INTERNATIONAL STANDARD

ISO/IEC 23004-6

First edition
2008-02-15

Information technology — Multimedia Middleware —

Part 6: Fault management

Technologies de l'information — Intergiciel multimédia —

Partie 6: Gestion des anomalies

Reference number
ISO/IEC 23004-6:2008(E)

© ISO/IEC 2008

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS

Not for Resale

PDF disclaimer

This PDF file may contain embedded typefaces. In accordance with Adobe's licensing policy, this file may be printed or viewed but shall not be edited unless the typefaces which are embedded are licensed to and installed on the computer performing the editing. In downloading this file, parties accept therein the responsibility of not infringing Adobe's licensing policy. The ISO Central Secretariat accepts no liability in this area.

Adobe is a trademark of Adobe Systems Incorporated.

Details of the software products used to create this PDF file can be found in the General Info relative to the file; the PDF-creation parameters were optimized for printing. Every care has been taken to ensure that the file is suitable for use by ISO member bodies. In the unlikely event that a problem relating to it is found, please inform the Central Secretariat at the address given below.



COPYRIGHT PROTECTED DOCUMENT

© ISO/IEC 2008

All rights reserved. Unless otherwise specified, no part of this publication may be reproduced or utilized in any form or by any means, electronic or mechanical, including photocopying and microfilm, without permission in writing from either ISO at the address below or ISO's member body in the country of the requester.

ISO copyright office
Case postale 56 • CH-1211 Geneva 20
Tel. + 41 22 749 01 11
Fax + 41 22 749 09 47
E-mail copyright@iso.org
Web www.iso.org

Published in Switzerland

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale
© ISO/IEC 2008 – All rights reserved

Contents

Page

Foreword iv

Introduction v

1 Scope 1

2 Normative references 1

3 Terms and definitions 1
3.1 Specification terms and definitions 1
3.2 Realization terms and definitions 6

4 Abbreviated terms 6

5 Overview of interface suites 6

6 Fault Management interface suites 8
6.1 Interfaces for controlling Fault Management 8
6.2 Interfaces for Fault Management coordination 15

7 Realization overview 23
7.1 Introduction 23
7.2 Concepts and terminology 24
7.3 Assumptions and capabilities 25
7.4 Entities and responsibilities 26

8 Fault Management realization 26
8.1 Initial Situation 26
8.2 The basic principle 27
8.3 Roles 28
8.4 Interaction with the System Integrity Management Framework 35

Annex A (informative) **Fault tolerance techniques** 36

Annex B (informative) **Example scenarios** 42

Bibliography 44

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Foreword

ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission) form the specialized system for worldwide standardization. National bodies that are members of ISO or IEC participate in the development of International Standards through technical committees established by the respective organization to deal with particular fields of technical activity. ISO and IEC technical committees collaborate in fields of mutual interest. Other international organizations, governmental and non-governmental, in liaison with ISO and IEC, also take part in the work. In the field of information technology, ISO and IEC have established a joint technical committee, ISO/IEC JTC 1.

International Standards are drafted in accordance with the rules given in the ISO/IEC Directives, Part 2.

The main task of the joint technical committee is to prepare International Standards. Draft International Standards adopted by the joint technical committee are circulated to national bodies for voting. Publication as an International Standard requires approval by at least 75 % of the national bodies casting a vote.

Attention is drawn to the possibility that some of the elements of this document may be the subject of patent rights. ISO and IEC shall not be held responsible for identifying any or all such patent rights.

ISO/IEC 23004-6 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

ISO/IEC 23004 consists of the following parts, under the general title *Information technology — Multimedia Middleware*:

— *Part 1: Architecture*

— *Part 2: Multimedia application programming interface (API)*

— *Part 3: Component model*

— *Part 4: Resource and quality management*

— *Part 5: Component download*

— *Part 6: Fault management*

— *Part 7: System integrity management*

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Introduction

The Multimedia Middleware (M3W) Fault Management Framework provides mechanisms for adding fault tolerance techniques (e.g. error detection and recovery) to an existing system. These techniques deal with errors occurring in Service Instances created from Components that are not trusted, without modifying these Components and without relying on reflection. The Fault Management Framework is restricted to the definition of these mechanisms; the actual fault tolerance techniques are system dependent and outside its scope.

Fault management is done by means of intercepting and redirecting Service instantiation and interface method invocations by inserting a new entity in-between, called the "Middleman". The Fault Management Framework also defines a Fault Manager for coordinating several Middlemans[1)]. Collectively, the Middlemans and Fault Managers provide the fault tolerance techniques.

Finally, the Fault Management Framework may interact with the (remote) System Integrity Management Framework in order to support fault removal.

The purpose of the Fault Management Framework is to enable component-based systems to operate failure-free despite the presence of faults in their Components (fault-tolerance). For a running system, fault-tolerance is based on error detection, confinement, and recovery. The Fault Management Framework aims to (enable) stopping error propagation by handling the detected errors and not letting them lead to a failure.

The Fault Management Framework provides mechanisms to add fault management to existing systems and Components. It does not address the design and implementation of new Components that may incorporate some fault management techniques such as those found in fault-tolerant systems.

The M3W Fault Management Framework should allow a large number of known or future fault-tolerance techniques to be incorporated.

1) The word “Middlemans” is used as the plural of Middleman, since a Middleman is not a human.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Information technology — Multimedia Middleware —

Part 6:
Fault management

1 Scope

This part of ISO/IEC 23004 defines the MPEG Multimedia Middleware (M3W) technology Fault Management Architecture. It contains the specification of the part of the M3W application programming interface (API) related to Fault Management as well as the realization. The M3W API specification provides a uniform view of the Fault Management functionality provided by M3W. The specification of the realization is relevant for those who are making an implementation of a Fault Management framework for M3W.

2 Normative references

The following referenced documents are indispensable for the application of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

ISO/IEC 23004-1, *Information technology — Multimedia Middleware — Part 1: Architecture*

ISO/IEC 23004-3, *Information technology — Multimedia Middleware — Part 3: Component model*

3 Terms and definitions

For the purposes of this document, the following terms and definitions apply.

3.1 Specification terms and definitions

3.1.1
API specification
specification of a collection of software interfaces providing access to coherent streaming-related functionality

3.1.2
interface suite
collection of mutually related interfaces providing access to coherent functionality

3.1.3
logical component
coherent unit of functionality that interacts with its environment through explicit interfaces only

3.1.4
role
abstract class defining behavior only

3.1.5
role instance
object displaying the behavior defined by the role

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.6
attribute
instance variable associated with a role

NOTE Attributes are used to associate state information with roles.

3.1.7
signature
definition of the syntactic structure of a specification item such as a type, interface or function in IDL

NOTE For C functions, signature is equivalent to prototype.

3.1.8
specification item
entity defined in a specification

NOTE Data type, role, attribute, interface and function are examples of specification items.

3.1.9
qualifier
predefined keyword representing a property or constraint imposed on a specification item

3.1.10
constraint
restriction that applies to a specification item

3.1.11
execution constraint
constraint on multi-threaded behavior

3.1.12
model type
data type used for specification (modeling) purposes only

NOTE Set, map and entity are examples of model types.

3.1.13
model constant
constant used for specification (modeling) purposes only

3.1.14
enum element type
enumerated type whose values can be used to construct sets (bit vectors) of at most 32 values by logical or-ing

3.1.15
enum set type
32-bit integer data type representing sets of enumerated values

3.1.16
set type
data type whose values are mathematical sets of values of a specific type

NOTE Unlike enum sets, these sets may be infinite.

3.1.17
map type
data type whose values are tables mapping values of one type (the domain type) to values of another type (the range type)

NOTE Maps are a kind of generalized array.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.18
entity type
class of objects that may have attributes associated with them

3.1.19
interface-role model
extended Unified Modeling Language class diagram showing the roles and interfaces associated with a logical component, and their mutual relations

3.1.20
logical component instance
incarnation of a logical component: a configuration of objects displaying the behavior defined by the logical component

3.1.21
provides interface
interface that is provided by a role or role instance

3.1.22
requires interface
interface that is used by a role or role instance

3.1.23
specialization
behavioral inheritance
definition, by a role, of behavior which implies the behavior defined by another role

NOTE A role S specializes a role R if the behavior defined by S implies the behavior defined by R, i.e. if S has more specific behavior than R.

3.1.24
diversity
set of all parameters that can be set at instantiation time of a logical component, and that will not change during the lifetime of the logical component

3.1.25
mandatory interface
provides interface of a role that should be implemented by each instance of the role

3.1.26
optional interface
provides interface of a role that need not be implemented by each instance of the role

3.1.27
configurable item
parameter that can be set at instantiation time of a logical component, usually represented by a role attribute

3.1.28
diversity attribute
role attribute that represents a configurable item

3.1.29
instantiation
process of creating an instance (an incarnation) of a role or logical component

3.1.30
initial state
state of a role instance or logical component instance immediately after its instantiation

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.31
observable behavior
behavior that can be observed at the external software and streaming interfaces of a logical component

3.1.32
function behavior
behavior of the functions in the provides interfaces of a role

3.1.33
streaming behavior
input-output behavior of the streams associated with a role

3.1.34
active behavior
autonomous behavior that is visible at the provides and requires interfaces of a role

3.1.35
instantiation behavior
behavior of a role at instantiation time of a logical component

3.1.36
independent attribute
attribute whose value may be defined or changed independently of other attributes and entities

3.1.37
dependent attribute
attribute whose value is a function of the values of other attributes or entities

3.1.38
invariant
assertion about a role or logical component that is always true from an external observer's point of view

NOTE In reality, the assertion may temporarily be violated.

3.1.39
callback interface
interface provided by a client of a logical component whose functions are called by the logical component

NOTE A notification interface is an example of this, but there may be other call-back interfaces as well, e.g. associated with plug-ins.

3.1.40
callback-compliance
general constraint that the functions in a callback interface should not interfere with the behavior of the caller in an undesirable way, such as by blocking the caller, or by delaying it too long

3.1.41
event notification
act of reporting the occurrence of events to 'interested' objects

3.1.42
event subscription
act of recording the types of event that should be notified to objects

3.1.43
cookie
special integer value that is used to identify an event subscription

NOTE Clients pass cookies to a logical component when subscribing to events. Logical components pass cookies back to clients when notifying the occurrence of the events.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.1.44
event-action table
table associating events that can occur to actions that will be performed in reaction to the events

NOTE This is used to specify event-driven behavior.

