the semantics of all component interfaces have to be specified.

Component interfaces

- Provides interface
  - Defines the services that are provided by the component to other components.

- Requires interface
  - Defines the services that specifies what services must be made available for the component to execute as specified.
Component Model... more complex

Component interface

Component Y

Event sinks

Attributes

Event sources

Provides

Requires

Component composition

- The process of assembling components to create a system.
- Composition involves integrating components with each other and with the component infrastructure.
- Normally you have to write ‘glue code’ to integrate components.
Types of composition

- **Sequential composition**
  where the composed components are executed in sequence. This involves composing the provides interfaces of each component.

- **Hierarchical composition**
  where one component calls on the services of another. The provides interface of one component is composed with the requires interface of another.

- **Additive composition**
  where the interfaces of two components are put together to create a new component.
What is intended for composition

- To build a new Component by using existing ones

The CBSE process

Outline system requirements → Identify candidate components → Modify requirements according to discovered components

Architectural design → Identify candidate components → Compose components to create system
Component models

- A component model is a definition of standards for component implementation, documentation and deployment.
- Examples of component models
  - EJB model (Enterprise Java Beans)
  - COM+ model (.NET model)
  - Corba Component Model
- The component model specifies how interfaces should be defined and the elements that should be included in an interface definition.

Elements of a component model

- Composition
  - Interface definition
  - Specific interfaces
- Customisation
  - Naming convention
- Usage information
- Deployment and use
  - Documentation
    - Meta-data access
    - Packaging
  - Evolution support
  - Component model
Middleware support

- Component models are the basis for middleware that provides support for executing components.
- Component model implementations provide:
  - Platform services that allow components written according to the model to communicate;
  - Horizontal services that are application-independent services used by different components.
- To use services provided by a model, components are deployed in a container. This is a set of interfaces used to access the service implementations.

MPEG MultiMedia Middleware
M3W is split into several parts.

M3W Parts:
1. Architecture
2. Multimedia API
3. Component Model
4. Resource Management
5. Download / Delivery
6. Fault Management
7. Integrity Management

Architecture

Applications

Middleware

Support Platform

Multimedia Platform

Computing Platform

= Part 2
= ‘Multimedia’ API, aka ‘Functional’ API
= Parts 3, 4, 5, 6, & 7
= ‘Support’ API, aka
‘Non-functional’ API

= This API is not defined, and is implementation-specific for each M3W implementation.
Architecture

Applications

OCAP, DASE, MHP, OMA...

Support Platform

Multimedia Platform

Computing Platform

Specification vs Realization

Applications and other middleware

M3W API

REMI-P

REMI-R

Target

Terminal

Conn Mgr

Logical component

Role

Realization

Runtime Environment

Fault Management

Quality Manager

Service Manager

Fault Management

Runtime Environment

Computing Platform

Sistemi Distribuiti, Paolo Nesi 2006-2007
Uniform View

Applications and other middleware

Uniform View

M3W API

Support

REMI-P
REMI-R
Target
Terminal
Conn Mgr

Service Manager

Quality Manager

Runtime Environment

Computing Platform

Implementation freedom

Attribute also accessible via service specific interface
M3W Realization Technology

Component Model and Core Framework

Fault Management

Integrity Management

Download

Resource Management

Auxiliary Services

Applications and other middleware

How to create and interact with...

Runtime Environment

Computing Platform

Service Manager

REMIP

REMIP-R

Core Mgr
Resource Management

Applications and other middleware

Quality Manager

Runtime Environment

Computing Platform

Download

Applications and other middleware

Target

Initiator

Computing Platform

How to get the desired components on the terminal...
Fault Management

- Applications and other middleware
- Support
- Functional
- Fault Management
- How to prevent failures in the presence of faults...
- Runtime Environment
- Computing Platform

Integrity Management

- Applications and other middleware
- Support
- Functional
- Fault Management
- How to maintain a consistent software configuration
- Runtime Environment
- Computing Platform
MM API does not define implementation

- Logical Component = API specification
- Implementation services need to implement logical components
- Implementer is free to choose the architecture of the implementation services
  - Implementation can for instance use hardware streaming, software streaming or a combination
  - Client does not need to be aware of implementation services

Logical services and implementations

- Implementation services do not need to map 1-to-1 to logical components
**Application portability: third-party binding**