3.1.45
non-standard event notification
event notification that is accompanied by other actions (such as state changes of the notifying logical component

3.1.46
client role
role modeling the users of a logical component

3.1.47
actor role
role (usually a client role) whose active behavior consists of calling functions in interfaces without any a priori constraints on when these calls will occur

3.1.48
control interface
interface provided by a logical component that allows the logical component's functionality to be controlled by a client

3.1.49
notification interface
interface provided by a client of a logical component that is used by the logical component to report the occurrence of events to the client

3.1.50
specialized interface
interface of a role R that is inherited from another role and is further constrained by R

3.1.51
precondition
assertion that should be true immediately before a function is called

3.1.52
action clause
part of an extended precondition and postcondition specification defining the abstract action performed by a function

NOTE The abstract action usually defines which variables are modified and/or which out-calls are made by the function.

3.1.53
out-call
An out-going function call of an object on an interface of another object

3.1.54
postcondition
assertion that will be true immediately after a function has been called

3.1.55
asynchronous function
function with a delayed effect

NOTE The effect of the function will occur some time after returning from the function call.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

3.2 Realization terms and definitions

3.2.1
error
unwanted software state that is liable to lead to a failure

3.2.2
failure
event that occurs when the delivered service of a system deviates from correct service

3.2.3
fault
adjudged or hypothesized cause of an error

3.2.4
unit of failure
entity of the system on which a failure can be observed

3.2.5
unit of fault management
entity of the system that will be subject to Fault Management as a whole

3.2.6
middleman
Fault Management entity used for wrapping untrusted services

NOTE This wrapper can be used to insert fault tolerance mechanisms.

3.2.7
fault manager
Fault Management responsible for coordinating fault management between different middlemans in the system

3.2.8
fault management instantiation policy
extension to the runtime environment that enables instantiation of a middleman when an untrusted service is requested

4 Abbreviated terms

API Application Programming Interface
IDL Interface Definition Language
M3W Multimedia Middleware
OS Operating System
UML Unified Modeling Language
UoF Unit of Failure
UoFM Unit of Fault Management
USI Untrusted Service Instance
UUID Universally Unique Identifier

5 Overview of interface suites

This clause is informative. The M3W Fault Management Framework enables the addition of fault tolerance techniques to realization elements (service instances or compositions of service instances). This addition of fault tolerance techniques can be fully transparent for the applications and other middleware. In this case the

-`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

fault management interfaces would not be visible as part of the M3W API (see Figure 1 — Fault Management is fully transparent for applications and other middleware).

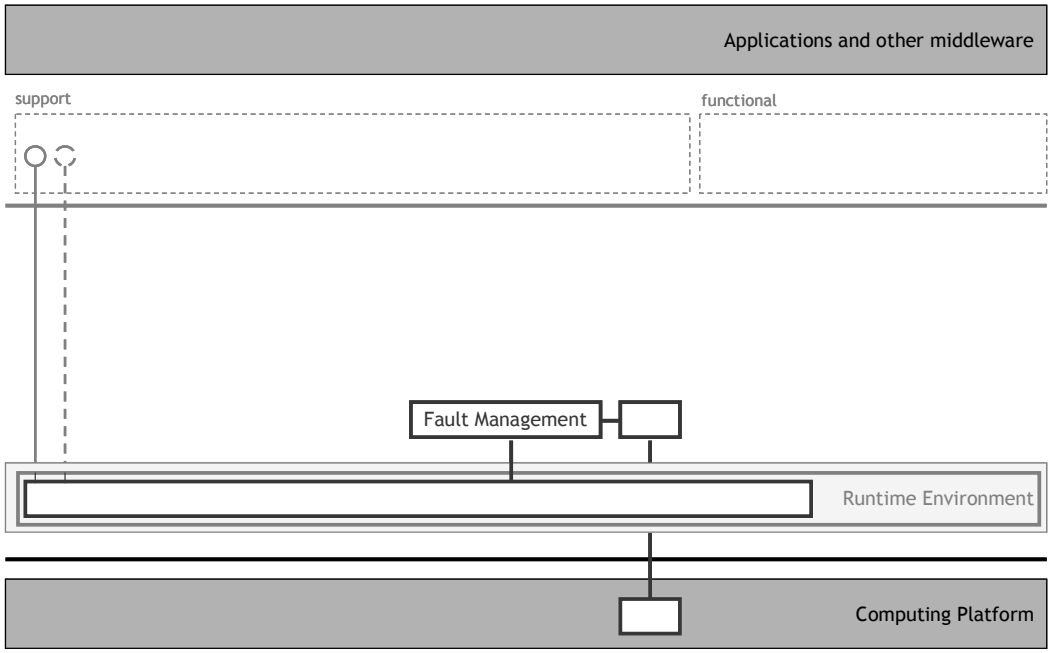

Figure 1 — Fault Management is fully transparent for applications and other middleware

M3W exposes the fault management interfaces as (an optional) part of the M3W API. These interfaces enable the development of a Fault Manager (responsible for coordination of fault management) or a Middleman (wrapper with fault tolerance mechanisms) by an application or other middleware developer. Finally, it also enables control of the Fault Management framework (which realization entities need to be managed and how). Integration with the overall architecture shall be done according to ISO/IEC 23004-1.

The Fault Management API consists of the following parts:

— Fault Management Instantiation Policy

— Fault Manager / Middleman (Coordination)

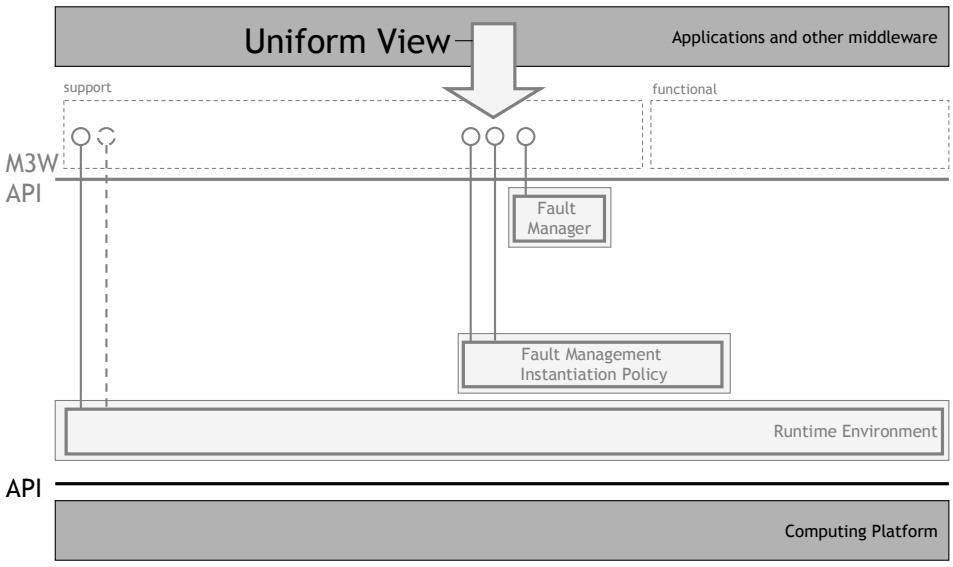

Figure 2 — Fault Management API

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6 Fault Management interface suites

6.1 Interfaces for controlling Fault Management

6.1.1 Fault Management

6.1.1.1 Concepts

The Fault Management Framework enables wrapping "un-trusted" services with a middleman. The middleman contains fault tolerance mechanism such as error detection and error recovery. The Fault Management Framework can be controlled in the sense that we can specify which services need to be wrapped by which middlemans.

In a system that supports Fault Management the Runtime Environment must be extended with a Fault Management Instantiation Policy. This logical extension to the Runtime Environment provides an interface to register a Middleman Service for a registered service that needs to be managed.

6.1.1.2 Types and Constants

6.1.1.2.1 Public Types and Constants

6.1.1.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_FAULTMANAGEMENT_NOT_ALLOWED = 0x00000001;
```

Qualifiers

— Error-codes

Description

The non-standard error codes that can be returned by functions of this logical component

Constants

Table 1

Name	Description
RC_ERR_FAULTMANAGEMENT_NOT_ALLOWED	The operation that is invoked is not allowed. This error is returned by the setMiddleMan operation, for example when it is not allowed to wrap the service due to security reasons.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.2.1.2 rcFaultManagement_MiddlemanRecord_t

Signature

```
struct _rcFaultManagement_MiddlemanRecord_t {
           pUUID middlemanId;
           pUUID svcId;
           pUUID dummyId;
} rcFaultManagement_MiddlemanRecord_t, *prcFaultManagement_MiddlemanRecord_t;
```

Qualifiers

— struct-element

Description

A data structure that is used to describe a middleman in the Fault Management framework. This structure contains the id of the middleman, the id of the service that is managed by the middleman and an optional dummy Id that can be used by the middleman. This dummy id is needed in order for the middleman to be able to instantiate the service that is managed (avoiding an infinite loop of instantiating middlemans).

6.1.1.2.2 Model Types and Constants

None

6.1.1.3 Logical Component

6.1.1.3.1 Interface Role Model

Figure 3 — Interface Role Model depicts the interface-role model of the FaultManagement logical component (grey box). It shows the interfaces and the roles involved with this component.

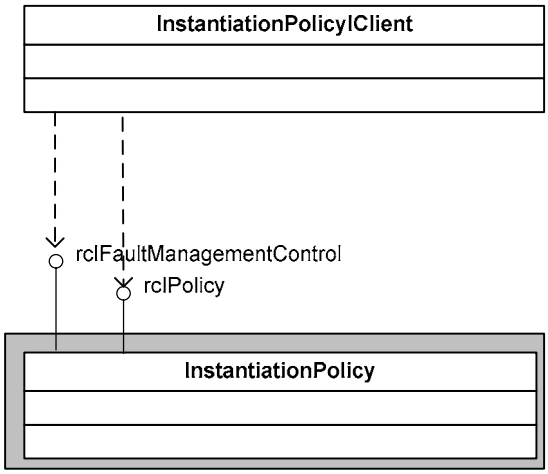

Figure 3 — Interface Role Model

A Fault Management logical component contains the `InstantiationPolicy` role that provides the `rcIFaultManagementControl` interface. This interface can be used by a client to control which services are managed by which middleman. The `InstantiationPolicy` role also provides the `rcIPolicy` interface. This interface is used by a client (usually the Runtime Environment) to ask the `InstantiationPolicy` which service needs to be instantiated (middleman or the original service)

© ISO/IEC 2008 All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.3.2 Diversity

6.1.1.3.2.1 Provided Interfaces

Table 2

Role	Interface	Presence
InstantiationPolicy	rcIFaultManagementControl	Mandatory
InstantiationPolicy	rcIPolicy	Mandatory

6.1.1.3.2.2 Configurable Items

The behavior of the InstantiationPolicy depends on the contents of the Fault Management registry. This registry contains information about which services are managed by which middleman and information that enables the middleman to instantiate the managed service while all other clients will get the middleman on request of the service that is managed. This registry is specified using the following attribute.

Table 3

Role	Attribute
InstantiationPolicy	middlemanRecords

6.1.1.3.2.3 Constraints

None.

6.1.1.3.3 Instantiation

6.1.1.3.3.1 Objects Created

The logical component FaultManagement and the InstantiationPolicy role are always available. They are not created by the clients of M3W.

Table 4

Type	Object	Multiplicity
InstantiationPolicy	instantiationPolicy	1

6.1.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— none

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.3.4 Execution Constraints

The `FaultManagement` logical component is thread-safe.

6.1.1.4 Roles

6.1.1.4.1 InstantiationPolicy

Signature

```
role InstantiationPolicy {
   rcFaultManagement_MiddlemanRecord_t     middlemanRecords[];
}
```

Qualifiers

— Root

Description

This role is the single role of the `FaultManagement` logical component. This role enables controlling Fault Management by controlling which services are wrapped by which middleman. The middleman determines which fault tolerance mechanisms are used.

Independent Attributes

Table 5

Attribute	Description
`middlemanRecords`	This attribute is used to store which services are managed by which middleman. It also contains information that enables the middleman to instantiate the managed service without getting into an infinite loop of instantiation of middlemans.

Invariants

— None

Instantiation

The `InstantiationPolicy` role is always created as part of the `FaultManagement` logical component.

Active Behavior

The `InstantiationPolicy` role has no active behavior.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.1.1.5 Interfaces

6.1.1.5.1 rclFaultManagementControl

Qualifiers

— None

Description

This interface enables control of Fault Management by identifying which services need to be wrapped by which middleman. The interface provides an operation for registration of a new middleman for a service (from that moment on all new instances of the services that are instantiated will be wrapped by an instance of the middleman) and an operation for un-registration of a middleman for a service.

Interface ID

```
{ 6bf7c2b7-ed1b-4f5d-941d-8c9d7c5bb6a5 }
```

6.1.1.5.1.1 setMiddleman

Signature

```
rcResult setMiddleman (
    [in] pUUID middleman,
    [in] pUUID untrusted,
    [in] pUUID dummy
);
```

Qualifiers

— synchronous

— thread-safe

Description

Add a middleman record to the Fault Management registry.

Parameters

Table 6

Name	Description
`middleman`	Reference to the UUID of the middleman that will be wrapping (adding fault tolerance mechanisms) the un-trusted service.
`untrusted`	Reference to the UUID of the un-trusted service that will be wrapped by the middleman
`dummy`	Reference to a dummy UUID that will be used by the middleman in order to instantiate the un-trusted service.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Only the non-standard error codes are listed.

Table 7

Name	Description
RC_ERR_FAULTMANAGEMENT_NOT_ALLOWED	This operation is not allowed. Reason is not specified.

Precondition

middlemanRecords = M

Action

None

Postcondition

middlemanRecords = M ∪ <middleman,untrusted,dummy>

6.1.1.5.1.2 clearMiddleman

Signature

```
rcResult clearMiddleman (
    [in] pUUID middleman,
    [in] pUUID untrusted
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove a middleman record from the Fault Management registry.

Parameters

Table 8

Name	Description
middleman	Reference to the UUID of the middleman that is wrapping (adding fault tolerance mechanisms) the un-trusted service.
untrusted	Reference to the UUID of the un-trusted service that is wrapped by the middleman

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Return Values

Standard.

Precondition

`middlemanRecords = M`

Action

None.

Postcondition

`middlemanRecords ∩ <middleman,untrusted,X> = ∅ for all X`

6.1.1.5.2 rclPolicy

Qualifiers

— None

Description

This interface is used by the client (usually Runtime Environment) to ask the `InstantiationPolicy` which service should be instantiated based on a requested service or a service proposed by another policy.

Interface ID

`{ 8751b464-f875-4339-8195-85ded910c82d }`

6.1.1.5.2.1 getServiceUUID

Signature

```
rcResult getServiceUUID (
    [in]  pUUID svcId,
    [out] pUUID *retValue
);
```

Qualifiers

— Synchronous

Description

Return the UUID of the service that should be created according to this `InstantiationPolicy` based on the UUID of the proposed UUID.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2008 – All rights reserved
Not for Resale

Parameters

Table 9

Name	Description
svcId	Reference to the UUID of the service that is proposed for instantiation.
retValue	Reference to the UUID of the service that is the proposal by this policy

Return Values

Default.

Precondition

middlemanRecords = M

Action

None

Postcondition

(∃ m,d : <m,svcId,d> ∈ M) ⇒ retValue = m

(∃ m,s : <m,s,svcId> ∈ M) ⇒ retValue = s

otherwise retValue = svcId

6.2 Interfaces for Fault Management coordination

6.2.1 Fault Manager

6.2.1.1 Concepts

The scope of a Middleman is restricted to the Service it wraps. In case where fault management techniques must be coordinated over multiple Middlemans, a coordinating entity, called Fault Manager may be used. This interface suite specifies the interaction pattern between the Middlemans and the Fault Manager but leaves the handled information and the internal logic of the Fault Manager and Middlemans out of scope, as this is system specific.

6.2.1.2 Types and Constants

6.2.1.2.1 Public Types and Constants

6.2.1.2.1.1 Error Codes

Signature

None

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers

— Error-codes

Description

The non-standard error codes that can be returned by functions of this logical component

Constants

None

6.2.1.2.1.2 rcFaultManager_MiddlemanInstanceRecord_t

Signature

```
struct _rcFaultManager_MiddlemanInstanceRecord_t {
           prcIMiddleman middlemanIntf;
           pUUID middlemanId;
           Int32 Id;
} rcFaultManager_MiddlemanInstanceRecord_t,
  *prcFaultManager_MiddlemanInstanceRecord_t;
```

Qualifiers

— struct-element

Description

Data structure used to store registration of a middleman instances in the FaultManager. This structure contains the interface reference of the middleman instance, the UUID of the middleman, and the id of the middleware instance.

6.2.1.2.2 Model Types and Constants

None

6.2.1.3 Logical Component

6.2.1.3.1 Interface Role Model

Figure 3 — Interface Role Model depicts the interface-role model of the Fault Manager logical component (grey box). It shows the interfaces and the roles involved with this component.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

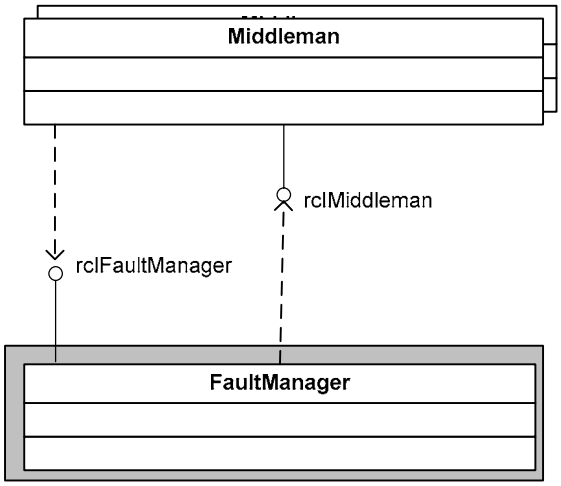

Figure 4 — Interface Role Model

A `FaultManager` Logical Component contains the `FaultManger` role that provides the `rcIFaultManager` interface. This interface can be used by a client to register a `Middleman`. The same interface can be used by the middleman to escalate problems to the `FaultManager`. The other role in this interface suite is the `Middleman` which provides the `rcIMiddleman` interface. This interface can be used by the `FaultManager` to indicate desired fault management operations to a `Middleman`.

6.2.1.3.2 Diversity

6.2.1.3.2.1 Provided Interfaces

Table 10

Role	Interface	Presence
`FaultManager`	`rcIFaultManager`	`Mandatory`
`Middleman`	`rcIMiddleman`	`Mandatory`

6.2.1.3.2.2 Configurable Items

The behavior of the FaultManager is dependent on the Middleman (instances) that are registered at the `Fault Manager`. These references to these instances are stored internally and are modeled by the `middlemanInstanceRecords`.

Table 11

Role	Attribute
`FaultManager`	`middlemanInstanceRecords`

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

6.2.1.3.2.3 Constraints

None.

6.2.1.3.3 Instantiation

6.2.1.3.3.1 Objects Created

The logical component `FaultManager` and the `FaultManger` role are singletons. At most one instance of these entities is created. The `FaultManager` role is created on creation of the `FaultManager` logical component.

Table 12

Type	Object	Multiplicity
`FaultManager`	`faultManger`	`1`

6.2.1.3.3.2 Initial State

The following constraints apply to the initial state of a logical component instance:

— none

6.2.1.3.4 Execution Constraints

The `FaultManager` Logical Component is thread-safe.

6.2.1.4 Roles

6.2.1.4.1 FaultManager

Signature

```
role FaultManager {
    rcFaultManager_MiddlemanInstanceRecord_t       middlemanInstanceRecords[];
}
```

Qualifiers

— Root

Description

The `FaultManager` is a singleton (at most 1 instance will be created). This role is responsible for coordination of fault management activities. The scope of a `Middleman` is local. The scope of the FaultManager is system wide.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Independent Attributes

Table 13

Attribute	Description
`middlemanInstanceRecords`	This attribute is used to store the middleman instances that are registered at the `FaultManager`.

Invariants

— None

Instantiation

The `FaultManager` role is always created as part of the `FaultManager` logical component.

Active Behavior

None.

6.2.1.5 Interfaces

6.2.1.5.1 rcIFaultManager

Qualifiers

— None

Description

The `rcIFaultManager` interface is used by `Middleman`s. It enables error notification to the `FaultManager`. The `FaultManager` can coordinate error handling at the system level. The `rcIFaultManager` interface also contains operations for registration and un-registration of middleman.

Interface ID

```
{ aace1505-e854-4a62-bb5a-d6673f6650a4 }
```

6.2.1.5.1.1 registerMiddleman

Signature

```
rcResult registerMiddleman (
    [in]  prcIMiddleman middlemanIntf,
    [in]  pUUID middlemanId,
    [out] Int32 *retValue
);
```

Qualifiers

— Synchronous

— Thread-safe

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Add a new middleman instance to the `middlemanInstanceRecords`.