- Applications should only depend on the logical components they control
- This makes them portable across a range of platform instances
- Platform instance specific code should be isolated

**Third-party binding is a key technique to enable this**

**The use of third-party binding is recommended, not mandatory**

---

**Third-party binding: “The players”**

- The “third-party”: isolates platform instance specific code
- Each application requires a set of interfaces to do its work

---

**Application**

- **Conn mgr**
- **Platform instance specific**

---

**MM API**

- **Tuning 1**
- **Channel recorder**
- **ATSC decoder**
- **Audio processing**
- **Video processing**
- **Video mixing**
- **Speaker set**
- **Video processing 2**
Multimedia API

- Audio and Video API
  - Front-end components
  - Encoders / Decoders
  - Video processing
  - Audio processing
  - Generic
- IPMP API

Logical Components

Audio Video Logical Components

(Version 1.0)

- Front-end component (11)
  - Channel Decoding
  - Tuning
  - HdmiIn
- Decoders/encoders (5)
  - ATSC Decoder
  - Transport Stream Demultiplexing
  - ...
- Video processing (16)
  - Basic Video Featuring
  - Color Transient Improvement
  - Sharpness Measurement
  - Video Mixing
  - ...
- Audio processing (10)
  - Audio Bass Enhancements
  - Audio Dynamic Range Control
  - Audio Volume Control
  - ...
- Generic (8)
  - Connection Management
  - Fatal Error Handling
  - Unknown
  - ...

Sistemi Distribuiti, Paolo Nesi
Multimedia API

- Audio and Video API
- IPMP API
  - Trust Management Interfaces
    - Key Management
    - Signature Management
    - Licence Management
    - Certificate Management
  - Tool Interfaces
    - General Tool processing
    - Tool Function
    - Tool Update
    - Tool Communication

IPMP Service Model

- IPMP Service Model
- IPMP Terminal Service
- IPMP Tool Service
- IPMP Tool Service
- IPMP Tool Service

- General Tool processing
- Tool update
- Tool Function
- Tool Communication
- Trust management
Objects & Interfaces

The M3W programming model is based on the Object-Oriented paradigm.

Objects are opaque representations of functionality (i.e., purely functional).
- Behavior (code)
- State (data)

The functionality is accessed through 1 or more interfaces:
- Sequence of parameterized operations
- The object "is-a" of the interface type

At run-time, an interface instance represents an interface implementation:
- Is typed by the interface
- Refers to the code & data

An Interface Reference (iref) is the client’s access point to the interface instance:
- Multiple irefs may refer to the same interface instance

Note: this is classical Microsoft COM ☺
Objects & Interfaces (3)

Interface definitions may inherit from other interface definitions
- Extend function ➔ subtype

All interface definitions implicitly inherit from rcIUnknown

An object that implements multiple Interface types is polymorph
- Navigation queryInterface(iid)
- Dynamic discovery ➔ flexibility

Distributed co-operative lifetime mgt
- AddRef()
- Release()

```
interface rcIUnknown { UUID }
{
    native queryInterface(iid);
    long AddRef();
    long Release();
};
```

Dynamic discovery

Object (server)

Interface

-uuid

Operation

-returnType

Attribute

- type
- scope
- name

- enable access to

Parameter

- type
- scope
- name

scope in (in, out, in out)
**Interface Reference**

![Diagram of Interface Reference]

- **Client** has 1...0 Interface References which in turn point to 0...1 Interface Instances.

**Inner Class**

![Diagram of Inner Class]

- Class has 1...0 inner classes.

**Inner Objects**

![Diagram of Inner Objects]

- Object has 1...0 inner objects.
Intuition: Service

Basic Architectural element
Unit of Instantiation
- Service has 0 or more named Provides ports
- Service has 0 or more named Requires ports
- Ports are of an Interface type
- Interface has 0 or more Operations
- Service implements “Service” Interface
  - Obtain Interface reference to Provides ports
  - Bind Interface references to Requires ports

Example: SStereoAmplifier Service

```java
service SStereoAmplifier {GUID}
implements IVolume {
    provides{
        ISpeaker left;
        ISpeaker right;
    };
};
```

SStereoAmplifier
interface RcIUnknown { GUID }
{
    native QueryInterface(in guid);
    long AddRef();
    long Release();
};

interface RcIService { GUID }
{
    void start();
    boolean isStarted();
    void stop()
    raises RcXCannotStop;
};
Service Specific Interface