Parameters

Table 14

Name	Description
`middlemanIntf`	Reference to the `rcIMiddleman` interface of the middleman that is registered
`middlemanId`	UUID of the middleman that is registered
`retValue`	Return value that contains the id of the middleman instance given by the `FaultManager`. This id can be used for example for un-registration.

Return Values

Default.

Precondition

`middlemanInstanceRecords = M`

Action

None

Postcondition

`middlemanInstanceRecords = M ∪ <middlemanIntf, middlemanId, id>`

`(∀ mi, md : <mi,md, id> ∉ M)`

`retValue = id`

6.2.1.5.1.2 unregisterMiddleman

Signature

```
rcResult unregisterMiddleman (
    [in] Int32 middlemanId
);
```

Qualifiers

— Synchronous

— Thread safe

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Description

Remove middleman instance from `middlemanInstanceRecords`.

Parameters

Table 15

Name	Description
`middlemanId`	Id that was assigned by the `FaultManager` to the middleman instance.

Return Values

Default.

Precondition

[todo]

Action

None

Postcondition

[todo]

6.2.1.5.1.3 escalate

Signature

```
rcResult escalate (
    [in] Int32 middlemanId,
    [in] String notificationId
);
```

Qualifiers

— Synchronous

— Thread-safe

Description

Notification of errors to the FaultManger when for example the error cannot be handled locally.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Parameters

Table 16

Name	Description
`middlemanId`	Id of the middleman instance that is raising the error notification. This id is assigned upon registration of a middleman instance.
`notificationId`	The format and semantics of the `notificationId` are specific to the type of the Middleman and FaultManager; their specification is outside the scope of this interface suite.

Return Values

Standard.

Precondition

true

Action

None.

Postcondition

Platform specific.

6.2.1.5.2 rclMiddleman

Qualifiers

— None

Description

Interface used by the Fault Manager to control the Middleman's fault management logic.

Interface ID

```
{ cdd56af6-7d0c-4031-b1e3-57cc43342037 }
```

6.2.1.5.2.1 designate

Signature

```
rcResult designate (
    [in]  String notificationId,
);
```


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Qualifiers

— Synchronous

Description

Issues fault management notifications to the Middleman. The notification consists of a `notificationId` which identifies the desired fault management operation.

Parameters

Table 17

Name	Description
`notificationId`	Platform specific identification of the desired fault management operation. The format and semantics are specific to the Middleman and Fault Manager. Specification of format and semantics is outside the scope of this interface suite.

Return Values

Default.

Precondition

True

Action

None

Postcondition

Platform specific

7 Realization overview

7.1 Introduction

This is an informative clause that gives an overview of the Fault Management framework realization. The realization is based on the core realization technology specified in ISO/IEC 23004-3.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.2 Concepts and terminology

7.2.1 Failure, Error, Fault

We adopt the basic terminology of dependable systems introduced in [13]. For the intuitive definition of wrong-doing being something incorrect, illegal, mistaken, improper, unsuitable or out of order, we have the following basic definitions that relate to fault management.

A *failure* is an event that occurs when the delivered service of a system deviates from correct service [2)]. Examples of failures are unexpected behavior of a system with respect to its specification such as, for example, delivery of wrong data to the user or loss of communication over a network that needed to drop packets due to a buffer overflow.

An *error* is an unwanted software state that is liable to lead to a failure. Examples of errors are invalid memory states, e.g. detectable by checksums, or the drop of packets from a network due to a buffer overflow.

A *fault* is the adjudged or hypothesized cause of an error. Often faults are mistakes made by a human, but they can also be caused by external factors such as defective hardware. Examples of faults include design/programming errors by a system developer, the toggling of memory bits due to electromagnetic disturbances in the environment where a given hardware operates, and a network congestion beyond its buffer capacities.

Fault tolerance is the technique of designing a system that is failure-free despite the existence of faults. Fault tolerance mechanisms are based on detecting, resolving and hiding errors. Note that the faults in a fault-tolerant system remain but the resulting errors are resolved to prevent failures.

7.2.2 Unit of Failure

The term *Unit of Failure (UoF)* is used to denote an entity of the system on which a failure can be observed, i.e. it is not possible to decompose this system entity to constituent entities and justifiably associate the observed failure with one of them. Then a UoF experiences a failure atomically, as a single whole.

One failure is bound to a single UoF. This property does not prohibit a failure on a given UoF to cause a fault, an error and another failure on a different UoF. Rather, it prohibits two distinct UoFs to experience the same failure (failure as an instance of some event). Hence, if two distinct UoFs experience simultaneously the same type of failure, these two failures are not causally related to each other, although they may have a common fault in the *fault* $\rightarrow$ *error* $\rightarrow$ *failure* $\rightarrow$ *fault* $\rightarrow$ … chains that led to each of them.

In the case of internal faults, it follows from the above definition that the fault and the error that are immediate predecessors of a failure in a *fault* $\rightarrow$ *error* $\rightarrow$ *failure* chain respectively reside in and are detected on the same UoF that exposes the given failure. As a consequence, a fault (respectively error) resides in a single UoF. The same type of fault (respectively error) may reside in different UoFs but that corresponds to a number of different instances of the fault type (respectively error type).

In the context of M3W, a Method off an Interface is the Unit of Failure.

7.2.3 Unit of Fault Management

The term *Unit of Fault Management (UoFM)* is used to denote an entity of the system that will be subject to Fault Management as a whole. For the M3W Fault Management Framework, the UoFM is a Service Instance. This means that fault management operates independently for each instance of each un-trusted Service.

Note, however, that the Fault Manager role provides options for coordinating the operations between various Fault Managed Service Instances.

---

2) For the purpose of the Fault Management Framework, correct service is service that meets the specification (i.e. the framework does not consider faults in the specification).

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.3 Assumptions and capabilities

7.3.1 Assumptions

The characteristics of this framework are that the unit of Fault Management is a Service Instance and that such a Service Instance has no reflection or inspection capability.

The following assumptions are made:

— The Components exist and cannot be inspected or changed, no inside knowledge of the Services is used or assumed, but they do comply with the M3W specification, in particular:

— the creator of the fault managed Service Instance cannot be modified,

— the clients of the fault managed Service Instance cannot be modified,

— the fault managed Service cannot be modified,

— the servers, i.e. Services bound to requires ports, of the fault managed Service Instance cannot be modified.

— The IDL definitions of the fault managed Service and of all its interfaces are known.

— The fault-managed Components and Services do not need to provide any reflection or inspection capability to any other entities. The fault managed Service Instances are therefore black-boxes (i.e. there exists no possibility of observation or intervention on the internals of the Service Instance).

— The fault-managed Services do not have to provide any specific Interface(s) related to fault management, nor do their Components.

7.3.2 Capabilities

Given a set of Services that cannot be inspected or changed and are possibly linked by uses/requires relationships, the Fault Management Framework provides the capability to add fault tolerance without either side of the relationship being modified. The Fault Management Framework enables:

— Intercepting and stopping method invocations from other Service Instances on a fault managed Service Instance.

— Intercepting and stopping a fault managed Service Instance method invocations through its requires ports.

— Replacing a fault managed Service Instance X by another Service Instance X’ by redirecting to X’ all bindings to X.

— Isolating a fault managed Service Instance from the running system.

— Coordinating fault management operations between multiple fault managed Service Instances.

— Adding new detection mechanisms,

— Adding new recovery mechanisms.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

7.4 Entities and responsibilities

The Fault Management framework enables adding fault tolerance mechanisms to Service Instances or a composition of Service instances (Services according to the specification in ISO/IEC 23004-3). There are three important entities in the Fault Management Framework:

— Middleman: Interception and redirection, which is called wrapping, is done by a "Middleman". This Middleman intercepts calls to its interface methods which it may redirect to the wrapped Service Instance or to an alternate implementation (which may contain the fault tolerance mechanisms).

— Fault Manager: The scope of a Middleman is restricted to the Service it wraps. Where fault management techniques must be coordinated over multiple Middlemans, a coordinating entity called the Fault Manager may be used. The M3W Fault Management Framework specifies the interaction pattern between the Middlemans and the Fault Manager but leaves the internal logic of the Fault Manager and Middlemans out of the scope, as this is system-specific.

— Fault Management Instantiation Policy: In a system that supports Fault Management the Runtime Environment must be extended with the Fault Management Instantiation Policy. This logical extension enables instantiation of a Middleman upon the instantiation request for a Service instance.

8 Fault Management realization

8.1 Initial Situation

In absence of Fault Management, the following roles may be present in a system, relative to a given (untrusted) Service Instance:

— The Untrusted Service Instance (USI), which may have implemented interfaces, provides ports, attributes, and requires ports.

— The creator of the USI that initiates the instantiation.

— The Runtime Environment, including the standard instantiation policy, that determines the appropriate Component, loads it, and requests the creation of the Service Instance.

— The configurator of the USI (i.e. the entity that binds interface references to the requires ports which is known as 3rd party binding).

— The clients, i.e. Service Instances or applications that have references to interfaces (implemented interfaces and/or provided ports) implemented by USI.

— The servers, i.e. Service Instances from which interface references have been bound to USI's requires ports.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

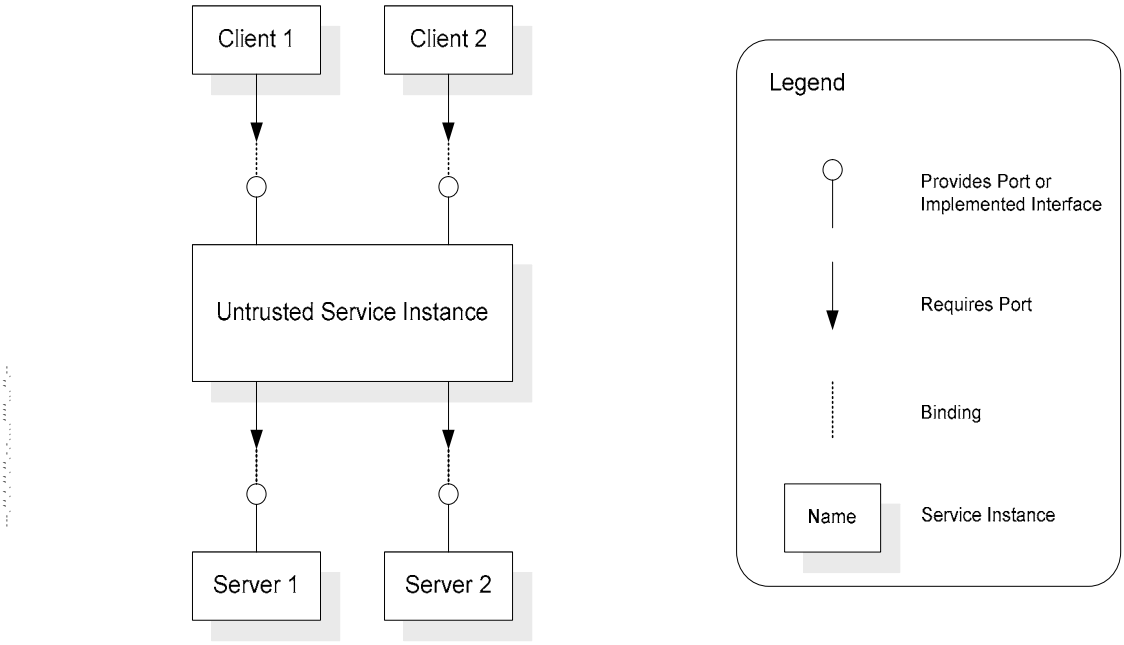

Figure 5 — Initial situation of bindings

NOTE Only explicit dependencies that are expressed through requires ports are taken into consideration here. The USI can also have implicit dependencies, e.g. through the direct creation of Service Instances using the Runtime Environment.