Used for:
- getting to provided ports
- connecting requires ports
- setting & getting attributes

Implemented Interfaces

Represents:
- Is A relations

Properties:
- Navigation between implemented interfaces using QueryInterface()
Provided Ports

- Represents:
  - Has A relations

- Properties:
  - Navigation between ports is NOT possible
  - Reference to ports are obtained using the service specific interface

Class – Physical Service

At least:
- rcIUnknown
- rcIService
- the ServiceGenericInterface
Physical Service and Ports & Attributes

rcIServiceGeneric Interface
Service Factory

Interface
-uuid
0..*

Class

2..*

-implements

ServiceFactory

-Factory for

PhysicalService

-uuid
1

0..*

-implements

0..*

-implements

1

1

instantiates

0..*

instance of

PhysicalServiceInstance

0..*

At least:
- IUnknown
- IServiceFactory

recIServiceGenericFactory Interface

Interface

-raises presetting of

0..*

2..*

-implements

ServiceFactory

-Factory for

PhysicalService

-uuid
1

0..*

-raises

PhysicalService

-uuid
1

0..*

Attributes

-has

Port

0..*

1
Intuition: Executable Component

Set of Services

Unit of Loading
- Form depends on the OS, e.g.
  - Static in-process (LIB)
  - Dynamic in-process (DLL)
  - Dynamic out-of-process (EXE)

Also contains the “factory” logic for Services Instances, the Service Factory.

Example: Executable Component

component CStereoAmplifier {UUID} {
  provides SStereoAmplifier;
};

RclUnknown
RclServiceFactory
RclServiceGenericFac.

SStereoAmp. Factory

SStereoAmp.
interface RCIComponent { UUID } {
  void initialize();
  boolean isInitialized();
  RCIService getServiceFactory( in uuid svcId);
  void finalize();
  boolean canUnload();
};

csStereoAmplifier
SStereoAmp.
Factory
RClUnknown
RClServiceFactory
RClServiceGenericFac

interface RCIUnknown { } {

};

csStereoAmplifier
SStereoAmp.
Factory
RClUnknown
RClServiceFactory
RClServiceGenericFac

interface RCIUnknown { } {

};
Executable Component

Set of Models & Relations

Unit of Trading

- Some model relations defined
- Typically: one (or more) of those Models is executable on a platform
Models

Model Examples
Relations

Standardization Element

Model Type
1
-typed by

Model
0..*

Model Relation Type
1
-typed by

Model Relation
0..*

Model Relation Type

Model Relation

M3W Component

Component

Model

Manifest

Model

Manifest

Model Relation

Model Relation

Manifest

Component

Model

Manifest

Model Relation

Model Relation

Manifest

Component

Model

Manifest

Model Relation

Model Relation

Manifest

Component

Model

Manifest

Model Relation

Model Relation

Manifest
Runtime Environment

- Single Device
- May be connected
- OS + HW = Platform

External World

Robocop Device

Applications

Middleware

Runtime Environment (M3W RE)

OS, Network & Drivers

Device Hardware

M3W RE (Instantiation of Physical Services)

interface RcIClient (UUID)
{
    RcIService getServiceInstance( in uuid svcId );
    RcIServiceFactory getServiceFactory( in uuid svcId );
    RcIGuidItr getCompliesList( in uuid blueprint );
};

Optional Interfaces
interface RcIRegistryControl { UUID }
{
    void registerComponent( in uuid cmpId, in string location );
    void unregisterComponent( in uuid cmpId );
    void registerService( in uuid cmpId, in guid svcId )
    void unregisterService( in uuid svcId );
    void setComplies( in uuid complying, in uuid blueprint );
    void clearComplies( in uuid complying, in uuid blueprint );
};
**Physical Service Registration**

1. **ExecutableComponent**
   - container
   - PhysicalServiceRegistration

2. **PhysicalService**
   - 0..* -container

**Complies Registration**

1. **CompliesRegistration**
   - blueprint
   - PhysicalService

2. **PhysicalService**
   - 0..* -complying
   - 1
Service Creation using M3W RE

Client
- Asks M3W RE to create a physical service instance
  - Using RcIClient Interface
  - Giving UUID of service

RcIClient

M3W RE

getServiceInstance(svcId)

Asks M3W RE to create a physical service instance.

Create the service instance

Return a reference to the RcIService interface of the created instance.