NOTE The above situation does not provide any fault management and may lead to failures.

8.2 The basic principle

A client of the USI calls one of the interface methods of the USI. The thread of control goes inside the USI code and may in turn go to (some of) the USI's servers. The basic idea for adding fault tolerance is that the thread of control between the client and the USI and between the USI and its servers can be

— intercepted and

— redirected

without any of the entities implementing the roles listed in the previous paragraph being modified in any manner.

This basic interception and redirection provides the following capabilities:

— A number of fault tolerant technique implementations may be inserted in the thread of control without modifying the original Component.

— The USI can be dynamically replaced by another instance of the same Service or of a compliant Service.

— Under certain conditions [3)], the Component that hosts the USI may be un-initialized, de-activated and un-installed from the system without risk to the clients or to the system integrity.

3) If no other service instances have been created from this Component.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3 Roles

8.3.1 Middleman

Interception and redirection, which is called "wrapping", is done using a new entity, called the Middleman. This Middleman poses as the USI, intercepting calls to its interface methods. It may redirect calls to the USI or to an alternate implementation. Furthermore it may insert implementations of fault management techniques before and after redirected calls. Finally it may notify a Fault Manager (introduced below) of certain conditions.

A Middleman instance MI is dedicated to the interception of calls directed to the USI. The Middleman definition shall comply with the Untrusted Service (US) definition, in order to guarantee a similar appearance to the binder and clients of the USI. However, it can also implement additional interfaces and provide additional ports.

A Middleman is a Service as any other, i.e. it is part of a Component. As such, in order to be used, a Middleman must be registered and instantiated, just as with any other Service. For each USI, an instance of the Middleman is created to wrap that USI.

If diversity in fault managing a given untrusted Service is required, this can be realized in the following two ways:

— Hard coded in the Middleman, potentially controlled by a dedicated interface or attribute

— Multiple Middleman implementations in different Components that are selectively registered with the Runtime Environment (but only one at any given point in time).

Figure 6 shows the same example as described in Figure 5, with the addition of a Middleman, i.e. the grey box, that intercepts all incoming calls on the provides ports of the un-trusted Service. Note that this can also be done for implemented interfaces.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

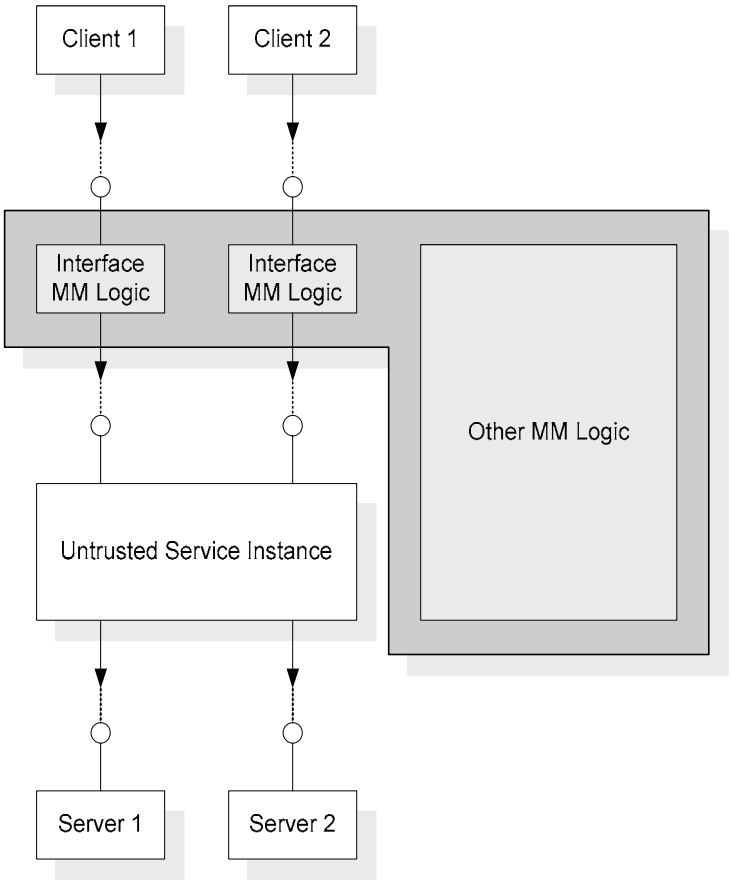

Figure 6 — Fault Managed Situation

Figure 7 shows the same example as described in Figure 6, with the addition that the Middleman also intercepts all outgoing calls through the requires ports of the USI.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

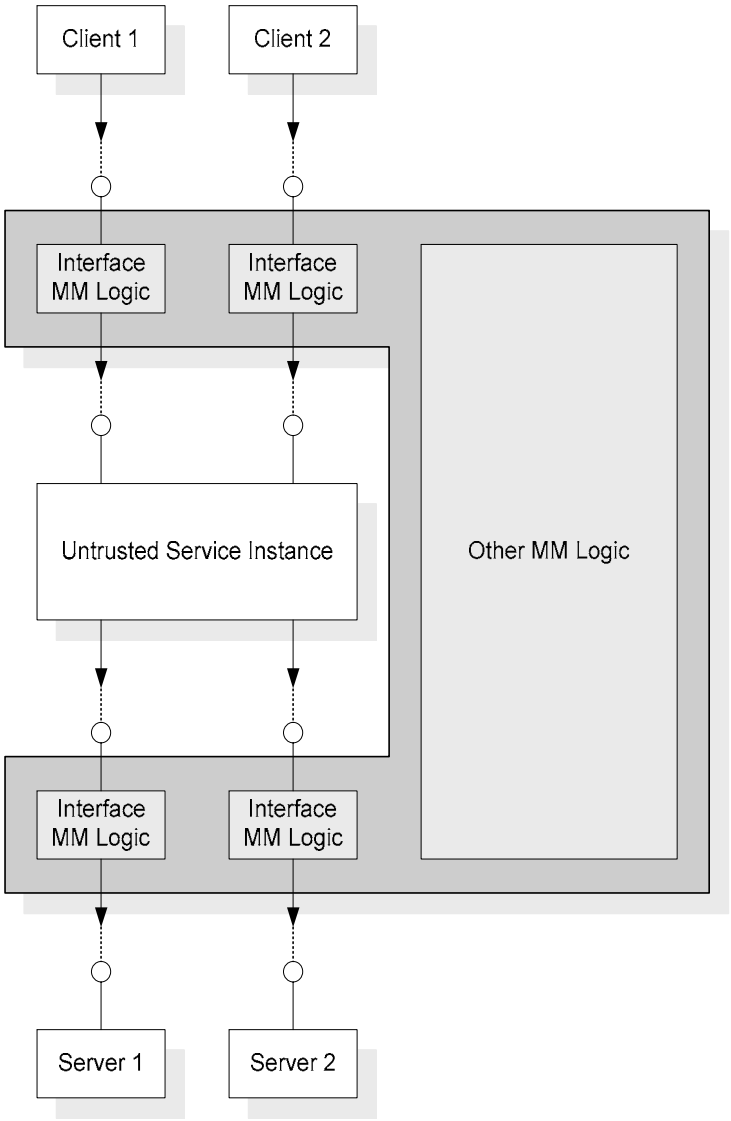

Figure 7 — Fault Managed Situation

A Middleman complies with the Service it wraps. Therefore it appears to its clients as identical to the wrapped Service (i.e. exposes the same interfaces), and cannot be distinguished from it. It also means that the Middleman's Service Factory must comply with the Untrusted Service's Service Factory.

As can be seen from the above figures, the Middleman has inserted some logic between its interfaces and the corresponding interfaces of the USI, in the figure denoted as "Interface MM logic". This interception logic can include any fault management technique. Furthermore there may be fault management logic not directly in the interface call path, as denoted by "Other MM Logic".

In dealing with each of the USI's requires ports, the Middleman has two principle options:

— Forward the `bindTo<portname>()` and `unbind<portname>()` operations from its Service Specific interface to the USI's Service Specific interface. This option is depicted in Figure 6. Effectively this means that the binding is done on the USI's requires ports. As a consequence there is no interception of method calls to the USI's servers.

— Bind an interface it provides to the USI's requires port, and effectuate the `bindTo<portname>()` and `unbind<portname>()` operations on its own corresponding requires port. This option is depicted in Figure 7 for both requires ports. Effectively this means that the outgoing calls made by the USI on its

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

requires port can be intercepted by the Middleman instance. It may, as part of its operation, forward those calls to the USI's server.

A Middleman must be registered with the Fault Management Instantiation Policy.

8.3.1.1 Interfaces

8.3.1.1.1 Introduction

The Middleman is a Service that takes the place of, and complies with, a Service that is untrusted. From the perspectives of the creator, the configurator, and the clients of the untrusted Service Instance the Middleman is indistinguishable from the untrusted Service Instance. From the perspective of the Fault Management Framework, the Middleman may have the following additional fault management interface:

— `rcIMiddleman`, which is used by the Fault Manager to control the Middleman's fault management logic. In the case where the Middleman is entirely autonomous, this interface does not have to be provided.

8.3.1.1.2 rcIMiddleman

```
interface rcIMiddleman { cdd56af6-7d0c-4031-b1e3-57cc43342037 } {
    void designate( in String notificationId );
};
```

The `designate()` operation issues fault management notifications to the Middleman. The notification consists of a `notificationId` which identifies the desired fault management operation. The format and semantics of the `notificationId` are specific to the type of the Middleman and its Fault Manager; their specification is outside the scope of the Fault Management Framework.

8.3.2 Fault Manager

The scope of a Middleman is restricted to the Service it wraps. In case where fault management techniques must be coordinated over multiple Middlemans, a coordinating entity, called Fault Manager may be used. The M3W Fault Management Framework specifies the interaction pattern between the Middlemans and the Fault Manager but leaves the handled information and the internal logic of the Fault Manager and Middlemans out of the scope, as this is system specific.

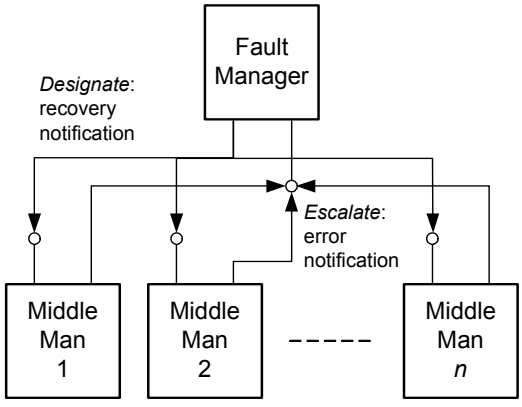

Figure 8 — Fault Manager coordinating multiple Middlemans

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A Fault Manager is a singleton Service. The runtime environment returns the same instance every time a Middleman asks for a Fault Manager instance. Therefore all Middlemans that use the same type of Fault Manager get the same Fault Manager Instance [4)]. During execution a Middleman may notify its Fault Manager about some state that requires fault management actions that exceed its scope of its control. Based on the its fault management policy, the Fault Manager takes appropriate action, e.g. communicate back to the involved Middlemans the fault management actions they should take.

8.3.2.1 Interfaces

8.3.2.1.1 Introduction

The Fault Manager is a singleton Service. It shall implement the following interface:

— `rcIFaultManager`, which is used by Middlemans to register and unregister with the Fault Manager, and to notify the Fault Manager about some state that requires fault management actions that are beyond the scope of the Middleman.

8.3.2.1.2 rcIFaultManager

```
interface rcIFaultManager { aace1505-e854-4a62-bb5a-d6673f6650a4 } {
    Int32 registerMiddleman( in rcIMiddleman middlemanIntf
                             in pUUID middlemanId );
    void unregisterMiddleman( in Int32 middlemanId );
    void escalate( in Int32 middlemanId, in String notificationId );
};
```

The `registerMiddleman()` operation registers a Middleman Instance with its Fault Manager. The Middleman must supply its `rcIMiddleman` interface reference as the `middlemanIntf` parameter, and must supply the UUID of the Middleman Service type as the `middlemanId` parameter. The Fault Manager may use the `middlemanUuid` value to determine what type of Middleman is being registered. The Fault Manager shall assign a unique identification to this Middleman instance and return it for subsequent identification of the Middleman.

The `unregisterMiddleman()` operation unregisters a previously registered Middleman, using its `middlemanId` as key.

The `escalate()` operation issues fault management notifications to the Fault Manager. The notification consists of a `middlemanId`, to identify the calling Middleman instance and a `notificationId`, to identify the desired fault management operation. The format and semantics of the `notificationId` are specific to the type of the Middleman and its Fault Manager; their specification is outside the scope of the Fault Management Framework.

Fault Management Instantiation Policy

The Fault Management Framework extends the standard Runtime Environment instantiation policy in order to create a Middleman instead of the requested fault managed Service Instance. The following interfaces are specified:

— `rcIFaultManagementControl`, to register and unregister Middlemans for a particular Service type.

4) Typically a Middleman will use the Runtime Environment's getServiceInstance() operation to obtain a reference to the instance of its Fault Manager. Because the Fault Manager is a singleton, the first Middleman will create the instance and subsequent Middlemans get a reference to that instance.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

8.3.2.1.3 rclFaultManagementControl

```
interface rcIFaultManagementControl
{ 6bf7c2b7-ed1b-4f5d-941d-8c9d7c5bb6a5 }  {
    void  setMiddleman( in pUUID middleman, in pUUID untrusted, in pUUID dummy )
        raises RcXNotAllowed;
    void clearMiddleman( in pUUID middleman, in pUUID untrusted )
};
```

This interface is provided as a system library, just as the other direct RRE interfaces.

The `setMiddleman()` operation registers the `Middleman` uuid for the `untrusted` uuid, so that subsequent instantiations of the untrusted Service Factory will create the Middleman Service Factory instead. It also registers the dummy uuid that is used by the middleman in order to create an instance of the untrusted service. This dummy uuid is translated into the uuid of the untrusted service by the Fault Management Instantiation Policy.

The `clearMiddleman()` operation clears the registration between the `middleman` uuid and the `untrusted` uuid.

8.3.3 Interaction between roles

Figure 9 shows two sequences of interactions between one Middleman instance, one un-trusted Service Instance (USI) and one client instance calling a method of this un-trusted Service. They are given as examples and show two possible behaviors of the client thread of control.

The first sequence (up to step 6) involves an un-trusted Service that does return from the call of its method. In this case, the un-trusted Service may behave incorrectly but it does terminate, possibly with an error code or any erroneous result. In this case, the Middleman indicates the situation to the Fault Manager, which takes a decision in accordance to the system policy, and returns its decision to the Middleman. The recovery action is then applied by the Middleman to the un-trusted Service (for example a method retry).

The second sequence (form step 9) involves an un-trusted Service that does not return from the call of its method. Such a situation is detected at the Middleman end by means of (for example) a timer that elapses before the call is returned. This situation is reported to the Fault Manager, which takes a decision in accordance to the system policy, and returns the decision to the Middleman. An example of a recovery action is the deletion of the un-trusted Service Instance.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2008 All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

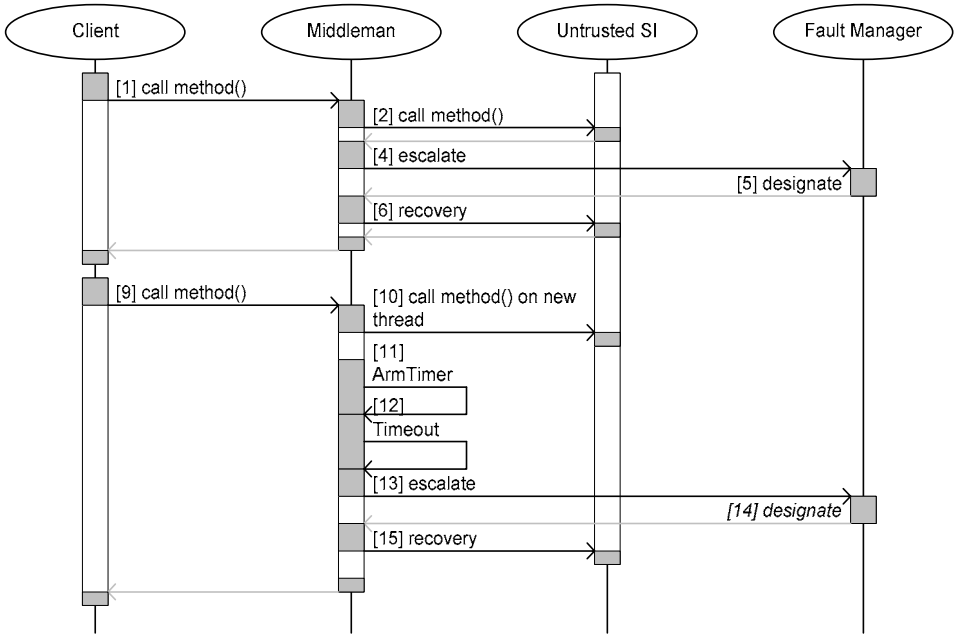

Figure 9 — Sequence chart involving one Middleman and the Fault Manager

8.3.4 Fault Management Instantiation Policy

In a system that supports Fault Management the Runtime Environment must be extended with the Fault Management Instantiation Policy. This logical extension to the Runtime Environment provides an interface to register a Middleman Service for a registered Service that needs to be fault managed. The Fault Management Instantiation policy records these relations in a registry that is a logical extension of the Runtime Environment registry.

Policies can be considered as a function on a uuid. Based on an input uuid (proposed or requested) a policy will give the uuid of the service that needs to be created according to that policy. This output can be used as input for another policy. The Fault Management Instantiation Policy will cause the registered Middleman's uuid to be returned for every uuid of a Service for which a Middleman has been registered.

8.3.4.1 Interfaces

8.3.4.1.1 Introduction

The Fault Management Instantiation Policy is a logical extension to the Runtime Environment. It shall implement the following interface.

— `rcIPolicy`, which is used by the Runtime Environment to ask policies what service should be instantiated based on the uuid of the (currently) proposed uuid. The first proposed uuid is the uuid of the requested service. The Runtime Environment can use a number of policies to determine the uuid of the service that in fact will be instantiated.

8.3.4.1.2 rcIPolicy

```
interface rcIPolicy { 8751b464-f875-4339-8195-85ded910c82d }  {
    pUUID getServiceUUID( in pUUID svcId )
};
```


© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The `getServiceUUID` operation returns the uuid of the service that should be instantiated according to the Fault Management Instantiation Policy based on the uuid (`svcId`) of the proposed / requested Service. When a middleman is registered for svcId the policy returns the uuid of the middleman. When `svcId` is equal to a dummy id of an untrusted service the policy returns the uuid of the untrusted service. Otherwise the policy returns `svcId`.

8.4 Interaction with the System Integrity Management Framework

The entities in the Fault Management Framework may notify the System Integrity Management Framework of any relevant fault management information. The System Integrity Management Framework is specified in ISO/IEC 23004-7.

--`,,```,,,,````-`-`,,`,,`,`,,`---

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex A
(informative)

Fault tolerance techniques

A.1 Introduction

Fault tolerance techniques may be categorized into error detection and error recovery, including confinement, techniques. The techniques described hereafter are given as examples of what could be implemented in the context of the Fault Management Framework.

In the remainder of this clause the implementation of a (combination of) fault tolerance techniques is called a "building block". The intent is that the building blocks are generic so that their specification may be reused in Middlemans and Fault Managers without change.

A.2 Objectives

Introducing the Fault Management Framework as described in the previous paragraphs is not sufficient to detect and recover from faults occurring in an un-trusted Service. The basic objective of the building blocks is to provide these mechanisms.