Instantiation using M3W RE

Client

M3W RE

Component

ServiceFactory - sf

ServiceInstance - si

start()

getServiceInstance(SvId)

RcComponent(SvId)

Activate(SvId)

Initialize(SvId)

getServiceFactory()

return(sf)

getServiceImplerance()

return(si)
Service Creation using Service Factory

Client
- Asks M3W RE to create a service factory for a physical service
  - Using RclClient Interface
  - Giving UUID of service
- Asks the service factory to create the service instance

M3W RE
- Create the service factory
- Return a reference to the RclServiceFactory interface of the created instance

M3W RE
- Calls RclServiceFactory
- getServiceProvider(svcId)

Sistemi Distribuiti, Univ. Firenze, Paolo Nesi 2006-2007

Instantiation using Service Factory

Client
- getServiceManager(svId)
  - FindComponent(svId)
  - Activate()
  - Initialize()
  - getServiceFactory()
  - return(s)
  - start()

M3W RE
- getServiceInstance()
- return(s)

Component
- ServiceFactory - sf
- ServiceInstance - si
### Setting Bindings

**Third Party**
- Binds required ports using the service generic interface implemented by a (physical) service

**Physical Service**
- The `rcIServiceGeneric` interface contains the `BindTo(<name>,<ref>)` operation

### Setting Attributes

**Third Party**
- Accesses service attributes using the service generic interface implemented by a (physical) service

**Physical Service**
- The service generic interface contains the operation:
  
  ```
  set(<attributeName>,<value>)
  get(<attributeName>,<value>)
  ```
Logical Service

<table>
<thead>
<tr>
<th>Interface</th>
<th>-offers</th>
<th>Logical Component</th>
</tr>
</thead>
<tbody>
<tr>
<td>-uuid</td>
<td>0..*</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>0..*</td>
</tr>
</tbody>
</table>

- offers

- is implemented by

0..* 1..*

PhysicalService

-uuid

Service Manager

<table>
<thead>
<tr>
<th>Logical Component</th>
<th>-Handles requests for</th>
<th>ServiceManager</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>0..*</td>
<td>0..*</td>
</tr>
</tbody>
</table>

- is implemented by

0..*

1..*

PhysicalService

-uuid

Use Runtime environment for instantiation and the service generic interfaces for binding
Service Manager Interfaces

- Ask for instantiation of logical component
- Registration of metadata for logical components and physical services

rcISMClient

- Bool isServiceAvailable(uuid logicalComponentId)
- RclService getInstanceForService(uuid logicalComponentId)
rcISMControl

- Bool registerPhysicalServiceMetadata (in String metadataModelPath)
- Bool unregisterPhysicalServiceMetadata (in uuid physicalServiceID)
- Bool registerLogicalComponentMetadata (in String metadataModelPath)
- Bool unregisterLogicalComponentMetadata (in uuid logicalComponentId)

Service Manager

Service Manager can be implemented as a physical service

PhysicalService
- uuid

ServiceManager
**Remote method invocation**

- REMI-R and REMI-P enable
  - Instantiating and using services in remote systems
  - Controlling the life-cycle of remote active instances
- Generated Proxies and Wrappers enable
  - Transparently use of remote services
    - using REMI-R and REMI-P to issue invocation requests

**Use of remote services**
REMI-R responsibilities

- REMI-R manages the forwarding of method execution, creation and releasing of a remote instance
- REMI-R is able to create Proxy instances for a given service
  - Enables transparent usage by the client

REMI-P responsibilities

- REMI-P manages the incoming requests of method execution, creation and releasing of instances
- REMI-P is able to create Wrapper for a given service
- REMI-P manages the repository of services which are available for remote usage
Proxy and Wrapper concepts

MyInterfaceProxy \( \langle \text{uses} \rangle \) MyInterfaceWrapper

Service

MyServiceProxy \( \langle \text{uses} \rangle \) MyServiceWrapper

MyService

Example invocation request

MyInterfaceProxy

Call specific method

marshal input parameters

issue generic request

generic response

un-marshall output parameters

un-marshall return value

return value

REMIL-R

RPC message
Example handling of invocation request

- REMI-P
  - handle generic request
  - get wrapper
  - handle generic request
- MyInterfaceWrapper
  - un-marshall input parameters
  - call specific method
  - return value
  - marshal output parameters
  - marshal return value
- response