Using the Fault Management Framework, it is now possible to insert a Middleman between the client (an application or another Service Instance) and an un-trusted Service Instance. This means that the call directed to the un-trusted Service Instance is intercepted by the Middleman, which, in turn will call the un-trusted Service. While doing this, it is possible to insert in the Middleman some logic for error detection and recovery before and after the call to the un-trusted Service. This is depicted in the following picture.

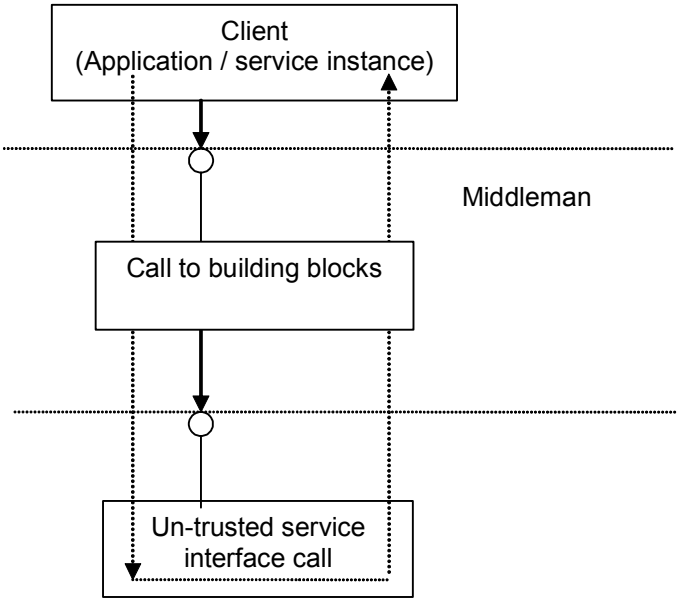

Figure A.1 — Middleman calls to building blocks

In terms of execution path, the client calls the corresponding Middleman method, which is configured to call/execute any planned building blocks, then the un-trusted Service Instance method is called and, when the

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

call returns, any combination of building blocks can again be executed on the return path. This corresponds to the kind of wrapping depicted on Figure 6 of the un-trusted Service.

If the un-trusted Service Instance invokes methods of another Service Instance (server) through its requires ports, it is also possible to intercept these calls in the Middleman and execution of the building blocks can also be performed prior and after these calls. This corresponds to the case depicted in Figure 7.

A.3 Threading model

Depending on the error detection mechanism, the invocation of the untrusted Service Instance method may or may not be done on the thread of control of the client. It may be necessary to create a new thread on which the call to the un-trusted Service Instance will be carried. This design decision will depend on the implementation of the different building blocks, on the available target resources and on the required performance of the system.

For example, this is applicable to the timer building block. The occurrence of a time-out breaks the thread of execution, because the ongoing processing has to be interrupted and the execution flow has to jump to the specified time-out procedure. Although this is by no means the only solution, we suggest using a dedicated thread on which the call to the un-trusted Service is performed. In case of timeout, this thread would be interrupted and possibly killed, leaving the client thread intact.

A.4 Interfaces required by the building blocks

The building blocks may require interfacing with the operating system in order to access system resources, e.g. create timer, create thread or lock/unlock a semaphore.

A.5 Detection building blocks

Before being able to take countermeasures, it is necessary to detect errors. The following paragraphs depict a number of possible detection techniques that can be used in the Fault Management Framework described above. These blocks have as sole task to detect errors. This is then used to take countermeasures as described by the recovery building blocks that will be detailed later. The list given below is not intended to be exhaustive. Rather, the objective was to show the richness of the approach. Obviously, other building blocks may be found.

A.5.1 Timeout

Timeout is the action provoked by an elapsed timer or, equivalently, when a specified waiting time has been reached. When such an event occurs, the corresponding timer is said to have "timed-out". A timer can be used to limit the time taken by an un-trusted Service method to execute. Since the un-trusted Service is a "black box", the interpretation of a timeout is necessarily vague. It can be that

— the Service is taking too much time to execute (but it will finish eventually), or

— the Service entered an infinite loop, or

— the Service is blocked (on resource, interlocked with other Services, etc.)

In all cases it is up to the entity that implements the policy (fault manager or middleman) to decide on the corrective action to be taken.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

A.5.2 Hop counter

The Hop Counter mechanism sets a circular call limit for a method. It prevents the system from running out of stack space when there is an endless circular loop. The maximum calling depth can be specified. If this depth is exceeded, the mechanism will indicate an error. Care should be taken to distinguish between recursive calls and parallel calls from different threads. The counters will have to be stored adequately.

A.5.3 Thread safeness

A thread-safeness building block is used to detect whether a method is called by a single thread at a time or concurrently be different threads. This may be used when there is suspicion that the un-trusted Service is not thread-safe.

A.5.4 Buffer overflows / memory invariant

A buffer overflow is an anomalous condition where a program somehow writes data beyond the allocated end of a buffer in memory. A buffer overflow typically results when a programmer fails to do bounds checking when writing data into a fixed length buffer. Buffer overflows usually arise as a consequence of a bug, and are a commonly exploited computer security risk since program code often sits in the memory areas adjacent to data buffers. As a consequence of a buffer overflow condition, a computer can be made to execute arbitrary (and potentially malicious) code that is fed to the buggy program as data.

The building block can perform memory CRC at start / end of buffers (provided these buffers are given as argument to the interface function) before calling the un-trusted Service and when the Service returns, the same check is performed. Both CRC shall be identical. If not, an error may have occurred.

Similarly, some part of the memory may be checked before and after un-trusted Service calls and, if check fails, an error has occurred.

A.5.5 Exceptions

Depending on the used language, the type of processor and the type of operating system used, an un-trusted Service method execution may cause an exception. This is the case for example for a division by 0, which is handled differently by every operating system.

The exception building block may be used to trap these exceptions and allow correction at later stage (using a corrective building block).

A.5.6 Pre- and post conditions

Pre- and Post condition mechanisms can check method parameters before and respectively after a method call. A condition can be specified per parameter. If a condition is not met, the mechanism may override the parameter with a default value. This may also be considered as an error that will be corrected using one of the recovery building blocks.

Expression of pre and post-conditions may span over a single parameters or over multiple in complex cases. For instance, an expression may be evaluated at runtime. For method: `foo(a1, a2)`, we may have precondition `p1:"a1<a2"`, override `p1: "a1=a2-1"`.a

A.6 Recovery building blocks

When an error is detected by any of the detection building blocks, it may be necessary to provide some correctives actions. These may also be implemented as building blocks such as those described below. Note that the decision to take a given countermeasure after detecting a given error may be performed locally in the middleman or by the fault manager.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2008 – All rights reserved
Not for Resale

A.6.1 Retries

Retrying is a mechanism for hiding temporary errors. It encapsulates a method with functionality that is transparent to the client. The mechanism tries the method it encapsulates for a specified maximum number of times before giving up. If the method does not succeed before the maximum number of retries is reached without success in the invocation of the un-trusted method, an error has occurred that may be reported to the Fault Manager or used locally to decide to use another recovery mechanism.

A.6.2 Service de/reactivation

The reactivation mechanism instantiates a fresh instance for a Service. The Middleman no longer uses the current instance and rebinds to the new instance. The runtime environment will possibly clean up the old instance that is no longer used (as all bindings to it are released by the Middleman). The new instance may be of the same service type or a different Service with the same interface signature. A different Service with the same interface signature may be a compliant Service or an alternative Service explicitly referenced by a Uuid.

A.6.3 Check-pointing and backward recovery

The principle is based on the return, when an error or fault has been identified, to a state where no fault or error has been encountered in the Service Instance.

— The static restart recovery returns the restarted Service Instance to a pre-defined state. This return state can be the initial state, another state identified as such during the design of the Service, or a given unique state identified when the Service Instance is started.

— The dynamic restart recovery returns to a dynamically computing state, identified as such during the execution of the Service Instance. Such states are computed at regular intervals, on specific events external to Service Instance, or on specific internal events. There are three strategies for the events interval: equidistant (periodic), modular (linked to the execution of given operations inside the Service Instance) or random.

Coordinated checkpoints avoid the domino effect and minimize the stable storage requirement in the context of distributed check-pointing. Nevertheless, the requirement for permanent storage may not always be fulfilled, especially by small or mobile devices. The mutable checkpoint can be saved anywhere, in RAM, disk, etc. They help in reducing the number of messages between mobile systems and other entities in the mobile system (such as mobile support stations, etc.), thus avoiding the avalanche effect by using a non-blocking algorithm for coordinated check-pointing.

An experience in the Unix environment is described in [14]. It lists the following information that represents program behavior:

— volatile state, such as the program stack, static and dynamic data segments, and possibly OS-dependent structures,

— whole or part of the persistent state, representing all the user files (actively used, already closed or not yet open files) used by the program,

— OS environment, i.e. the resources that the user processes access through the OS, such as the swap space, the file systems, communication means, etc.

The different combinations of the above information pieces into the checkpoint are analyzed, in particular whether the process state (i.e. the checkpoint related to one process) should include or not all persistent information of the process, e.g. the user files. For example, when the persistent state is included into the checkpoint, the rollback mechanism is more transparent and consistent. However, excluding parts of the persistent state leaves more flexibility when restarting because these parts can be used differently (bypassed, used with different input data, etc.) as during the previous execution attempt. It also discusses the volatility vs. the persistence of the memory in which the checkpoints must be stored.

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

The check-pointing technique finds its limits with the existence of unrecoverable situations, which are in general linked to some irreversible actions (erased information, etc.). In this case, some additional processing must be done, such as the recreation of the deleted information, etc. in order to return to an acceptable, previously unreachable state.

A.6.4 Off-line analysis

Tracing and off-line analysis is done to detect and locate latent errors in a system, such as memory leaks and/or thread starvation. For this type of analysis to be possible, the runtime behavior of the system must be recorded. This is achieved by logging the relevant system events in a chronological way to a location that is accessible even if the system encounters a crash failure (persistent storage). These logs, describing the runtime behavior of the system can then be analyzed to find patterns identifying the latent errors.

Traces can be created either by a Middleman wrapping the Service to be logged or, if possible, a programmer can explicitly insert them to the appropriate Service. In any case, the logging system should be well defined and used in a consistent way.

The general requirements for the log entries include a timestamp and a specified description or key for each type of entry. Timestamps are used to determine the chronological relation between the entries whereas the descriptions are used in the analysis phase to locate and distinguish different events from each other. In addition, the entries can and should contain additional information about the event to enable and aid the off-line analysis.