Resource Management Framework

Objectives:
- Robustness in the time domain
- Resource management of memory, processor, bus, network, ...
- Guaranteed execution of tasks with time requirements
- Quality of Service Management
Some RM definitions

- Resource-aware (RA) applications:
  - Know their resource needs.
- Quality-aware apps (QA): Are RA apps
  - Provide a number of output quality levels (QL)
  - Can change their quality level dynamically
- Real-Time apps (RTA): Are RA apps
  - Applications with time requirements.
  - To guarantee them, know required resources
  - HRT apps. provides two QLs: all and nothing

NOTE
The word “app” in this context is a collection of Service Instances and/or Applications that form a logical whole from RM point of view.

Resource awareness

- Resource-Aware Apps (RA)
- Quality-Aware Apps (QA)
  - Multiple, dynamic QLs
- Real-Time Apps (RTA)
  - HRT: Binary QLs

Non-Resource-Aware Apps (NRA)
Approach to Resource & Quality Management

- Based on a contract model
  - Resource Management Framework provides resource
  - Applications provide a certain Quality Level
- Negotiation based
  - Applications provide <quality level, resource needs> options
  - The RMF (selects the option and) assigns resources to Resource Aware applications.
  - A portion of the available resources is reserved for Non-Resource Aware apps

Resource Management Architecture

- Support several QLs
- Negotiate contract with QoSM
- Use the assigned budgets
- Adapt to deviations.
- Handle overruns and errors
- Monitor the system to ensure that QLs are provided & to know precise resource usage
- Optimise the global quality
- Resource share setting
- Basic budget
- Events for accounting & enforcing

Negotiate contract with apps
Monitor the system to ensure that QLs are provided & to know precise resource usage
Optimise the global quality
Resource share setting
Basic budget
Events for accounting & enforcing
**Interaction Model**

QoS manager is a M3W Physical Service

- **QoS Management**
  - QoS Manager
  - Resource Manager
  - Quality Manager
  - Network RC
  - CPU RC

1. Interface obtained from the M3W RE
2. Interface provided by the QC when registering in the QM

**Composition of QoS Information**

- App 1
- App 2
- Other Service
- QoS Manager
- FilterSvcA
- FilterSvcB
- FilterSvcC
- Quality Chief B
- Quality Chief C
- Resource Chief

Q1=(Qb1, Qc1) ; R1=(Rb1+Rc1)
Q2=(Qb1, Qc2) ; R1=(Rb1+Rc2)
Q3=(Qb2, Qc2) ; R1=(Rb2+Rc2)
Download Framework

- Objective: controlled download of Components
- Entities
  - Repository: where component to be downloaded resides
  - Target: device where the component will be downloaded to
- Roles
  - Initiator: identifies the need for a download and contacts the involved parties to initiate the process
  - Locator: locates Target, Repository and Decider for a download
  - Decider: performs the feasibility analysis for the download: business fit & technical fit & resource fit

Process Steps (1, negotiation)

- Initiator
- Locator
- Decider
- Repository
- Target

- LocateAll
  - status + locations
- MakeDownloadDecision
  - status + decisions
- GetComponentProfile
  - status + profile
- GetTargetProfile
  - status + profile
- DownloadDecision
  - status
- DownloadDecision
  - status
- GetComponentProfile
  - status + profile
Decision is based on Profiles

Component profile compatible with Target profile

- **Target profile**: snapshot of the Target's state and properties
- **Component profile**: the component's requirements for
  - installation on the Target
  - execution on the Target
- If -given the Decider's rules- both profiles are matching, the Download can proceed. If not, another component can be selected in the list gathered from the repository

Profile attributes

<table>
<thead>
<tr>
<th>Component profile</th>
<th>Target profile</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of bytes transferred to Target for component installation</td>
<td>Available memory on the device</td>
</tr>
<tr>
<td>Stored component size</td>
<td>Id.</td>
</tr>
<tr>
<td>Installed component size</td>
<td>Id.</td>
</tr>
<tr>
<td>Instantiated component size</td>
<td>Id.</td>
</tr>
<tr>
<td>Maximum amount of dynamic memory required by a component instance</td>
<td>Id.</td>
</tr>
<tr>
<td>CPU information for which the code was generated</td>
<td>Device CPU information</td>
</tr>
<tr>
<td>Specific hardware requirements (e.g., clock, display, MPEG decoder)</td>
<td>Specific hardware features</td>
</tr>
<tr>
<td>Specific services of the OS (like semaphores, process priorities, timers, etc.)</td>
<td>OS information (maker, version, type, capabilities, services, etc.)</td>
</tr>
<tr>
<td>Target platform/operating system (family, etc.) for which the component is suited (can be manufacturer dependent)</td>
<td>Id.</td>
</tr>
<tr>
<td>Platform/operating system versions acceptable for the component, e.g. a communication stack required by the component</td>
<td>List of specific platform services</td>
</tr>
<tr>
<td>List of specific platform services (not specifically linked to M3W),</td>
<td></td>
</tr>
<tr>
<td>Transferred file format (e.g., zip, jar, ...)</td>
<td>List of supported transfer formats</td>
</tr>
<tr>
<td>Executable file format (e.g. ELF)</td>
<td>List of supported executable formats</td>
</tr>
<tr>
<td>Language of the installed component code (and maybe the version of the language)</td>
<td>Supported versions, families, etc.</td>
</tr>
<tr>
<td>Versioning information like some major/minor component version number,</td>
<td></td>
</tr>
<tr>
<td>CPU/thread usage, expressed in terms of required QoS,</td>
<td>Available resources</td>
</tr>
<tr>
<td>Trading information, like a manufacturer identification,</td>
<td>Identification, signature</td>
</tr>
</tbody>
</table>
Process steps (2 - PUSH & PULL models)

Download API

- The Download Process is accessed through an API
- One API for each role / entity:
  - Locator API
  - Decider API
  - Repository API
  - Target API
- Initiator API
- Target API

General characteristics of the API:
- Blocking calls.
- RPC-like syntax so that it can be projected to RMI, Web services, etc.
- Transparently local or distant.
Fault Management Framework objectives

- Ability to make systems composed of M3W Service Instances fault tolerant.
  - Fault-manage (instances of) "black box" Service Instances
- Be transparent to creator, server, and the Service Instance being fault-managed
- Ability to co-ordinate Fault handling between multiple Service Instances

Fault Management - Initial Situation

- Client uses the Service
- Service uses a Server
- Binding through Interfaces
- Service may be “untrusted”
Fault Management - Managed Situation

- Intercept creation of the Untrusted Service Instance
  - Fault Management policy
- Replace with Middleman
  - Middleman may use the Untrusted Service Instance
  - May intercept Interface Bindings
  - May contain other logic

Fault Management - Coordinating Multiple MM

Fault Manager

Delegate → Escalate

Middleman 1

Middleman 2

⋯

Middleman N
**System Integrity Management Objectives**

- Maintain a terminal in a consistent & sane state on behalf of the user and/or service provider
  - Also in the view of new information becoming available
- Manage upgrades & updates
  - Also based on context information (e.g. bus-stop)
- Provide a “reporting point” for fault management

**Maintaining Software Integrity**

**Approach:** Maintaining software integrity using 3 roles!

**Responsibilities:** of the individual roles …

- **Externalize Model of Current Configuration**
  - Offer Basic Configuration Facilities
- **Monitoring**
  - Diagnosis
  - Repairing
    - Script generation
    - Script execution
- **Provide rules**
  - Provide solutions
Maintaining Software Integrity

1. Terminal is not working properly
2. Model is retrieved by Terminal Manager
3. Terminal Manager will do a number of checks. These checks might require knowledge from the database
4. Terminal Manager will generate a repair script based on the outcome of the checks. This might require some knowledge from the database
5. Terminal Manager will execute repair script using the basic configuration facilities offered by the terminal.

System Integrity Management Overview

Monitoring
- monitor
- fetch model

Diagnosing
- diagnose
- fetch rules

Repairing
- repair
- execute commands
Uniform View

M3W API

Service Manager

Instantiation and binding based on logical component UUID

Instantiation based on service UUID

Applications and other middleware

Runtime Environment

Computing Platform

Enable the invocation operations of interfaces of service on this M3W device by service on another M3W device

Enable the invocation operations of interfaces of service on another M3W device by service on this M3W device

Sistemi Distribuiti, Univ. Firenze, Paolo Nesi 2006-2007
Uniform View

Applications and other middleware

Uniform View

Functional

Target

Initiator

Computing Platform

M3W API

API

Support

References

- M3W
  http://www.chiariglione.org/mpeg/technologies/mpe-m3w/index1.htm