The off-line analysis is based on logs gathered during the execution of the system using the logging Service. Using these logs the analysis aims at defining and finding patterns in the logs that implicate different latent heap management errors, such as memory leaks, double free's and bad programming practice.

A.7 Complex strategies

The strategies presented below for information can be used locally by the Middleman and / or by the Fault Manager (for example if coordination is required).

A.7.1 Graceful degradation

In this part of ISO/IEC 23004, graceful degradation refers to the decrease of the system functionality in the presence of errors, i.e. in the presence of an error the system functionality decreases to that which has not been affected by the error. The intention is to ensure that an error will affect only the system functionality that is provided by the Service Instance where the error occurs and those Service Instances that depend on it, but the error will not be propagated to the rest of the system. A graph can be used to capture the dependencies of the Service Instances in the system. A traversal of the graph starting at the Service Instance where the error occurred is then used to identify those Service Instances that are affected by the error. Subsequently, the affected Service Instances can be removed from the executing system. The removal from the system of the affected Service Instances is achieved using the dynamic reconfiguration capabilities of the M3W Runtime Environment and the Fault Management Framework.

A.7.2 Layers of defense

The layers of defense concept can be used to define more complex recovery strategies. System states can be mapped to layers of defense. Per defense layer different error recovery mechanisms may be triggered. Based on a detected error the Middleman internal logic or the Fault Manager will take some recovery actions depending on the current defense layer. For example, the internal logic may decide to raise the layer of defense when it repeatedly has to re-activate Service Instance *x*. Next time upon receiving the error notification, it will activate Service Instance *y* instead. Service *y* is a different implementation with the same interface signature.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

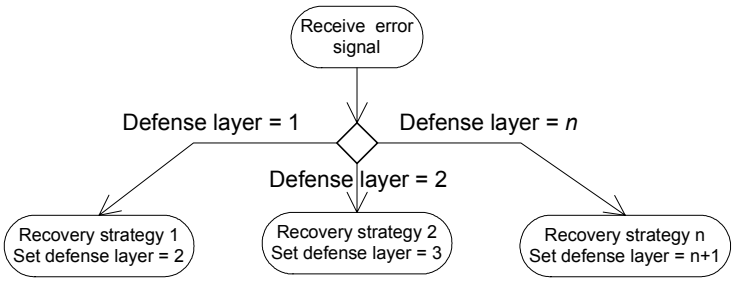

Figure A.2 — Example of complex recovery strategy

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Annex B
(informative)

Example scenarios

B.1 Fault Tolerance in Television Application

In this annex we discuss addition of Fault Tolerance mechanisms to a television application. The structure of the application is depicted in Figure B.1 — TV application before adding Fault Tolerance mechanisms. The component `TunerControl` can be used for selecting channels, `AnalogueIn` enables access to the input provided by the TV card, `ImageDecode` enables access to images that are on local storage, `VideoMixer` can combine multiple video inputs, and `SWVideoFeaturer` can be used to adjust brightness, saturation, hue, etc.

For demonstration purposes a number of faults have been introduced in `TunerControl` and `SWVideoFeaturer`. `TunerControl` does not always lock onto a signal when used to switch between channels. As a result switching channels can result in only noise on the screen. Some of the data structures in `SWVideoFeaturer` are not initialized and precondition is not checked when the saturation is adjusted. As a result adjusting brightness can result in a full black or white screen and adjusting saturation can result in an over-saturated image.

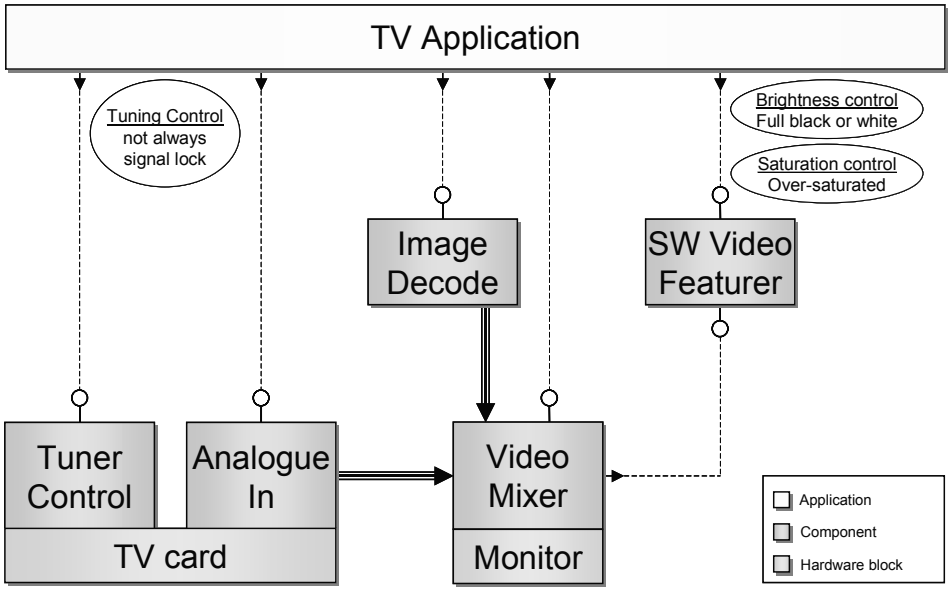

Figure B.1 — TV application before adding Fault Tolerance mechanisms

Fault Tolerance mechanisms (see Annex A) have been added in order to prevent the faults in `TunerControl` and `SWVideoFeaturer` to lead to a failure. The application structure with the Fault Tolerance mechanisms is depicted in Figure B.2 — TV Application with Fault Tolerance mechanisms. For `TunerControl` a retry building block is used. As a result switching channels will be retried until there is a signal lock and therefore a good image. For SWVideoFeaturer the building blocks pre/post condition check and strategy are used. The Strategy building block actually is a customizable combination of building blocks. As a result of the pre condition check there saturation will be clipped on a maximum value. As a result of the strategy the SWVideoFeaturer will be re-instantiated the first time we have a full black or white screen, an attempt to

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

activate a compliant service based on the registered executable components will be made the second time, the third time the Integrity Management framework (see ISO/IEC 23004-7) will be notified. The integrity management framework will verify the configuration and can decide to retrieve a compliant service from a remote location and replace the original service.

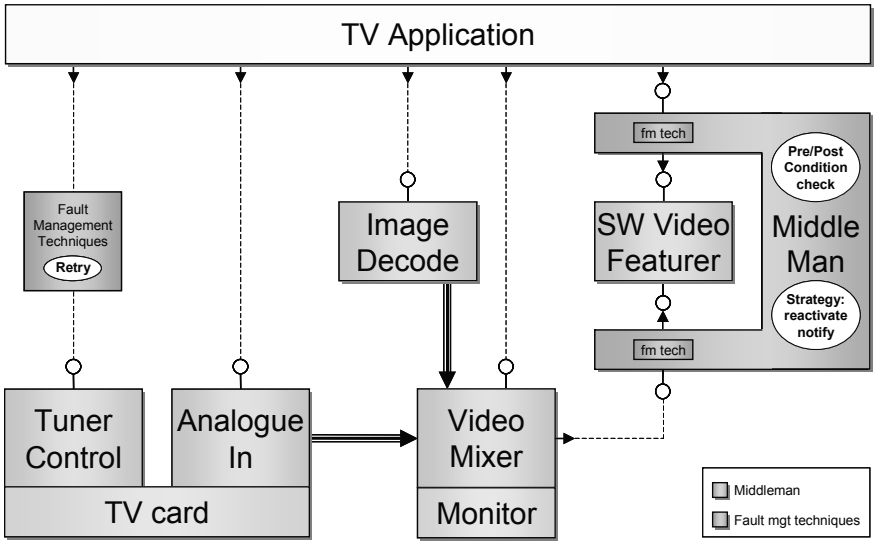

Figure B.2 — TV Application with Fault Tolerance mechanisms

The Fault Tolerance building blocks in the television application example are generated based on a fault tolerance definition language. This avoids introduction of faults in handwritten code for fault tolerance mechanisms and the middleman. An example fragment of this language is listed below. The example shows the addition of the precondition check for adjusting the saturation in the middleman for the `SWVideoFeaturer`.

```
<MiddleMan uuid="7fb340cb-94bc-4054-9888-07506de27669"
           service="4b624587-388d-4513-b117-ad33618e99a7">
   <Interface uuid="a63119f3-19bd-4996-ab17-1c33ca0d14c5">
      <Method name="SetSaturation">
          <PreCondition name="SaturationOutOfBounds">
              <Parameter name="saturation" condition=">20" override="20"/>
          </PreCondition>
      </Method>
   </Interface>
</MiddleMan>
```

© ISO/IEC 2008 – All rights reserved

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Bibliography

[1] ISO/IEC Directives, Part 2, *Rules for the structure and drafting of International Standards*, 2001

[2] ISO/IEC TR 10000-1, *Information technology — Framework and taxonomy of International Standardized Profiles — Part 1: General principles and documentation framework*

[3] ISO 10241, *International terminology standards — Preparation and layout*

[4] ISO 128-30, *Technical drawings — General principles of presentation — Part 30: Basic conventions for views*

[5] ISO 128-34, *Technical drawings — General principles of presentation — Part 34: Views on mechanical engineering drawings*

[6] ISO 128-40, *Technical drawings — General principles of presentation — Part 40: Basic conventions for cuts and sections*

[7] ISO 128-44, *Technical drawings — General principles of presentation — Part 44: Sections on mechanical engineering drawings*

[8] ISO 31 (all parts), *Quantities and units*

[9] IEC 60027 (all parts), *Letter symbols to be used in electrical technology*

[10] ISO 1000, *SI units and recommendations for the use of their multiples and of certain other units*

[11] ISO 690, *Documentation — Bibliographic references — Content, form and structure*

[12] ISO 690-2, *Information and documentation — Bibliographic references — Part 2: Electronic documents or parts thereof*

[13] Jean-Claude Laprie, Brian Randell, Algirdas Avizienis, and Carl Landwehr. Basic concepts and taxonomy of dependable and secure computing. IEEE Trans. Dependable Secure Computing, 1(1), pp. 11-33, 2004

[14] Wang Y.-M., Huang Y., Vo K.-P., Chung P.-Y., Kintala C., Checkpointing and its applications, in Proc. Twenty-fifth International Symposium on Fault-Tolerant Computing (FTCS-25), pp. 22-31, IEEE, June 1995.

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

ICS 35.040

Price based on 44 pages

© ISO/IEC 2008 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale