INTERNATIONAL STANDARD

ISO/IEC 23004-3

First edition
2007-09-15

Information technology — Multimedia Middleware —

Part 3: Component model

Technologies de l'information — Intergiciel multimédia —

Partie 3: Modèle de composant

Reference number
ISO/IEC 23004-3:2007(E)

© ISO/IEC 2007

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 2007

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

© ISO/IEC 2007 – All rights reserved

Not for Resale

Contents

Page

Foreword iv

Introduction v

1 Scope 1

2 Organization of this document 1

3 Normative references 2

4 Terms and definitions 2

5 Overview of interface suites 3

6 General support interface suites 4
6.1 Instantiation of realization elements 4
6.2 Interaction with and between realization elements 51

7 Overview of realization 108
7.1 General 108
7.2 Guiding principles 108
7.3 Assumptions 109
7.4 Architecture overview 110
7.5 M3W stakeholders 111
7.6 Technical context 116

8 Development framework 118
8.1 Overview 118
8.2 Concepts 118
8.3 Behaviour 127

9 Iterator idiom 129
9.1 Overview 129

10 Execution framework 130
10.1 Concepts 130
10.2 Behaviour 149

11 Service Manager 154
11.1 Description 154
11.2 Service Manager's APIs 154
11.3 Logical component and service 155
11.4 Hierarchical structure of service 156
11.5 Service Manager usages 157
11.6 Metadata 162

12 Remote Method Invoker 191
12.1 Overview 191
12.2 Proxies and wrappers 193
12.3 Marshalling parameters 196
12.4 REMI-R logical component 199
12.5 REMI-P logical component 200
12.6 Remotable service development support 201

Annex A (normative) **Service Manager interface definition** 203

Annex B (normative) **Logical Service metadata schema** 205

Annex C (normative) **Service metadata schema** 207

Bibliography 211

© ISO/IEC 2007 – 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 23004-3 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*

— *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 2007 – 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

MPEG, ISO/IEC JTC 1/SC 29/WG 11, has produced many important standards (MPEG-1, MPEG-2, MPEG-4, MPEG-7, and MPEG-21). MPEG feels that it is important to standardize an application programming interface (API) for Multimedia Middleware (M3W) that complies with the requirements found in the annex to the Multimedia Middleware (M3W) Requirements Document Version 2.0 (ISO/IEC JTC 1/SC 29/WG 11 6981).

The objectives of Multimedia middleware (M3W) are to allow applications to execute multimedia functions with a minimum knowledge of the middleware and to allow applications to trigger updates to the middleware to extend the middleware API. The first goal can be achieved by standardizing the API that the middleware offers. The second goal is much more challenging, as it requires mechanisms to manage the middleware API and to ensure that this functions according to application needs. The second goal can support the first, by reducing the needed standard API to those that provide middleware management. Consequently, applications can use these standard management APIs to generate the multimedia system they require.

ISO/IEC 23004 provides the following:

1) a *vision* for a multimedia middleware API framework to enable the transparent and augmented use of multimedia resources across a wide range of networks and devices to meet the needs of all Users;

2) a method to facilitate the integration of APIs to software components and services in order to harmonize *technologies* for the creation, management, manipulation, transport, distribution and consumption of content;

3) a *strategy* for achieving a multimedia API framework by the development of specifications and standards based on well-defined functional requirements through collaboration with other bodies.

© ISO/IEC 2007 – 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 3:
Component model

1 Scope

This part of ISO/IEC 23004 defines the Multimedia Middleware (M3W) Component Model and Core Framework. The context of the M3W Component Model and Core Framework is described in ISO/IEC 23004-1.

2 Organization of this document

This part of ISO/IEC 23004 has the following high level structure:

— Clause 1 defines the scope of this part of ISO/IEC 23004.

— Clause 3 gives an overview of documents that are indispensable for the application of this part of ISO/IEC 23004.

— Clause 4 gives the terms and definitions used in this part of ISO/IEC 23004.

— Clause 5 gives an overview of the interface suites that are part of the Core Framework.

— Clause 6 contains the detailed specification of the interfaces of the Core Framework that are part of the M3W API. These interfaces are structured as follows:

  — Instantiation of realization elements: This subclause contains the interface specifications for instantiation of realization elements based on the uuid of a Service (Run Time) as well as based on the uuid of a logical component (Service Manager).

  — Interaction with and between realization elements: This subclause contains the interface specifications for invocation of an operation on a remote Service as well as enabling remote entities to invoke an operation on a local Service.

— Clause 7 gives an overview of the realization of the component model and core framework.

— Clause 8 describes the development and packaging of realization elements.

— Clause 9 describes the iterator idiom that is used in the execution framework.

— Clause 10 describes the M3W execution framework. This contains the concepts of the component model that are relevant at runtime as well as the elements of the core framework that are needed at runtime for instantiation.

— Clause 11 describes an optional service that enables the instantiation and binding of realization elements based on the uuid of a logical component.

© ISO/IEC 2007 – 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

— Service, Meta Data, and Composition: Informative clause that describes the Meta Data of Services and logical components that enables the Service Manager to instantiate and bind realization elements (services) based on the uuid of a logical component.

— Clause 12 describes the optional services that enable the remote invocation of operations of a service.

3 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*

W3C REC-xml-20001006, Extensible Markup Language (XML) 1.0 (Second Edition), W3C Recommendation 6 October 2000.

W3C REC-xmlschema-1-20041028, XML Schema Part 1: Structures Second Edition, W3C Recommendation 28 October 2004

W3C REC-xmlschema-2-20041028, XML Schema Part 2: Datatypes Second Edition, W3C Recommendation 28 October 2004

4 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 23004-1 apply.

© ISO/IEC 2007 – 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

5 Overview of interface suites

This clause is informative and gives an overview of the interface suites defined in this International Standard. The interface suites defined in this document belong to the core component model and support framework that is specified in M3W. The interface suites that belong to the core component model and support framework deal with instantiation of realization elements and interaction with (and between) realization elements.

Subclause 6.1 specifies the interfaces suites for Instantiation of realization elements. This contains the specification of the Runtime Environment and Service Manager role. The Runtime Environment logical component enables the instantiation of realization element based on an UUID of the realization element (service). The Service Manager logical component enables the instantiation and binding of a number of realization elements that realize a certain logical component (based on the UUID of the logical component).

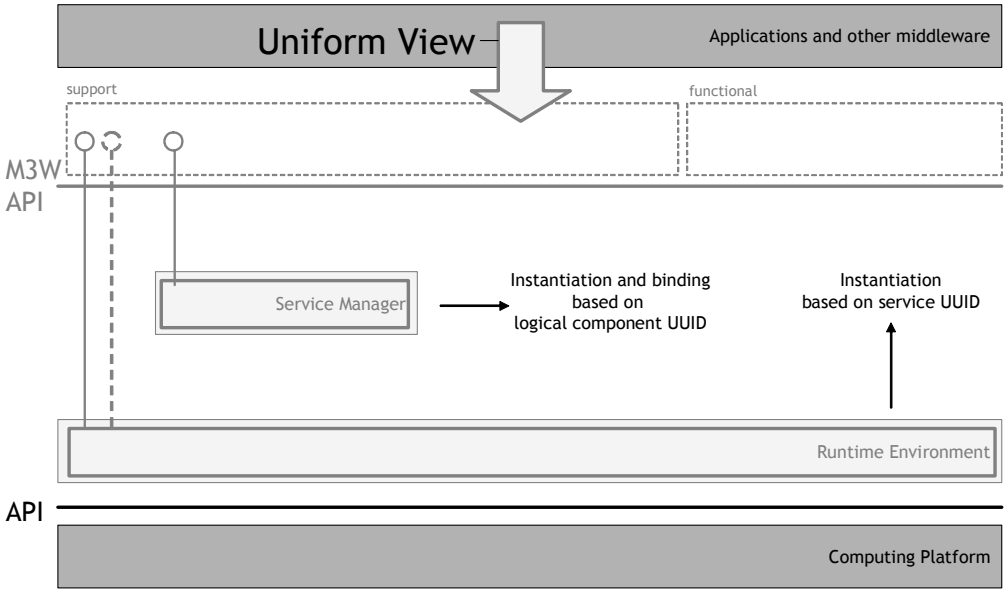


Figure 1 — Overview of the interface suites (logical components) for creation of realization elements

Subclause 6.2 specifies the interface suites for remote method invocation. This contains the specification of REMI-P used for providing operations that can be invoked from a remote M3W device. It also contains the specification of REMI-R used for invocation of operations provided by services on a remote M3W device. A realization of REMI-P must be available on the M3W device that hosts the service that is invoked. A realization of REMI-R needs to be available on the M3W device that has the invoking entity.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

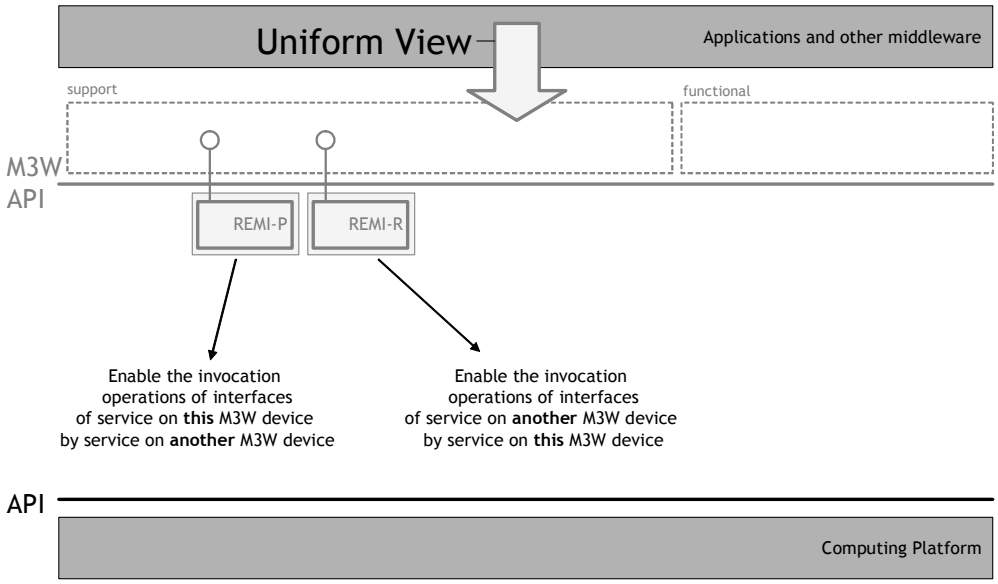


Figure 2 — Overview of interface suites (logical components) for remote method invocation

6 General support interface suites

6.1 Instantiation of realization elements

6.1.1 Runtime Environment

6.1.1.1 Concepts

The function of a Runtime Environment logical component is to enable the instantiation of realization elements (Services, see 10.1.7). It can instantiate Services on request of a client. Furthermore the Runtime Environment offers a number of interfaces for the inspection and manipulation of the Registry that contains information on the executable components (see 10.1.14) that contain the Services that can be instantiated.

6.1.1.2 Types & Constants

6.1.1.2.1 Public Types & Constants

6.1.1.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001
const rcResult RC_ERR_RUNTIME_CANNOT_INITIALIZE = 0x00000002
const rcResult RC_ERR_RUNTIME_CANNOT_FINALIZAE = 0x00000004
const rcResult RC_ERR_RUNTIME_NO_SUCH_ELEMENT = 0x00000008
const rcResult RC_ERR_RUNTIME_NO_SUCH_SERVICE = 0x00000010
const rcResult RC_ERR_RUNTIME_INVALID_LOCATION = 0x00000020
const rcResult RC_ERR_RUNTIME_INVALID_COMPONENT = 0x00000040
const rcResult RC_ERR_RUNTIME_ALREADY_REGISTERED = 0x00000080
const rcResult RC_ERR_RUNTIME_UNKNOWN_COMPONENT = 0x00000100
const rcResult RC_ERR_RUNTIME_NOT_ALLOWED = 0x00000200
```


© ISO/IEC 2007 – 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

Table 1

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This error is returned when a particular function is not implemented by the runtime environment
RC_ERR_RUNTIME_CANNOT_INITIALIZE	This error is returned when a component containing a service cannot be initialized
RC_ERR_RUNTIME_CANNOT_FINALIZE	This error is returned when an executable component cannot be finalized.
RC_ERR_RUNTIME_NO_SUCH_ELEMENT	This error is returned when an (instance of an) element is requested that is not available
RC_ERR_RUNTIME_NO_SUCH_SERVICE	This error is returned when a service instance is requested and the service is not available
RC_ERR_RUNTIME_INVALID_LOCATION	This error is returned on an attempt to use an invalid location. For example for registration of an executable component
RC_ERR_RUNTIME_INVALID_COMPONENT	This error is returned on an attempt to register an invalid executable component.
RC_ERR_RUNTIME_ALREADY_REGISTERED	This error is returned on an attempt to register an executable component multiple times
RC_ERR_RUNTIME_UNKNOWN_COMPONENT	This error is returned on an attempt to modify the registered information related to an executable component that is not known to the runtime environment
RC_ERR_RUNTIME_NOT_ALLOWED	This error is returned when the operation is not allowed

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

© ISO/IEC 2007 – 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 rcRuntime_ComponentRecord_t

Signature

```
struct _rcRuntime_ComponentRecord_t {
           UUID cmpId;
           String location;
} rcRuntime_ComponentRecord_t, *prcRuntime_ComponentRecord_t;
```

Qualifiers

— struct-element

Description

Data structure used to pass and store registration of an executable component.

6.1.1.2.1.3 rcRuntime_ContainerRecord_t

Signature

```
struct _rcRuntime_ContainerRecord_t {
           UUID cmpId;
           UUID svcId;
} rcRuntime_ContainerRecord_t, *prcRuntime_ContainerRecord_t;
```

Qualifiers

— struct-element

Description

Data structure used to pass and store registration of the containment of a service by an executable component.

6.1.1.2.1.4 rcRuntime_CompliesRecord_t

Signature

```
struct _rcRuntime_CompliesRecord_t {
           UUID complying;
           UUID blueprint;
} rcRuntime_CompliesRecord_t, *prcRuntime_CompliesRecord_t;
```

Qualifiers

— struct-element

Description

Data structure used to pass and store registration of the complies relation between services.

6.1.1.2.2 Model Types & Constants

None

© ISO/IEC 2007 – 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 Logical Component

6.1.1.3.1 Interface-Role Model

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

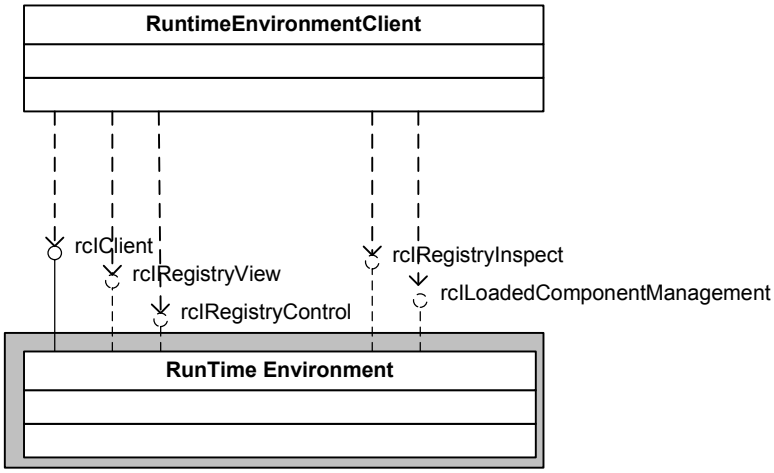


Figure 3 — Interface-Role Model

A Runtime Environment Logical Component contains the Runtime Environment role that provides the rclClient interface. This interface can be used by a client to create instances of services. The Runtime Environment role also has a number of optional interfaces to modify and inspect the registry of the Runtime Environment.

6.1.1.3.2 Diversity

6.1.1.3.2.1 Provided Interfaces

Table 2

Role	Interface	Presence
Runtime Environment	`rcIClient`	`Mandatory`
Runtime Environment	`rcIRegistryView`	`Optional`
Runtime Environment	`rcIRegistryControl`	`Optional`
Runtime Environment	`rcIRegistryInspect`	`Optional`
Runtime Environment	`rcILoadedComponentManagement`	`Optional`

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

© ISO/IEC 2007 – 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

6.1.1.3.2.2 Configurable Items

The behaviour of the Runtime Environment depends on the registry. This registry contains the information the available executable components, which services are contained by these executable components and which services are compatible. The registry contents is specified using the following attributes.

Table 3

Role	Attribute
Runtime Environment	`componentRecords`
Runtime Environment	`containerRecords`
Runtime Environment	`compliesRecords`

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 Runtime Environment and the Runtime Environment role are always available. They are not created by the clients of the M3W.

Table 4

Type	Object	Multiplicity
`RuntimeEnvironment`	`runtime`	`1`

6.1.1.3.3.2 Initial State

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

— none

6.1.1.3.4 Execution Constraints

The logical component Runtime Environment is thread-safe.

6.1.1.4 Roles

6.1.1.4.1 Runtime Environment

Signature

```
role RuntimeEnvironment {
  rcRuntime_ComponentRecord_t        componentRecords[];
  rcRuntime_ContainerRecord_t        containerRecords[];
  rcRuntime_CompliesRecord_t         compliesRecords[];
}
```


© ISO/IEC 2007 – 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

— root

Description

A Runtime Environment enables the instantiation of service based on the registry contents (`componentRecords`, `containerRecords` and `compliesRecords`). Furthermore it provides interfaces for the inspection and manipulation of this registry.

Independent Attributes

Table 5

Name	Description
`componentRecords`	`Used to store the uuid's of the registered executable components and the location on locally accessible storage.`
`containerRecords`	`Used to store the binding between services and executable components`
`compliesRecords`	`Used to store the complies relation between services.`

Invariants

— When the `containerRecords` contain the association between a service and an executable component then `componentRecords` contains the association between this executable component and a location on locally accessible storage.

Instantiation

The Runtime Environment role is always instantiated as part of the Runtime Environment logical component.

Active Behaviour

The Runtime Environment has no active behaviour

6.1.1.5 Interfaces

6.1.1.5.1 rcIClient

Qualifiers

None.

Description

This interface of a Runtime Environment role provides facilities for instantiation of realization elements (services).

Interface ID

uuid(41c235af-0417-4731-8627-c11c3d51d359)

© ISO/IEC 2007 – 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

Remarks

None

6.1.1.5.1.1 getServiceInstance

Signature

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

Qualifiers

— synchronous

— thread-safe

Description

Creates an instance of the requested service.

Parameters

Table 6

Name	Description
svcId	Reference to the UUID of the requested service
retValue	Reference to the rcIService interface of the created service instance.

Return Values

Only non-standard error values are listed

Table 7

Name	Description
RC_ERR_RUNTIME_NO_SUCH_SERVICE	Service UUID could not be found in the registry.
RC_ERR_RUNTIME_CANNOT_INITIALIZE	Executable component containing the service could not be initialized

Precondition

— True

© ISO/IEC 2007 – 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

Action

Creation of an instance of the requested service (or a compliant) service, based on the information in the registry.

Postcondition

```
retValue == <reference to rcIService intf of created instance>
```

6.1.1.5.1.2 getServiceFactory

Signature

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

Qualifiers

synchronous

thread-safe

Description

Creates a factory for the requested service.

Parameters

Table 8

Name	Description
svcId	Reference to the UUID of the service for which a factory has to be created
retValue	Reference to the rcIService interface of the created service instance.

Return Values

Only non-standard error values are listed

Table 9

Name	Description
RC_ERR_RUNTIME_NO_SUCH_SERVICE	Service UUID could not be found in the registry.
RC_ERR_RUNTIME_CANNOT_INITIALIZE	Executable component containing the service could not be initialized

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

© ISO/IEC 2007 – 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

Precondition

True

Action

Creation of a factory for the requested service (or a compliant) service, based on the information in the registry.

Postcondition

retValue == <reference to rclServiceFactory intf of

factory for the requested service>

6.1.1.5.1.3 getCompliesList

Signature

```
rcResult getCompliesList (
    [in] pUUID blueprint,
    [out] prcIUuidItr *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Returns the UUID the services registered in the registry that are compliant with the blueprint service.

Parameters

Table 10

Name	Description
blueprint	Reference to the UUID of the blueprint service
retValue	Reference to rcIUuidItr interface that can be used to walk through the UUIDs of the compliant services.

Return Values

Only non-standard error values are listed

Table 11

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This operation is not implemented by the runtime environment

© ISO/IEC 2007 – 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

Precondition

True

Action

None.

Postcondition

`retValue == <reference to rcIUuidItr intf>`

6.1.1.5.2 rclRegistryView

Qualifiers

None.

Description

This interface of a Runtime Environment role provides facilities basic inspection of the registry contents.

Interface ID

uuid(17745387-4687-4bf9-a9dd-7f958d27dc72)

Remarks

None

6.1.1.5.2.1 isComponent

Signature

```
rcResult isComponent (
    [in] pUUID ident,
    [out] Bool *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Returns whether the entity identified by `ident` is an executable component.

© ISO/IEC 2007 – 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 12

Name	Description
`ident`	`Reference to the UUID of the identifier`
`retValue`	`Bool that indicates whether the entity identified by the identifier is an executable component`

Return Values

Standard

Precondition

True

Action

None.

Postcondition

— `retValue == True` when entity identified by `ident` is an executable component

— `retValue == False` otherwise

6.1.1.5.2.2 getComponentList

Signature

```
rcResult getComponentList (
    [out] prcIUuidItr *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Returns the UUID of the executable components registered in the registry.

Parameters

Table 13

Name	Description
`retValue`	`Reference to rcIUuidItr` interface that can be used to walk through the UUIDs of the executable components.

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 14

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This operation is not implemented by the runtime environment

Precondition

True

Action

None.

Postcondition

```
retValue == <reference to rcIUuidItr intf>
```

6.1.1.5.2.3 getComponentLocation

Signature

```
rcResult getComponentLocation (
    [in] pUUID ident,
    [out] String *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Returns the location on locally accessible storage of the executable component.

Parameters

Table 15

Name	Description
ident	Reference to the UUID of the executable component
retValue	String that contains the location of the executable component

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

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 16

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This operation is not implemented by the runtime environment

Precondition

True

Action

None.

Postcondition

```
retValue == Location of executable component
```

6.1.1.5.2.4 getContainedList

Signature

```
rcResult getContainedList (
    [in]  pUUID cmpId,
    [out] prcIUuidItr *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Returns the UUID of the services contained by the executable component identified by `cmdId`.

Parameters

Table 17

Name	Description
retValue	`Reference to rcIUuidItr` interface that can be used to walk through the UUIDs of the services.

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 18

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This operation is not implemented by the runtime environment

Precondition

True

Action

None.

Postcondition

```
retValue == <reference to rcIUuidItr intf>
```

6.1.1.5.2.5 getContainingList

Signature

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

Qualifiers

synchronous

thread-safe

Description

Returns the UUID of the executable components that contain the service identified by `svcId` or a compliant service.

Parameters

Table 19

Name	Description
retValue	`Reference to rcIUuidItr` interface that can be used to walk through the UUIDs of the executable components.

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 20

Name	Description
RC_ERR_RUNTIME_NOT_IMPLEMENTED	This operation is not implemented by the runtime environment

Precondition

True

Action

None.

Postcondition

retValue == <reference to rcIUuidItr intf>

6.1.1.5.3 rclRegistryControl

Qualifiers

None.

Description

This interface of a Runtime Environment role provides facilities manipulation of the registry contents.

Interface ID

uuid(4e7b2b79-e440-462d-9c6c-ccfc02f348ae)

Remarks

None

6.1.1.5.3.1 registerComponent

Signature

```
rcResult registerComponent (
    [in]  pUUID cmpId,
    [in]  String location
);
```

Qualifiers

synchronous

thread-safe

Description

Add the executable component to the registry so that it can be loaded dynamically at a later time.

Parameters

© ISO/IEC 2007 – 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

Table 21

Name	Description
cmpId	Reference to the identified of the executable component.
location	Location on locally accessible storage where the executable component is located

Return Values

Only non-standard error values are listed

Table 22

Name	Description
RC_ERR_RUNTIME_INVALID_LOCATION	The location is not valid
RC_ERR_RUNTIME_INVALID_COMPONENT	The executable component is not valid (does not comply with the M3W component model).
RC_ERR_RUNTIME_ALREADY_REGISTERED	The component is already registered

Precondition

componentRecords = A

Action

None.

Postcondition

componentRecords = A ∪ <cmpId, location>

6.1.1.5.3.2 unregisterComponent

Signature

```
rcResult unregisterComponent (
    [in]  pUUID cmpId
);
```

Qualifiers

synchronous

thread-safe

© ISO/IEC 2007 – 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 the executable component from the registry.

Parameters

Table 23

Name	Description
cmpId	Reference to the identifier of the executable component.

Return Values

Standard

Precondition

componentRecords = A ∪ <cmpId, location>

Action

None.

Postcondition

componentRecords = A

6.1.1.5.3.3 registerService

Signature

```
rcResult registerService (
    [in]  pUUID cmpId,
    [in]  pUUID svcId
);
```

Qualifiers

synchronous

thread-safe

Description

Add the service to the registry so that it can be instantiated at a later time. This operation creates the binding between the executable component and the service.

© ISO/IEC 2007 – 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 24

Name	Description
cmpId	Reference to the identifier of the executable component.
svcId	Reference to the identifier of the service

Return Values

Only non-standard error values are listed

Table 25

Name	Description
RC_ERR_RUNTIME_UNKNOWN_COMPONENT	The executable component is not registered yet.
RC_ERR_RUNTIME_ALREADY_REGISTERED	The component is already registered

Precondition

containerRecords = A

A ∩ < X, svcId> = ∉ for all X

Action

None.

Postcondition

containerRecords = A ∪ <cmpId, svcId>

6.1.1.5.3.4 unregisterService

Signature

```
rcResult unregisterService (
    [in]  pUUID svcId
);
```

Qualifiers

synchronous

thread-safe

Copyright International Organization for Standardization © ISO/IEC 2007 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Description

Remove the service from the registry. This operation removes the binding between the executable component and the service.

Parameters

Table 26

Name	Description
svcId	Reference to the identifier of the service

Return Values

Standard

Precondition

containerRecords = A

Action

None.

Postcondition

containerRecords = A

A ∩ < X, svcId> = ∉ for all X

6.1.1.5.3.5 setComplies

Signature

```
rcResult setComplies (
    [in]  pUUID complying,
    [in]  pUUID blueprint
);
```

Qualifiers

synchronous

thread-safe

Description

Add a complies relation to the registry.

© ISO/IEC 2007 – 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 27

Name	Description
complying	Reference to the identifier of the compliant service.
blueprint	Reference to the identifier of the blueprint service

Return Values

Only non-standard error values are listed

Table 28

Name	Description
RC_ERR_RUNTIME_NOT_ALLOWED	This operation is not allowed

Precondition

compliesRecords = A

Action

None.

Postcondition

containerRecords = A ∪ <complying, blueprint>

6.1.1.5.3.6 clearComplies

Signature

```
rcResult clearComplies (
    [in]  pUUID complying,
    [in]  pUUID blueprint
);
```

Qualifiers

synchronous

thread-safe

Description

Remove a complies relation to the registry.

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

© ISO/IEC 2007 – 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 29

Name	Description
complying	Reference to the identifier of the compliant service.
blueprint	Reference to the identifier of the blueprint service

Return Values

Standard

Precondition

```
compliesRecords = A
```

Action

None.

Postcondition

```
containerRecords = A \ <complying, blueprint>
```

6.1.1.5.4 rcIComponentItr

Qualifiers

Iterator.

Description

This interface is returned as a result of an operation of the `rcIRegistryInspect` interface. The interface can be used to walk through the executable components and their locations that are in the registry.

Interface ID

uuid(e18f5bee-ea1f-44a6-ac7d-26a2255a71a9)

Remarks

None

6.1.1.5.4.1 reset

Signature

```
rcResult reset (
);
```

Qualifiers

synchronous

© ISO/IEC 2007 – 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

Reset cursor to initial position.

Parameters

None

Return Values

Standard

Precondition

`True`

Action

None.

Postcondition

`Cursor is on intial position`

6.1.1.5.4.2 next

Signature

```
rcResult next (
  [out] rcRuntime_ComponentRecord_t *retValue
);
```

Qualifiers

synchronous

Description

Return the next componentRecord.

Parameters

Table 30

Name	Description
`retValue`	`Returned componentRecord`

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 31

Name	Description
RC_ERR_RUNTIME_NO_SUCH_ELEMENT	There is no next element

Precondition

Cursor is at position N

Action

Return next componentRecord, as a result the cursor position is increased by 1.

Postcondition

Cursor is at position N+1

6.1.1.5.4.3 atEnd

Signature

```
rcResult atEnd (
  [out] Bool *retValue
);
```

Qualifiers

synchronous

Description

Return whether the cursor is at the last postion.

Parameters

Table 32

Name	Description
retValue	Return value

Return Values

Standard

Precondition

Cursor is at position N

Action

None.

© ISO/IEC 2007 – 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

Postcondition

`retValue == True` when N is the last position

`retValue == False` otherwise

6.1.1.5.5 rclContainerItr

Qualifiers

Iterator.

Description

This interface is returned as a result of an operation of the `rcIRegistryInspect` interface. The interface can be used to walk through the executable components and their contained services that are in the registry.

Interface ID

uuid(2292732b-5f9b-44e6-92ec-4dbbf65031ad)

Remarks

None

6.1.1.5.5.1 reset

Signature

```
rcResult reset (
);
```

Qualifiers

synchronous

Description

Reset cursor to initial position.

Parameters

None

Return Values

Standard

Precondition

`True`

Action

None.

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

© ISO/IEC 2007 – 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

Postcondition

```
Cursor is on intial position
```

6.1.1.5.5.2 next

Signature

```
rcResult next (
  [out] rcRuntime_ContainerRecord_t *retValue
);
```

Qualifiers

synchronous

Description

Return the next containerRecord.

Parameters

Table 33

Name	Description
retValue	Returned containerRecord

Return Values

Only non-standard error values are listed

Table 34

Name	Description
RC_ERR_RUNTIME_NO_SUCH_ELEMENT	There is no next element

Precondition

```
Cursor is at position N
```

Action

Return next containerRecord, as a result the cursor position is increased by 1.

Postcondition

```
Cursor is at position N+1
```


© ISO/IEC 2007 – 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

6.1.1.5.5.3 atEnd

Signature

```
rcResult atEnd (
  [out] Bool *retValue
);
```

Qualifiers

synchronous

Description

Return whether the cursor is at the last postion.

Parameters

Table 35

Name	Description
`retValue`	`Return value`

Return Values

Standard

Precondition

`Cursor is at position N`

Action

none.

Postcondition

`retValue == True` when N is the last position

`retValue == False` otherwise

6.1.1.5.6 rclCompliesItr

Qualifiers

Iterator.

Description

This interface is returned as a result of an operation of the `rcIRegistryInspect` interface. The interface can be used to walk through the complies relations that are in the registry.

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

© ISO/IEC 2007 – 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

Interface ID

uuid(9da241cb-8f14-4ff0-9296-e379abbe0b68)

Remarks

None

6.1.1.5.6.1 reset

Signature

```
rcResult reset (
);
```

Qualifiers

synchronous

Description

Reset cursor to initial position.

Parameters

None

Return Values

Standard

Precondition

```
True
```

Action

None.

Postcondition

```
Cursor is on intial position
```

6.1.1.5.6.2 next

Signature

```
rcResult next (
  [out] rcRuntime_CompliesRecord_t *retValue
);
```

Qualifiers

synchronous

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2007 – All rights reserved
Not for Resale

Description

Return the next compliesRecord.

Parameters

Table 36

Name	Description
retValue	Returned compliesRecord

Return Values

Only non-standard error values are listed

Table 37

Name	Description
RC_ERR_RUNTIME_NO_SUCH_ELEMENT	There is no next element

Precondition

Cursor is at position N

Action

Return next compliesRecord, as a result the cursor position is increased by 1.

Postcondition

Cursor is at position N+1

6.1.1.5.6.3 atEnd

Signature

```
rcResult atEnd (
  [out] Bool *retValue
);
```

Qualifiers

synchronous

Description

Return whether the cursor is at the last position.

© ISO/IEC 2007 – 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 38

Name	Description
retValue	Return value

Return Values

Standard

Precondition

Cursor is at position N

Action

none.

Postcondition

retValue == True when N is the last position

retValue == False otherwise

6.1.1.5.7 rclRegistryInspect

Qualifiers

None.

Description

This interface of a Runtime Environment role provides facilities inspection of the complete registry contents.

Interface ID

uuid(c845e1b3-8176-4279-98e2-dd7e6cf5c9a5)

Remarks

None

6.1.1.5.7.1 getComponentRecords

Signature

```
rcResult getComponentRecords (
    [out]  prcIComponentItr *retValue
);
```

Qualifiers

synchronous

© ISO/IEC 2007 – 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

thread-safe

Description

Get all componentRecords in the registry returns a reference to an instance of a componentRecord iterator or NULL if the list is empty.

Parameters

Table 39

Name	Description
`retValue`	`Reference instance of componentRecord iterator`

Return Values

Standard

Precondition

`componentRecords = A`

Action

None.

Postcondition

`retValue == NULL if A == ∅`

`retValue` is iterator for `A` otherwise

6.1.1.5.7.2 getContainerRecords

Signature

```
rcResult getContainerRecords (
    [out]  prcIContainerItr *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Get all containerRecords from the registry returns a reference to an instance of a containerRecord iterator or NULL if the list is empty.

© ISO/IEC 2007 – 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 40

Name	Description
retValue	Reference instance of containerRecord iterator

Return Values

Standard

Precondition

containerRecords = A

Action

None.

Postcondition

retValue == NULL if A == ∅

retValue is iterator for A otherwise

6.1.1.5.7.3 getCompliesRecords

Signature

```
rcResult getCompliesRecords (
    [out]  prcICompliesItr *retValue
);
```

Qualifiers

synchronous

thread-safe

Description

Get all compliesRecords in the registry returns a reference to an instance of a compliesRecord iterator or NULL if the list is empty.

Parameters

Table 41

Name	Description
retValue	Reference instance of compliesRecord iterator

© ISO/IEC 2007 – 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

`compliesRecords = A`

Action

None.

Postcondition

`retValue == NULL if A == ∅`

`retValue` is iterator for `A` otherwise

6.1.1.5.8 rclUuidItr

Qualifiers

Iterator.

Description

This interface is returned by an operation of the rclLoadedComponentManagement of a Runtime Environment role. The interface can be used to walk through a list of UUIDs.

Interface ID

uuid(d3bab695-7796-4755-b8ba-09dff6a1edeb)

Remarks

None

6.1.1.5.8.1 reset

Signature

```
rcResult reset (
);
```

Qualifiers

synchronous

Description

Reset cursor to initial position.

Parameters

None

© ISO/IEC 2007 – 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

`True`

Action

None.

Postcondition

`Cursor is on intial position`

6.1.1.5.8.2 next

Signature

```
rcResult next (
  [out] pUUID *retValue
);
```

Qualifiers

synchronous

Description

Return the next reference to UUID.

Parameters

Table 42

Name	Description
`retValue`	`Returned UUID ref`

Return Values

Only non-standard error values are listed

Table 43

Name	Description
`RC_ERR_RUNTIME_NO_SUCH_ELEMENT`	`There is no next element`

Precondition

`Cursor is at position N`

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

© ISO/IEC 2007 – 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

Action

Return next UUID ref, as a result the cursor position is increased by 1.

Postcondition

`Cursor is at position N+1`

6.1.1.5.8.3 atEnd

Signature

```
rcResult atEnd (
  [out] Bool *retValue
);
```

Qualifiers

synchronous

Description

Return whether the cursor is at the last position.

Parameters

Table 44

Name	Description
`retValue`	`Return value`

Return Values

Standard

Precondition

`Cursor is at position N`

Action

none.

Postcondition

`retValue == True` when N is the last position

`retValue == False` otherwise

6.1.1.5.9 rclLoadedComponentManagement

Qualifiers

None.

© ISO/IEC 2007 – 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

This interface of the Runtime Environment role can be used to manage the lifetime of components. It can be used to find out which components are loaded, whether they are initialized and to unload these components

Interface ID

uuid(f4a9a915-a2dc-4a28-89a7-f1b96312ca1f)

Remarks

None

6.1.1.5.9.1 getLoadedList

Signature

```
rcResult getLoadedList (
  [out] prcIUuidItr *retValue
);
```

Qualifiers

Synchronous

Thread-safe

Description

Get all loaded components. Returns a reference to an UUID iterator instance

Parameters

Table 45

Name	Description
retValue	Return value

Return Values

Standard

Precondition

A = set of loaded components

Action

none.

Postcondition

retValue == NULL if A == ∅

retValue == iterator for a

© ISO/IEC 2007 – 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

6.1.1.5.9.2 isInitialized

Signature

```
rcResult isInitialized (
  [in] pUUID cmpId,
  [out] Bool *retValue
);
```

Qualifiers

Synchronous

Thread-safe

Description

Return whether the component is initialized.

Parameters

Table 46

Name	Description
`cmpId`	`Reference to identifier of executable component`
`retValue`	`Return value`

Return Values

Standard

Precondition

`True`

Action

none.

Postcondition

`retValue == True` when executable component identified by `cmpId` is initialized

`retValue == False` otherwise

6.1.1.5.9.3 unloadComponent

Signature

```
rcResult isInitialized (
  [in] pUUID cmpId,
);
```

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

© ISO/IEC 2007 – 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

Thread-safe

Description

Unload a component.

Parameters

Table 47

Name	Description
`cmpId`	`Reference to identifier of the executable component`

Return Values

Only non-standard error values are listed

Table 48

Name	Description
`RC_ERR_RUNTIME_CANNOT_FINALIZE`	`Executable component cannot be finalized and therefore not be unloaded`

Precondition

Executable component identified by `cmpId` is loaded

Action

none.

Postcondition

Executable component identified by `cmpId` is unloaded.

6.1.2 ServiceManager

6.1.2.1 Concepts

The function of Service Manager is to enable client to access functionality offered by services without having knowledge of how the detail implementation of the service. The access is requested by using the standardized logical component identification. If the requested service has been instantiated before, a response to an access request will be simply returning the reference to the already instantiated service. If the service has not been instantiated before, Service Manager will instantiate it (by requesting to the Runtime Environment through service id) first, and then, hands the reference to the service to the requesting client.

© ISO/IEC 2007 – 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

Service Manager works base on metadata. It keeps the metadata information about logical components supported by the M3W implementation and also metadata information about services that are installed in the M3W. Base on these metadata, Service Manager upon a request from client to access certain functionality, decides which service or services needs to be instantiated (and linked if there are dependencies among services).

6.1.2.2 Types & Constants

6.1.2.2.1 Error Codes

Signature

```
const rcResult RC_ERR_SM_NOT_IMPLEMENTED = 0x00001001
const rcResult RC_ERR_SM_CANNOT_INITIALIZE = 0x00001002
const rcResult RC_ERR_SM_CANNOT_FINALIZE = 0x00001004
const rcResult RC_ERR_RUNTIME_NO_SUCH_ELEMENT = 0x00001008
const rcResult RC_ERR_SM_NO_SUCH_SERVICE = 0x00001010
const rcResult RC_ERR_SM_ALREADY_REGISTERED = 0x00001080
const rcResult RC_ERR_SM_UNKNOWN_COMPONENT = 0x00001100
const rcResult RC_ERR_SM_NOT_ALLOWED = 0x00001200
```

Qualifiers

— Error-codes

Description

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

Constants

Table 49

Name	Description
RC_ERR_SM_NOT_IMPLEMENTED	This error is returned when a particular function is not implemented by the runtime environment
RC_ERR_SM_CANNOT_INITIALIZE	This error is returned when a component containing a service cannot be initialized
RC_ERR_SM_CANNOT_FINALIZE	This error is returned when an executable component cannot be finalized.
RC_ERR_SM_NO_SUCH_ELEMENT	This error is returned when an (instance of an) element is requested/referred that does not exist.
RC_ERR_SM_NO_SUCH_SERVICE	This error is returned when a access to a service is requested and the service is not available
RC_ERR_SM_ALREADY_REGISTERED	This error is returned on an attempt to register an same object multiple times
RC_ERR_SM_UNKNOWN_COMPONENT	This error is returned on an attempt to request access to a service by give an unknown logical component as parameter
RC_ERR_SM_NOT_ALLOWED	This error is returned when the operation is not allowed

© ISO/IEC 2007 – 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

6.1.2.3 Logical Component

6.1.2.3.1 Interface-Role Model

Figure 4 — Service Manager's Interface-Role Model, depicts the interface-role model of the ServiceManager (grey box). It shows the interfaces and the roles involved with this component.

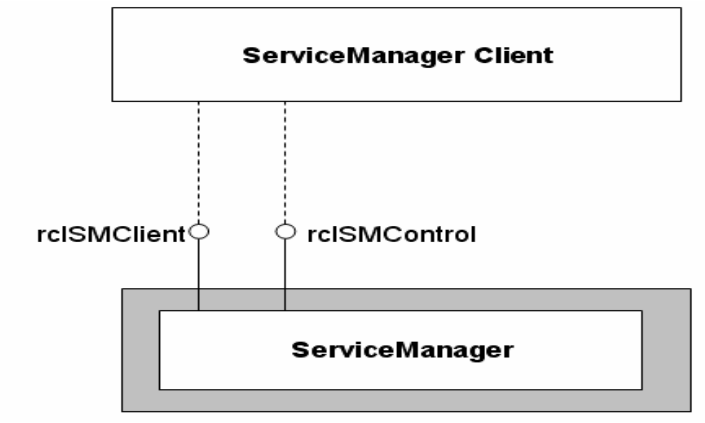


Figure 4 — Service Manager's Interface-Role Model

A Service Manager Logical Component contains the Service Manager role that provides the rclSMClient interface. This interface can be used by a client to check the availability of a certain multimedia functionality and/or request an instance of service. Service Manager role has an optional interface to control the knowledge/state of the Service Manager. This involves registering and un-registering the metadata.

6.1.2.3.2 Diversity

6.1.2.3.2.1 Provided Interfaces

Table 50

Role	Interface	Presence
Service Manager	`rcISMClient`	`Mandatory`
Service Manager	`rcISMControl`	`Optional`

© ISO/IEC 2007 – 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

6.1.2.3.2.2 Configurable Items

None

6.1.2.3.2.3 Constraints

None

6.1.2.3.3 Instantiation

6.1.2.3.3.1 Objects Created

Service Manager logical component and role are always available. They are not created by the clients of the M3W.

Table 51

Type	Object	Multiplicity
`ServiceManager`	`ServiceManager`	`1`

6.1.2.3.3.2 Initial State

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

— none

6.1.2.3.4 Execution Constraints

If implemented with thread, it shall be thread-safe.

6.1.2.4 Roles

6.1.2.4.1 Service Manager

Signature

```
role ServiceManager {}
```

Qualifiers

— root

Description

Service Manager provides access to the functionalities provided by the services. The access request is made base on standardized logical component.

Instantiation

Service Manager role is always instantiated as part of the Service Manager logical component.

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

© ISO/IEC 2007 – 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

6.1.2.5 Interfaces

6.1.2.5.1 rcISMClient

Qualifiers

None

Description

This interface of Service Manager role provides facilities for instantiation of realization elements (services).

Interface ID

```
f877fbb1-e2e2-4c3e-ba7a-3ba1000361af
```

Remarks

None

6.1.2.5.1.1 isServiceAvailable

Signature

```
rcResult isServiceAvailable (
    [in] pUUID lcId,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Inspect whether there is service instance available for a certain logical component.

Parameters

Table 52

Name	Description
lcId	Reference to the UUID of the logical component.
retValue	A boolean value. True -> there exist service that implements that logical component; False -> otherwise.

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 53

Name	Description
RC_ERR_SM_UNKNOWN_COMPONENT	The logical component that is given in the input parameter is not recognized by the Service Manager.

Precondition

— True

Action

Inspect the existence implementation of the requested logical component base on the knowledge from the registered metadata.

Post condition

— `retValue == True` when there is at least one service that implements the logical component

— `retValue == False` otherwise

6.1.2.5.1.2 getInstanceForLogicalComponent

Signature

```
rcResult getInstanceForLogicalComponent (
    [in] pUUID lcId,
    [out] prcIService *retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Get an instance of service that implements the requested logical component.

Parameters

Table 54

Name	Description
lcId	Reference to the UUID of the requested logical component.
retValue	Reference to the rcIService interface of the created service instance.

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 55

Name	Description
RC_ERR_SM_NO_SUCH_SERVICE	There is not such service implementation for the requested logical component.
RC_ERR_SM_UNKNOWN_COMPONENT	The logical component that is given in the input parameter is not recognized by the Service Manager.
RC_ERR_SM_CANNOT_INITIALIZE	The service instance cannot be initialized
RC_ERR_SM_CANNOT_FINALIZE	Instantiation of the service instance cannot be finalized. Typical problem of this error is because the failure in linking/binding several service instances (if several instances are needed to fulfil the requested logical component)

Precondition

— True

Action

Get (instantiate if no instance existed yet) the handler to the instance of a service for the requested logical component.

Postcondition

```
retValue == <reference to rcIService implements requested logical component>
```

6.1.2.5.2 rcISMControl

Qualifiers

None

Description

This interface of Service Manager role provides facilities for register and un-register necessary metadata both for logical component and service.

Interface ID

```
8d026b36-d4c7-43ee-a211-e158fd8cb9fc
```


--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2007 – All rights reserved
Not for Resale

Remarks

None

6.1.2.5.2.1 registerLogicalComponentMetadata

Signature

```
rcResult registerLogicalComponentMetadata (
    [in] String path,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Register logical component metadata so that Service Manager implementation knows what are the logical components recognized by this M3W implementation.

Parameters

Table 56

Name	Description
path	Path to the file where the metadata is defined
retValue	A boolean value indicating whether the operation success or not.

Return Values

Only non-standard error values are listed

Table 57

Name	Description
RC_ERR_SM_NOT_IMPLEMENTED	This method is not implemented yet by the Service Manager.
RC_ERR_SM_ALREADY_REGISTERED	The metadata has been registered before.
RC_ERR_SM_NOT_ALLOWED	This operation is not allowed to be executed by the caller.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

Precondition

— True

Action

Register the logical component metadata. The implementation of ServiceManager will locate the metadata file location, load it to its memory/stored (i.e., in the form of DOM object).

Post condition

```
Elements of logical component == Current elements of logical component + element(s) from the metadata
```

6.1.2.5.2.2 unregisterLogicalComponentMetadata

Signature

```
rcResult unregisterLogicalComponentMetadata (
    [in] pUUID lcId,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove the information of a logical component.

Parameters

Table 58

Name	Description
lcId	Reference to the UUID of the logical component whose metadata will be un-registered/removed
retValue	A boolean value indicating whether the operation success or not.

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 59

Name	Description
RC_ERR_SM_NOT_IMPLEMENTED	This method is not implemented yet by the Service Manager.
RC_ERR_SM_NO_SUCH_ELEMENT	There is no element that represents that logical component.
RC_ERR_SM_NOT_ALLOWED	This operation is not allowed to be executed by the caller.

Precondition

— True

Action

Remove the element the represents the logical component whose id is given in the input parameter.

Postcondition

```
Elements of logical component == Current elements of logical component – removed element
```

6.1.2.5.2.3 registerServiceMetadata

Signature

```
rcResult registerServiceMetadata (
    [in] String path,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

Description

Register service metadata so that Service Manager implementation knows the service that resides in the M3W. Later it will be used in decision to determine whether the service can be used when a multimedia functionality is requested.

© ISO/IEC 2007 – 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 60

Name	Description
Path	Path to the file where the metadata is defined
retValue	A boolean value indicating whether the operation success or not.

Return Values

Only non-standard error values are listed

Table 61

Name	Description
RC_ERR_SM_NOT_IMPLEMENTED	This method is not implemented yet by the Service Manager.
RC_ERR_SM_ALREADY_REGISTERED	The metadata has been registered before.
RC_ERR_SM_NOT_ALLOWED	This operation is not allowed to be executed by the caller.

Precondition

— True

Action

Register the service metadata. The implementation of Service Manager will locate the metadata file location, load it to its memory/stored (i.e., in the form of DOM object).

Post condition

```
Elements of service == Current elements of service + element(s) from the metadata
```

6.1.2.5.2.4 unregisterServiceMetadata

Signature

```
rcResult unregisterLogicalComponentMetadata (
    [in] pUUID srvId,
    [out] Bool retValue
);
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – 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 the information of a service.

Parameters

Table 62

Name	Description
srvId	Reference to the UUID of the service whose metadata will be un-registered/removed
retValue	A boolean value indicating whether the operation success or not.

Return Values

Only non-standard error values are listed

Table 63

Name	Description
RC_ERR_SM_NOT_IMPLEMENTED	This method is not implemented yet by the Service Manager.
RC_ERR_SM_NO_SUCH_ELEMENT	There is no element that represents that service.
RC_ERR_SM_NOT_ALLOWED	This operation is not allowed to be executed by the caller.

Precondition

— True

Action

Remove the element the represents the service whose id is given in the input parameter.

Postcondition

```
Elements of service == Current elements of service – removed element
```

6.2 Interaction with and between realization elements

6.2.1 MarshalTypeFactory

6.2.1.1 Concepts

The function of a MarshalTypeFactory logical component is to create objects which implements the generic interface rcIParam. This is needed in order to marshal parameter values within network messages. It exposes

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

© ISO/IEC 2007 – 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

several creation methods in order to allow client code to prepare remote method execution requests on the REMI-R logical component.

6.2.1.2 Types & Constants

6.2.1.2.1 Public Types & Constants

6.2.1.2.1.1 rcParamType_t

Signature

```
typedef enum _rcParamType_t {
        ParamInt8,
        ParamInt16,
        ParamInt32,
        ParamUInt8,
        ParamUInt16,
        ParamUInt32,
        ParamBool,
        ParamChar,
        ParamDouble,
        ParamFloat,
        ParamAggregate,
        ParamSequence,
        ParamUUID,
        ParamHandle,
} rcParamType_t, *prcParamType_t;
```

Qualifiers

- enum

Description

Enumeration of the supported types for marshalling.

6.2.1.3 Logical Component

6.2.1.3.1 Interface-Role Model

Figure 5 — Interface-Role Model, depicts the interface-role model of the MarshalTypeFactory (grey box). It shows the interfaces and the roles involved with this component.

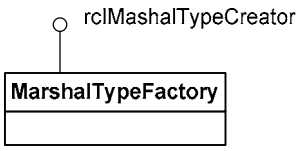


Figure 5 — Interface-Role Model

A MarshalTypeFactory Logical Component contains the Marshal Type Creator role that provides the rclMarshalTypeCreator interface. This interface can be used by a client to create instances which implements rclParam interface and one of its derived ones.

© ISO/IEC 2007 – 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 Diversity

6.2.1.3.2.1 Provided Interfaces

Table 64

Role	Interface	Presence
Marshal Type Creator	rclMarshalTypeCreator	Mandatory

6.2.1.3.2.2 Configurable Items

None

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 MarshalTypeFactory is created by the clients of the M3W, if they need to create marshal types. The MarshalTypeFactory is a Singleton of the M3W, after the first instantiation any other client will obtain a reference to the unique MarshalTypeFactory instance.

Table 65

Type	Object	Multiplicity
Marshal Type Creator	marshalTypeFactory	0…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 logical component MarshalTypeFactory is thread-safe.

6.2.1.4 Roles

6.2.1.4.1 Marshal Type Creator

Signature

```
role MarshalTypeCreator {
}
```

© ISO/IEC 2007 – 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

— root

Description

A Marshal Type Creator enables the create instances which implements rcIParam for marshalling remote invocation parameters.

Invariants

— None

Instantiation

The Marshal Type Creator role is always instantiated as part of the MarshalTypeFactory logical component.

Active Behaviour

— None

6.2.1.5 Interfaces

6.2.1.5.1 rcIMarshalTypeCreator

Qualifiers

None.

Description

This interface of a Marshal Type Creator role provides facilities for creation of instance of rcIParam interface which are treated by REMI-R for passing parameters of interface method invocation requests.

Interface ID

uuid(5f5bda4c-dd30-43bf-ac8c-1e2c748029a4)

Remarks

None

6.2.1.5.1.1 createParam

Signature

```
rcResult createParam (
    [in] rcParamType_t type,
    [out] prcIParam *param_iref);
);
```

Qualifiers

— synchronous

— thread-safe

© ISO/IEC 2007 – 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

It returns an interface reference to an instance which implements a specific interface for managing the encapsulation of the given type.

Parameters

Table 66

Name	Description
Type	Type to be encapsulated
param_iref	Reference to the rcIParam interface

Return Values

Standard

Precondition

true

Action

Creation of an instance which implements rcIParam specific interface for type `type`.

Postcondition

param_iref = <reference to the created instance>

6.2.1.5.2 rcIParam

Qualifiers

None.

Description

This interface is used as a return type of rcIMarshalTypeCreator methods. This interface provides facilities for manipulating parameters in a generic manner.

Interface ID

uuid(58f442b1-ab75-4324-904d-29ce40211b03)

Remarks

None

6.2.1.5.2.1 getType

Signature

```
rcResult getType (
    [out] prcParamType_t type
);
```

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

© ISO/IEC 2007 – 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

— thread-safe

Description

It returns the parameter type that is encapsulated inside this rcIParam interface.

Parameters

Table 67

Name	Description
Type	Reference to rcParamType_t enumeration which identifies the parameter type

Return Values

standard error values

Precondition

true

Action

Retrieving of encapsulated type.

Postcondition

type = <reference to the encapsulated parameter type>

6.2.1.5.3 rcIParamInt8

Qualifiers

None.

Description

This interface inherits from rcIParam. It provides facilities for managing a marshal type which encapsulate an integer value.

Interface ID

uuid(67c001e5-f471-40a0-be26-2aaf8b0c836a)

Remarks

None

© ISO/IEC 2007 – 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

6.2.1.5.3.1 getInt8

Signature

```
rcResult getInt8 (
    [out] int *value
);
```

Qualifiers

— synchronous

— thread-safe

Description

It returns the value which is encapsulated.

Parameters

Table 68

Name	Description
Value	Reference to integer to be filled with the encapsulated value.

Return Values

standard error values

Precondition

encapsulatedValue = x

Action

Retrieving of encapsulated value.

Postcondition

value = <reference to 8 bit signed integer containing x>

6.2.1.5.3.2 setInt8

Signature

```
rcResult setInt8 (
    [in] Int8 *value
);
```

Qualifiers

— synchronous

— thread-safe

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

© ISO/IEC 2007 – 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

It returns the value which is encapsulated.

Parameters

Table 69

Name	Description
Value	Reference to integer which contains the value to be encapsulated.

Return Values

standard error values

Precondition

true

Action

Copying the value.

Postcondition

encapsulatedValue = value

6.2.1.5.4 rclParamAggregate

Qualifiers

None.

Description

This interface inherits from rclParam. It provides facilities for managing a marshal type which encapsulate an user-defined type.

Interface ID

uuid(722c161d-804b-448f-b4f2-2118855b298b)

Remarks

None

6.2.1.5.4.1 createIterator

Signature

```
rcResult createIterator (
    [out] prcIParamIterator *iterator;
);
```


© ISO/IEC 2007 – 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

— thread-safe

Description

It returns an iterator in order to inspect the aggregation of rcIParam interface references.

Parameters

Table 70

Name	Description
iterator	Reference to be filled with the corresponding rcIParamIterator interface reference

Return Values

standard error values

Precondition

aggregatedParams = A

Action

Retrieving of the iterator.

Postcondition

iterator == NULL if A == ∅

iterator is iterator for `A` otherwise

6.2.1.5.4.2 getCount

Signature

```
rcResult getCount (
    [out] int *count
);
```

Qualifiers

— synchronous

— thread-safe

Description

It returns the count of the rcIParam interface references which are aggregated by this instance.

© ISO/IEC 2007 – 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 71

Name	Description
Count	Reference to integer which contains the number of aggregation elements.

Return Values

standard error values

Precondition

aggregatedParams = A

Action

Copying the value.

Postcondition

count = <cardinality of A>

6.2.1.5.4.3 addElement

Signature

```
rcResult addElement (
    [in] prcIParam element
);
```

Qualifiers

— synchronous

— thread-safe

Description

It adds a rcIParam interface reference in the parameter aggregation.

Parameters

Table 72

Name	Description
Element	Reference to an rcIParam instance to be added to the current aggregation.

Return Values

standard error values

© ISO/IEC 2007 – 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

Precondition

aggregatedParams = A

Action

Adding the parameter reference.

Postcondition

aggregatedParams = A ∪ <element>

6.2.1.5.5 rcIParamSequence

Qualifiers

None.

Description

This interface inherits from rcIParamAggregate. It provides facilities for managing a marshal type which encapsulate an homogeneous sequence of parameters (i.e. an array).

Interface ID

uuid(ee487131-fdef-4bd4-8b4e-02ba5dbb1388)

Remarks

None

6.2.1.5.5.1 getElementType

Signature

```
rcResult getElementType (
    [out] prcParamType_t elementType
);
```

Qualifiers

— synchronous

— thread-safe

Description

It returns the parameter type of the elements that are encapsulated inside this rcIParamSequence interface.

Parameters

Table 73

Name	Description
elementType	Reference to rcParamType_t enumeration which identifies the parameter type of the elements

© ISO/IEC 2007 – 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 error values

Precondition

true

Action

Retrieving of encapsulated type.

Postcondition

type = <reference to the encapsulated parameter type>

6.2.1.5.6 rclParamHandle

Qualifiers

None.

Description

This interface inherits from rclParam. It provides facilities for managing a marshal type which encapsulate an instance handle (i.e. a logical reference to a remote instance).

Interface ID

uuid(1306ff0c-ac65-4867-8849-9549829efc33)

Remarks

None

6.2.1.5.6.1 getHandle

Signature

```
rcResult getHandle (
    [out] prcObjHandle *handle
);
```

Qualifiers

— synchronous

— thread-safe

Description

It returns the instance handle which is encapsulated.

© ISO/IEC 2007 – 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 74

Name	Description
handle	Reference to a prcObjHandle to be filled with the encapsulated instance handle.

Return Values

standard error values

Precondition

encapsulatedInstanceHandle = h

Action

Retrieving of encapsulated instance handle.

Postcondition

handle = <reference to prcObjHandle containing h>

6.2.1.5.6.2 setHandle

Signature

```
rcResult setHandle (
    [in] prcObjHandle handle
);
```

Qualifiers

— synchronous

— thread-safe

Description

It sets the instance handle which is encapsulated.

Parameters

Table 75

Name	Description
Handle	Reference to rcObjHandle to be encapsulated.

Return Values

standard error values

© ISO/IEC 2007 – 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

Precondition

true

Action

Copying the value.

Postcondition

encapsulatedInstanceHandle = handle

6.2.1.5.7 rclParamIterator

Qualifiers

Iterator.

Description

This interface is returned by an operation of the rclParamAggregate. The interface can be used to walk through a list of rclParam interface references.

Interface ID

uuid(2739e01b-42a4-4cee-b3be-d0e44609b4c1)

Remarks

None

6.2.1.5.7.1 reset

Signature

```
rcResult reset (
);
```

Qualifiers

synchronous

Description

Reset cursor to initial position.

Parameters

None

Return Values

Standard

Precondition

True

© ISO/IEC 2007 – 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

Action

None.

Postcondition

Cursor is on intial position

6.2.1.5.7.2 next

Signature

```
rcResult next (
  [out] prcIParam *element
);
```

Qualifiers

synchronous

Description

Return the next reference to UUID.

Parameters

Table 76

Name	Description
element	Returned rcIParam reference

Return Values

Only non-standard error values are listed

Table 77

Name	Description
RC_ERR_RUNTIME_NO_SUCH_ELEMENT	There is no next element

Precondition

Cursor is at position N

Action

Return next rcIParam reference, as a result the cursor position is increased by 1.

Postcondition

Cursor is at position N+1

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

© ISO/IEC 2007 – 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

6.2.1.5.7.3 atEnd

Signature

```
rcResult atEnd (
  [out] Bool *isAtEnd
);
```

Qualifiers

synchronous

Description

Return whether the cursor is at the last position.

Parameters

Table 78

Name	Description
isAtEnd	It returns a Boolean value notifying if the current position is the last one

Return Values

Standard

Precondition

Cursor is at position N

Action

none.

Postcondition

`isAtEnd == True` when N is the last position

`isAtEnd == False` otherwise

6.2.2 REMI-R

6.2.2.1 Concepts

The function of a REMI-R logical component is to enable the usage of realizations elements (physical services, see 10.1.7), which are located in other M3W Systems. More in details, it can request the execution of a method of an interface exposed by an instantiated element in a remote M3W System. The realization element it contacts is reached by means of an available TCP transport established among the M3W Systems.

© ISO/IEC 2007 – 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

6.2.2.2 Types & Constants

6.2.2.2.1 Public Types & Constants

6.2.2.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE = 0x00000001
const rcResult RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE = 0x00000002
const rcResult RC_ERR_INVOKER_WRONG_PARAMS = 0x00000004
const rcResult RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION = 0x00000008
const rcResult RC_ERR_INVOKER_PROXY_ALREADY_REGISTERED =  0x00000010
const rcResult RC_ERR_INVOKER_PROXY_NOT_REGISTERED =  0x00000020
```

Qualifiers

— Error-codes

Description

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

Constants

Table 79

Name	Description
RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE	This error is returned when an invocation asks for a service that is not registered in any M3W System in the distributed environment
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_INVOKER_WRONG_PARAMS	This error is returned when an invocation is performed with an input parameters set not matching the interface method signature
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of a remote method invocation an exception has been raised
RC_ERR_INVOKER_PROXY_ALREADY_REGISTERED	This error is returned on registration of a proxy when an association already exist for a given service identifier
RC_ERR_INVOKER_PROXY_NOT_REGISTERED	This error is returned when an association with a proxy does not exist for a given service identifier

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

© ISO/IEC 2007 – 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

6.2.2.2.1.2 URI

Signature

```
typedef String URI;
```

Qualifiers

none

Description

Data structure used to store the information related to a reachable M3W System.

6.2.2.2.1.3 rcObjHandle_t

Signature

```
struct _rcObjHandle_t {
           UInt32 instanceId;
           URI location;
} rcObjHandle_t, *prcObjHandle_t;
```

Qualifiers

— struct-element

Description

Data structure used to store the information related to a reachable M3W System.

6.2.2.2.1.4 rclParam

See 6.2.1.5.2

6.2.2.3 Logical Component

6.2.2.3.1 Interface-Role Model

Figure 6 — Interface-Role Model, depicts the interface-role model of the REMI-R (grey box). It shows the interfaces and the roles involved with this component.

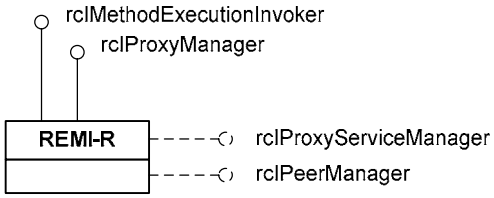


Figure 6 — Interface-Role Model

A REMI-R Logical Component contains the Remote Requester role that provides the rclMethodExecutionInvoker interface. This interface can be used by a client to invoke methods on remote instances of services. The Remote Requester role also provides the rclProxyManager interface which is used

© ISO/IEC 2007 – 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

by clients to create/manage transparent proxies for the remote instances. Other two optional interfaces can be used to configure the REMI-R in order to select the suitable proxy implementation for a given service/interface and to manage the “contact list” for retrieving and using remote instances.

6.2.2.3.2 Diversity

6.2.2.3.2.1 Provided Interfaces

Table 80

Role	Interface	Presence
Remote Requester	rclMethodExecutioInvoker	Mandatory
Remote Requester	rclProxyManager	Mandatory
Remote Requester	rclProxyServiceManager	Optional
Remote Requester	rclPeerManager	Optional

6.2.2.3.2.2 Configurable Items

The behaviour of the REMI-R depends on the list of the registered Proxy Services (i.e. Phisical Services implementing Proxy behaviour) and on the list of the network peers which are known by the local M3WSystems. The former contains association records among service/interface identifiers and proxy service identifiers. The latter contains the network addresses that has to be contacted in order to reach the corresponding REMI-P service instances (see 6.2.3). The registry contents is specified using the following attributes.

Table 81

Role	Attribute
Remote Requester	proxyServicesRecords
Remote Requester	Peers

6.2.2.3.2.3 Constraints

None

6.2.2.3.3 Instantiation

6.2.2.3.3.1 Objects Created

The logical component REMI-R are created by the clients of the M3W, if they need to exploits services in a distributed environment. The REMI-R is a Singleton of the M3W, after the first instantiation any other client will obtain a reference to the unique REMI-R instance.

© ISO/IEC 2007 – 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

Table 82

Type	Object	Multiplicity
Remote Requester	remi-r	0…1

6.2.2.3.3.2 Initial State

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

— none

6.2.2.3.4 Execution Constraints

The logical component REMI-R is thread-safe.

6.2.2.4 Roles

6.2.2.4.1 Remote Requester

Signature

```
role RemoteRequester {
  rcRemote_ProxyServiceRecord_t  proxyServiceRecords
  URI  peers [];
}
```

Qualifiers

— root

Description

A Remote Requester enables the execution of interface methods of remote services. The method invocations are served by service instances which are present in remote M3W Systems. The Remote Requester contains the list of the reachable M3W Systems via the current underline network (`peers`).

Independent Attributes

Table 83

Name	Description
proxyServicesRecords	Association records among service/interface uuid's and proxy service uuid's.
peers	Network addresses of M3W Systems where a REMI-P service instance has been activated

Invariants

— None

© ISO/IEC 2007 – 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

Instantiation

The Remote Requester role is always instantiated as part of the REMI-R logical component.

Active Behaviour

None

6.2.2.5 Interfaces

6.2.2.5.1 rclMethodExecutionInvoker

Qualifiers

None.

Description

This interface of a Remote Requester role provides facilities for invocation of interface methods on remote service instances.

Interface ID

uuid(15196bb8-a888-42e9-a16a-19bad6778ed6)

Remarks

None

6.2.2.5.1.1 invokeMethodOnNewService

Signature

```
rcResult invokeMethodOnNewService (
    [in] pUUID serviceId,
    [in] URI activatorLocation,
    [in] pUUID interfaceId,
    [in] String methodName,
    [in] prcIParam inParams[],
    [inout] prcIParam inOutParams[],
    [out] prcIParam *outParams[],
    [out] prcIParam *retVal,
    [out] rcObjHandle_t *instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Invokes an interface method of the requested service on a remoter newly created instance and returns the handle in order to be able to identify that instance in successive calls `through invokeMethodOnInstance method`. It is able to target which M3W System has to activate the service instance.

© ISO/IEC 2007 – 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 84

Name	Description
serviceId	Reference to the UUID of the requested service
activatorLocation	(Optional) Location of the required M3W System which have to activate the remote instance of the requested service
interfaceId	Reference to the UUID of the requested interface
methodName	The requested method name
inParams	Reference to an array of input parameters to be passed to method invocation
inOutParams	Reference to an array of input/output parameters to be passed to (and modified by) method invocation
outParams	Reference to an array of output parameters to be returned by method invocation
retVal	Reference to the rcIParam interface which represent the return value of method invocation.
instanceHandle	Reference to the handle identifying the remote newly created instance, to be used in successive calls

Return Values

Only non-standard error values are listed

Table 85

Name	Description
RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE	This error is returned when an invocation asks for a service that is not registered in any M3W System in the distributed environment
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_INVOKER_WRONG_PARAMS	This error is returned when an invocation is performed with an input parameters set not matching the interface method signature
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of a remote method invocation an exception has been raised

© ISO/IEC 2007 – 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

Precondition

N remote interface references of the required service exist

Action

Invocation of a method on a newly created instance of the requested service (by passing the specified parameters).

Postcondition

N+1 remote interface references of the required service exist

inOutParams = <param values modified by the method execution>

outParams = <param values modified by the method execution>

retValue = <reference to the rcIParam interface of the return value of the method execution>

instanceHandle = <handle identifying the remote newly created instance>

6.2.2.5.1.2 invokeMethodOnInstance

Signature

```
rcResult invokeMethodOnInstance (
    [in] prcObjHandle_t instanceHandle,
    [in] pUUID interfaceId,
    [in] String methodName,
    [in] prcIParam inParams[],
    [inout] prcIParam inOutParams[],
    [out] prcIParam *outParams[],
    [out] prcIParam *retVal
);
```

Qualifiers

— synchronous

— thread-safe

Description

Invokes an interface method of the requested service on an existing instance (i.e. previously activated by an invocation of *invokeMethodOnNewService* or *createRemoteService*).

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

© ISO/IEC 2007 – 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 86

Name	Description
instanceHandle	Reference to the handle identifying the requested instance
interfaceId	Reference to the UUID of the requested service
methodName	The requested method name
inParams	Reference to an array of input parameters to be passed to method invocation
inOutParams	Reference to an array of input/output parameters to be passed to (and modified by) method invocation
outParams	Reference to an array of output parameters to be returned by method invocation
retValue	Reference to the rcIParam interface which represent the return value of method invocation.

Return Values

Only non-standard error values are listed

Table 87

Name	Description
RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE	This error is returned when an invocation asks for a service that is not registered in any M3W System in the distributed environment
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_INVOKER_WRONG_PARAM	This error is returned when an invocation is performed with an input parameters set not matching the interface method signature
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of a remote method invocation an exception has been raised

Precondition

True

© ISO/IEC 2007 – 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

Action

Invocation of a method on the instance identified by the handle `instanceHandle` (by passing the specified parameters).

Postcondition

inOutParams == <param values modified by the method execution>

outParams == <param values modified by the method execution>

retValue == <reference to the rcIParam interface of the return value of the method execution>

6.2.2.5.1.3 createRemoteService

Signature

```
rcResult createRemoteService (
    [in] pUUID serviceId,
    [in] URI activatorLocation,
    [out] prcObjHandle_t instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Create a remote instance of the requested service. It is able to target the location of M3W System responsible of instantiation. If the target is not specified one of the reachable M3W Systems, which are able to remote such a service, will instantiate the service.

Parameters

Table 88

Name	Description
serviceId	Reference to the UUID of the requested service
activatorLocation	(Optional) Location of the required M3W System which have to activate the remote instance of the requested service
instanceHandle	Reference to the handle identifying the remote newly created instance, to be used in successive calls

Return Values

Only non-standard error values are listed

Copyright International Organization for Standardization © ISO/IEC 2007 All rights reserved
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS Not for Resale

Table 89

Name	Description
RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE	This error is returned when an invocation asks for a service that is not registered in any M3W System in the distributed environment
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable

Precondition

N remote interface references of the required service exist

Action

Creation of an instance of the requested service.

Postcondition

N+1 remote interface references of the required service exist

instanceHandle = <handle identifying the remote newly created instance>

6.2.2.5.1.4 releaseInstance

Signature

```
rcResult releaseInstance (
    [in] prcObjHandle_t instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Notifies to the remote M3W System that an active instance is not used anymore by client code which is local to the M3W.

Parameters

Table 90

Name	Description
instanceHandle	Reference to the handle identifying the remote instance to be released

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 91

Name	Description
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of a remote method invocation an exception has been raised

Precondition

the remote instance identified by instanceHandle exists and its reference counter has value N

Action

Notification of a release request for the instance identified by the handle `instanceHandle`.

Postcondition

the reference counter of the remote instance has value N-1

6.2.2.5.2 rclProxyManager

Qualifiers

None.

Description

This interface of a Remote Requester role provides facilities for obtaining proxies of services and interfaces in order to let applications use transparently remote services/objects. It also allows to check the presence of proxy availability.

Interface ID

uuid(5e808bf4-327a-4ece-b27c-99140f947e8b)

Remarks

None

6.2.2.5.2.1 createServiceProxy

Signature

```
rcResult createServiceProxy (
    [in] pUUID serviceId,
    [out] prcIUnknown proxy_iref,
    [in] URI activatorLocation,
);
```

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

© ISO/IEC 2007 – 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

— thread-safe

Description

Creates a proxy of a remote service and returns a corresponding `iref` to that proxy implementing the generic interface `rcIUnknown`. It is able to target a specific service. It is able to target which M3W System has to activate the service instance.

Parameters

Table 92

Name	Description
serviceId	Reference to the UUID of the requested service
proxy_iref	Reference to the iref of the requested service proxy
activatorLocation	(Optional) Location of the required M3W System which have to activate the remote instance of the requested service

Return Values

Only non-standard error values are listed

Table 93

Name	Description
RC_ERR_INVOKER_SERVICE_NOT_AVAILABLE	This error is returned when an invocation asks for a service that is not registered in any M3W System in the distributed environment
RC_ERR_INVOKER_PROXY_NOT_REGISTERED	This error is returned when an association does not exist for a given service identifier

Precondition

True

Action

Creation of a proxy of the service identified by `serviceId` and returning of an `iref` to that proxy.

Postcondition

A proxy of the service identified by serviceId has been created

proxy_iref == <reference to the proxy `of the service serviceId`>

© ISO/IEC 2007 – 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

6.2.2.5.2.2 createInterfaceProxy

Signature

```
rcResult createInterfaceProxy (
    [in] pUUID interfaceId,
    [in] prcObjHandle_t instanceHandle,
    [out] prcIUnknown proxy_iref);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Creates a proxy of a remote instance (identified by `instanceHandle`) of an object implementing the interface identified by `interfaceId` and returns a corresponding `iref` to the proxy implementing the generic interface `rcIUnknown`.

Parameters

Table 94

Name	Description
interfaceId	Reference to the UUID of the requested interface
instanceHandle	Reference to the handle identifying the specific instance of the object implementing the requested interface
proxy_iref	Reference to the iref of the requested interface proxy

Return Values

Only non-standard error values are listed

Table 95

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_REGISTERED	This error is returned when the execution request asks for a service that is not registered in the local M3W System

Precondition

True

Action

Creation of a proxy of the interface identified by `interfaceId` and returning of an `iref` to that proxy.

© ISO/IEC 2007 – 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

Postcondition

A proxy of the interface identified by interfaceId has been created

proxy_iref == <reference to the proxy `of the interface interfaceId`>

6.2.2.5.2.3 retrieveProxiedInstance

Signature

```
rcResult retrieveProxiedInstance (
    [in] prcIUnknown proxy_iref,
    [out] Bool *found,
    [out] prcObjHandle_t instanceHandle,
);
```

Qualifiers

— synchronous

— thread-safe

Description

Retrieves an interface reference in the list of proxies. It returns whether the passed iref it is actually a proxy of a remote instance. If yes, it returns the related istance handle (i.e. the information regarding the remote instance).

Parameters

Table 96

Name	Description
proxy_iref	Any iref
Found	Reference to a Boolean value which returns if the iref is in the list of proxies (i.e. is the iref of a proxy)
instanceHandle	If the iref is found in the list of proxies, it contains a reference to the proxied instance handle

Return Values

Standard errors

Precondition

True

Action

Retrieving of the information related to a proxy which is identified by proxy_iref.

© ISO/IEC 2007 – 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

Postcondition

found == true if a proxy oin the list is identified by proxy_iref

found == false else

instanceHandle == <reference to rcObjHandle_t containing the information of the instance which is proxied by proxy_iref>

6.2.2.5.3 rcIProxyServiceManager

Qualifiers

Optional.

Description

This interface of a Remote Requester role provides facilities for managing proxy services (i.e. services which are proxies for services and interfaces), in order to eventually let configuration tools determine which services are to be used as proxies.

Interface ID

uuid(d2647af6-f860-4129-99f1-66da22406a3e)

Remarks

This interface is not required but can be implemented to make the REMI-R more flexible.

6.2.2.5.3.1 registerServiceProxyService

Signature

```
rcResult registerServiceProxyService (
    [in] pUUID serviceId,
    [in] pUUID proxyServiceId);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Registers a service proxy identified by `proxyServiceId` as proxy for the service identified by `serviceId`.

© ISO/IEC 2007 – 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 97

Name	Description
serviceId	Reference to the UUID of the service for which the new service will be a proxy
proxyServiceId	Reference to the UUID of the new service proxy

Return Values

Only non-standard error values are listed

Table 98

Name	Description
RC_ERR_INVOKER_PROXY_ALREADY_REGISTERED	This error is returned when an association already exist for a given service identifier

Precondition

$\forall$A: <serviceId, A > $\notin$ proxyServiceRecords

Action

Creation of an association between the service `serviceId` and the proxy `proxyServiceId`.

Postcondition

<serviceId, proxyServiceId > $\in$ proxyServiceRecords

6.2.2.5.3.2 unregisterServiceProxyService

Signature

```
rcResult unregisterServiceProxyService (
    [in] pUUID serviceId);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove a record from `proxyServiceRecords`.

--`,,```,,,,````-`-`,,`,,`,`,,`---
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2007 – All rights reserved
Not for Resale

Parameters

Table 99

Name	Description
serviceId	Reference to the UUID of the service proxy to be unregistered

Return Values

Only non-standard error values are listed

Table 100

Name	Description
RC_ERR_INVOKER_PROXY_NOT_REGISTERED	This error is returned when an association does not exist for a given service identifier

Precondition

`<serviceId, proxyServiceId>` ∈ proxyServiceRecords

Action

Deletion of the record associating the service `serviceId` and the proxy `serviceProxyId`.

Postcondition

<serviceId, serviceProxyId> ∉ proxyServiceRecords

6.2.2.5.3.3 registerInterfaceProxyService

Signature

```
rcResult registerInterfaceProxyService (
    [in] pUUID interfaceId,
    [in] pUUID proxyServiceId);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Registers an interface proxy identified by `proxyServiceId` as proxy for the interface identified by `interfaceId`.

© ISO/IEC 2007 – 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 101

Name	Description
interfaceId	Reference to the UUID of the interface for which the new service will be a proxy
proxyServiceId	Reference to the UUID of the new interface proxy

Return Values

Only non-standard error values are listed

Table 102

Name	Description
RC_ERR_INVOKER_PROXY_ALREADY_REGISTERED	This error is returned when an association already exist for a given service identifier

Precondition

$\forall$A: <interfaceId, A> $\notin$ proxyServiceRecords

Action

Creation of an association between the interface `interfaceId` and the proxy `proxyServiceId`.

Postcondition

<interfaceId, proxyServiceId> $\in$ proxyServiceRecords

6.2.2.5.3.4 unregisterInterfaceProxyService

Signature

```
rcResult unregisterInterfaceProxyService (
    [in] pUUID interfaceId);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove a record from `proxyServiceRecords`.

© ISO/IEC 2007 – 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 103

Name	Description
interfaceId	Reference to the UUID of the interface proxy to be unregistered

Return Values

Only non-standard error values are listed

Table 104

Name	Description
RC_ERR_INVOKER_PROXY_NOT_REGISTERED	This error is returned when an association does not exist for a given service identifier

Precondition

<interfaceId, proxyServiceId> $\in$ proxyServiceRecords

Action

Deletion of the record associating the interface `interfaceId` and the proxy `proxyServiceId`.

Postcondition

<interfaceId, proxyServiceId> $\notin$ proxyServiceRecords

6.2.2.5.4 rclPeerManager

Qualifiers

Optional.

Description

This interface of a Remote Requester role provides facilities for managing peers list, in order to eventually let configuration tools determine which peers are to be used as remote M3W Systems.

Interface ID

uuid(d886b020-210a-4150-b584-c8dbd1156ef8)

Remarks

This interface is not required but can be implemented to make the REMI-R more flexible.

© ISO/IEC 2007 – 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

6.2.2.5.4.1 registerPeer

Signature

```
rcResult registerPeer (
    [in] URI location);
);
```

Qualifiers

— synchronous

— thread-safe

Description

Registers an M3W System at location `URI` as peer providing remote services.

Parameters

Table 105

Name	Description
location	URI identifying the location of an M3W System to be used as peer

Return Values

Standard

Precondition

peers = A

Action

Add a record in the `peers` list.

Postcondition

peers = A ∪ <location>

6.2.2.5.4.2 unregisterServiceProxyService

Signature

```
rcResult unregisterPeer (
    [in] URI location);
);
```

Qualifiers

— synchronous

— thread-safe

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

© ISO/IEC 2007 – 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

Unregisters an M3W system as peer providing remote services.

Parameters

Table 106

Name	Description
Location	URI identifying the location of an M3W System to not be used as peer anymore

Return Values

Standard

Precondition

peers = A

Action

Remove a record from `peers`.

Postcondition

peers = A \ <location>

6.2.2.5.4.3 getPeerList

Signature

```
rcResult getPeerList (
    [out]  URI **peerList
);
```

Qualifiers

synchronous

thread-safe

Description

Get the list of peers, i.e. the list of other M3W Systems registered as peers providing remote services.

Parameters

Table 107

Name	Description
peerList	Reference to an array of URIs

© ISO/IEC 2007 – 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

peers = A

Action

None.

Postcondition

peerList = <array filled with all the records contained in A>

6.2.3 REMI-P

6.2.3.1 Concepts

The function of a REMI-P logical component is to enable the usage of realizations elements (physical services, see 10.1.7), from clients which are located in other M3W Systems. More in details, it can serve a remote request of interface method execution redirecting such a request to a local instance of the requested service. The request is transported over a TCP connection with the client M3w System.

6.2.3.2 Types & Constants

6.2.3.2.1 Public Types & Constants

6.2.3.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_PROVIDER_SERVICE_NOT_AVAILABLE = 0x00000001
const rcResult RC_ERR_PROVIDER_INSTANCE_NOT_AVALIABLE = 0x00000002
const rcResult RC_ERR_PROVIDER_WRONG_PARAMS = 0x00000004
const rcResult RC_ERR_PROVIDER_METHOD_RAISED_EXCEPTION = 0x00000008
const rcResult RC_ERR_PROVIDER_WRAPPER_ALREADY_REGISTERED =  0x00000010
const rcResult RC_ERR_PROVIDER_WRAPPER_NOT_REGISTERED =  0x00000020
const rcResult RC_ERR_PROVIDER_METADATA_NOT_REGISTERED =  0x00000040
```

Qualifiers

— Error-codes

Description

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

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

© ISO/IEC 2007 – 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

Constants

Table 108

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_AVAILABLE	This error is returned when the execution request asks for a service that is not registered in the local M3W System
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_PROVIDER_WRONG_PARAM	This error is returned when the interface method execution is performed with an input parameter set that is not matching the signature
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of release an exception has been raised
RC_ERR_PROVIDER_WRAPPER_ALREADY_REGISTERED	This error is returned when attempting to register a wrapper that is already registerd
RC_ERR_PROVIDER_WRAPPER_NOT_REGISTERED	This error is returned when the corresponding wrapper service has not been registered in the local M3W System
RC_ERR_PROVIDER_METADATA_NOT_REGISTERED	This error is returned when the corresponding metadata could not been retrieved in the local M3W System

6.2.3.2.1.2 rcHandler_WrapperAssoc_t

Signature

```
struct _rcHandler_WrapperAssoc_t {
           UUID entityId;
           UUID wrapperServiceId;
} rcHandler_WrapperAssoc_t, *prcHandler_WrapperAssoc_t;
```

Qualifiers

— struct-element

Description

Data structure used to store the information related to a reachable M3W System.

© ISO/IEC 2007 – 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

6.2.3.3 Logical Component

6.2.3.3.1 Interface-Role Model

Figure 7 — Interface-Role Model, depicts the interface-role model of the REMI-P (grey box). It shows the interfaces and the roles involved with this component.

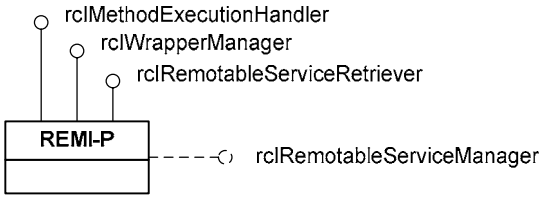


Figure 7 — Interface-Role Model

A REMI-P Logical Component contains the Remote Provider role that provides the rclMethodExecutionHandler interface. This interface can be used by a protocol server implementation to handle remote invocations of methods on instances of services. The Remote Provider role also provides the rclProxyWrapper interface which is used by clients to create wrapper services for the hosted instances. Another interface is needed in order to allow remote inspection request about the services which are available for remote execution. Another optional interfaces can be used to configure the REMI-P in order to select the suitable wrapper implementation for a given service/interface.

6.2.3.3.2 Diversity

6.2.3.3.2.1 Provided Interfaces

Table 109

Role	Interface	Presence
Remote Provider	rclMethodExecutionHandler	Mandatory
Remote Provider	rclWrapperManager	Mandatory
Remote Provider	rclRemotableServiceRetriever	Mandatory
Remote Provider	rclRemotableServiceManager	Optional

6.2.3.3.2.2 Configurable Items

The behaviour of the REMI-P depends on the list of the registered Wrapper Services (i.e. Phisical Services implementing wrapper behaviour). This list contains association records among service/interface identifiers and wrapper service identifiers. The list is specified using the following attribute.

Table 110

Role	Attribute
Remote Provider	wrapperServiceRecords

© ISO/IEC 2007 – 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

6.2.3.3.2.3 Constraints

None

6.2.3.3.3 Instantiation

6.2.3.3.3.1 Objects Created

The logical component REMI-P is created by the clients of the M3W, if they need to make available services in a distributed environment. The REMI-P is a Singleton of the M3W, after the first instantiation any other client will obtain a reference to the unique REMI-P instance.

Table 111

Type	Object	Multiplicity
Remote Provider	remi-p	0…1

6.2.3.3.3.2 Initial State

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

— none

6.2.3.3.4 Execution Constraints

The logical component REMI-P is thread-safe.

6.2.3.4 Roles

6.2.3.5 Interfaces

6.2.3.5.1 rclMethodExecutionInvoker

Qualifiers

None.

Description

This interface of a Remote Provider role provides facilities for handling interface methods execution request on service instance which are local to the M3W System.

Interface ID

uuid(6938f1bf-5d25-4021-a3f0-bd284a27b674)

Remarks

None

© ISO/IEC 2007 – 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

6.2.3.5.1.1 handleMethodOnNewService

Signature

```
rcResult handleMethodOnNewService (
    [in] pUUID serviceId,
    [in] pUUID interfaceId,
    [in] String methodName,
    [in] String serializedInParams,
    [inout] String *serializedInOutParams,
    [out] String *serializedOutParams,
    [out] String *serializedRetValue
    [out] prcObjHandle_t *instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Handles an interface method execution request which targets a new Service instance. It is able to generally perform the execution of the requested interface method by also asking the local system to create a new instance of a given service (by means of service identifier).

Parameters

Table 112

Name	Description
serviceId	Reference to the UUID of the requested service
interfaceId	Reference to the UUID of the requested interface
methodName	The requested method name
serializedInParams	Reference to a String which contains the serialization of input parameters to be passed to interface method invocation
serializedInOutParams	Reference to a String which contains the serialization of input/output parameters to be passed to (and modified by) interface method invocation
serializedOutParams	Reference to a String which contains the serialization of output parameters to be modified by interface method invocation
serializedRetValue	Reference to a String which contains the serialization of the return value of method invocation.
instanceHandle	Reference to an object handle, i.e. the unique logical reference to the newly created instance. It can be used to retrieve this instance and execute next interface method requests.

© ISO/IEC 2007 – 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 non-standard error values are listed

Table 113

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_REGISTERED	This error is returned when the execution request asks for a service that is not registered in the local M3W System
RC_ERR_PROVIDER_WRONG_PARAM	This error is returned when the interface method execution is performed with an input parameter set that is not matching the signature
RC_ERR_PROVIDER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of the interface method an exception has been raised

Precondition

— True

Action

Execution of a interface method on a new instance of the requested service (by passing the specified parameters). It perform a deserialization of the input and input/output parameters. The execution previously creates a Wrapper Service on the basis of the requested service identifier. Than the Wrapper Service instance is used to decode the “by name” request in a real call on the actual service instance.

Postcondition

serializedInOutParams == <serialization String containing param values modified by the method execution>

serializedOutParams == <serialization String containing param values modified by the method execution>

serializedRetValue == <serialization String containing the return value of the method execution>

instanceHandle == <reference to the rcObjHandle <instanceID, M3WSystem network address>>

6.2.3.5.1.2 handleMethodOnInstance

Signature

```
rcResult handleMethodOnInstance (
    [in] prcObjHandle_t instanceHandle,
    [in] pUUID interfaceId,
    [in] String methodName,
    [in] String serializedInParams,
    [inout] String *serializedInOutParams,
    [out] String *serializedOutParams,
    [out] String *serializedRetValue
);
```

© ISO/IEC 2007 – 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
— thread-safe

Description

Handles an interface method execution request which targets an existing instance. It is able to generally perform the execution of the requested interface method on the target instance (by means of instance handle).

Parameters

Table 114

Name	Description
instanceHandle	Reference to an object handle, i.e. the unique logical reference to the existing instance.
interfaceId	Reference to the UUID of the requested interface
methodName	The requested method name
serializedInParams	Reference to a String which contains the serialization of input parameters to be passed to interface method invocation
serializedInOutParams	Reference to a String which contains the serialization of input/output parameters to be passed to (and modified by) interface method invocation
serializedOutParams	Reference to a String which contains the serialization of output parameters to be modified by interface method invocation
serializedRetValue	Reference to a String which contains the serialization of the return value of method invocation.

Return Values

Only non-standard error values are listed

Table 115

Name	Description
RC_ERR_PROVIDER_INSTANCE_NOT_AVAILABLE	This error is returned when an invocation asks for an instance that is not active in the local M3W System
RC_ERR_PROVIDER_WRONG_PARAM	This error is returned when the interface method execution is performed with an input parameter set that is not matching the signature
RC_ERR_PROVIDER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of the interface method an exception has been raised

© ISO/IEC 2007 – 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

Precondition

— instanceHandle is referring to an existing instance.

Action

Execution of a interface method on an existing instance (by passing the specified parameters). It perform a deserialization of the input and input/output parameters. The instance handle is used to obtain the wrapper service instance which is connected to the underline target instance. The wrapper is used to decode the “by name” request in a real call on the actual instance.

Postcondition

serializedInOutParams == <serialization String containing param values modified by the method execution>

serializedOutParams == <serialization String containing param values modified by the method execution>

serializedRetValue == <serialization String containing the return value of the method execution>

6.2.3.5.1.3 handleCreateService

Signature

```
rcResult handleCreateService (
    [in] pUUID serviceId,
    [out] prcObjHandle_t instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Handles a creation request for a new Service instance. It is able to ask the local system to create a new instance of a given service (by means of service identifier).

Parameters

Table 116

Name	Description
serviceId	Reference to the UUID of the requested service
instanceHandle	Reference to an object handle, i.e. the unique logical reference to the newly created instance. It can be used to retrieve this instance and execute next interface method requests.

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 117

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_REGISTERED	This error is returned when the execution request asks for a service that is not registered in the local M3W System

Precondition

— True

Action

Creation of a new instance of the requested service. Creation of a Wrapper Service on the basis of the requested service identifier. Than the Wrapper Service and the service instance are bound.

Postcondition

instanceHandle == <reference to the rcObjHandle <instanceID, M3WSystem network address>>

6.2.3.5.1.4 handleInstanceRelease

Signature

```
rcResult handleInstanceRelease (
    [in] rcObjHandle_t *instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Handles a release request for an active instance identified by `instanceHandle`.

Parameters

Table 118

Name	Description
instanceHandle	Reference to the handle identifying the instance to be released

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 119

Name	Description
RC_ERR_INVOKER_INSTANCE_NOT_AVALIABLE	This error is returned when an invocation asks for a service instance in a specific M3W System which is not reachable
RC_ERR_INVOKER_METHOD_RAISED_EXCEPTION	This error is returned when during the execution of release an exception has been raised

Precondition

the remote instance identified by instanceHandle exists and its reference counter has value N

Action

Handling a release request for the instance identified by the handle `instanceHandle`.

Postcondition

the reference counter of the remote instance has value N-1

6.2.3.5.2 rclWrapperManager

Qualifiers

None.

Description

This interface of a Remote Provider role provides facilities for creating and managing wrapper service instances which are needed in order to allow existing local instances to generically handle interface method execution requests.

Interface ID

uuid(d1c53894-6acd-45ec-bec1-ce05e374ce99)

Remarks

None

6.2.3.5.2.1 createServiceWrapper

Signature

```
rcResult createServiceWrapper (
    [in] pUUID serviceId,
    [out] prcIReflection *wrapper,
    [out] prcObjHandle_t *instanceHandle
);
```

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

© ISO/IEC 2007 – 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

— thread-safe

Description

Create a service wrapper instance which is specific for a service. It returns a interface reference which allow to perform interface method execution request “by name”. (by means of interface identifier, method name and a general parameter list).

Parameters

Table 120

Name	Description
serviceId	Reference to the UUID of the related service
wrapper	Interface reference to a rcIReflection implementation which is already wrapping a service instance.
instanceHandle	Reference to the instance handle which uniquely identifies the newly created instance in the distributed environment.

Return Values

Only non-standard error values are listed

Table 121

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_REGISTERED	This error is returned wrapper creation asks for a service that is not registered in the local M3W System
RC_ERR_PROVIDER_WRAPPER_NOT_REGISTERED	This error is returned when the corresponding wrapper service has not been registered in the local M3W System

Precondition

— serviceID has been registered the local M3W System

— An wrapper service association record is present as <serviceID, wrapperServiceID>

— wrapperServiceID has been registered to the local M3W System

© ISO/IEC 2007 – 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

Action

Creation of a service instance and of a Wrapper Service instance. Binding the service instance to the wrapper bind point.

Postcondition

wrapper == < reference to rcIReflection intf implemented by wrapper service instance>

instanceHandle == < reference to rcObjHandle_t which uniquely identifies the created instance >

6.2.3.5.2.2 createInterfaceWrapper

Signature

```
rcResult createInterfaceWrapper (
    [in] prcIUnknown interfaceRef,
    [in] pUUID interfaceId,
    [out] prcIReflection *wrapper,
    [out] prcObjHandle *instanceHandle
);
```

Qualifiers

— synchronous

— thread-safe

Description

Create a interface wrapper instance which is specific for a given interface. It returns a interface reference which allow to perform interface method execution request “by name”. (by means of interface identifier, method name and a general parameter list).

Parameters

Table 122

Name	Description
interfaceRef	Reference to the wrapped interface reference
interfaceId	Reference to the UUID of the related interface
wrapper	Interface reference to a rcIReflection implementation which is wrapping the interface
instanceHandle	Reference to the instance handle which uniquely identifies the newly created instance in the distributed environment.

Return Values

Only non-standard error values are listed

© ISO/IEC 2007 – 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

Table 123

Name	Description
RC_ERR_PROVIDER_WRAPPER_NOT_REGISTERED	This error is returned when the corresponding wrapper service has not been registered in the local M3W System

Precondition

— A wrapper service association record is present as <interfaceID, wrapperServiceID>

— wrapperServiceID has been registered to the local M3W System

Action

Creation of a Wrapper Service instance. Binding the passed interface reference to the wrapper bind point.

Postcondition

wrapper == < reference to rcIReflection intf implemented by wrapper service instance>

instanceHandle == < reference to rcObjHandle_t which uniquely identifies the created instance >

6.2.3.5.2.3 retrieveWrappedInterface

Signature

```
rcResult retrieveWrappedInterface (
    [in] prcObjHandle instanceHandle,
    [out] Bool *found,
    [out] prcIRCUnknown *interface
);
```

Qualifiers

— synchronous

— thread-safe

Description

Create a interface wrapper instance which is specific for a given interface. It returns a interface reference which allow to perform interface method execution request “by name”. (by means of interface identifier, method name and a general parameter list).

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

© ISO/IEC 2007 – 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 124

Name	Description
instanceHandle	Reference to the instance identifier
found	Boolean value which state is the required association has been found
interfaceRef	If association has been found, it returns the interface reference to a rclUnknown which has been wrapped and associate to the instance identifier

Return Values

Only non-standard error values are listed

Precondition

— true

Action

Retrieving of an association among instance handles and wrapper service instance. Gathering of the found wrapper bind point.

Postcondition

found == < association instanceHandle, wrapper_iref is present>

interfaceRef == if found, < reference to rclUnknown related to instanceHandle>

6.2.3.5.3 rclRemotableServiceRetriever

Qualifiers

None.

Description

This interface of a Remote Provider role provides facilities for inspecting and retrieving metadata from the service list which is at disposal for remote execution. Basically it is able to answer if the local M3W system can create a service instance of a given type and it can provide information about the hosted service (i.e. interface method signatures).

Interface ID

uuid(c1e580cd-0dbb-444e-9718-d756d7a1f6b6)

Remarks

None

© ISO/IEC 2007 – 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

6.2.3.5.3.1 isServiceAvailable

Signature

```
rcResult createServiceWrapper (
    [in] pUUID serviceId,
    [out] Bool *available,
);
```

Qualifiers

— synchronous

— thread-safe

Description

It return a Boolean value which state that such a service is present in the local M3W System and is registered as available for remote execution.

Parameters

Table 125

Name	Description
serviceId	Reference to the UUID of the requested service
Available	Boolean value which indicate if the requested service is at disposal for remote execution

Return Values

Standard

Precondition

— true

Action

Checking the presence of the service identifier as a key of the wrapper service record list.

Postcondition

available == < serviceID can be created and managed for remote execution >

6.2.3.5.3.2 getServiceInfo

Signature

```
rcResult getServiceInfo (
    [in] pUUID serviceId,
    [out] String *info,
);
```


-`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

— thread-safe

Description

Retrieve service information for a given service identifier. The returned metadata include information which is needed for type-checking and for parameter marshalling.

Parameters

Table 126

Name	Description
serviceId	Reference to the UUID of the requested interface
Info	String contaning service metadata

Return Values

Only non-standard error values are listed

Table 127

Name	Description
RC_ERR_PROVIDER_METADATA_NOT_REGISTERED	This error is returned when the corresponding metadata could not been retrieved in the local M3W System

Precondition

— A wrapper service association record is present as <serviceID, wrapperServiceID>

— Metadata related to serviceID has been registered to the local M3W System

Action

Retrieving of the service metadata document.

Postcondition

info == < String containinig serviceID metadata>

6.2.3.5.4 rclRemotableServiceManager

Qualifiers

None.

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

© ISO/IEC 2007 – 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

This interface of a Remote Provider role provides facilities for inspecting and retrieving metadata from the service list which is at disposal for remote execution. Basically it is able to answer if the local M3W system can create a service instance of a given type and it can provide information about the hosted service (i.e. interface method signatures).

Interface ID

uuid(e1d6ce36-785c-466e-b571-52627a5c7919)

Remarks

None

6.2.3.5.4.1 registerServiceWrapperService

Signature

```
rcResult registerServiceWrapperService (
    [in] pUUID serviceId,
    [in] pUUID wrapperServiceId
);
```

Qualifiers

— synchronous

— thread-safe

Description

Add an association between a given service identifier and the identifier of the related Wrapper Service.

Parameters

Table 128

Name	Description
serviceId	Reference to the UUID of the service
wrapperServiceId	Reference to the UUID of the wrapper service

Return Values

Only non-standard error values are listed

Table 129

Name	Description
RC_ERR_PROVIDER_WRAPPER_ALREADY_REGISTERED	This error is returned when an association already exist for a given service identifier

© ISO/IEC 2007 – 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

Precondition

∀W: <interfaceId, W > ∉ wrapperServiceRecords

Action

Adding a record containing the association between service and wrapper service identifiers.

Postcondition

<serviceId, wrapperServiceId> ∈ wrapperServiceRecords

6.2.3.5.4.2 unregisterServiceWrapperService

Signature

```
rcResult unregisterServiceWrapperService (
    [in] pUUID serviceId,
);
```

Qualifiers

— synchronous

— thread-safe

Description

Remove an association between a given service identifier and the identifier of the related Wrapper Service.

Parameters

Table 130

Name	Description
serviceId	Reference to the UUID of the service

Return Values

Only non-standard error values are listed

Table 131

Name	Description
RC_ERR_PROVIDER_SERVICE_NOT_PRESENT	This error is returned when an association does not exist for a given service identifier

Precondition

— <serviceId, wrapperServiceId> ∈ wrapperServiceRecords

© ISO/IEC 2007 – 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

Action

Deleting a record containing the association between service and wrapper service identifiers.

Postcondition

<serviceId, wrapperServiceId> ∉ wrapperServiceRecords

6.2.3.5.4.3 registerInterfaceWrapperService

Signature

```
rcResult registerServiceWrapperService (
    [in] pUUID interfaceId,
    [in] pUUID wrapperServiceId
);
```

Qualifiers

— synchronous

— thread-safe

Description

Add an association between a given interface identifier and the identifier of the related Wrapper Service.

Parameters

Table 132

Name	Description
interfaceId	Reference to the UUID of the interface
wrapperServiceId	Reference to the UUID of the wrapper service

Return Values

Only non-standard error values are listed

Table 133

Name	Description
RC_ERR_PROVIDER_WRAPPER_ALREADY_REGISTERED	This error is returned when an association already exist for a given service identifier

Precondition

∀W: <interfaceId, W> ∉ wrapperServiceRecords

© ISO/IEC 2007 – 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

Action

Adding a record containing the association between interface and wrapper service identifiers.

Postcondition

`<interfaceId, wrapperServiceId> ∈` wrapperServiceRecords

6.2.3.5.4.4 unregisterInterfaceWrapperService

Signature

```
rcResult unregisterInterfaceWrapperService (
    [in] pUUID interfaceId,
);
```

Qualifiers

- synchronous
- thread-safe

Description

Remove an association between a given service identifier and the identifier of the related Wrapper Service.

Parameters

Table 134

Name	Description
interfaceId	Reference to the UUID of the interface

Return Values

Only non-standard error values are listed

Table 135

Name	Description
RC_ERR_PROVIDER_WRAPPER_NOT_REGISTERED	This error is returned when the corresponding wrapper service has not been registered in the local M3W System

Precondition

- `<interfaceId, wrapperServiceId> ∈` wrapperServiceRecords

© ISO/IEC 2007 – 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

Action

Deleting a record containing the association between interface and wrapper service identifiers.

Postcondition

<interfaceId, wrapperServiceId> ∉ wrapperServiceRecords

7 Overview of realization

7.1 General

This clause is informative. Given the wide range of potential devices targeted by the M3W, there are a wide number of domain specific properties that need to be supported. Not all of these properties need to be supported on all M3W systems. This creates a clear variation point within the overall architecture. For this purpose M3W divides the overall architecture into a core (development and execution) framework and additional frameworks, which support other properties, such as resource management, component download fault management and system integrity management. The additional frameworks are described in other parts of this standard:

— Resource Management (Part 4)

— Download (Part 5)

— Fault Management (Part 6)

— System Integrity Management (Part 7)

In the remainder of this document we describe the realization of the core development and execution framework. The remainder of this clause gives an overview of the guiding principles, assumptions, architecture overview and the stakeholders of an M3W system. The development framework is described in Clause 8. The core execution framework is described in Clause 10. An extension to the core framework that enables automated instantiation and binding of realization elements (services) that realize a specification artefact (logical component) is described in Clause 11. An extension to the core framework that enables remote invocation of operations provided by remote services is described in Clause 12.

7.2 Guiding principles

7.2.1 Simplicity

A simple solution that covers a large part of the desired functionality is preferred over a considerable more complex solution that covers a slightly larger part of the desired functionality. This does not mean that inadequate solutions are accepted.

7.2.2 Minimizing assumptions

As few assumptions as possible are made (OS features, Hardware, contents of components, etc.), thus reducing the dependency of M3W frameworks on the system context. This is done in order to make the applicable scope as wide as possible.

7.2.3 Make assumptions explicit

Assumptions that are necessary are made explicit.

© ISO/IEC 2007 – 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

7.2.4 Favour open over closed solutions

Whenever a reasonable option exists to select an open (denoting both non-proprietary and extendable) solution over a closed solution the former is chosen. This is in line with the aim of M3W, but also reflects the belief that architectures need to be able to evolve.

7.2.5 No penalties for un-used features

Various features are optional.

EXAMPLE not all systems require Fault Management. The architecture for the feature is constructed in such a way that there is no (performance) penalty for not using the feature.

7.2.6 Implementation preference: run-time, servers, and then clients

In a M3W system, there is one infrastructure, several servers and more clients. This implies that implementing a feature in the infrastructure enables the feature to be used system-wide without additional effort from component and application developers.

As multiple clients can use a component, there are potentially more clients than components in a system. Implementing a feature in a client will thus need more (development) resources compared to a server side implementation.

This principle is balanced with the simplicity principle; the infrastructure etc. must be made as simple as possible.

7.3 Assumptions

7.3.1 Platform functionality

The Device's Platform will at least provide the following services:

7.3.1.1 Scheduling

The basic schedulable entity of the OS is called "thread" in M3W. M3W makes no assumption on the OS scheduling policy (e.g. pre-emptive priority based or earliest deadline first).

If CPU resource management is to be supported, the OS must provide the necessary "hooks" to allow the resource manger to function.

7.3.1.2 Process

If spatial robustness of components is to be supported, the OS must provide a "process" concept supported by a hardware facility such as a memory management unit (MMU). Within the M3W context a process is a container for threads with full memory protection against access from other processes. In the absence of a process concept, the Device has one single (virtual) process.

7.3.1.3 Activation

The OS provides a means to load (if necessary) a component from persistent storage, bind to it, and activate it when necessary. When the OS has the notion of a process (see previous paragraph), a component can be loaded and activated in two ways: it can be loaded and activated in the process of the client requesting the component, or the component can be loaded in its own process.

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

© ISO/IEC 2007 – 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

7.3.1.4 Persistent storage

If dynamic downloading of components is to be supported, the device will provide a means to store downloaded components and their registration data persistently and to address this storage. Examples of persistent storage include battery back-up RAM, Flash memory, and a file system on disk.

7.3.2 Platform standards

For a given Platform, there will be a standard representation for such concepts as (dynamic link) libraries and executable file layouts, thereby allowing interoperability of development tools.

For a given Platform there will be a standard mapping of high-level languages to the processor architecture, thereby allowing binary compatibility between independently compiled (and linked) components. This mapping permits the standard layout of M3W artefacts on the Platform.

7.3.3 Hosts connected to suitable media

It is assumed that host systems are connected a suitable media when needed

7.4 Architecture overview

In the development view, a component is a collection of models and their relations. A model is a self contained set of information about the component. Models are managed as a single entity and are the 'unit of deployment'. In the M3W architecture, the set of model types is open. Models can take any form, they may be human and/or computer readable and may be executable. One model of particular importance is the 'executable model', which is typically called the 'executable component'. This is a model of the component that can be loaded and executed on a target platform.

Note A component may contain multiple executable components (models), e.g. for different target platform architectures or for different purposes such as debug and production. Other possible model types include resource models, simulation models, licensing models, documentation, and source code.

An executing M3W system consists of a platform with a set of software that provides features to a user. An application is the primary interface with the user that provides the end features. Executable components and Services provide an extra layer of functionality onto of the platform. A component is a unit of trading. Services are a unit of execution and provide functionality to applications. To ensure the integration all M3W elements (components, models and model elements) need to conform to the M3W component model. The component model states requirements on the M3W elements. In order to connect the applications to components and components to the system a run-time environment is required. The connections between all these conceptual entities are shown in Figure 8.

This conceptual structure identifies the key elements of the M3W architecture:

— Component Model – this provides the set of rules that must be adhered to by every M3W element

— Component – The unit of trading. A container for models and their relations.

— Executable Components – these act as containers for Services. In certain cases these are similar to DLLs, but their content may include system information for inclusion in simulators or other data.

— Services – these are normally instances of executing entities.

— Run-time Environment – this is the infrastructure artefact used to support executable components and executable component content management. In an executing M3W system this is responsible for allowing application to access Service, as well as, managing Executable Component instantiation, initialization and clean-up. Inside a Simulation environment this might be a description of the Run-time or its properties.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

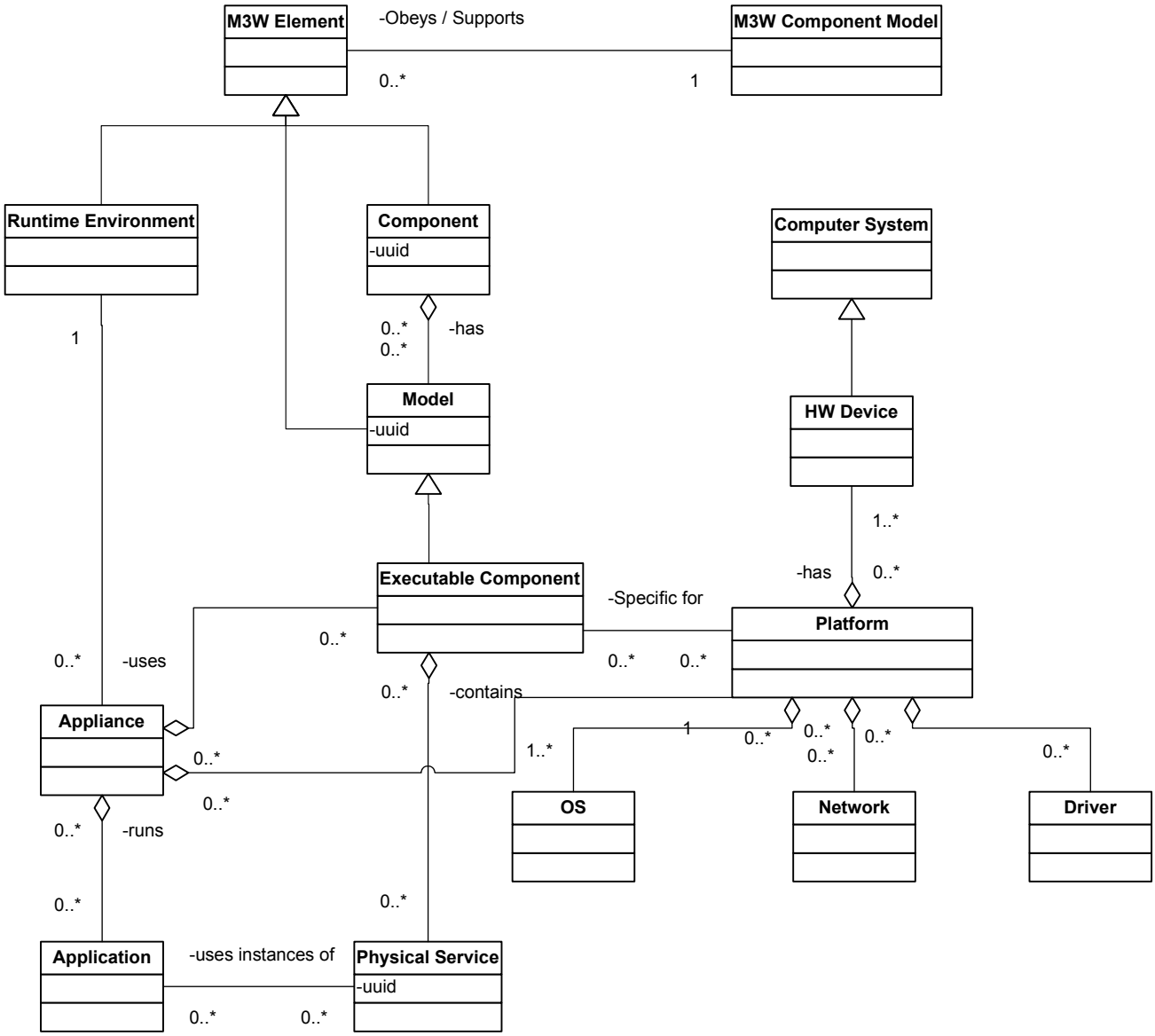


Figure 8 — M3W Conceptual Diagram

While not explicitly mentioned in the conceptual diagram, all interaction between the elements in M3W systems is defined by interfaces. The interface concept applies to all the elements of a M3W system. Interfaces define the set of operations provided by an element, creating the global concepts of servers (the providers of interfaces) and clients (the users of interfaces).

7.5 M3W stakeholders

The M3W architecture must address the concerns of many different stakeholders within the embedded software development industry. Therefore, prior to generating and discussing the architecture these need to be defined and the impact of the architecture on the concerns identified.

In order to support the embedded software industry M3W must support the movement of components from 3rd party developers to device manufacturers. For this to be effective, component developers must have the ability to sell their components to different manufacturers. Application developers must have methods for specifying the underlying platform used by their software. To support these users, an execution environment for component operation, and a model of the platform for application support must be included into the M3W system. These create two additional users of the M3W system; an infrastructure developer who creates the

© ISO/IEC 2007 – 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

component framework according to the M3W specification; and, a platform developer who integrates M3W components with a M3W run-time to support application development.

Additional to the development of these software structures, support for the selling of these elements is within the scope of the M3W project. As this process is different from the development of M3W software, the requirements and the viewpoints of the users change. o emphasize this fact, additional stakeholders need to be added to the list of M3W users: the component vendor, the component environment vendor and the system vendor. In order to reassure component, system and application developers, some guarantee of operational characteristics is needed. This can be provided by certifiers that check different properties of the developed artifacts. Fortunately, not all these stakeholders interact with each other, there is a clear path of software construction/distribution starting from the device manufacturer and ending with the application developer. This is illustrated in Figure 9.

This analysis produces 14 basic stakeholders related to the M3W system, which can be broadly categorized into 4 groups, external users, vendors, certifiers and developers. Other stakeholders that relate to the marketing of components exist that are not considered primary in the development of the M3W system; their interaction with the primary stakeholders is included below, where they are shown in italics.

The division of the stakeholders into the four broad categories helps define the types of activities that the M3W system must support. The key activities are the vending/certification and development of platforms, run-times and components. Of these activities, the one that places technical requirements on the M3W architecture and component model is the developmental process. The support for the vending activity by the framework consists of providing information that can be combined with tools for the selection and integration of software into a system.

© ISO/IEC 2007 – 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

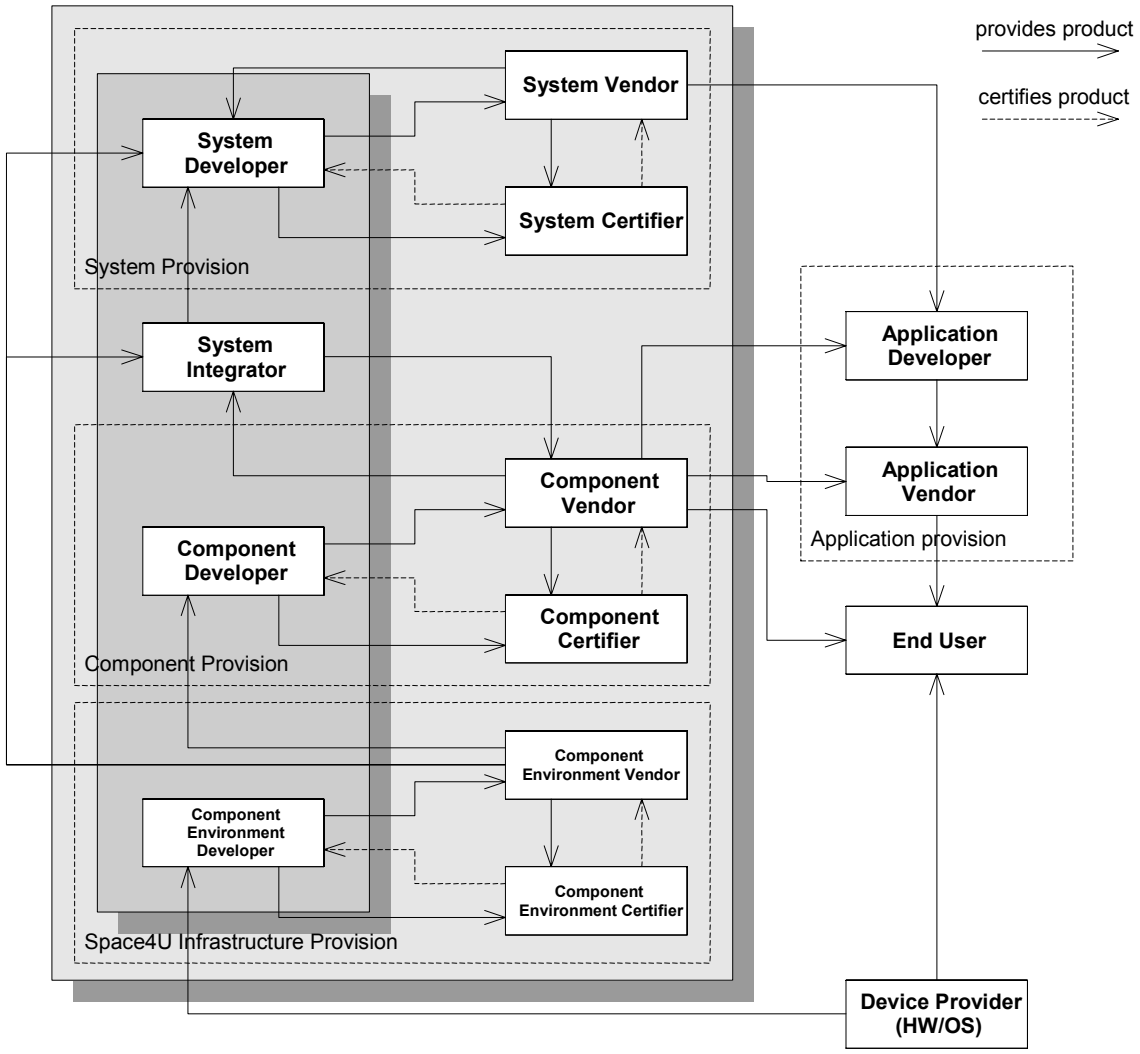


Figure 9 —Stakeholder-Relation diagram of the M3W business model.

7.5.1 External user stakeholders

The concerns of these stakeholders are not directly addressed within the core infrastructure architecture, these are addressed in the supporting frameworks for fault management, resource awareness, terminal management and system development.

— **End-user** – This user is not directly related to the M3W system, however these place demands on application vendors. The application vendor produces requirements for the application developer, who demands features from the M3W platform. Their interaction with the platform vendor arises from their desired application's requirements that must be satisfied. They also place requirements on device providers in the form of desired features. The end-user is the ultimate source of requirements, and their desires can be used to predict the type of components needed inside a platform. The end-user purchases information from service and content providers.

— **Device Provider** – Although not a part of the M3W system the device provider is the source of the underlying hardware and operating system. In the current stakeholder model, these act as OS and hardware suppliers. This affects the performance and capabilities available to the M3W framework developer. Furthermore, they respond to the needs of the end-user and require that their device

© ISO/IEC 2007 – 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

capabilities be utilized. The device provider's interaction with the end-user means that in certain cases a company will act as all of the elements inside the M3W arena of operation, as well as the application provider.

— **Application Vendor** – This stakeholder has little direct interaction with the M3W system, it serves as a conduit for requirements from the end-user to the application developer. However, as the applications may require additional run-time components it is quite probable that these will act as resellers for M3W components.

— **Application Developer** – This stakeholder is responsible for the technical requirements on the underlying system. If this supports the installation of components according to the needs of applications, these requirements can be placed on components inside the system itself. As the application developer might need to package specialized component with their application, they act as component customers. Since the applications must run on a M3W system, they also act as system vendor customers. To provide end-users with dynamic applications these developers work with service and content providers, who provide information for end-users. An application vendor can also act as a component specifier by defining a set of requirements that a component developer should fulfill.

7.5.2 Vendor stakeholders

These stakeholders have a set of concerns related to the distribution, marketing and selling of components. Consequently, their concerns are not addressed explicitly here, but their requirements have been taken into consideration when designing the overall framework.

— **System Vendor** – This stakeholder needs to advertise and distribute platforms to application developers. The platforms distributed may depend on a given device, therefore the device provider and platform vendor may be the same company. The system vendor’s primary interest is to provide systems to application developers. As one feature of the system is its certification, it is possible that the system vendor obtains this by using the services of a system certifier.

— **Component Vendor** – The component vendor needs to provide application and system developers with operations to select and get a component. The selection criteria may reflect the target domain of the platform. In certain circumstances a component may have to be adapted to a given system, this relies on the services of a system integrator, who tailors the component. Once a component is ready for sale, it can be certified by a component certifier, who checks its logical and operation properties. Component developers act as the source of components to the vendor. In certain circumstances, when a component developer transfers their copyright privileges, a component vendor can act as a component owner. A component vendor may interact with other component vendors (sometimes called component brokers) who act as an intermediary in component trading.

— **Component Environment Vendor** – Similar to the platform vendor the component environment vendor needs to advertise and distribute a run-time for a given platform. This may require some tools for allowing system and component developers select the appropriate run-time for their software. Due to the close relationship between a run-time and the underlying OS and hardware, it is likely that the device provider will also sponsor the development of the environment. Similar to component vendors, this vendor can interact with certifiers, for checking run-time properties and functionality, and developers who provide the environment. The component environment itself may consist of both tools and a run-time element; these can be produced by different developers.

— **SDE Tool Vendor** – Several of the stakeholders may require tooling to support them in their objectives. With the aid of a component standard, the tools can be interoperable and complementary to each other. As the tool vendor can support all stakeholders, they are not depicted in Figure 4.

7.5.3 Certification stakeholders

Like the previous sets of stakeholders, certifiers are not a primary concern for the M3W framework. However, the use of models and clear specification enables the requirements of these stakeholders to be somewhat supported within the current M3W architecture.

© ISO/IEC 2007 – 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

— **System Certifier** – In order to build an application with a desired set of characteristics the properties of the underlying platform need to analyzed and verified. This is implemented by these stakeholders, who use a variety of tools to profile a developed application platform. This stakeholder can provide services to system developers and to system vendors, who wish to advertise their products capabilities. This stakeholder will probably interact with component certifiers and component verifiers in order to extract information necessary for system certification.

— **Component Certifier** – This stakeholder measures and profiles the operational characteristics of a component. This process must be supported by tools that can quantify the necessary aspects of its operation. The information supplied may depend on the underlying run-time and the needs of the component domain. The results of the certification are used by system developers to predict a component's performance inside a system, and by system certifiers to certify the system. In this case, a certifier acts as a component verifier, who checks that a component satisfies a specification. Another role for the certifier is to check that a component matches a set of standards, in order to grant a component certification. This involves interaction with standardization bodies, which provide an appropriate set of standards for the certification process.

— **Component Environment Certifier** – The use of components inside a system depend on the linking infrastructure, which during operation resolves to the supporting run-time environment. In order to effectively verify a system this crucial element also needs to be certified. This is achieved by these stakeholders, which analyze a run-time to extract its properties. These stakeholders provide services to component environment developers and to vendors who wish to ensure that a run-time environment meets certain criteria.

7.5.4 Developmental stakeholders

These stakeholders are the primary concern of this document. The goal is to support developers as much as possible within the scope of the M3W project.

— **System Developer** – The system developer is responsible for integrating available components into a system that can support application development. This can be supported by tools, the component run-time and a component model that facilitates easy integration. They get their components from component vendors who may provide components that need to be adapted to the system. System integrators provide adapted components and tailor components for a given system. System developers can act as component specifiers in order to sub-contract system development.

— **System Integrator** – In cases where a component cannot be directly installed in a system, the system integrator adapts the component to match the desired system. An example of this would be the compilation of a source code component for a specific OS/HW combination. These system specific components can be resold to component vendors, or passed to system developers. Due to the close relationship between a component and its system, it is likely that system developers will also do the integration It is also likely that system integrators will interact with component certifiers and component verifiers, when the system specific component needs to be checked against a standard or a specification.

— **Component Developers** – The component developer is responsible for creating a component that conforms to the M3W model. This constrains and guides the developer in creating components that can be used within a M3W system. The components developed rely on the run-time. In order to support the component vendor, this product must be packaged in a manner that allows component characteristics be presented and used by system developers. A component developer can use the services of a component certifier or component verifier to check that a component conforms to a standard or a specification. The primary role of this stakeholder is to provide components to component vendors. In order to ensure a market for their components it can be expected that component developers will interact with component specifiers who describe a component’s desired characteristics. Another key interaction for component developers will be with IP owners (patent holders) of the algorithms and techniques embedded in a component implementation. These developers also develop services and libraries for inclusion inside a component according to the M3W model.

— **Component Environment Developer** – The infrastructure developer's primary role is to produce the M3W run-time on a device. In order to support the run-time vendor, the characteristics of the run-time

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

must be packaged with it. As a component environment can contain development tools, these can be packaged with the run-time as part of the environment. One source of product differentiation between environments will be the tool support for modeling, testing and other activities. As the environment is built on an underlying platform, this stakeholder will have considerable interaction with device providers and OS suppliers.

The M3W architecture cannot address all the concerns of all the stakeholders, so the focus of this architecture is to handle all the possible developmental concerns of M3W system developers, while not precluding support for the other stakeholders. This requires addressing system-level, platform-level and component-level issues relating to the efficient design, development and distribution of M3W components, services and libraries. Therefore, the M3W architecture directly addresses some of the issues related to End-User, Vendor and Certifying stakeholders.

7.6 Technical context

This section describes the main context of the M3W elements. M3W elements are in the middleware of the software stack.

The M3W context is here divided along three axes:

— The platform axis: M3W elements run on a platform and may be specifically developed for that platform. The platform consists of an operating system, drivers, hardware devices, and optionally a network. A hardware device can be any computer system.

— The development axis: M3W elements are developed on hosts. Repositories are computer systems that are used for the publication and retrieval of M3W Components, Services and Libraries.

— The appliance axis: M3W elements exist to form a part of an appliance. Next to the M3W elements an appliance consists of applications, and a platform.

© ISO/IEC 2007 – 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

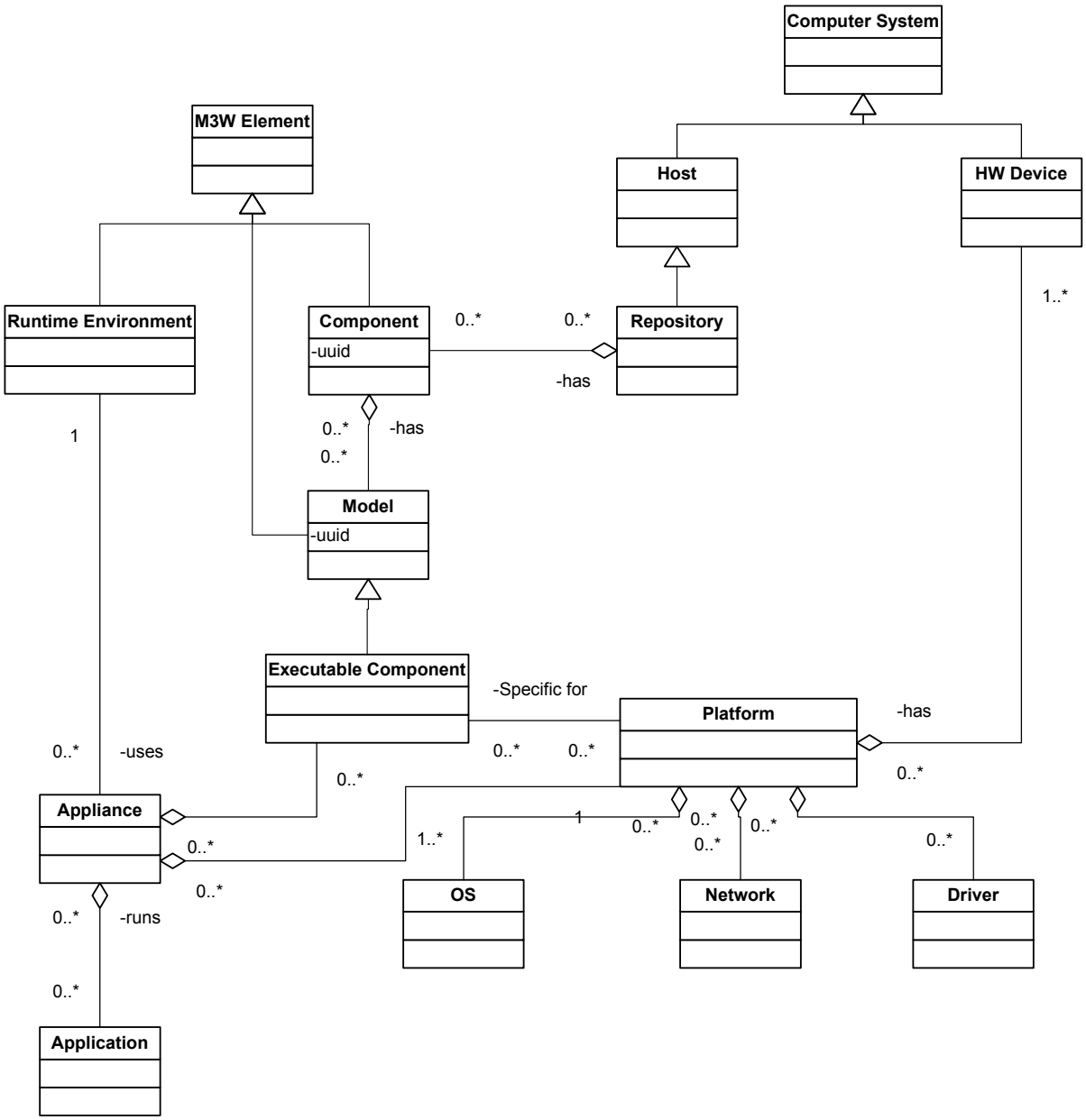


Figure 10 — M3W technical context

Table 136

Concept	Definition	Example
Computer system	Any system that enables the execution of software.	My own DVD Player
Host	A computer system that is used to develop software, in particular M3W elements	A PC running Linux
Repository	A computer system that is used to publish M3W elements to and retrieve them from	A website connected to a database that contains M3W components.
M3W element	An artifact that complies with the M3W standard. It is an abstract class; elements are always M3W components, models (model elements) or Runtime Environments	n.a.

© ISO/IEC 2007 – 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

Concept	Definition	Example
Hardware device	The computer system at which the M3W software is targeted.	A set-top box or mobile phone.
M3W runtime environment	Software to support the use of M3W components in an appliance	n.a.
M3W component	A component that complies with the M3W component model. It consists of a set of models that specify the component's properties.	A M3W MPEG4 decoder, a file-system component.
Appliance	Device that the end-consumer acquires.	A TV, set-top box, mobile phone.
Platform	Abstraction to allow the use of M3W components on devices that have different hardware and/or OS. It consists of a HW device, an operating system, several drivers, and optionally a network.	Linux running on a various different hardware systems; VxWorks in a set-top box.
Application	The implementation of one or more specific user oriented functions	The electronic programming guide.
Operating system	Support for Task/thread management and synchronization primitives, this may also include a process concept	WinCE, Linux
Driver	Software that provides an interface for a hardware element, such that the hardware element can be controlled by software.	Display driver in a PC
Network	Software that enables communication between computer systems.	TCP/IP stack

8 Development framework

8.1 Overview

This clause is informative. The development framework defines the roles and the relations between the various entities involved in the development, certification, trading, tailoring, and integration of Components and Runtime Environments.

8.2 Concepts

8.2.1 System

8.2.1.1 Description

From the development perspective the M3W architecture consists of several layers, as illustrated in Figure 11, these are:

— An application layer – this layer provides (access to) the functionality to the user of the device. It consists of software elements whose development relies on M3W supported middleware.

— A component layer – this is the layer of primary interest for M3W; it contains components that offer middleware services to applications. The development of this layer relies on the M3W layer.

— A M3W layer – this layer contains the core component infrastructure and supporting frameworks. Therefore, it is normally present before the development of M3W components. This layer relies on the implementation of the Platform layer.

© ISO/IEC 2007 – 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 Platform layer – this consists of the OS and HW device drivers that manage access to hardware resources from software.

Application provision is outside the scope of the M3W specification, so the application layer is enabled but not explicitly supported by the M3W architecture. The primary layer of interest for the development of M3W systems is the component layer. This layer places requirements on the M3W layer used to support the execution of M3W systems.

Figure 11 applies to many different types of M3W systems such as executing systems, simulators, analyzers and design environments. In each case the layers exist, albeit in an abstract form. For example, inside a M3W simulator the M3W layer, component layer and application layer may need to be taken into account to achieve an accurate simulation. In some models of a M3W system the role played by a layer maybe insignificant and discounted; this has to be an explicit choice made by the model designers. For example, in obtaining a estimation of the memory usage of a M3W system the M3W layer may be ignored (or abstracted away) with the analyzer focusing on the memory usage models contained in the component layer. However, in an executing M3W system all the layers will be present and used in system execution.

The idea of components as models is made clear in the section below. The important point made here is that a M3W system is the composition of applications with M3W Components supported by the M3W and OS layers.

8.2.1.2 Model

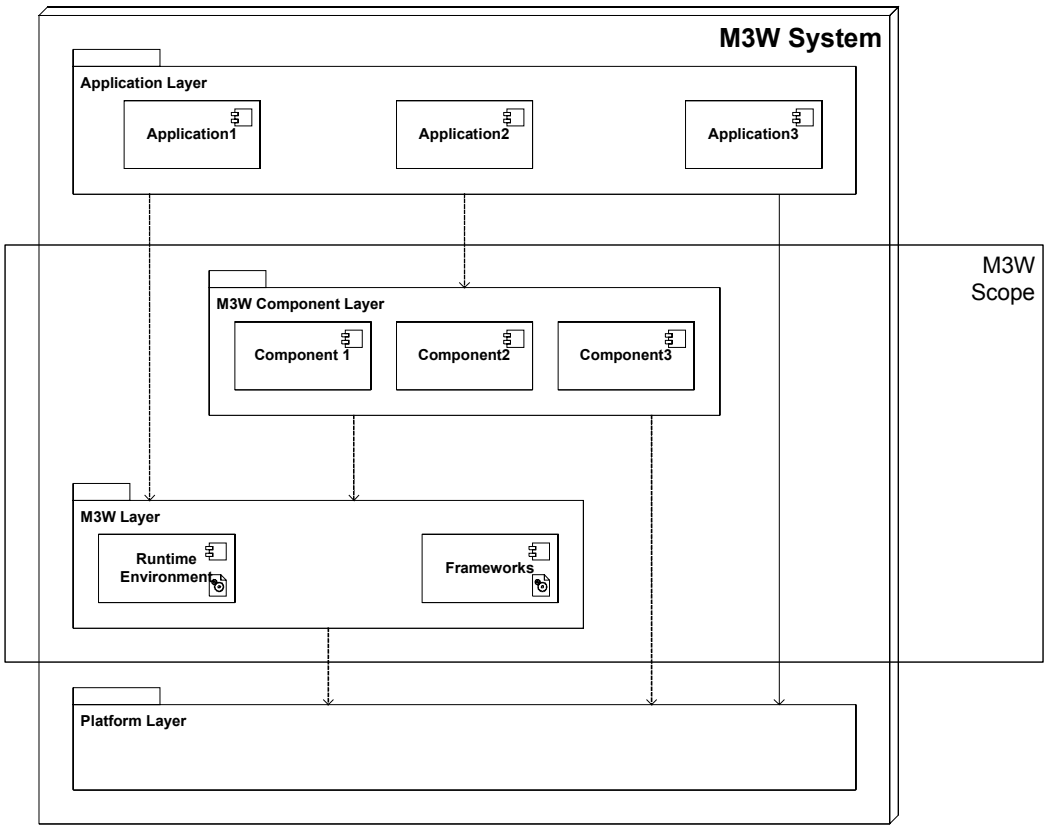


Figure 11 — System

© ISO/IEC 2007 – 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

8.2.2 M3W Model

8.2.2.1 Description

Models describe M3W components in a structured way. It is desirable that other pieces of software (like applications and platforms) are similarly structured. Therefore, we have generalized the idea of models and standards to the level of artifacts. It is allowed to read "M3W component" for the word "artifact".

Artifacts have properties. Models contain specifications that describe those properties. Models have relations.

A specification (or property or model) can be a specialization of another specification and can be built of other specifications. (Example a requirement can be built of sub-requirements and can be specialization of a general requirement. This also holds for properties, specifications, and models.)

When we set up a system for handling models, specifications, properties and artifacts, we need concepts that allow us to talk about the possible instances, their properties, and corresponding models and specifications. The type level concepts enable this.

An artifact may comply with an artifact standard. The artifact standard consists of rules about standardization elements. These are on type level, because the standard does not handle specific artifacts but is about artifact types.

Table 137

Concept	Definition	Example
Artifact	Anything made by mankind. (We may restrict it with anything that is described by a set of models).	A specific MPEG4 decoder
Artifact type	Used to type artifacts such that they can be classified and that rule can be specified to define standards for specific types.	M3W component, or a M3W MPEG4 decoder, a CPU resource model.
Artifact standard	A set of rules that hold (to a certain degree) for a set of artifacts.	The M3W standard for CPU resource models.
Property	A property of an artifact, independent of how it is specified.	n.a.
Property type	Stating that artifacts of a specific artifact type can have specific properties.	n.a.
Specification	Defined and structured statement about one or more properties of an artifact.	The recourse usage is between 4 and 5 Mbytes.
Specification type	Stating that one may specify certain properties in a certain way.	Resource usage specifications exist of a minimum and a maximum value expressed in Mbytes.
Model	A set of specifications that is consistent in itself and managed as one.	n.a.
Model type	A classification for models	A resource model
Rule	Stating that certain value must hold for one or more standardization elements.	n.a.
Standardization element	Element that can be used in the definition of a rule. The element is always an instance of one of the standardization element's subclasses.	n.a.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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 models that contain specification of properties of an artifact describe that artifact.

8.2.2.2 Model

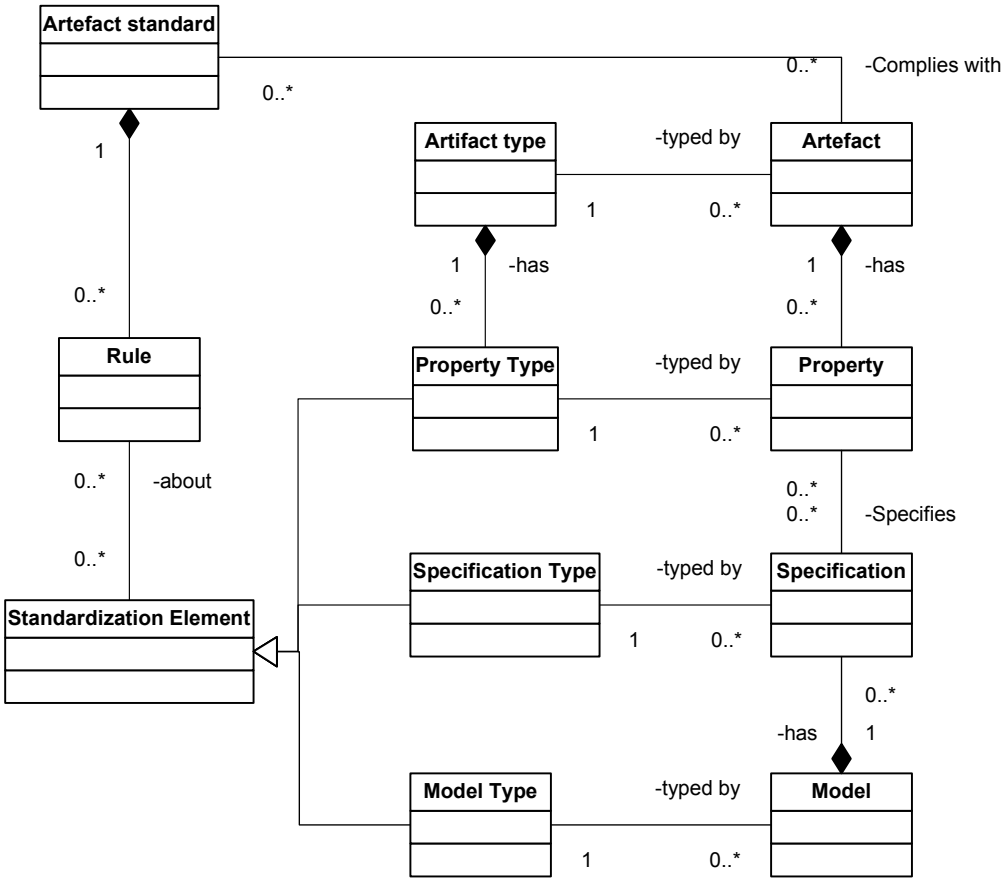


Figure 12 — Model

8.2.3 Model Relation

8.2.3.1 Description

The models that describe a M3W component are often related.

Table 138

Concept	Definition	Example
Model relation	A relation between two or more models	1. This resource model describes the resource usage in memory of that executable. 2. This model is an improvement/ refinement of that model.
Model relation type	A classification for model relations.	1. Memory usage of an executable. 2. Versioning

© ISO/IEC 2007 – 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

8.2.3.2 Model

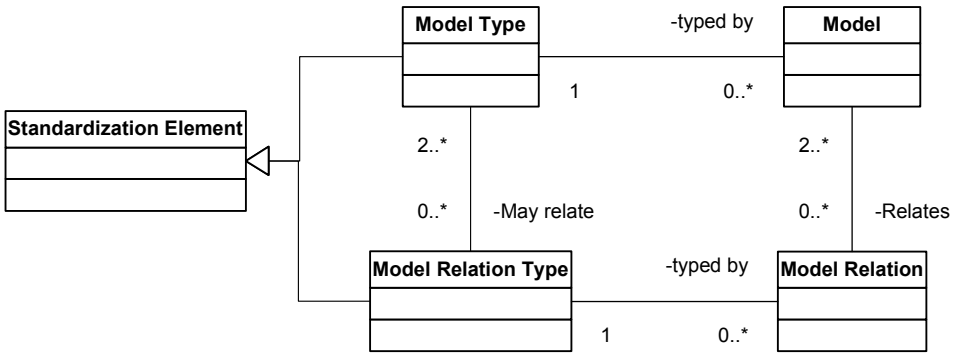


Figure 13 — Model Relation

8.2.4 Component

8.2.4.1 Description

A Component is a set of related Models. This abstracts the component concept away from a single representation and allows it to have many different facets that may have use beyond system execution. There are various usages for this concept, including trading, composition, and execution-time inspection of properties of Components. In this case a M3W system consists of an environment for combining models of M3W components with models of applications, M3W infrastructure elements and underlying platforms as depicted in Figure 11. It should be noted that in certain circumstances, some of these models may not impact the overall composition and can be excluded from the M3W system.

8.2.4.1.1 Trading support

Trading is the activity of buying and selling Components either directly or through a Component Vendor.

One of the usages of multiple Models is that evaluation versions of Components can be created, e.g. by just including the specification Models or by providing trading versions of (Executable) Models with certain limitations such as reduced performance, self-destruct after an evaluation period, etc.

Another usage is that the existence of certain Models and/or their content can be used as search criteria when identifying Components that could meet the needs of a System Integrator.

8.2.4.1.2 Composition support

One of the reasons of specifying a M3W Component as a set of interrelated Models is that it supports the reasoning on compositions of Components based on the composition of the Models. While this is an area still requiring significant research, we deem it an important and distinguishing feature of the M3W architecture.

EXAMPLE 1 To determine the total size of the executable code segment of all Components, one can easily add the sizes of the individual Component's executable code segments.

EXAMPLE 2 Given a behavioral simulation Model of a number of Components the behavior of the composition of these Components could be simulated.

Figure 14 shows an example of how tools could be used to create a Model for the composition of a number of Components based on the individual Models of the Components.

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

© ISO/IEC 2007 – 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

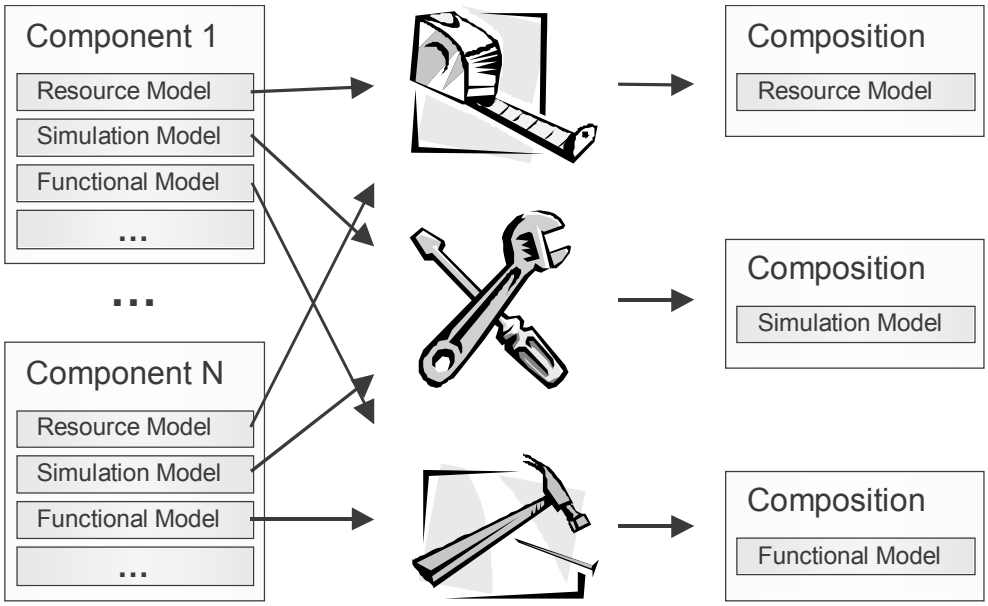


Figure 14 — Tool based composition of Models

8.2.4.1.3 Support execution-time inspection

At execution time, Applications, other Components, and the M3W Runtime Environment may inspect Models to determine certain attributes of a Component or Service. Based upon this inspection different actions may be taken.

EXAMPLE During the creation of a Service Instance, the RRE may inspect the Resource Model(s) of the Service to determine whether the Appliance still has sufficient resources (RAM, CPU cycles, etc.) to host the Service Instance.

NOTE These Models need to be accessible on the Device, either as target loaded Models or through a link to a Host.

© ISO/IEC 2007 – 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

8.2.4.2 Model

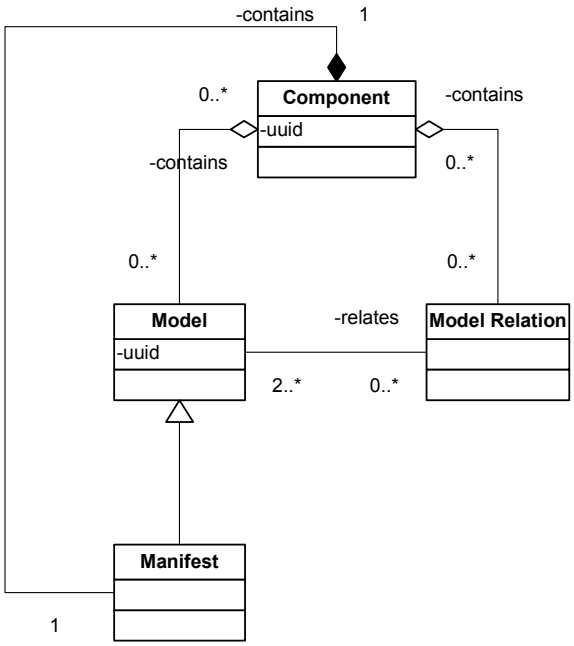


Figure 15 — Component

8.2.5 Package

8.2.5.1 Description

A Component consists of models and model relations. From a Component many representations can be made in the form of a package. A package is distributed in a distribution format. The distribution format is mostly determined by the standard that the Component complies with. The UUID is unique identifier of the component and the manifest describes the models and relations of the component.

Table 139

Concept	Definition	Example
Component standard	An artifact standards for artifacts of the type “M3W component”	The M3W component standard
Distribution format	A format in which package can be distributed. It may consist of syntax, a technology, and protocol.	ZIP format, TAR format.
Package	Representation of M3W component in a distribution format.	The models of that MPEG4 decoder in a ZIP-file.

© ISO/IEC 2007 – 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

8.2.5.2 Model

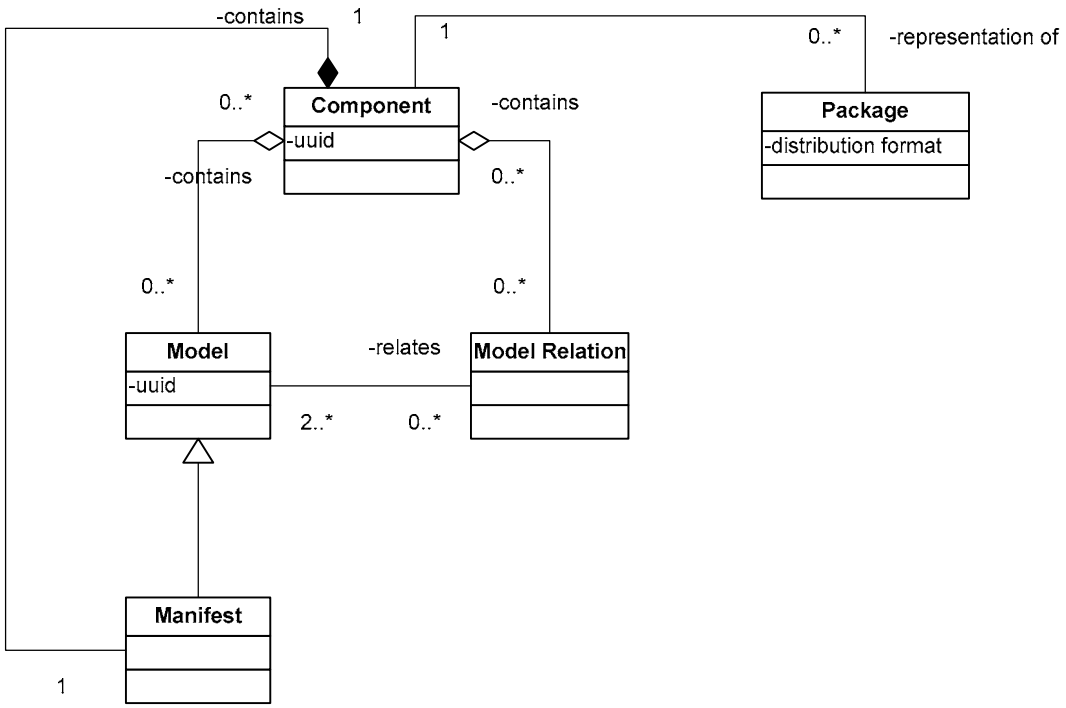


Figure 16 — Package

8.2.6 Repository

8.2.6.1 Description

Repositories store Components (so that they become Published) and support searching, browsing, uploading, and downloading. Searching entails mechanical identification based on given attributes such as key words, required Models, or supported Interfaces. Browsing allows a human to interactively inspect the (human readable) Models of the Component, its trading information and the like. Uploading is the copying of selected Models (some Models, e.g. the source code, may be excluded for upload) of the component to a Host for certification and/or later integration. Downloading is the copying of selected Models to a Device for dynamic integration.

Repositories are maintained by the Component Vendors role.

There may be a single or multiple Repositories. Published Components may be “generic” in the sense that they still need to be tailored to run on a specific Platform (i.e. Device hardware and OS). This may involve e.g. compilation and linking for that Platform. The Repository Caretaker, a trusted 3rd party, or the System Developer may carry out this process.

© ISO/IEC 2007 – 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

8.2.6.2 Model

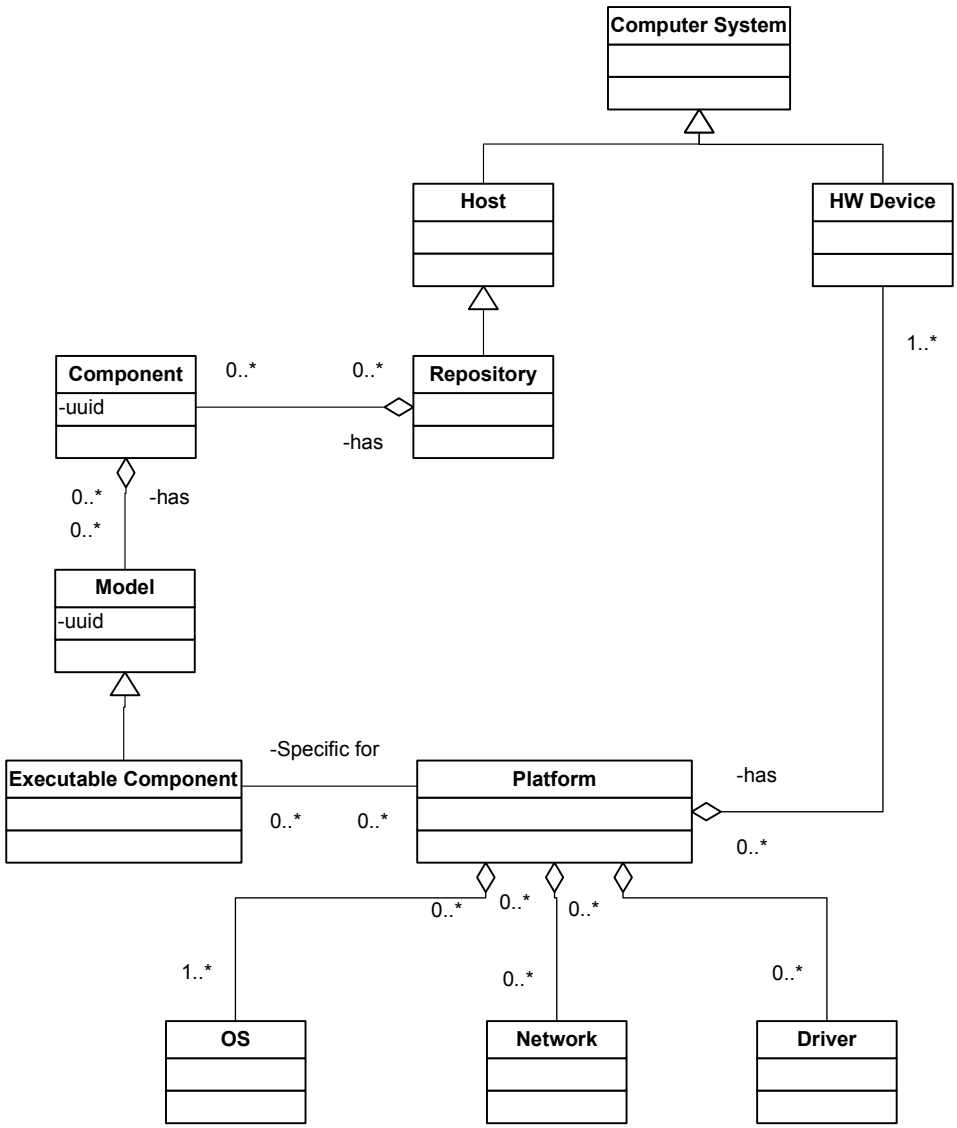


Figure 17 — Repository

© ISO/IEC 2007 – 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 Behaviour

Figure 18 illustrates a simplified development flow of a M3W Appliance with the M3W scope in bold lines.

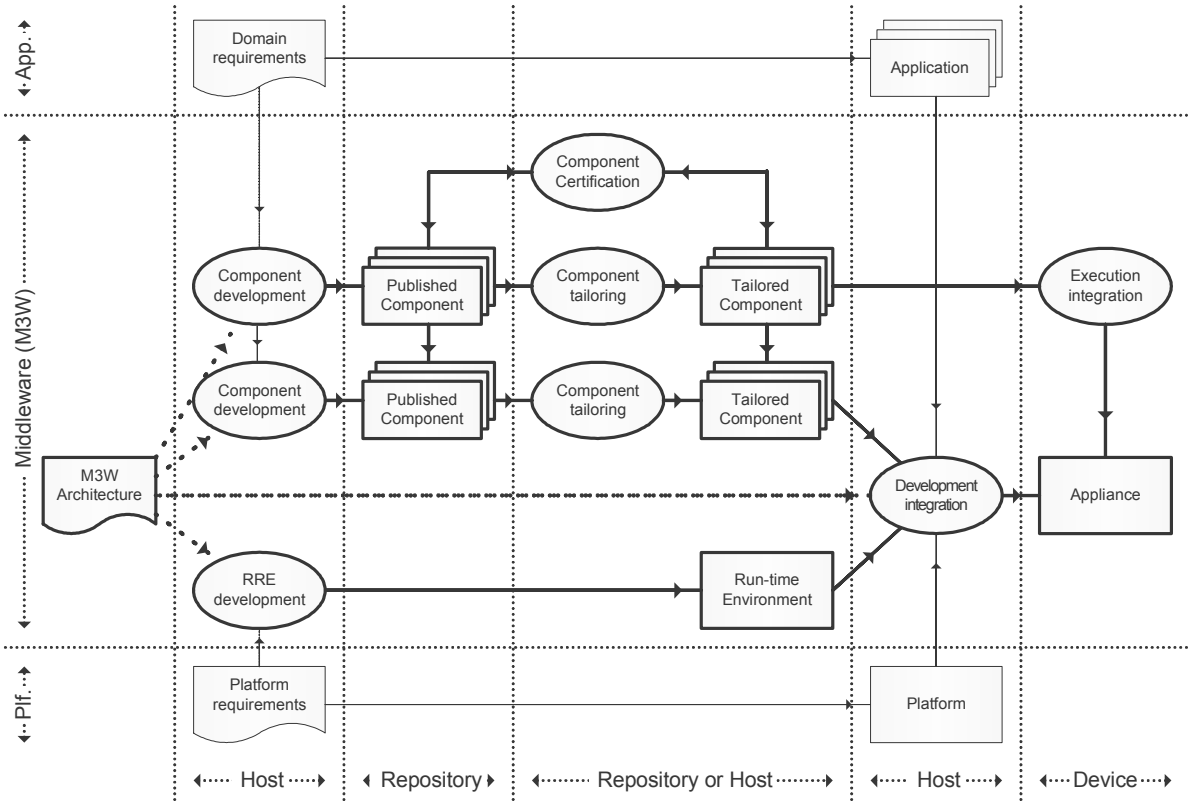


Figure 18 — Development Flow

A System Integrator develops the software for the Appliance. Typically, this includes the development of one or more Applications and some Components, the selection and potentially tailoring of externally available Components, the selection of a Runtime Environment and a Platform (type) as well as their integration. This process of integrating the Tailored Components with the Platform and adding Applications is called development integration. During this phase the System Integrator may make use of many different models contained in the system Components, such as resource models or timing models.

As part of the Appliance's manufacturing process, the total software stack is installed on the Device (instances).

During the lifetime of the system some Components may be replaced for improved functioning (upgrade) and/or new Components may be added for added capabilities (extension). This involves some execution time integration on the Device.

The following paragraphs discuss the various activities in more detail.

8.3.1 Component development

Based on the domain requirements for the Appliance(s) targeted Component Developers develop Components. Component development consists of developing the constituent Models and the associated Manifest.

It is envisioned that there will be many parties fulfilling the Component Developer roles. Those parties could be in-house System Integrators, OS and/or RRE vendors that desire to augment their base offering, or independent SW vendors (ISV).

© ISO/IEC 2007 – 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

How Components are developed (methodology, tools, languages, etc.) is left unspecified in M3W.

8.3.2 Component certification

Certification entails independent validation that a Component adheres to the M3W architecture and/or that it meets certain quality attributes, including complying with other standards. Certification is primarily a mechanism to increase the trust in a Component and as such supports the creation of an open market.

Components can be certified at any point during their lifecycle, e.g. before being published or after being tailored. The result of Certification action is a certification attribute of the Component or of –some of- its Models.

8.3.3 Component tailoring

Tailoring is the activity of specializing a Component to be deployed on a specific Platform. This may e.g. include compilation and linking for that Platform to create the Executable Component form that can be installed on the Device. It may also include adaptation and/or creation of other Models, e.g. a performance or resource Model that depends on the type of Platform.

Tailoring enriches a Component by adding Models in a consistent way. It does not fundamentally change the Component (i.e. the Services and Interfaces are not changed). A Component can have been tailored for multiple Platforms and can be published in a Repository and/or exist on the Host of the System Integrator.

Depending on the commercial conditions, tailoring could include removing of Models from the Component. Models containing proprietary information (such as the source code) or Models that are only meaningful for a different Platform (e.g. a MIPS instead of an Arm based Device) are good examples of Models that could be removed.

Tailoring creates a new component that “complies with” the original component.

Tailoring may be carried out by the original Component Developer, as a Repository Service, by a trusted 3rd party (i.e. a Component Vendor), or by the System Developer.

8.3.4 Development integration

Development integration covers all activities required to take existing Applications, Components and a Platform and make it all work together as a whole.

8.3.5 Execution integration

During the operational life of the Appliance, new Components may become available that can replace existing Components to improve operation (upgrade) or may be added to create new capabilities (extension). If the Appliance has the right provisions for this (such as a link to a Host), these Components can be loaded in the Device and registered with the M3W Runtime Environment so that they become available to the Appliance. This two-step process of integrating new Components with the existing middleware software is called execution integration.

In the first step, selected Models, amongst which at least the Executable Component, are loaded into the Device’s storage; this is called target loading. Examples of target loading include, but are not limited to, burning in [E]PROM or Flash memory, copying n a device resident disk drive, and downloading through the Download Framework.

In the second step, the Executable Component as well as its Services are registered with the Runtime Environment. Registration of the Executable Component establishes a link between the Component’s UUID and the storage location from which the OS can activate it. Registration of the Services establishes a link between the Service UUIDs and the Executable Component UUID. These links are made persistent by recording them in the Registry.

© ISO/IEC 2007 – 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

During both steps of the process, admission tests may be performed to ensure that the new Executable Component will fit and not break the Appliance.

EXAMPLE 1 Before starting a download the download manager ensures that the Device's storage system has sufficient free space to store the Executable Component.

EXAMPLE 2 Before registering the Executable Component in the Registry, the Component Support verifies that the Executable Component can run on the Device (e.g. that it has all resources required by the Executable Component such as e.g. the right version of the OS, sufficient memory etc.)

9 Iterator idiom

9.1 Overview

This is an informative clause. The iterator idiom is used in many different places throughout the M3W specification of the execution framework, so rather than describe similar functionality for each usage this subclause describes the idiom. This idiom allows the passing of a variable length list of data to a client in a well-encapsulated manner. A similar result could be achieved by the use of sequence types, but the iterator idiom has several advantages over the use of sequences to return the system data:

— The data returned may be very large: the iterator interface idiom does not mandate that the data be transferred to the client; it just defines how to support client access to the data.

— A NULL value can be returned in the case where no data is defined: using a sequence to return data means that a valid sequence of length 0 would have to be returned; this simplifies null implementations of optional functions in the System Interfaces.

9.1.1 The Iterator idiom

The iterator idiom can be described as a "macro" inside the RIDL language where there are two variation points for each instance of the iterator: its name (including its UUID) and its data type. The iterator interface is:

```
interface <iterator_name> <iterator_uuid> {
      void reset();
      <data_type> next() raises <prefix>_ERR_NoSuchElement;
      Bool atEnd();
};
```

The basic idea inside the iterator is that it has a cursor that traverses the underlying data list returning the value of the data that it points to.

There are 3 key functions inside every iterator interface:

— `reset()` is used to set the iterator cursor to the start of the data list.

— `next()` is used to read the data at the cursor position and to move the cursor, if the cursor is at the end of the data list it returns the last value and return the RcXNoSuchElement error code.

— `atEnd()` is used to check if the cursor is at the end of the data list.

9.1.2 Usage

The table below gives examples of where the iterator idiom is used inside M3W.

© ISO/IEC 2007 – 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

Table 140

Name/UUID	Data type	Where used
rcIUuidItr {d3bab695-7796-4755-b8ba-09dff6a1edeb}	UUID	rcIRegistryView rcILoadedComponentManagement
rcIEventItr {53980fdc-47c7-4de1-908b-4bfd2de6304a}	rcEvent	rcIEventChannel
rcIComponentItr {e18f5bee-ea1f-44a6-ac7d-26a2255a71a9}	rcComponentRecord	rcIRegistryInspect
rcIContainerItr {2292732b-5f9b-44e6-92ec-4dbbf65031ad}	rcContainerRecord	rcIRegistryInspect
rcICompliesItr {9da241cb-8f14-4ff0-9296-e379abbe0b68}	rcCompliesRecord	rcIRegistryInspect

10 Execution framework

10.1 Concepts

10.1.1 Type

10.1.1.1 Description

There is a set of basic types that is used in the execution framework.

Table 141

Type	Definition
UInt8	8 bit unsigned integer
UInt16	16 bit unsigned integer
UInt32	32 bit unsigned integer
Int8	8 bit signed integer
Int16	16 bit signed integer
Int32	32 bit signed integer
Double	64-bit double precision floating point
Float	32-bit single precision floating point

© ISO/IEC 2007 – 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

`Char`	8-bit character
`Bool`	A binary True / False. True = 1, False = 0 32-bit unsigned integer
`Void`	Nothing
`pVoid`	An opaque type with language mapping specific semantics
`rcIUnknown`	The rcIUnknown interface
`prcIUnknown`	Reference to rcIUnknown interface
`UUID`	128-bit unique identifier according to the COM specification [13].

Basic types can be used to construct user defined types. The execution framework supports the following categories of user defined types:

— Struct: A struct definition matches that of a C struct definition.

— Union: Unions allow the expression of structures whose content type changes according to some value or variable.

— Enum: Enums declare a defined set of constant integers starting from 0. The enumerators themselves are just labels.

— Sequence: This defines a structure that maybe of variable length. Unbounded sequences have a variable length; bounded sequences have a similar underlying storage structure but have a fixed index length.

— String: These define standard Strings, however they also define bounded Strings of a known length. The only different between wide Strings and Strings is that wide Strings contain wide characters.

— Interface: This defines an interface. An interface contains one or more operations and may inherit from another interface. Single inheritance is supported.

© ISO/IEC 2007 – 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

10.1.1.2 Model

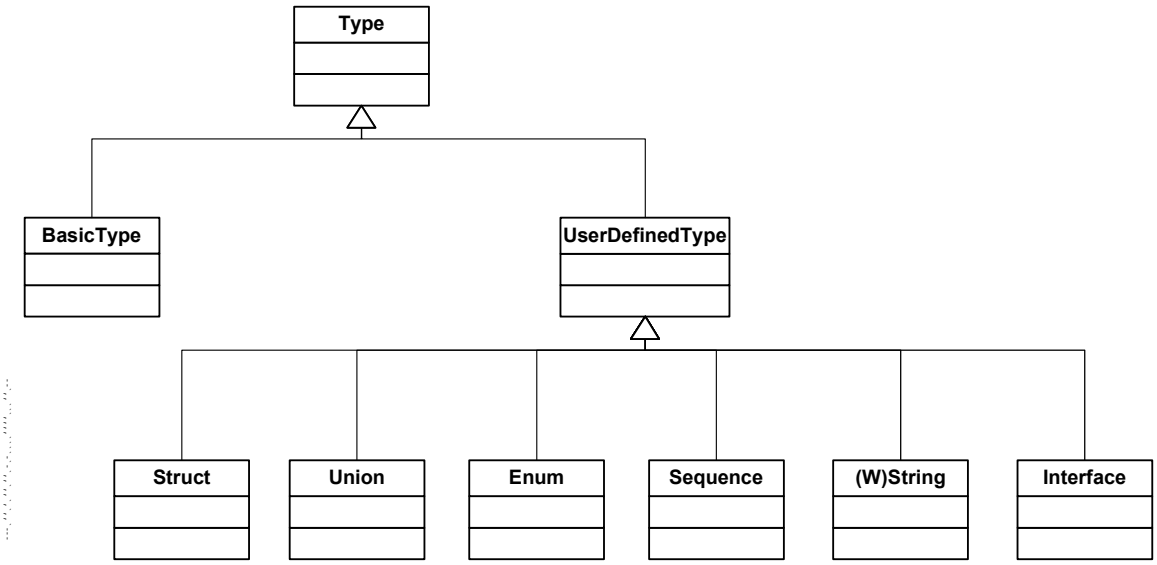


Figure 19 — Types

10.1.2 Interface

10.1.2.1 Description

Interfaces (interface definitions) have a number of operation. Operations can have a return value. Operations can have a number of parameters. Parameters have a type, name and a scope. Parameters can have the following scopes:

— Input: a.k.a. read only

— Output: a.k.a. write only

— Input and output: a.k.a. read and write

An interface is a type. This can be the basic type `rcIUnknown` or a user defined Interface. All interfaces are a specialization of `rcIUnknown` (see 10.1.6). Interfaces can also have attributes. Attributes are a short hand for the get and set operations associated with that attribute.

© ISO/IEC 2007 – 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

10.1.2.2 Model

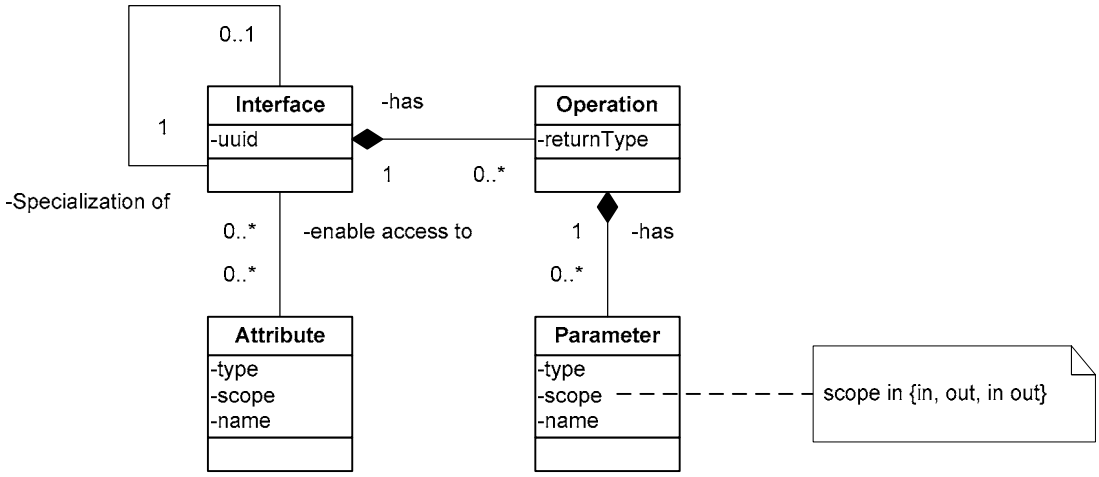


Figure 20 — Interface

10.1.3 Class and object

10.1.3.1 Description

Classes implement a number of operations. A number of operation implementations can be the implementation of an interface. The fact that a class implements an interface represents an “is a”-relation. Navigation between the implemented interfaces is done via the type (i.e. its uuid) and the `QueryInterface()` operation, which is present in all interfaces (as all interfaces are derived from `rcIUnknown`).

A class can be instantiated at runtime. The resulting runtime entity is called an object. An object has it’s own state.

10.1.3.2 Model

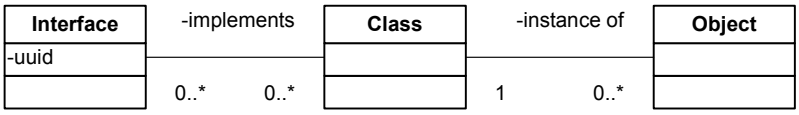


Figure 21 — Class and Object

10.1.4 Interface instance

10.1.4.1 Description

An interface (definition) can be implemented by a number of classes. This means that the class provides the implementation of the operations that are part of the interface. At runtime a class can be instantiated, the resulting entity is an object. This object has an interface instance for the interfaces (definition) implemented by the class. This interface instance gives access to the implementation of that interface. More precise it enables a client to invoke the operation implementations part of that interface implementation.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

10.1.4.2 Model

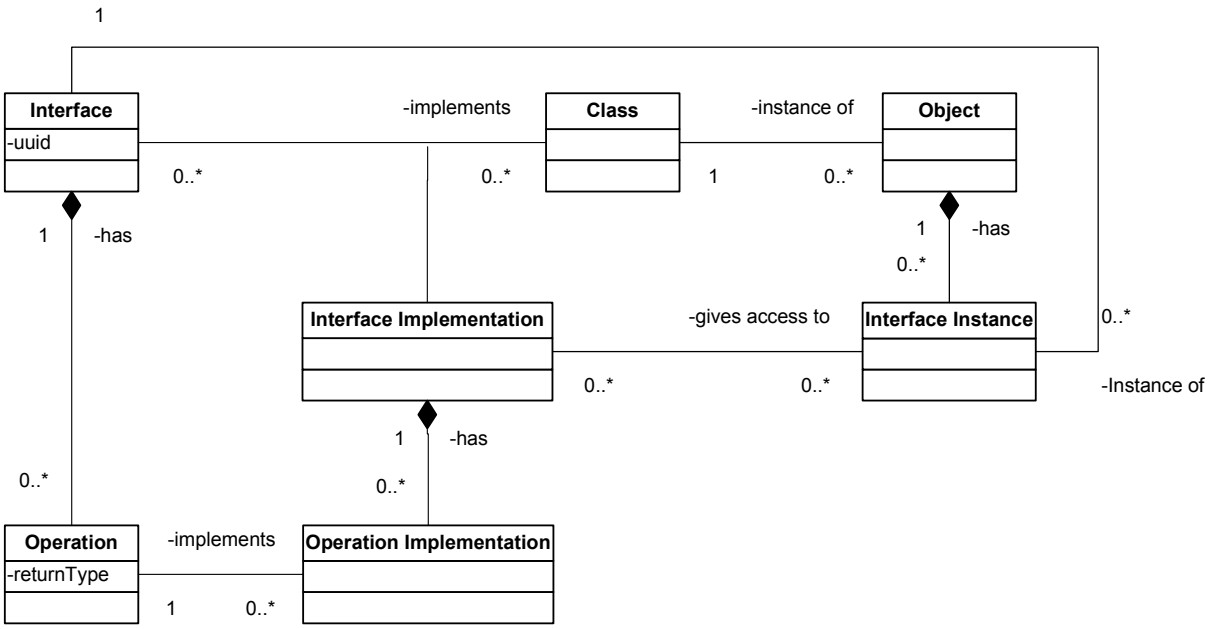


Figure 22 — Interface instance

10.1.5 Interface reference

10.1.5.1 Description

Interface Instances are the runtime (data) structures that give access to interface implementations (enable invocation of operation implementations). Objects offers these data structures. A client will use references to these data structures. An interface reference therefore is a pointer to an interface instance.

10.1.5.2 Model

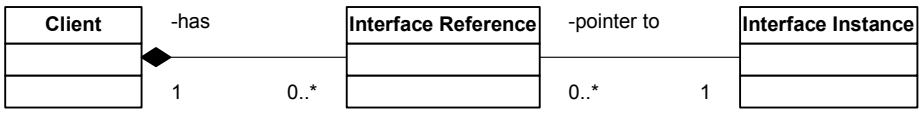


Figure 23 — Interface reference

10.1.6 Interface rclUnknown

10.1.6.1 Description

`rcIUnknown` is an interface that is implemented by every class. This interface enables navigation between implemented interfaces and management of the lifecycle of objects. This interface contains the `QueryInterface` operation to query the object for a specific interface. When the requested interface is implemented reference to this interface is returned, otherwise `NULL` is returned. The operations `AddRef()` and `Release()` are used for counting the number outstanding of references to the object.

© ISO/IEC 2007 – 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

```
interface rcIUnknown { 00000000-0000-0000-c000-000000000046 }
{
    pVoid QueryInterface(in UUID);
    UInt32 AddRef();
    UInt32 Release();
};
UUID_rcIUnknown = 00000000-0000-0000-c000-000000000046
```

There are three main rules that govern implementing the `rcIUnknown` operation on an object:

- Objects must have identity: For any given object instance, a call to `QueryInterface(UUID_rcIUnknown)` must always return the same physical pointer value. This allows you to call `QueryInterface(UUID_rcIUnknown)` on any two interfaces and compare the results to determine whether they point to the same instance of an object.

- The set of interfaces on an object instance must be static: The set of interfaces accessible on an object via `QueryInterface` must be static, not dynamic. Specifically, if `QueryInterface` returns an interface reference for a given `UUID` once, it must never return `NULL` on subsequent calls on the same object; and if `QueryInterface` returns `NULL` for a given `UUID`, subsequent calls for the same `UUID` on the same object must never return an interface reference.

- It must be possible to query successfully for any interface on an object from any other interface: This means `QueryInterface` must be reflexive, symmetric, and transitive with respect to the set of interfaces that are accessible.

There are rules on implementation of the operations for counting the number of outstanding references. Interface implementations must maintain a counter that is large enough to support (2^31)-1 outstanding pointer references to all the interfaces on a given object.

A client of an object must obey the following rules with respect to calling `AddRef` and `Release`.

Rule 1: `AddRef` must be called for every new copy of an interface reference, and Release called for every destruction of an interface reference, except where subsequent rules explicitly permit otherwise.

The following rules call out common nonexceptions to Rule 1.

- **Rule 1a:** In-out-parameters to operations. The caller must `AddRef` the actual parameter, since it will be Released by the callee when the out-value is stored on top of it.

- **Rule 1b:** Fetching a global variable. The local copy of the interface reference fetched from an existing copy of the reference in a global variable must be independently reference counted, because called operations might destroy the global copy while the local copy is still alive.

- **Rule 1c:** New references synthesized out of "thin air." A operation that synthesizes an interface reference using special internal knowledge, rather than obtaining it from some other source, must do an initial `AddRef` on the newly synthesized reference. Important examples of such routines include instance creation routines, implementations of `IUnknown::QueryInterface`, and so on.

- **Rule 1d:** Returning a copy of an internally stored reference. After the referencer has been returned, the callee has no idea how its lifetime relates to that of the internally stored copy of the reference. Thus, the callee must call `AddRef` on the reference copy before returning it.

Rule 2: Special knowledge on the part of a piece of code of the relationships of the beginnings and the endings of the lifetimes of two or more copies of an interface reference can allow AddRef/Release pairs to be omitted.

The return values of `AddRef` and `Release` should not be relied upon, and should be used only for debugging purposes.

© ISO/IEC 2007 – 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

If a client needs to know that resources have been freed, it must use a method in some interface on the object with higher-level semantics before calling `IUnknown::Release`.

10.1.6.2 Model

rclUnknown is an instance of an interface definition.

10.1.7 Service and service Instance

10.1.7.1 Description

Interfaces and classes give a fine-grained level of programming. A Service provides a higher level of programming than classes. Compared to classes, discussed in the previous sections, it has several extensions:

— Ports (see 10.1.8)

  — provides ports: an additional way to retrieve interface references

  — requires ports: expresses context dependencies

— attributes (see 10.1.8)

A Service is a specialization of a class. It is a class that implements the `rcIService` and a service specific interface (see 10.1.10). Just like classes Services can be instantiated. The resulting entity is called a Service Instance.

Each Service definition determines the so-called Service specific interface. This interface is the run-time view of the Service Instance. The interface is derived from `rcIService`. The `rcIService` interface has the following definition:

```
interface rcIService { d58f279c-c217-4730-b76b-48422b982835 } {
    Void Start( Void )
        raises RC_ERR_SERVICE_INSUFFICIENT_BOUND = 0x00000001;
    Bool isStarted( Void );
    Void Stop( Void )
        raises RC_ERR_SERVICE_CANNOT_STOP = 0x00000002;
}
```

As with any interface, it inherits from `rcIUnknown`.

The `Start()` operation needs to be called after the requires interfaces from the Service Instance are bound and the desired attributes are set. The `Start()` call indicates the context dependencies have been resolved, i.e. the binding to the ports is done, and the client wants to start using the Service Instance. Until the `Start()` method has returned, only calls on the Service interface of the Service Instance are allowed. After the `Start`, methods of other interfaces may be called. For so-called active services, the autonomous behavior starts when the `Start()` method is called.

The `isStarted()` operation can be used to determine whether a Service Instance has been started.

For instances of active Services, the `Stop()` operation can be called to stop the autonomous behavior. Depending on the implementation, or rather specification, of the Service the `Stop()` operation may need to be called when requires interfaces need to be rebound. Apart from those cases, there is no need to call the `Stop` explicitly. When all interfaces of the Service Instance, its provides ports, and any “inner” objects are released, the Service Instance can be removed.

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

© ISO/IEC 2007 – 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

10.1.7.2 Model

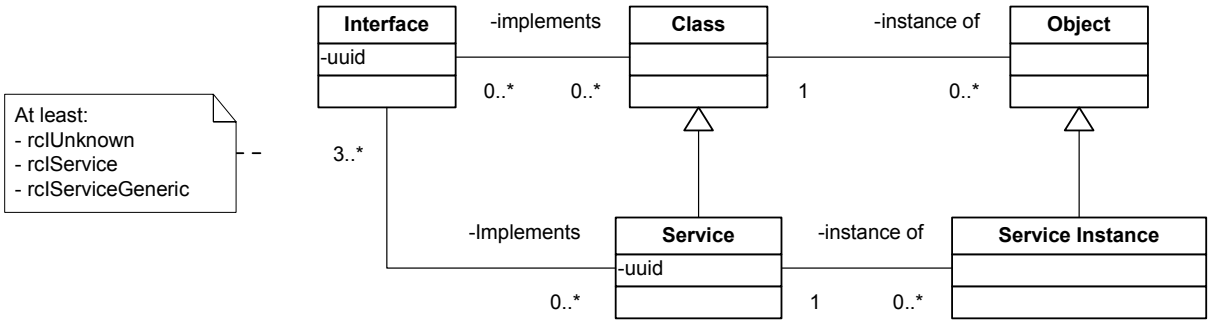


Figure 24 — Service (instance)

10.1.8 Service and attributes

10.1.8.1 Description

For a Service, it is possible to have attributes. These are named values that have a meaning for the Service. The attributes can be accessed via the `rcIServiceGeneric` interface.

10.1.8.2 Model

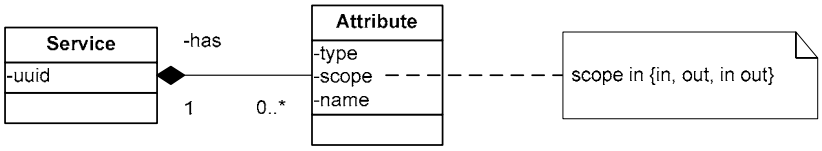


Figure 25 — Attributes

10.1.9 Ports

10.1.9.1 Description

Just like a class a Service can implement a number of Interfaces. Implementing an interface provides an “is-a” relation between the Service and the interface type. In addition there is also the possibility of providing a port. Ports can be used to obtain references of inner objects. These references are obtained using the `rcIServiceGeneric` (see 10.1.10). This interface contains a set of operations that provide access to the ports. A provides port models a “has-a” relation between the Service and the port interface type.

Services have explicit dependencies. These dependencies are represented by requires ports. Requires ports can be bound, by a third party, using the `rcIServiceGeneric` interface (see 10.1.10). Requires ports have *cardinality*: This cardinality can be 0..1 or 1. This is a ‘per requires port’ property: different requires ports of the same Service might have different cardinality.

It is up to the Service Instance if a bound interface, once bound, can be unbound again. It does imply that a Service should specify if unbinding is allowed or not; or if there are any special conditions that must be satisfied before unbinding is allowed. For example, the Service may allow unbinding only when in a particular state.

In short provides ports give access to inner objects, requires ports give access to outer objects. By binding different interfaces to the requires ports we change the outer objects that are used by the service instance.

© ISO/IEC 2007 – 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

10.1.9.2 Model

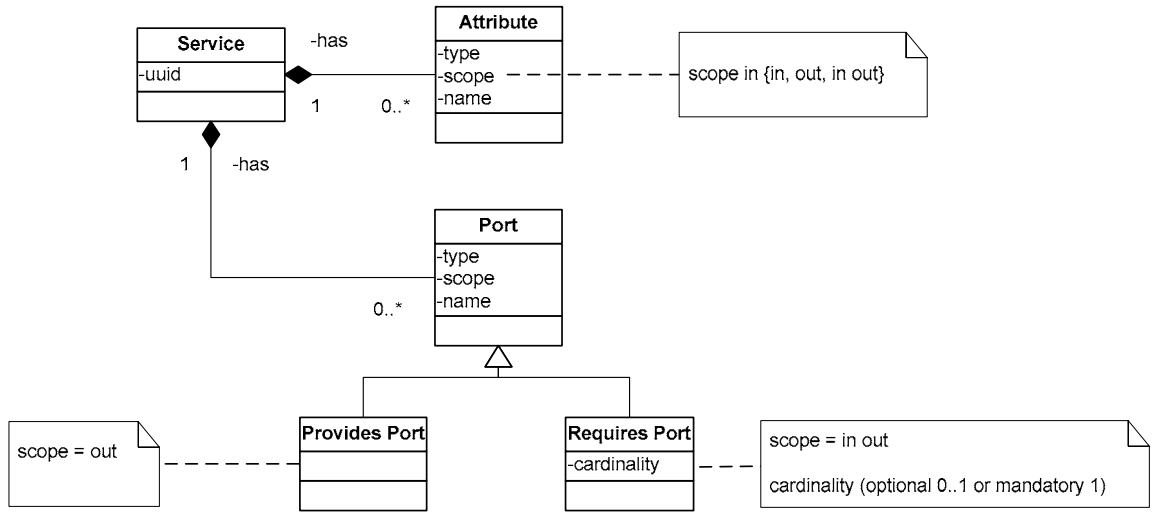


Figure 26 — Ports

10.1.10 rclServiceGeneric interface

10.1.10.1 Description

In addition to the standard `rcIService` operations, every service implements the `rcIServiceGeneric` interface. This interface enables unified access to the attributes and ports (provides and requires) of a service. This interface contains the `getProvides` operation that gives access to the provided ports. The interface contains the `setRequires` and `getRequires` operations that give access to the requires ports. The interface contains the `set` and `get` operations that give access to the attributes of the service.

```
enum RcEAttributeType
{
   UINT8,
   UINT16,
   UINT32,
   INT8,
   INT16,
   INT32,
   DOUBLE,
   FLOAT,
   CHAR,
   BOOL,
   VOID,
   PVOID,
   IUNKNOWN,
   PIUNKNOWN,
   UUID
};
```

```
    interface rcIServiceGeneric { 5083D1C4-0643-4ce6-B1EA-66467A65840B} {
    prcIUnknown getProvides( in String name )
         raises RC_ERR_SERVICE_NO_SUCH_NAME = 0x00000004;
    Void setRequires( in String name, prcIUnknown intf)
         raises RC_ERR_SERVICE_NO_SUCH_NAME  = 0x00000004;
    prcIUnknown getRequires( in String name )
```


© ISO/IEC 2007 – 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

```
     raises RC_ERR_SERVICE_NO_SUCH_NAME  = 0x00000004;
Void set( in String name, in rcEAttributeType type ,pVoid value)
     raises RC_ERR_SERVICE_NO_SUCH_NAME  = 0x00000004;
pVoid get( in String name, in rcEAttributeType type)
     raises RC_ERR_SERVICE_NO_SUCH_NAME  = 0x00000004;
}
```

10.1.10.2 Model

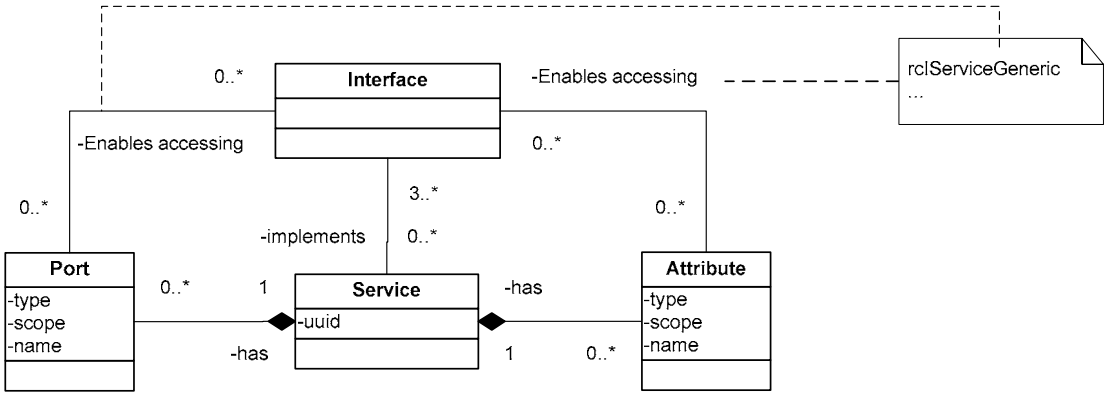


Figure 27 — rclServiceGeneric interface

10.1.11 ServiceFactory

10.1.11.1 Description

The creation and management of Service Instances is done via Service Factory (Instances). For each Service of a given type obtained from a Executable Component, there is exactly one Service Factory that can create Service Instances of that type and that can be obtained from the Executable Component. A Service Factory must support the `rcIServiceFactory` interface.

The primary responsibility of the Service Factory is in the creation of Service Instances. To that end, the `rcIServiceFactory` interface supports the following operation:

```
rcIService getServiceInstance();
```

NOTE There is no parameter to indicate the type of Service Instance that has to be created. A Service Factory can only create one type of Service. When creating a Service Instance, a (reference to) the `rcIService` interface is returned.

Service Factories support, through the `rcIServiceFactory` interface, the creation of Service Instances. The number of allowed instances is up to the implementation and the available resources on a given device. Once the maximum number of instances is reached, the function will return a `NULL` value to indicate a problem in Service creation.

A special case is in which a Service Factory will create at most 1 instance of the Service. The `getServiceInstance` will return a reference to the same instance on each call. The Service needs to specify this clearly.

© ISO/IEC 2007 – 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

10.1.11.2 Model

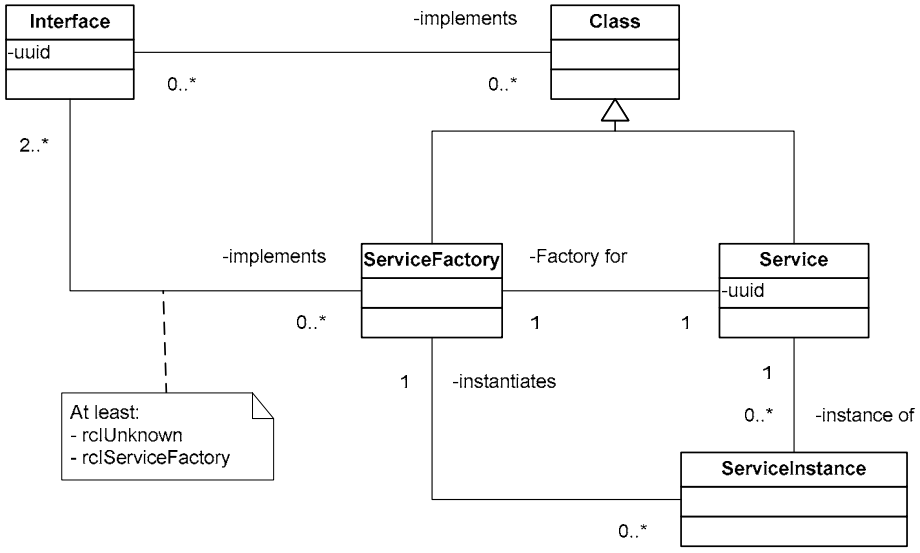


Figure 28 — Service Factory

10.1.12 rclServiceGenericFactory Interface

10.1.12.1 Description

Service Factories can have a `rcIServiceGenericFactory` Interface that supports the presetting of default values for a Service. The values that can be set include the Requires Port bindings and Attribute values. These values will be used to configure a Service that the Service Factory creates. The operations in this interface are the same as in the `rcIServiceGeneric` interface. Once a Service Instance has been retrieved, setting defaults in the Service Factory has no effect on that Service Instance anymore.

```
interface rcIServiceGenericFactory { 28B4E880-AF84-4c86-B5FD-FC82A9FE1746} {
prcIUnknown getProvides( in String name )
     raises RC_ERR_SERVICE_NO_SUCH_NAME  = 0x00000004;
Void setRequires( in String name, prcIUnknown intf)
     raises RC_ERR_SERVICE_NO_SUCH_NAME = 0x00000004;
prcIUnknown getRequires( in String name )
     raises RC_ERR_SERVICE_NO_SUCH_NAME = 0x00000004;
Void set( in String name, in rcEAttributeType type ,pVoid value)
     raises RC_ERR_SERVICE_NO_SUCH_NAME = 0x00000004;
 pVoid get( in String name, in rcEAttributeType type)
     raises RC_ERR_SERVICE_NO_SUCH_NAME = 0x00000004;
}
```


© ISO/IEC 2007 – 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

10.1.12.2 Model

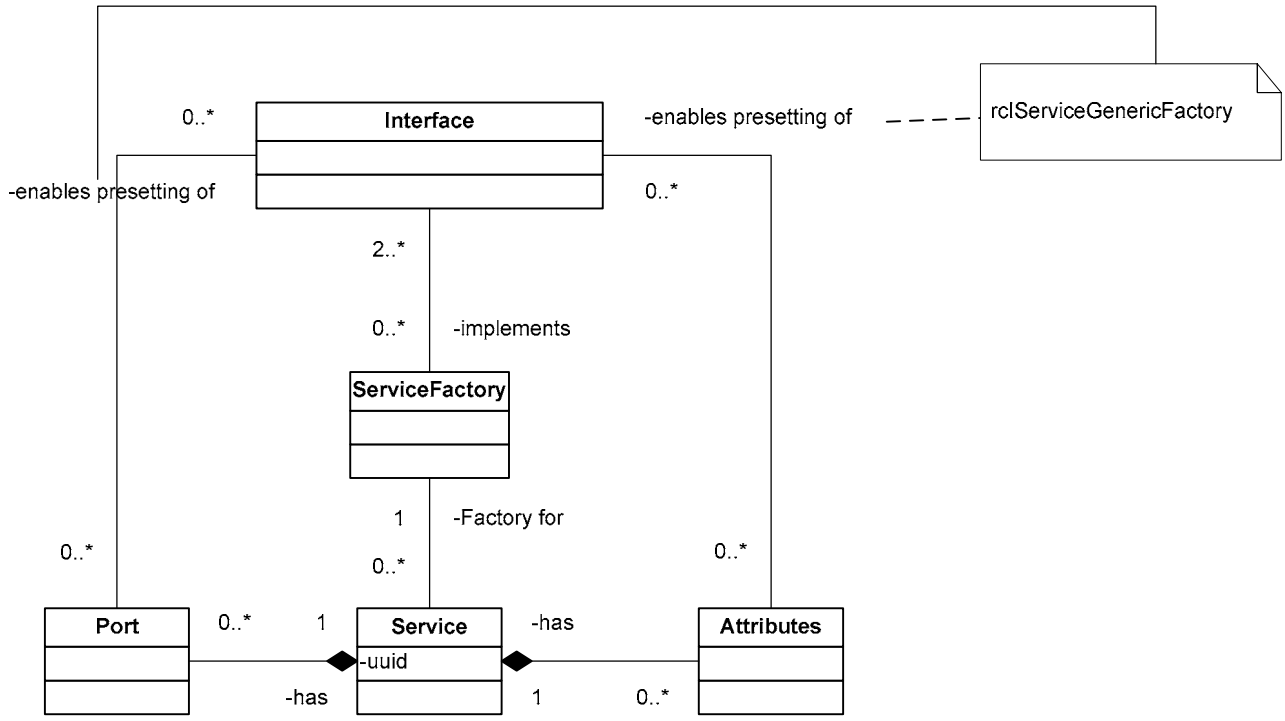


Figure 29 — rclServiceGenericFactory Interface

10.1.13 Service compliance

10.1.13.1 Description

In the realization technology used for M3W there is the notion of compliance (compatibility) between realization elements. Realization elements are services. Services are compliant when a client does not notice the difference. This means that a service must at implement at least the same interfaces, provide at least the same ports, have at most the same requires interfaces and have at least the same attributes. If the compliant service has less requires ports then binding to a port this is available in the blueprint service should not result in a failure.

10.1.13.2 Model

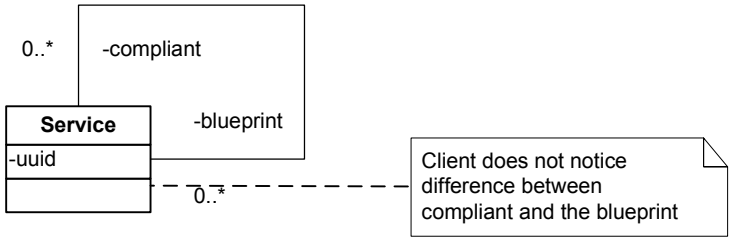


Figure 30 — Compliance between Services

© ISO/IEC 2007 – 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

10.1.14 Executable Component

10.1.14.1 Description

An Executable Component is a container for Services and the factory logic needed to create instances of these Services (Service Factories). From the viewpoint of the Runtime Environment, some interaction with the component is needed in order to create Service instances.

In order to obtain the Service Factories in the executable, every Executable Component must implement the `rcIComponent` interface. This interface contains all the functions that a Runtime Environment needs to manage an Executable Component and its contents. This interface contains the `getServiceFactory` operation to obtain a reference to a Service Factory, operations to related to initialization of the Executable Component and a safety operation by which the Runtime Environment can ask if there are any clients or other entities relying on the component.

```
interface rcIComponent { 24cfe19a-c143-4715-94c6-8b2cd8ee5eb4 }
{
    Void initialize()
        raises RC_ERR_RUNTIME_CANNOT_INTIALIZE = 0x00000002;
    Bool isInitialized();
    rcIServiceFactory getServiceFactory( in UUID svcId )
        raises RC_ERR_RUNTIME_NO_SUCH_SERVICE = 0x00000010
             , RC_ERR_RUNTIME_CANNOT_INTIALIZE = 0x00000002;
    Void finalize()
        raises RC_ERR_RUNTIME_CANNOT_FINALIZE = 0x00000004;
    Bool canUnload();
};
```

10.1.14.2 Model

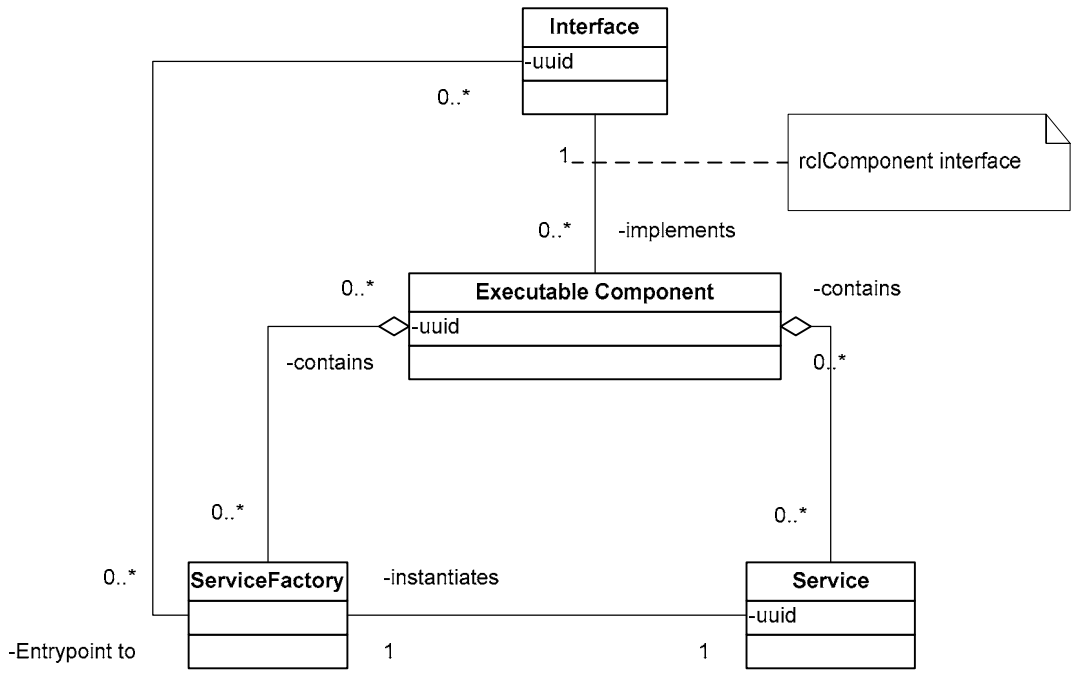


Figure 31 — Executable component

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

10.1.15 Runtime Environment

10.1.15.1 Description

The Runtime Environment is a native library that supports the use of Services within a M3W system. It supports several different interfaces that reflect different aspects of the M3W core framework. The Runtime Environment will implement these interfaces in a Platform specific way, typically as a statically linked library.

The core of the Runtime Environment functionality is defined by the `rcIClient` interface, a set of system operations that services and applications can use in order to create Service Instances. This is in fact the only part of the Runtime Environment that is mandatory and called the *client API*.

In addition to this mandatory core API, M3W has defined 4 more interfaces for supporting:

— Management of Component lifecycles

— Viewing, controlling and copying the Runtime Environment's registry of installed Executable Components

As some systems may not support or require the concepts related to these interfaces, these are optional for M3W Runtime Environments.

10.1.15.1.1 Client API

In order to instantiate Services and perform queries against the registry, the following interface is defined:

```
interface rcIClient { 41c235af-0417-4731-8627-c11c3d51d359 } {
    rcIService getServiceInstance( in UUID svcId )
        raises RC_ERR_RUNTIME_NO_SUCH_SERVICE = 0x00000010,
               RC_ERR_RUNTIME_CANNOT_INITIALIZE = 0x00000002;
    rcIServiceFactory getServiceFactory( in UUID svcId )
        raises RC_ERR_RUNTIME_NO_SUCH_SERVICE = 0x00000010,
               RC_ERR_RUNTIME_CANNOT_INITIALIZE = 0x00000002;
    rcIUuidItr getCompliesList( in UUID blueprint )
        raises RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001;
};
```

The method `getCompliesList` returns the list of entities that *comply to* the specified entity. This can be used to find out which Services support the specified Service. This function makes use of the Iterator idiom described to return a list of UUIDs.

10.1.15.1.2 Component Management API

In order to explicitly manage the lifecycle of Executable Components within a M3W process the following API is defined:

```
    interface rcILoadedComponentManagement
        { f4a9a915-a2dc-4a28-89a7-f1b96312ca1f } {
        rcIUuidItr getLoadedList()
          raises RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001;
        Bool isInitialized( in UUID cmpId );
        Void unloadComponent( in UUID cmpId )
          raises RC_ERR_RUNTIME_CANNOT_FINALIZE = 0x00000004;
};
```

The `getLoadedList()` function returns a list of UUIDs that uniquely identify the Executable Components loaded into the system. This function may return a null `rcIUuidItr` if there are no Loaded Executable Components or if the method is not implemented. If the method is not implemented the `RC_ERR_RUNTIME_NOT_IMPLEMENTED` error is returned.

© ISO/IEC 2007 – 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 `isInitialized()` function informs the caller whether an Executable Component is initialized or not. If a Executable Component is not loaded or not initialized the method will return `False`. If an Executable Component returned by the `getLoadedList` function is not initialized it must be in the Activated state.

The `unloadComponent()` method attempts to unload the Executable Component. When this method returns `True`, the Executable Component is no longer in use by any client of the Runtime Environment. If clients are using this Executable Component (indirect via interface references) the Executable Component will have the opportunity to negotiate the release of these interface references with the client. If this fails, the component is not unloaded and the method returns `False`.

10.1.15.1.3 Registry View API

This API allows a Client to view the content of a Registry, by allowing it to browse the contents of the Installed Components.

```
interface rcIRegistryView { 17745387-4687-4bf9-a9dd-7f958d27dc72 } {
    Bool isComponent( in UUID ident );
    rcIUuidItr getComponentList()
        raises RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001;
    String getComponentLocation( in UUID cmpId )
        raises RC_ERR_RUNTIME_UNKNOWN_COMPONENT = 0x00000100;
    rcIUuidItr getContainedList( in UUID cmpId )
        raises RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001;
    rcIUuidItr getContainingList( in UUID ident )
        raises RC_ERR_RUNTIME_NOT_IMPLEMENTED = 0x00000001;
};
```

The `isComponent()` method returns `True` if the given UUID refers to an Executable Component and the Executable Component is known to the Runtime Environment. This means that this will return `True` if and only if an Executable Component with the given UUID is registered with the Runtime Environment.

The method `getComponentList` returns the list of all Executable Components that are registered with the Runtime Environment.

The method `getComponentLocation` returns a String that describes the location of a registered (Installed) Executable Component. The format of this String is defined by the Runtime Environment implementation and is used in the `registerComponent` function used to register an Executable Component.

The method `getContainedList` returns the list of Services that are *directly* supported by the specified Executable Component.

The method `getContainingList` returns the list of Executable Components that support the specified Service.

10.1.15.1.4 Registry Control API

In order to register and de-register Executable Components from the Runtime Environment, the following interface is statically implemented by the Runtime Environment.

```
interface rcIRegistryControl { 4e7b2b79-e440-462d-9c6c-ccfc02f348ae } {
    Void registerComponent( in UUID cmpId, in String location )
        raises RC_ERR_RUNTIME_INVALID_LOCATION = 0x00000020,
            RC_ERR_RUNTIME_INVALID_COMPONENT = 0x00000040,
            RC_ERR_RUNTIME_ALREADY_REGISTERED = 0x00000080;
    Void unregisterComponent( in UUID cmpId );
    Void registerService( in UUID cmpId, in UUID svcId )
        raises RC_ERR_RUNTIME_UNKNOWN_COMPONENT = 0x00000100,
               RC_ERR_RUNTIME_ALREADY_REGISTERED = 0x00000080;
```


© ISO/IEC 2007 – 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

```
    Void unregisterService( in UUID svcId );
    Void setComplies( in UUID complying, in UUID blueprint )
        raises RC_ERR_RUNTIME_NOT_ALLOWED = 0x00000200;
    void clearComplies( in UUID complying, in UUID blueprint );
};
```

The method `registerComponent()` is used to specify the identity and the location of the Executable Component to the Runtime Environment. The location is specified in URL-like format. When the device hosting the Runtime Environment supports the notion of a file system, the `localUrl` can be of a file-type URL. For example: `file://sys/dowloaded/trusted/gsmstack.so`. When Executable Components are stored in memory, the location can be in the form of a memory address; e.g.: `address:0x1a2b3c4d`. Which formats are allowed to indicate the location of Executable components is up to the implementation of the Runtime Environment.

The method `registerComponent()` returns `false` when the Executable Component is already registered, and has no effect in that case. It can be the case that the Executable Component is already registered at a different location. The method `getComponentLocation()` can be used to obtain the location of an already registered Executable Component.

An Executable Component can be unregistered by the method `unregisterComponent()`. If an Executable Component is unregistered, all of its registered Services are unregistered as well: i.e. it is not needed to explicitly call `unregisterService` for the previously registered Services for that Executable Component. If an Executable Component was not previously registered, this method has no effect.

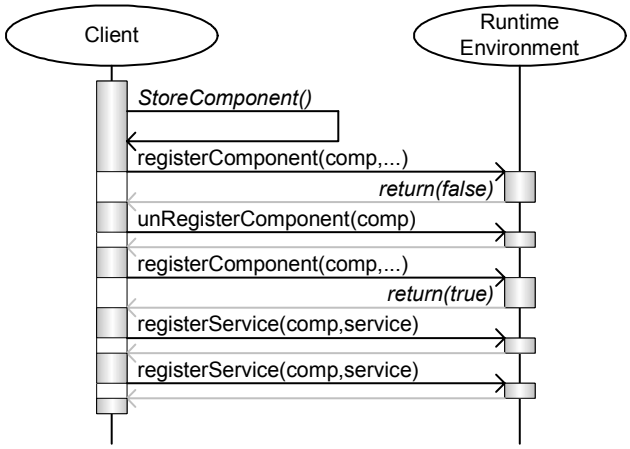


Figure 32 — Registering a previously registered executable component

The method `registerService()` registers a Service for a previously registered Executable Component. If the Executable Component is not registered an exception will be raised. If the Service is already registered to another Executable Component an exception will be raised. On a successful return, a Service will be registered with the Runtime Environment as being contained inside the specified Executable Component.

The method `unregisterService` can be used to un-register a Service. Note that it is still possible that Services Instances of the unregistered Service are used in the system (i.e. unregistering a Service on already instantiated elements).

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

© ISO/IEC 2007 – 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

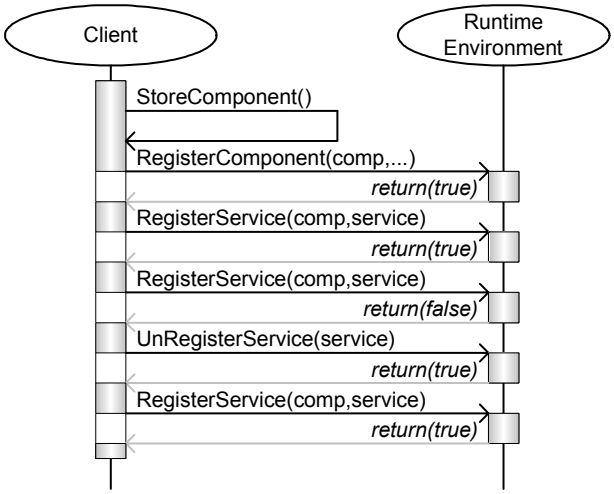


Figure 33 — Registering a Executable Component and its Services

The `setComplies` method can be used to inform the Registry that a Service complies with another. In this case the complying Service can substitute the blueprint. The Runtime Environment checks that the complies relationship refers to two Services, but is not required to do any further checking of the relationship.

The `clearComplies` method can be used to remove a complies relation between two entities store in the Registry. Clearing a non-existent relation always succeeds.

10.1.15.1.5 Registry Inspection API

This API is designed for use by system analyzers such as the System Integrity Management framework. It primary purpose is to allow a Client to access the “raw” data contained in the registry.

```
interface rcIRegistryInspect { c845e1b3-8176-4279-98e2-dd7e6cf5c9a5 }
{
    rcIComponentItr getComponentRecords();
    rcIContainerItr getContainerRecords();
    rcICompliesItr getCompliesRecords();
};
```

The `getComponentRecords` method returns an iterator that operates over a set of component records. These records contain (UUID, location) pairs, where the pair contains the UUID of a registered Executable Component and its registered location, which matches the format used by the `registerComponent` and `getComponentLocation` functions.

The `getContainerRecords` method returns an iterator that operates over a set of container records. These records contain (UUID,UUID) pairs, where the first UUID refers to a Executable Component and the second refers to a Service that the Component contains.

The `getCompliesRecord` method returns an iterator that operates over a set of complies records. These records contain (UUID,UUID) pairs, where the first UUID refers to a Service and the second refers to a Service that the first UUID complies to. This format is the same as that used by the `setComplies` function.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

10.1.15.2 Model

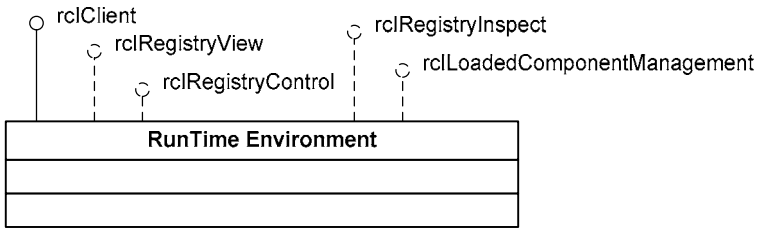


Figure 34 — Runtime Environment and Supported Interfaces

10.1.16 Registry

10.1.16.1 Description

In order for the Runtime Environment to instantiate the correct Services upon a request, the Runtime Environment stores information about Services, and the Executable Components. We use the term *registry* when referring to this (typically persistent) storage. In the registry, three relations are administered.

For each Executable Component, the location is stored. This location is in the form of an URL, and allows the Runtime Environment to locate the Executable Component. Note that this is a local URL. For example, for M3W devices that store the components in a file system the URL typically starts with '`file://`'.

Secondly, for each Executable Component it is administered which Services can be *supplied* by that Executable Component. This is a relation between Services and Executable Components, allowing the Runtime Environment to determine which Executable Component can be used upon request for an instance of a Service. For each Service, there can be at most one Executable Component registered to supply that Service.

The third relation is the *complies* relation between Services. When Service S1 can be used instead of (i.e. is a true extension of) Service S2, the Service S1 is said to comply with Service S2. This allows the Runtime Environment to use an Executable Component that can supply Service S1 when clients request a Service S2.

With the operations in the *registration API (*`rcIRegistryControl`*),* the Runtime Environment registry can be changed. The operations in the *client API (*`rcIClient`*)* use the Runtime Environment registry without changing it.

© ISO/IEC 2007 – 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

10.1.16.2 Model

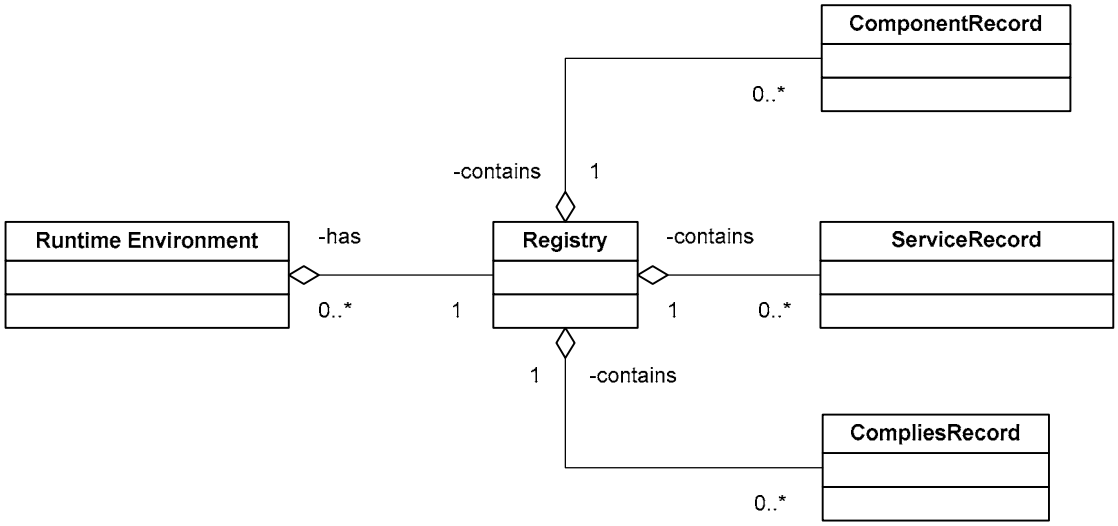


Figure 35 — Registry

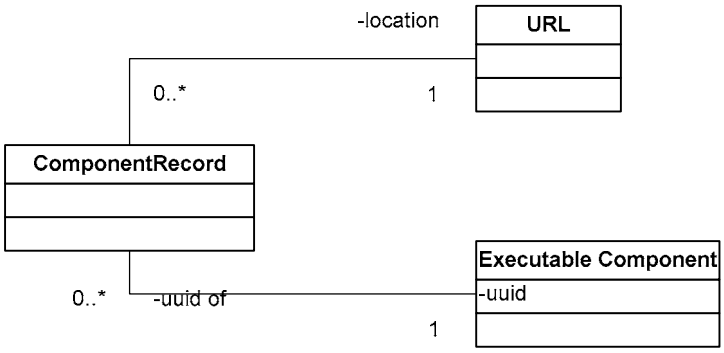


Figure 36 — Component Registration

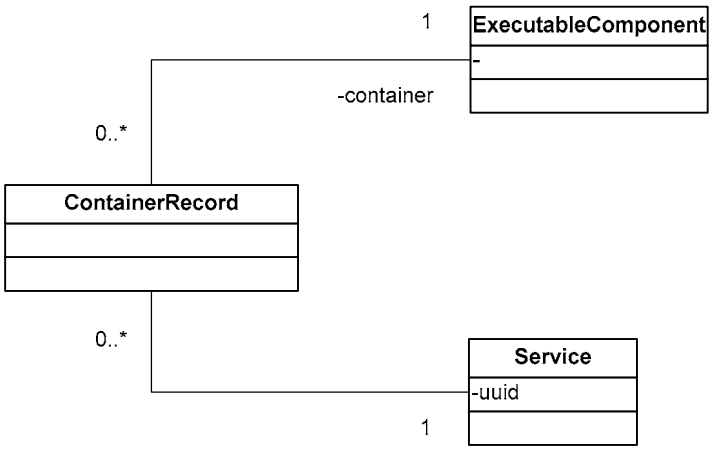


Figure 37 — Service Registration

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

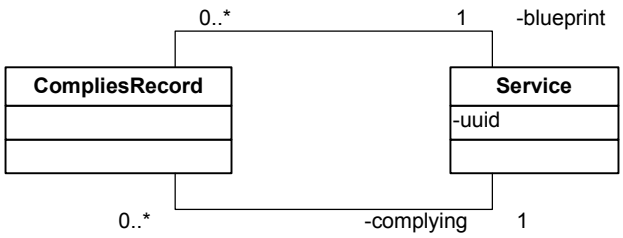


Figure 38 — Complies Registration

10.2 Behaviour

10.2.1 Executable Component life cycle

This section details the part of the life cycle when an Executable component is loaded onto the Device.

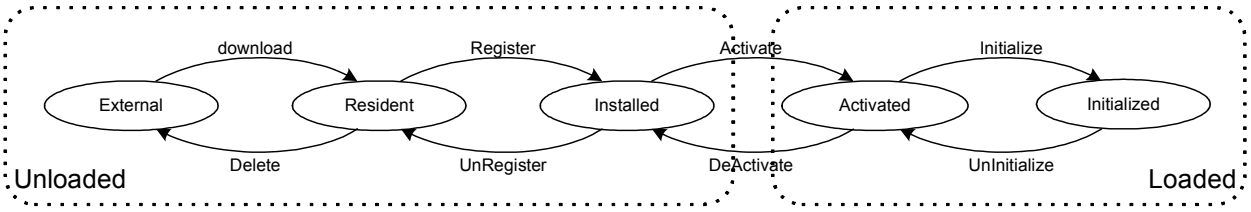


Figure 39 — Component life cycle

The component that is downloaded can contain several Models. It is not specified if all Models are downloaded to the target. However, it is safe to assume that at least the Executable Component will be downloaded. In the remainder of this section, we consider the Executable Component only.

After the Executable Component is *downloaded*, the Component is present on the target in some kind of (persistent) storage (Resident). It is not yet known by the Runtime Environment. Making the Executable Component known to the Runtime Environment is called *registering*. The location of the Executable Component is made known to the Runtime Environment, as well as the Services it can provide – the Component enters the Installed state. Only after that can a request for Services hosted by the component succeed.

In order to instantiate Services, the Executable Component may need to be transferred from its storage location into a memory space from which the code can be executed and static data for the component may need to be allocated. Typically, this is done by some low-level OS services. This step is called *Activation*, after which the component is in the Activated state.

The Executable Component is passive during these state changes. These state changes are triggered externally, and they either operate upon the Component or adapt the environment of the Component (e.g. registering the Component).

Once the activation is done, some additional *initialization* takes place before Services can be instantiated. At this point, the Executable Component gets actively involved in the management of its own lifecycle through the `rcIComponent` Interface. Firstly Runtime Environment calls the `initialize()` method of this interface. This gives the Component the opportunity to dynamically initialize itself, allocate any necessary resources etc. The `initialize()` method is not allowed to block.

After a successful call to `initialize()`, the Runtime Environment may call the other methods in this Interface. One operation will retrieve a Service Factory as described in 10.1.11.

© ISO/IEC 2007 – 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 Activated state and the Initialized state both occur when the Executable Component has been loaded into a memory space, and form a combined Loaded state.

The `isInitialized()` method checks the initialization status. The `finalize()` method asks the Executable Component to finalize itself; this is only a request and whether the Executable Component can indeed finalize depends on whether it is in use by clients (that have interface references to any of its Service Instances.

The moving of an Executable Component from a Loaded state to an Installed state requires interaction between the Runtime Environment and the Executable Component. This transition can be achieved in two ways:

— Explicit: external request to remove a Loaded Executable Component,

— Implicit: Executable Component requests for self-unloading.

In the first case the Runtime Environment provides an interface that allows clients to view the set of Loaded Executable Components and explicitly request that the Runtime Environment unloads an Executable Component, as shown in Figure 40. In the second case the Component maintains a record of the number of Service Instances that are currently operational, and requests the Runtime Environment to unload itself when there are no operational Services; this process is shown in Figure 41.

In both cases of Executable Component transition from the Loaded state, the Runtime Environment guarantees that the Executable Component's `finalize` function will be called. In the case that an Executable Component cannot be finalized an error will be reported to the Client. On success, the Runtime Environment will schedule the Executable Component for removal from the memory space, the exact timing of the removal procedure it left as a design decision for a Runtime Environment implementer. Prior to calling the Platform specific routines for deactivating a Executable Component the Runtime Environment will call the Executable Component's `canUnload` function to ensure that all resources required by the Executable Component have been released.

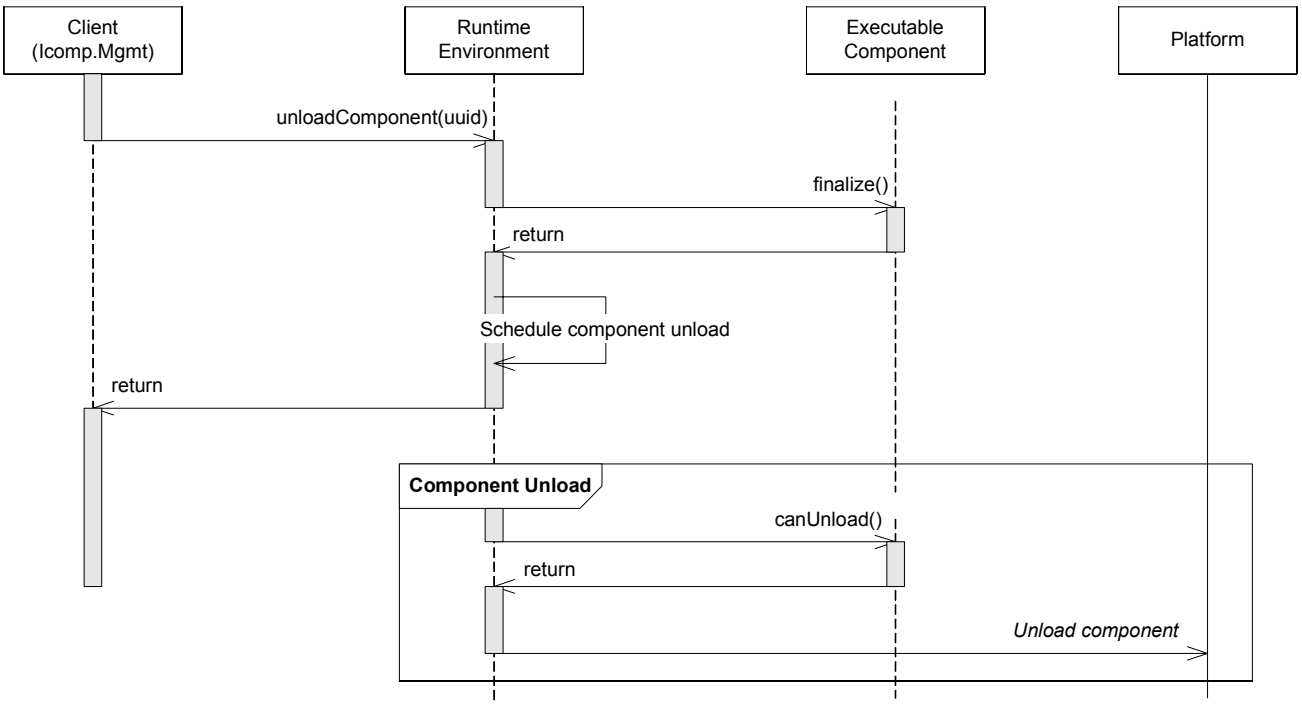


Figure 40 — Explicit client request for Executable Component transition into Unload state

© ISO/IEC 2007 – 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

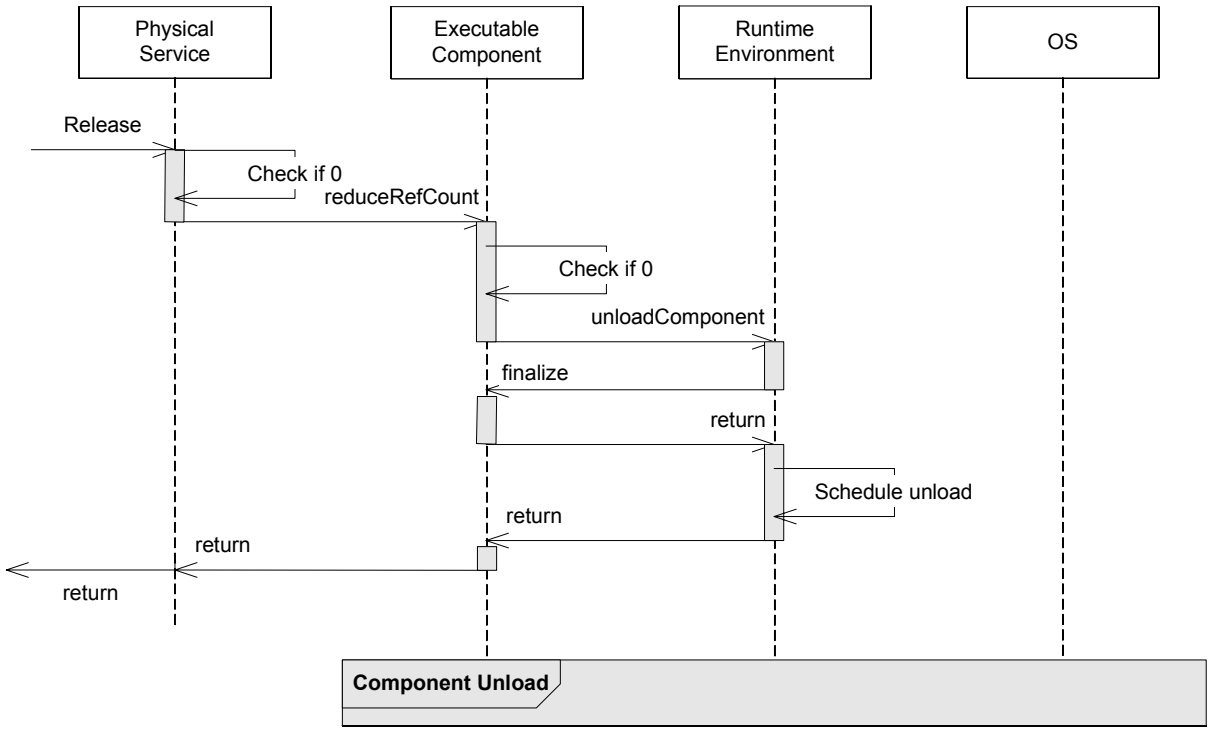


Figure 41 — Executable Component managed self-unload

10.2.2 Service life cycle

The Service instance life cycle is contained within the Executable Component life cycle. Service Instances can only be created when the Executable Component that hosts the services is *initialized*.

10.2.2.1 Service Instance creation

Clients can create instances of a Service with the aid of the Runtime Environment which provides the `rcIClient` Interface for clients to interact with the Runtime Environment. The method that creates a Service Instance and returns a reference to the `rcIService` interface supported by the Service instance is

```
rcIService getServiceInstance( in UUID svcID )
    raises RC_ERR_RUNTIME_NO_SUCH_SERVICE = 0x00000010,
           RC_ERR_RUNTIME_CANNOT_FINALIZE = 0x00000004;
```

This operation will create a Service Instance in several steps. The Figure below indicates all the steps during the execution of this operation.

© ISO/IEC 2007 – 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

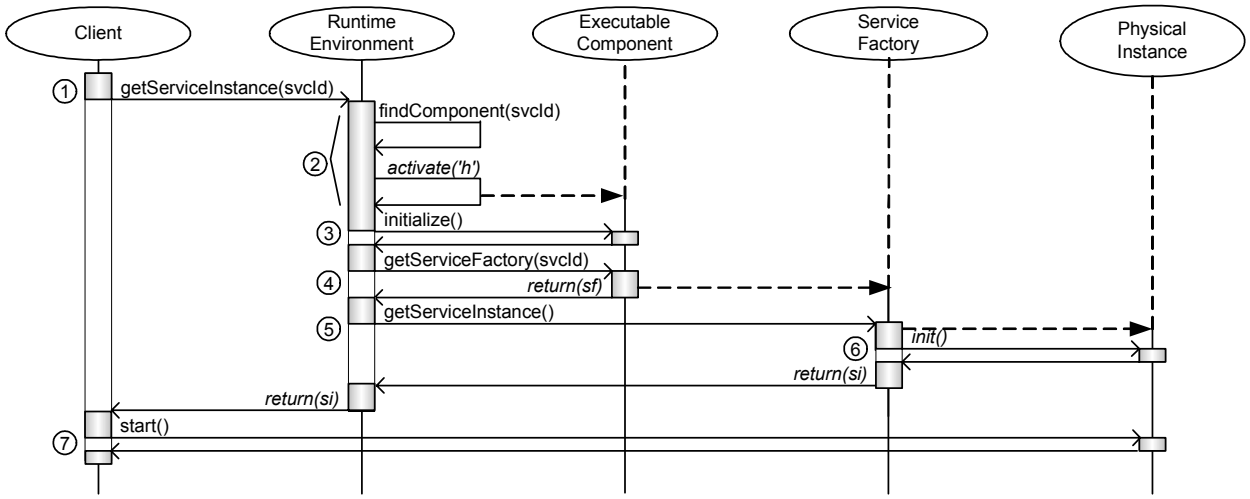


Figure 42 — Service Instance Creation in the Runtime Environment

When the client calls the Runtime Environment via the operation `GetServiceInstance` (1) it passes the identification UUID (`svcId`) for the Service required. The Runtime Environment will then determine the set of Executable Components that can provide this Service or a Service that complies with the requested Service as well as the location of these Executable Components. Using a private algorithm (e.g. one that takes into account which Executable Components in the set are already loaded and/or initialized) it selects which Executable Component will be used to obtain the requested Service Instance. If the Executable Component is not yet loaded, it will use the OS to load the Executable Component and make the entry point accessible to itself (2). It will then call the `initialize()` method on the Executable Component (3). If this initialization is successful, the Runtime Environment gets an interface reference to the `rcIServiceFactory` for the required Service Factory in the Executable Component (4). From this `rcIServiceFactory` interface reference, it will call the `GetServiceInstance` (5). The Service Factory in turn will create a Service Instance and Initialize the Service Instance (6). Note that this Initialization is internal to the Service Factory and Service Instance, i.e. there does not need to be an explicit `Init` method. In the diagram, it merely indicates the activity. When all this is successful, the call returns to the client and the `rcIService` interface reference is returned to the client. The Service Instance is not yet operational. The Service Instance needs to be started first by calling the `start()` method in the `rcIService` interface (7).

Before the `start()` method can be called, however, a compatible interface reference must be bound to all requires ports. Also the values of the Attributes may need to be set. This can be done by the client. To do so it can navigate (using `QueryInterface`) from the `rcIService` interface to the Service specific descendant and use its type safe operations.

A client needing more control or needing to instantiate multiple instances of a Service, can first use the Runtime Environment to get access to the Service Factory and subsequently create a Service Instance itself. See Figure 43. The calls labeled (1a) and (1b) illustrate this.

© ISO/IEC 2007 – 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

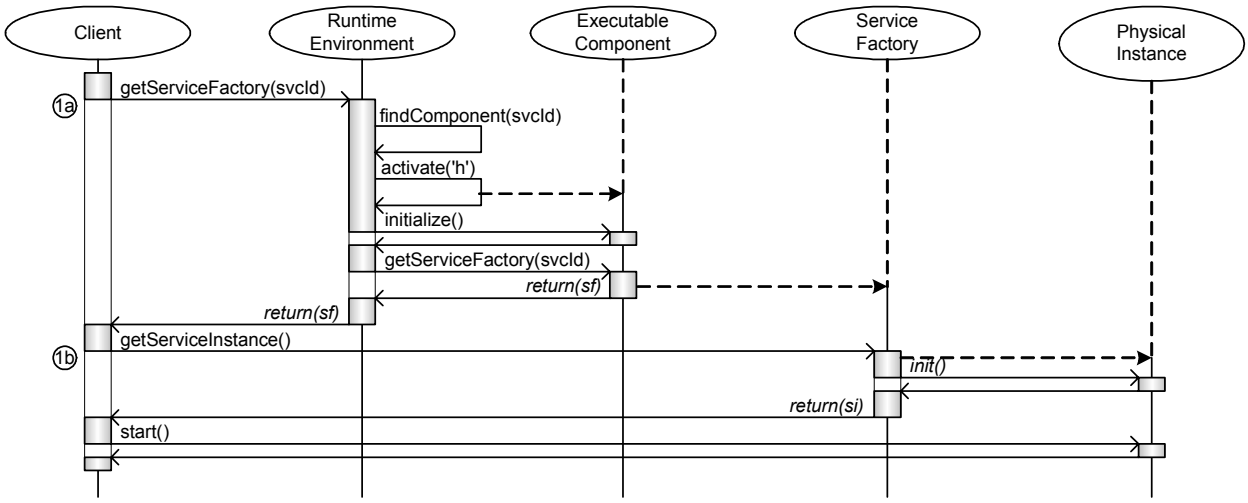


Figure 43 — Creating Service Instance explicit via Service Factory

The benefit of the latter Service Instance creation is to have more control over the binding of requires interface and setting or Attributes. A client that wants to set default bindings for each Service Instance can do that via the Service Factory (assuming the Service Factory supports this through the Service Specific Factory Interface. The alternative is to set it explicitly for each newly created Service Instance.

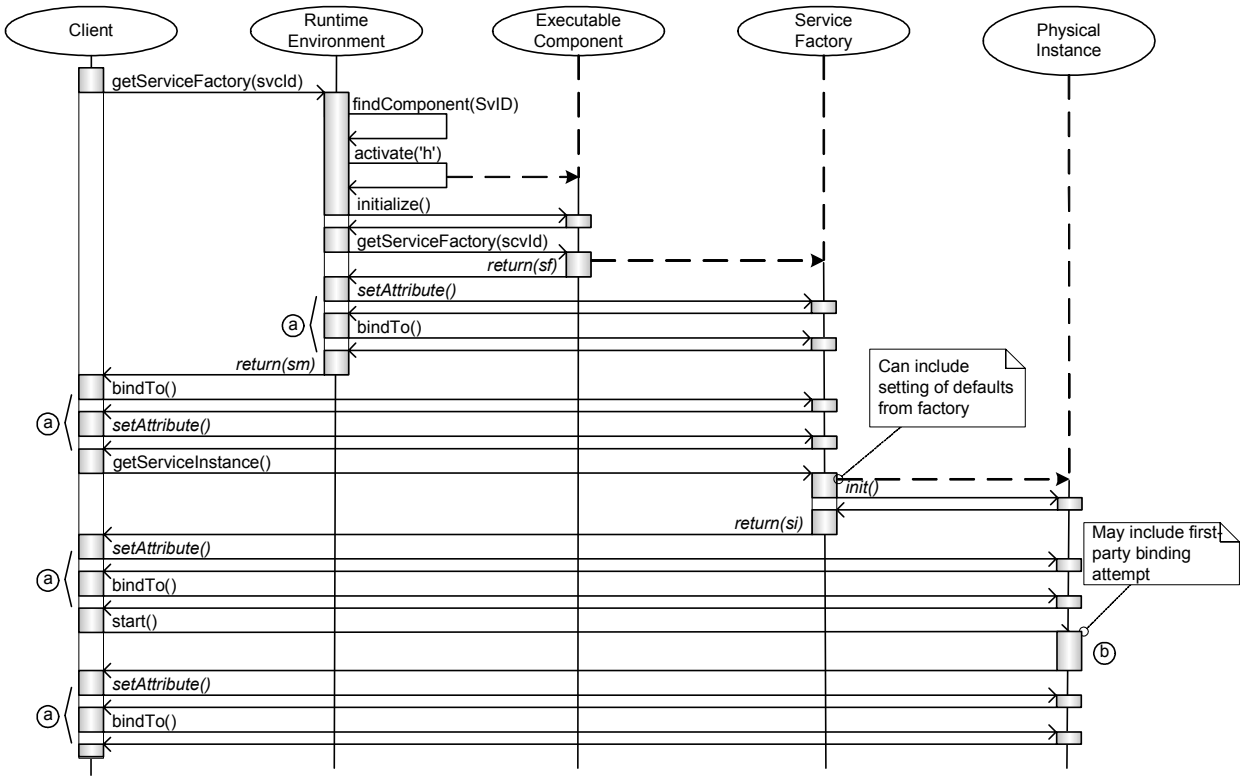


Figure 44 — Setting attributes and requires ports

When the Service Instance is started, it can be the case that not all properties/bindings needed are established. The Service Instance could try to establish the missing bindings by interacting with the Service Factory or the Runtime Environment directly or through any of the interfaces bound (b). If this fails it must raise the `RC_ERR_RUNTIME_INSUFFICIENT_BOUND` exception.

© ISO/IEC 2007 – 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 `start()` method must return in finite time. If the Service has autonomous behavior it is not allowed to start before the `start()` method is called. One way to guarantee this is to initiate the autonomous behavior from the `start()` method.

10.2.2.2 Service Instance life time

The lifetime of Service Instances (and any other object) is controlled implicitly by the outstanding references. As long as there are clients that have an interface reference to an object, the object should be available to the clients. By using the lifetime control mechanisms present in each interface, the `AddRef()` and `Release()` operations, clients can indicate whether they are (still) using the interface. Once all interface references to an object are released, the object supporting the interface references can be removed. The details on how this is done are internal to the component.

However, it is foreseen that objects, including Services Instances, have to be terminated while in use, e.g., when an Executable Component is to be replaced in a running system. Executable Components / Services that need to support such a kind of forceful removal may have to provide additional interfaces to support this.

The Service Instances adhere to the life cycle as depicted below.

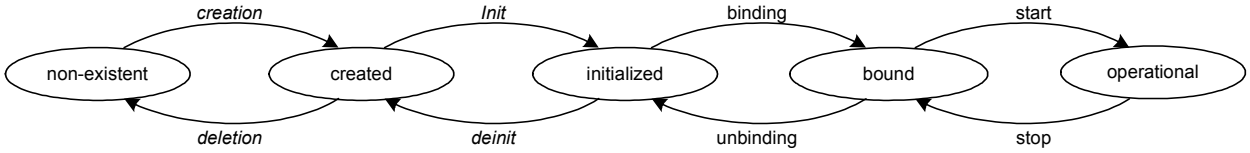


Figure 45 — Service Instance life cycle

Clients of a Service Instance external to the Executable Component containing the Service implementation do not have full visibility of all states. For example, it is not possible for a client to get a reference to a non-initialized Service Instance. The only states directly observable to a client holding a reference to a Service Instance are the *initialized, bound* and the *operational* state.

In order to support the Component lifecycle, Services may have to inform their containing Component about their deletion. This will allow the Component to keep an accurate count of the Service instances that depend on it.

11 Service Manager

11.1 Description

Service Manager (optional) provides its clients (i.e., user applications) a transparent access to the functionality offered by the services resides in the M3W. Transparent access means that the access is requested without requirement to have in depth knowledge how the services are implemented. The access is requested base on the logical component (group of multimedia APIs – see ISO/IEC 23004-2). Client of Service Manager requests access by giving the logical component as input parameter and in return Service Manager gives handler to the instance of service that implement the requested functionality. This transparent access is the key of application portability from one M3W implementation (on a certain platform) to other M3W implementation (on other platform).

11.2 Service Manager's APIs

11.2.1 Client API

Service Manager provides feature to its client to get access to the instance of service through rcISMClient interface. The interface contains APIs as shown in the boxes below.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

```
rcResult isServiceAvailable (in pUUID lcID, out Bool retValue);
```
By using “isServiceAvailable” API, client can inspect whether the implementation of the logical component (pUUID lcID) is available. Upon invocation of this API, Service Manager get the required interfaces to fulfill the logical component then checks them against the services that are in the M3W. If there is a service or a combination of services fulfills all the required interfaces, Service Manager shall return true, otherwise returns false.

```
rcResult getInstanceForLogicalComponent (in pUUID lcID, out prcIService
*retValue);
```
By using “getInstanceForLogicalComponent” API, client can request a (pointer) handler service instance that implements the requested logical component (pUUID lcID). Upon invocation of this API, Service Manager get information of which service or list of services that implements the logical component. It then checks the service is instantiated already or not, if there is no instance available, Service Manager may request Runtime Environment to instantiate it.

11.2.2 Management API

Service Manager provides optional APIs to manage the information (in form of metadata) about the logical component and service through rcISMControl interface. With these APIs, clients (i.e., middleware’s administrator, service) can register and un-register the metadata to the Service Manager.

```
rcResult registerLogicalComponentMetadata (in String path, out Bool retValue);
rcResult unregisterLogicalComponentMetadata (in pUUID lcID, out Bool retValue);
```
“registerLogicalComponentMetadata” and “unregisterLogicalMetadata” provide way to add and to remove metadata about logical component to the Service Manager. When registering the metadata, client (i.e., middleware’s admistrator) may formulate the metadata in an XML file first then put the path to the file in the parameter of the invoked API.

The logical component can be un-registered one by one. This is possible since the input parameter of the “unregisterLogicalComponentMetadata” receives the id of logical component.

```
rcResult registerServiceMetadata (in String path, out Bool retValue);
rcResult unregisterServiceMetadata (in pUUID srvID, out Bool retValue);
```
“registerServiceMetadata” and “unregisterServiceMetadata” provide way to add and to remove metadata about service to the Service Manager. When registering the metadata, client (i.e., service) may have the metadata already formulated (as part of its installer’s files) as the metadata in an XML file first then put the path to the file in the parameter of the invoked API.

The service can be un-registered one by one, for example when it is uninstalled from the middleware. This is possible since the input parameter of the “unregisterServiceMetadata” receives the id of service.

11.3 Logical component and service

An M3W logical component contains standardized group of multimedia APIs that performs a certain multimedia functions (see part 2 of specification). It provides a list of APIs’ definition while the implementations of those APIs are provided by service. Hence, in other word, it may be analogized as abstract class and instance. Class definition contains APIs that can be used to access its functionality while the real implementation is provided by the instance of that class.

Figure 46 — Logical component and service, shows an illustration of logical component and service in the M3W. From the application’s view point, the M3W is a collection of multimedia functionality that can be accessed through a set of predefined/standardized APIs grouped in logical component. Application does not have to know how these logical components (and their APIs) are implemented. When the application needs to access certain functionality provided by M3W, it simply requests it to the Service Manager by stating the logical component. Service manager handles the searching of appropriate service, instantiation, and binding (configure the hierarchy if the service has dependency to others). Service Manager provides linkage between logical component and service.

© ISO/IEC 2007 – 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

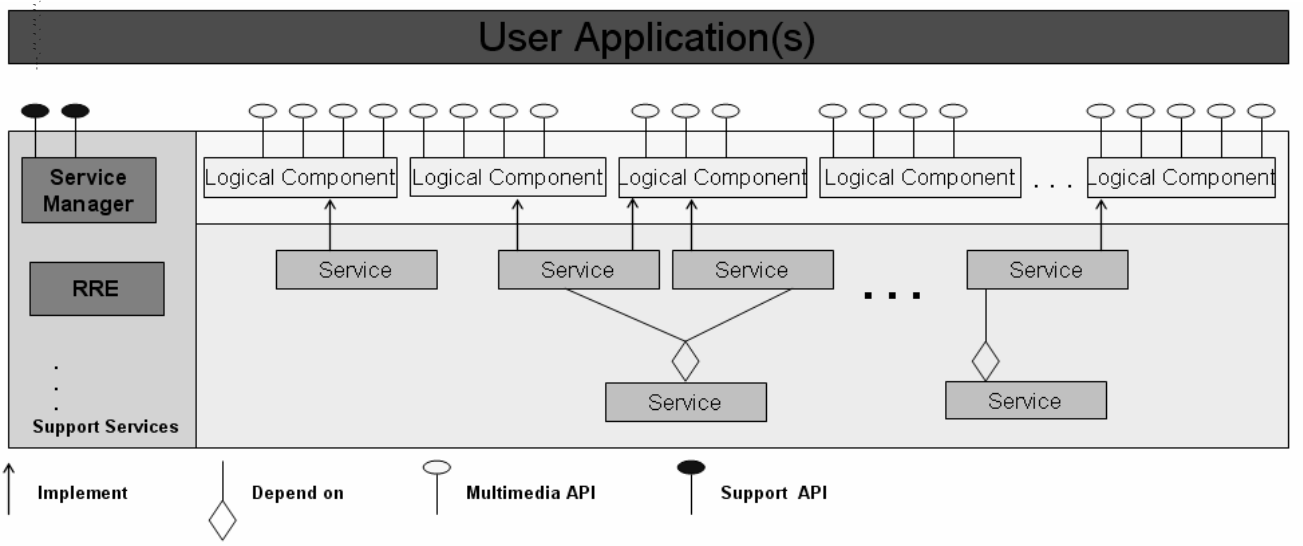


Figure 46 — Logical component and service

Set of logical component must be uniformly available in all implementation of M3W. It means that all M3W offer the same standardized multimedia APIs definition as defined in the part 2 of M3W specification. However, access availability to those multimedia APIs depends on the existence of service that implements them. As shown in Figure 46 — Logical component and service, at a snapshot of time, not all logical components has service that implements it (see the fourth logical component from the left). Hence, a request to use API member of that logical component cannot be fulfilled. There are some solutions for that kind of situation. For a static and not connected M3W case, Service manager can notify the requesting application that the requested functionality is not available. In other case, in connected (i.e., to internet) M3W case, if the location of service is known, download mechanism can be initiated followed by installation of the newly obtained service.

11.3.1 Logical component metadata

To provide linkage between logical component and service, Service Manager keeps information of both entities. The logical component metadata provides information about the list of logical component recognized by the M3W implementation. It contains a list of logical component in which hierarchically have roles, interfaces, and APIs (methods). See 11.6.1 (Logical Component Metadata) for the complete elements of the metadata together with their syntax and semantics. Annex B provides the XMLSchema definition of the metadata.

11.3.2 Service metadata

Service provides information about itself in the form of metadata to the Service Manager (can be at its installation stage). It contains information about its identification, which interface(s) it implements, the dependencies to other functionality (provided by other service(s)), rights information that governs its usage, environment property where it works well, etc. See 11.3.2 (Service Metadata) for the complete elements of the metadata together with their syntax and semantics. Annex C provides the XMLSchema definition of the metadata.

11.4 Hierarchical structure of service

Service is designed to be flexible, modular, and efficient in its implementation. This can be achieved by letting an implementation of a service exposes its interfaces to other service.

Figure 47 — Service composition example, illustrates the implementation of a service that efficiently uses the functionality from other services. A video decoder service, for example, may consist of several independent functions such as a variable length coding (VLC) function, discrete cosine transform (DCT) function, quantization function, motion estimation/compensation function, etc. The video decoder service can be

© ISO/IEC 2007 – 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

implemented as modular as possible by using other implementations that offers the required functionalities. Hence, in its implementation, the implementer needs to code the execution flow of those services.

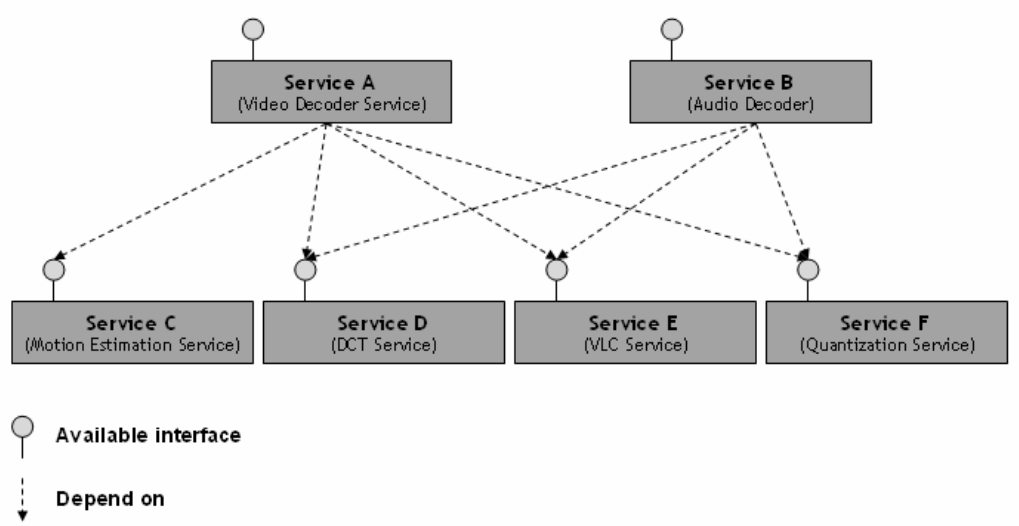


Figure 47 — Service composition example

In other cases, supposed that in the same middleware, an audio decoder service also exists at the same time. With the hierarchical concept of the service implementation, it may not be necessary to have two DCT, VLC, and quantization services. The same services that are used for the video decoder service can be used to serve the audio decoder service as well so that the middleware can be very efficient in the sense that there is no duplication in the offered function.

The binding between service and its dependencies is mediated by the Service Manager. During its instantiation and initialization, service can request to the Service Manager to provide its dependent service(s). Service Manager gives the best possible service by using the knowledge from the available metadata.

11.5 Service Manager usages

Service Manager provides functionalise regarding the logical component and service. The subsections below list some non-exhausted usages of Service Manager.

11.5.1 Validating service installation

M3W allows third party implementation for providing a service. This makes the service must be validated whether it implements interfaces that are belong to recognized logical component of M3W when it is installed in the middleware. The validation process involves checking to make sure that the functions and APIs implemented by that Service are conformance to what are standardized by M3W in logical components (and their multimedia APIs).

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

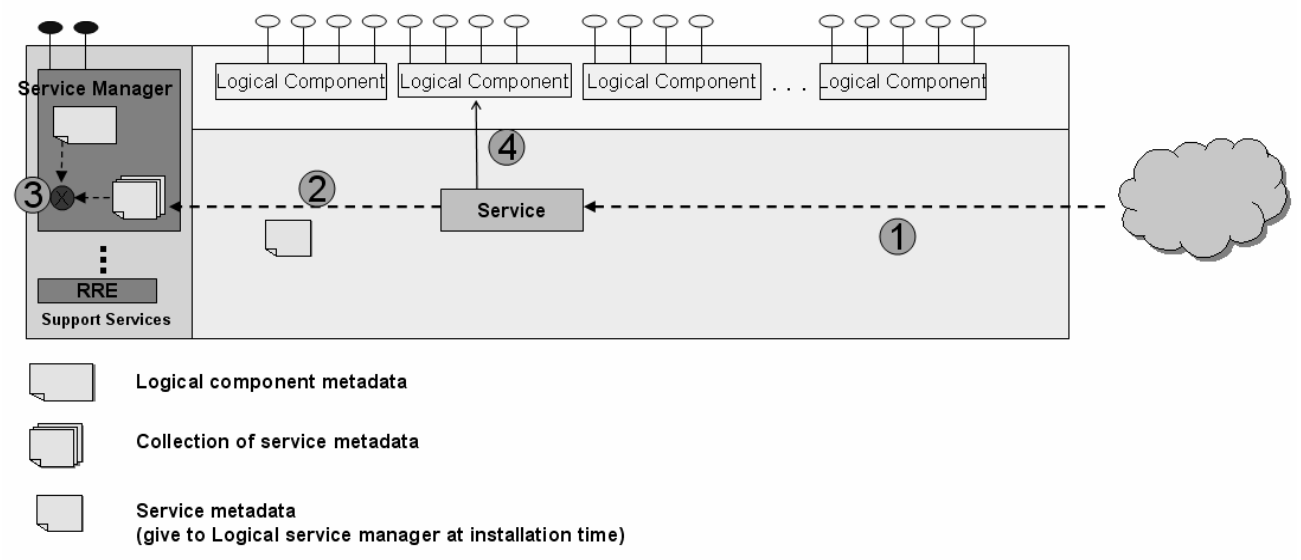


Figure 48 — Process in Service Manager when installing new service

Figure 48 — Process in Service Manager when installing new service, shows processes happen in Service Manager when a service is downloaded and installed into the middleware. Note that this use-case and illustration is simplified in term of suppressing the role of Runtime Environment. Assume that the middleware has been running in the terminal. The logical component has been configured with the definition of multimedia APIs by the logical component metadata. The following events happen when a service is obtained and installed to the middleware:

4) The service is obtained from middleware's external environment and ready to be installed.

5) At the installation process, the service gives its metadata to Service Manager. The service metadata provides information of what functions (interfaces of logical components) are implemented.

6) Service Manager checks the provided implementation (service metadata) against the standard ones (logical component metadata). A conformance service implementation is the one whose functions are defined by in the logical component. If there is function that is not match to the definition, Service Manager can reject the installation of that service.

7) If all the functions/interfaces implemented by the service are all in the logical component metadata, the installation can proceed to further step. After these steps, Service Manager knows which logical component(s) is/are implemented by the newly installed service.

11.5.2 Extending multimedia APIs

When the definition of the multimedia API is modified (i.e., adding new multimedia functions with newly defined logical service and APIs through MPEG amendment process) there should be no changes required in the Service Manager. Figure 49 — Illustration for MM APIs extension, shows the illustration of how to manage the extension of the multimedia API independent from the Service Manager. It is possible since logical component metadata provides a generic and standardized model that represents the multimedia API and serves as the configuration for the Service Manager. When the multimedia API definition is extended (in the figure, from v1.0 to v2.0) there is nothing needs to be changed in the Service Manager. The changes in the multimedia API definition are enough to be accommodated by re-modelling it into the new configuration file and feed it to the Service Manager.

© ISO/IEC 2007 – 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

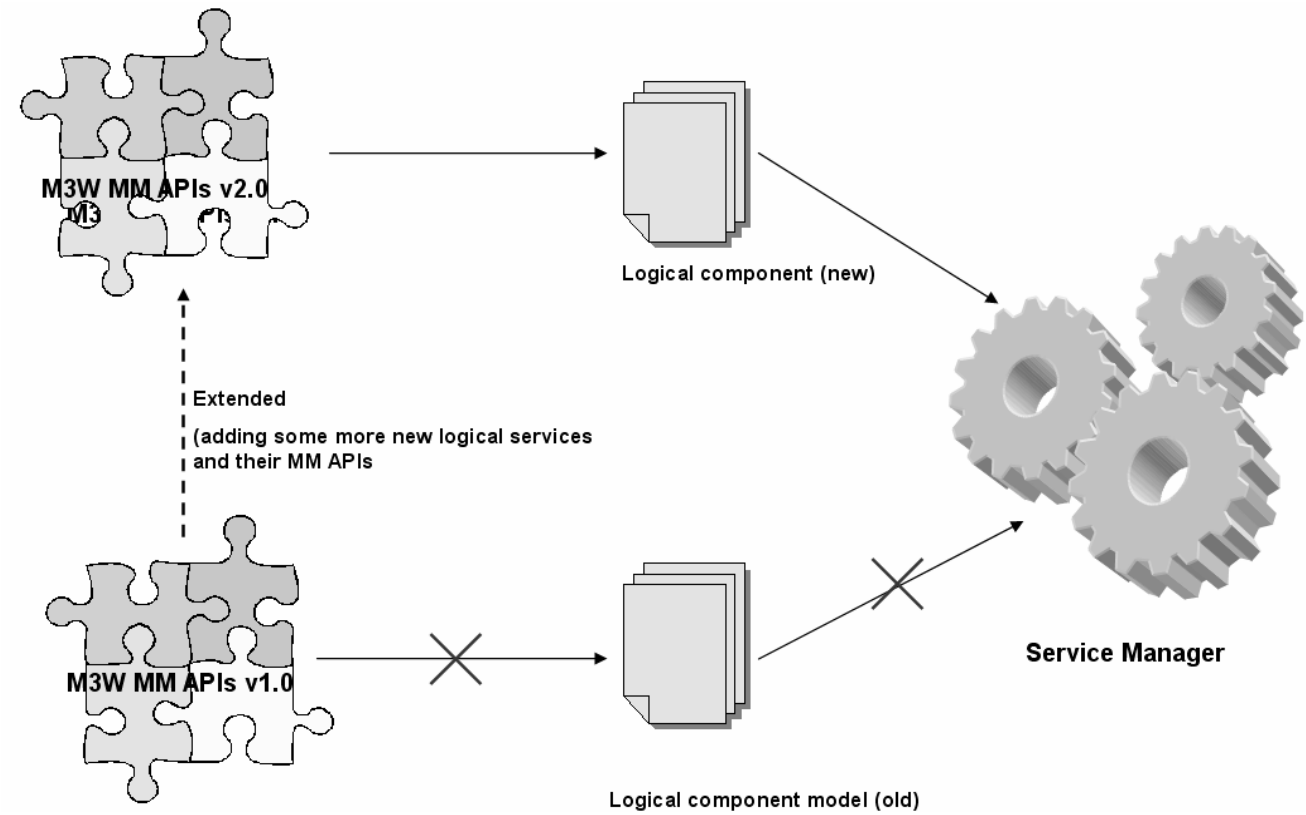


Figure 49 — Illustration for MM APIs extension

11.5.3 Accommodating non-standardized services

Any parties who implement the M3W may have their additional features which may not be standardized. This can be for sustaining their competitive advantages. M3W provides a means to accommodate this additional feature as long as it is built on top of the standard M3W.

One solution is by exploiting the logical component definition. The CE manufacturer can overwrite the metadata description of the logical component by adding the definition of its own service and then feed it to the Service Manager. Note that since the added functionality definition in the logical component is not standard then it is used only by applications that are made by that CE manufacturer itself.

© ISO/IEC 2007 – 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

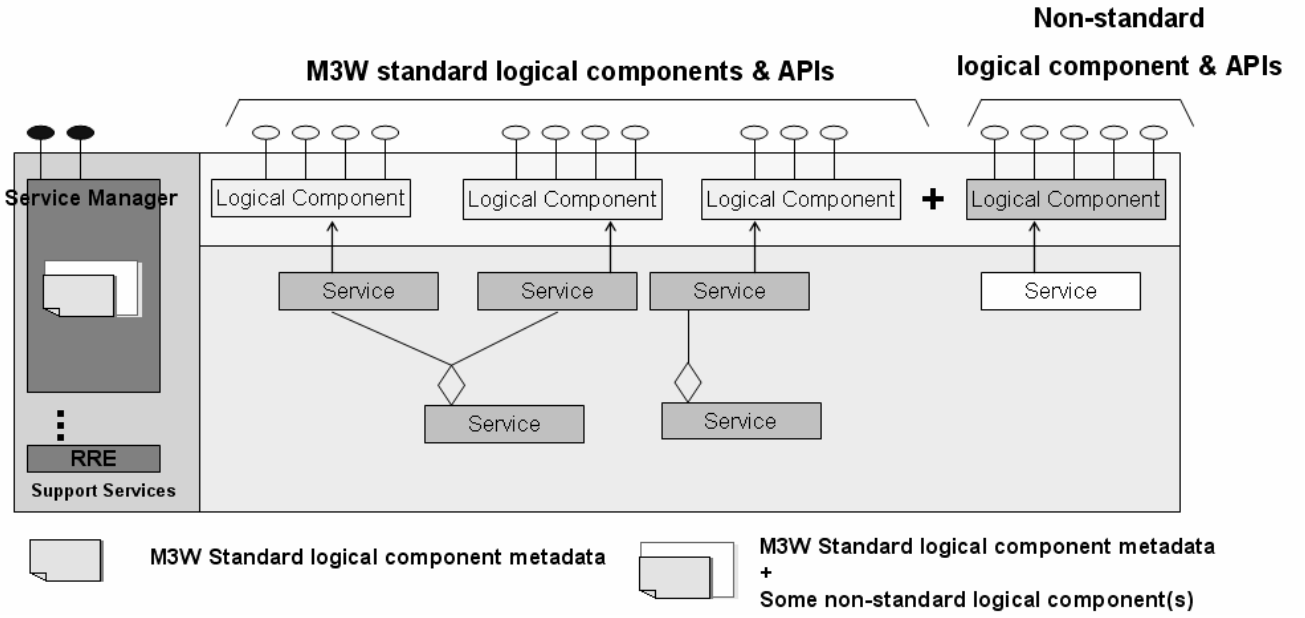


Figure 50 — Accommodating non-standardized logical services and APIs

Figure 50 — Accommodating non-standardized logical services and APIs, shows an illustration where a middleware is conformance to the standard M3W by providing all the standard logical components standardized by M3W and at the same time adding other logical component which is not standardized. This can be easily achieved by describing this non-standardized logical component together with the standardized ones. In this way, the middleware preserves its interoperability to all other M3W middleware while providing more capabilities to support special features of its manufacture vendor.

11.5.4 Configuring/Binding service dynamically

The flexible and modular hierarchy of service implementation can be used to dynamically configure a service at the instantiation phase. Configuring Service at the instantiation phase is done by linking a service to its services that implement the required interfaces and the process is repeated for each services.

Service Manager shall be the entity that is responsible to handle the dynamic configuration of service. When configuring service, Service Manager uses the information from service metadata to resolve dependency of service that needs to be instantiated.

Since it is possible to have several services that implement a certain interface, Service Manager takes the role in deciding which service should be used. Figure 51 — Choosing service to be linked, shows the simple illustration of decision taking by Service Manager in choosing services. For example, MPEG-2_Video_Decoder_Impl service has dependency on the APIs provided by interface DCT. From the collection of its metadata, Service Manager knows that there are three services from different vendors that implement interface DCT, they are DCT_VendorA_Impl, DCT_VendorB_Impl, DCT_VendorC_Impl. After some considerations, Service Manager chooses DCT_VendorC_Impl to be instantiate and linked to MPEG-2_Video_Decoder_Impl. The decision made by Service Manager in choosing the service depends on how it is implemented.

© ISO/IEC 2007 – 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

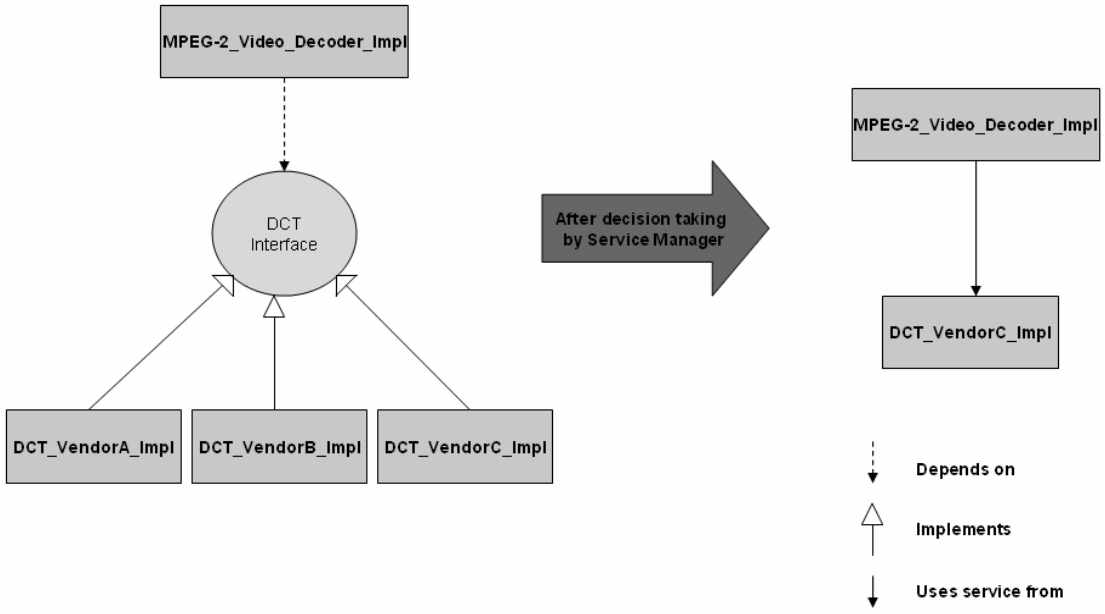


Figure 51 — Choosing service to be linked

Note that at a certain condition, implementation(s) of a required interface might not be available. So that Service Manager needs to download the implementation from somewhere else. In the case that the unavailable Service cannot be obtained, Service Manager cannot fully instantiate and configures the Service requested by client application. Hence, it has to return a message to client application saying that no service is available for the request.

Figure 52 — Sample of flowchart for Service Manager's actions, shows a simple flowchart that a Service Manager might use in configuring and instantiating a service. For efficiency reason, it is recommended that Service Manager makes a service instantiation plan prior by examining the metadata before taking the instantiation action. By doing this, Service Manager would know earlier if it cannot fulfil a request for a service and not wasting the middleware resource by instantiating something that cannot be successfully finalized.

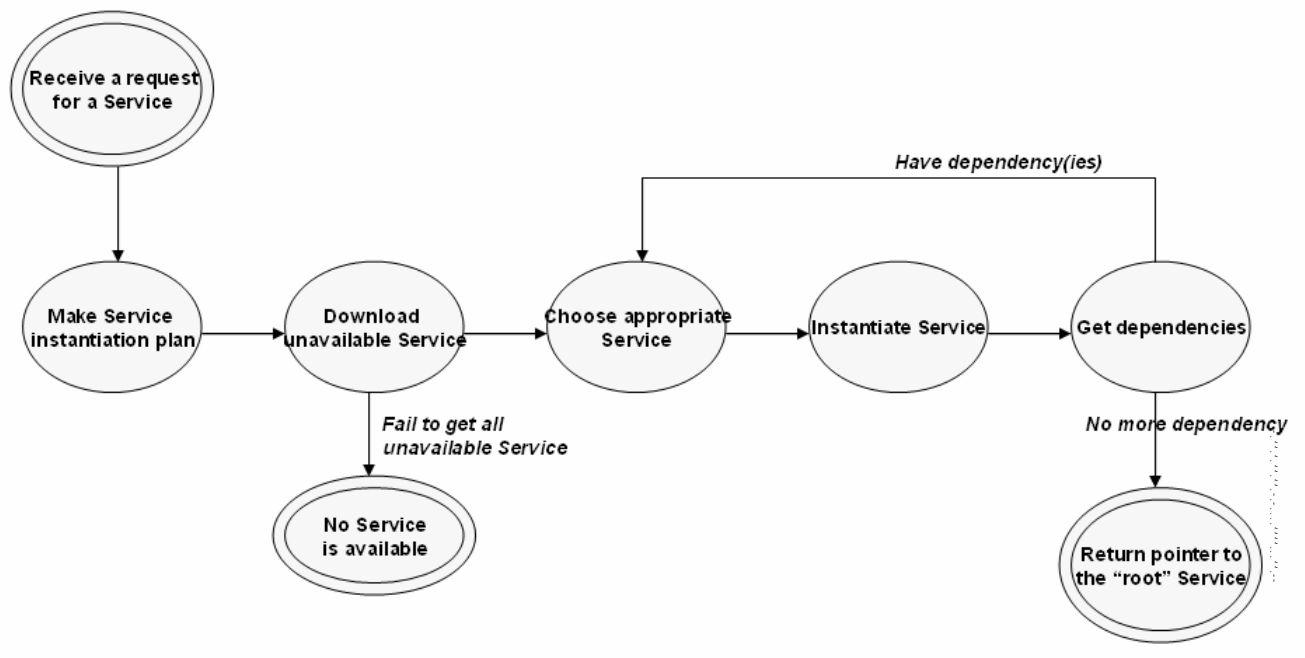


Figure 52 — Sample of flowchart for Service Manager's actions

© ISO/IEC 2007 – 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

11.6 Metadata

11.6.1 Logical Component metadata

11.6.1.1 Description

Logical Component Metadata describes the logical component. Figure 53 — Structure of logical component metadata, shows the hierarchical structure of logical component representation. The top element is a container that can carry a container of logical components and container for logical component record modification (add, remove, modify) history. A container of logical component can consist of zero to unbounded logical components. Respectively, the hierarchy of a logical component is as follow: Logical component – Role – Interface.

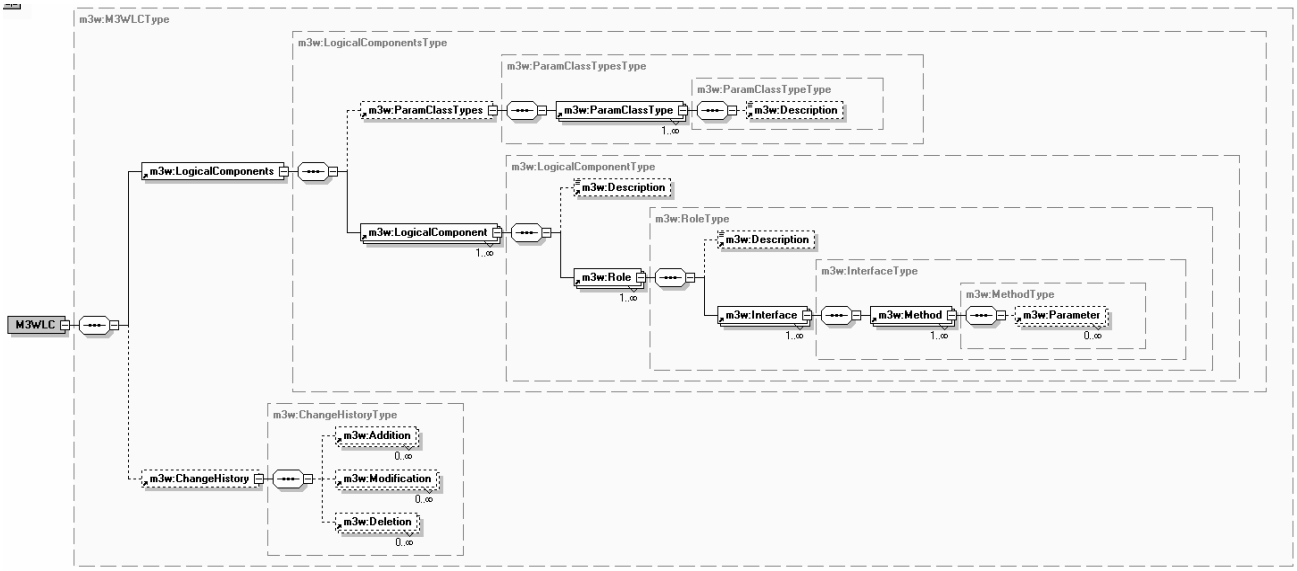


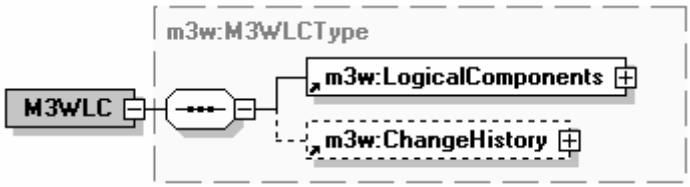
Figure 53 — Structure of logical component metadata

© ISO/IEC 2007 – 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

11.6.1.2 Syntax and semantics

11.6.1.2.1 M3WLC

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">-</td></tr>
<tr><td>Children</td><td colspan="4"><LogicalComponents> <ChangeHistory></td></tr>
<tr><td rowspan="4">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>version</td><td>String</td><td>Required</td><td>Version of the Logical Component modeled by this metadata.</td></tr>
<tr><td>date</td><td>date</td><td>Optional</td><td>Date of the release of the Logical Component that is modeled by this metadata</td></tr>
<tr><td>documentation</td><td>anyURI</td><td>Optional</td><td>Location where the specification can be obtained</td></tr>
<tr><td>Source</td><td colspan="4"><element name="M3WLC" type="m3w:M3WLCType"/>
<complexType name="M3WLSType">
<sequence>
<element ref="m3w:LogicalComponents"/>
<element ref="m3w:ChangeHistory" minOccurs="0"/>
</sequence>
<attribute name="version" type="String" use="required"/>
<attribute name="date" type="date" use="optional"/>
<attribute name="documentation" type="anyURI" use="optional"/>
</complexType></td></tr>
</table>

Semantics

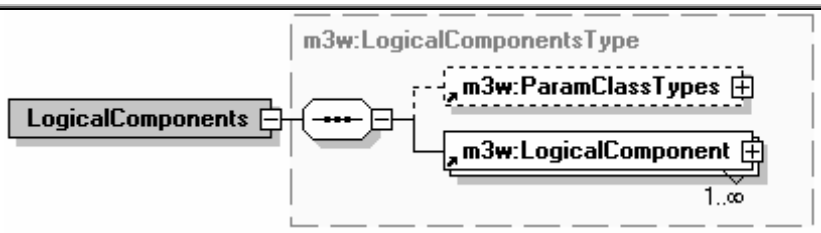
M3WLS is the root of metadata for describing the logical component of M3W.

© ISO/IEC 2007 – 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

11.6.1.2.2 LogicalComponents

Syntax

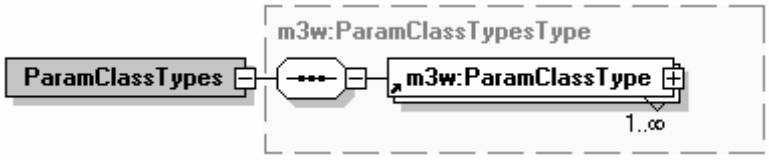
Diagram	
Used by	M3WLS
Children	<ParamClassTypes> <LogicalComponent>
Source	<element name="LogicalComponents" type="m3w:LogicalComponentsType"/> <complexType name=" LogicalComponentsType "> <sequence> <element ref="m3w:ParamClassTypes" minOccurs="0"/> <element ref="m3w:LogicalComponent " maxOccurs="unbounded"/> </sequence> </complexType>

Semantics

LogicalComponents is a container to carry the LogicalComponent

11.6.1.2.3 ParamClassTypes

Syntax

Diagram	
Used by	LogicalComponents
Children	<ParamClassType>
Source	<element name="ParamClassTypes" type="m3w: ParamClassTypesType"/> <complexType name=" ParamClassTypesType "> <sequence> <element ref="m3w:ParamClassType" maxOccurs="unbounded"/> </sequence> </complexType>

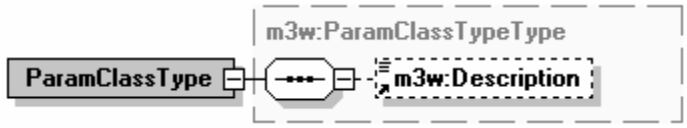
Semantics

ParamClassTypes is a container to carry the ParamClassType

© ISO/IEC 2007 – 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

11.6.1.2.4 ParamClassType

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">ParamClassTypes</td></tr>
<tr><td>Children</td><td colspan="4"><Description></td></tr>
<tr><td rowspan="2">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>name</td><td>String</td><td>Required</td><td>Name of the object. This object is used as the parameter type of the Method.</td></tr>
<tr><td>Source</td><td colspan="4"><element name="ParamClassType" type="m3w: ParamClassTypeType"/>
<complexType name="ParamClassType">
<sequence>
<element ref="m3w:Description" minOccurs="0"/>
</sequence>
<attribute name="name" type="String" use="required"/>
</complexType></td></tr>
</table>

Semantics

ParamClassType describes the class type of the Method's parameter. It describes other than primitive object types. For primitive object types definition such as Integer, String, Double, etc, they follow XMLSchema primitive data types definition.

11.6.1.2.5 Description

Syntax

<table>
<tr><td>Diagram</td><td> 
</td></tr>
<tr><td>Used by</td><td>MMObject, MMService</td></tr>
<tr><td>Children</td><td>-</td></tr>
<tr><td>Source</td><td><element name="Description" type="String"/></td></tr>
</table>

Semantics

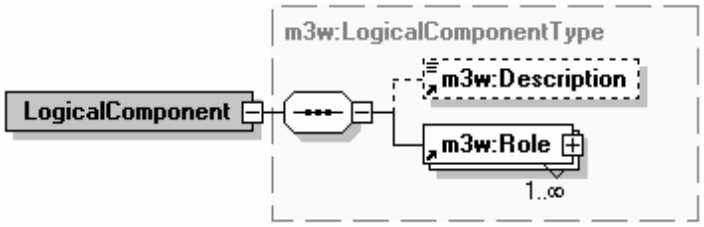
Description carries text description to describe its root element.

© ISO/IEC 2007 – 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

11.6.1.2.6 LogicalComponent

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">LogicalComponents</td></tr>
<tr><td>Children</td><td colspan="4"><Description><Role></td></tr>
<tr><td rowspan="4">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>id</td><td>String</td><td>Required</td><td>The identification of the logical component.</td></tr>
<tr><td>name</td><td>String</td><td>Required</td><td>Name of the logical component (i.e., MPEG-IPMP)</td></tr>
<tr><td>isStandard</td><td>boolean</td><td>optional</td><td>Describes whether this logical component is a M3W standard logical component or not. The default value of this attribute is ‘true’.</td></tr>
<tr><td>Source</td><td colspan="4"><element name="LogicalComponent" type="m3w:LogicalComponentType"/>
<complexType name="LogicalComponentType">
<sequence>
<element ref="m3w:Description" minOccurs="0"/>
<element ref="m3w:Role” maxOccurs="unbounded"/>
</sequence>
<attribute name="id" type="String" use="required"/>
<attribute name="name" type="String" use="required"/>
<attribute name="isStandard" type="boolean" use="optional" default="true"/>
</complexType></td></tr>
</table>

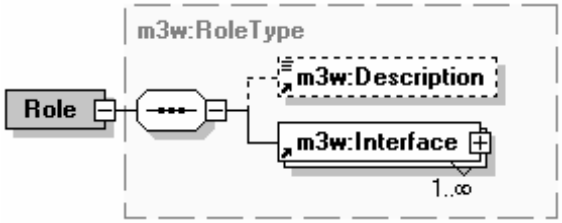
Semantics

LogicalComponent describes a logical component. A LogicalComponent contains a set of roles

© ISO/IEC 2007 – 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

11.6.1.2.7 Role

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">LogicalComponent</td></tr>
<tr><td>Children</td><td colspan="4"><Description><Interface></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>id</td><td>String</td><td>Required</td><td>The identification of the role.</td></tr>
<tr><td>name</td><td>String</td><td>Required</td><td>Name of the logical component (i.e., MPEG-IPMP)</td></tr>
<tr><td>Source</td><td colspan="4"><element name="Role " type="m3w:RoleType"/>
<complexType name="RoleType">
<sequence>
<element ref="m3w:Description" minOccurs="0"/>
<element ref="m3w:Interface” maxOccurs="unbounded"/>
</sequence>
<attribute name="id" type="String" use="required"/>
<attribute name="name" type="String" use="required"/>
</complexType></td></tr>
</table>

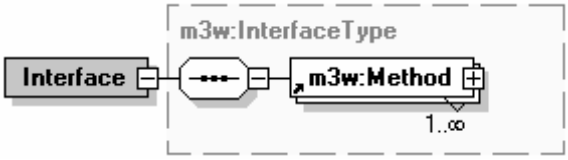
Semantics

Role describes a role. A Role contains a set of interfaces

© ISO/IEC 2007 – 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

11.6.1.2.8 Interface

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">LogicalComponent</td></tr>
<tr><td>Children</td><td colspan="4"><Method></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>id</td><td>String</td><td>Required</td><td>The identification of the interface.</td></tr>
<tr><td>name</td><td>String</td><td>Required</td><td>Name of the Interface</td></tr>
<tr><td>Source</td><td colspan="4"><element name="Interface" type="m3w:InterfaceType"/>
<complexType name="InterfaceType">
<sequence>
<element ref="m3w:Method" maxOccurs="unbounded"/>
</sequence>
<attribute name="id" type="String" use="required"/>
<attribute name="name" type="String" use="required"/>
</complexType></td></tr>
</table>

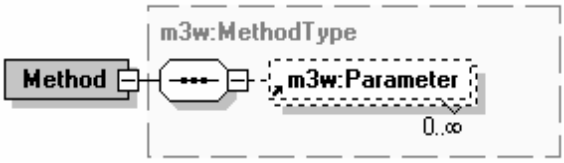
Semantics

Interface is a container to carry the list of methods

© ISO/IEC 2007 – 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

11.6.1.2.9 Method

Syntax

<table>
<tr><td>Diagram</td><td colspan="4"> 
</td></tr>
<tr><td>Used by</td><td colspan="4">Interface</td></tr>
<tr><td>Children</td><td colspan="4"><Parameter></td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>name</td><td>String</td><td>Required</td><td>Name of the API</td></tr>
<tr><td>output</td><td>String</td><td>Required</td><td>Output type of this API. i.e., integer, float, Boolean, etc. If this API does not output any data then it should be noted as ‘void’</td></tr>
<tr><td>Source</td><td colspan="4"><element name="Method" type="m3w:MethodType"/>
<complexType name="MethodType">
<sequence>
<element ref="m3w:Parameter" minOccurs="0" maxOccurs="unbounded"/>
</sequence>
<attribute name="name" type="String" use="required"/>
<attribute name="output" type="String" use="required"/>
</complexType></td></tr>
</table>

Semantics

Method describes a multimedia API. A method has name, output data type (‘void’ if it does not output any data), and sequence of parameters. The order of parameter in the implementation shall follow the order of the parameter describes by this element.

© ISO/IEC 2007 – 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

11.6.1.2.10 Parameter

Syntax

Diagram				
Used by	Method			
Children	-			
Attributes	**Name**	**Type**	**Use**	**Semantics**
	name	String	Required	Name of the parameter.
	type	String	required	Data type of this parameter. For primitive data type such as integer, String, float, double, etc this element follows the primitive data type defined in XMLSchema. For complex class data type, the class/object must be defined first in the ParamClassType.
	optional	boolean	Optional	Describes whether this parameter is optional or not.
Source	<element name="Parameter" type="m3w:ParameterType"/> <complexType name="ParameterType"> <attribute name="name" type="String" use="required"/> <attribute name="type" type="String" use="required"/> <attribute name="optional" type="boolean" use="optional" default="false"/> </complexType>			

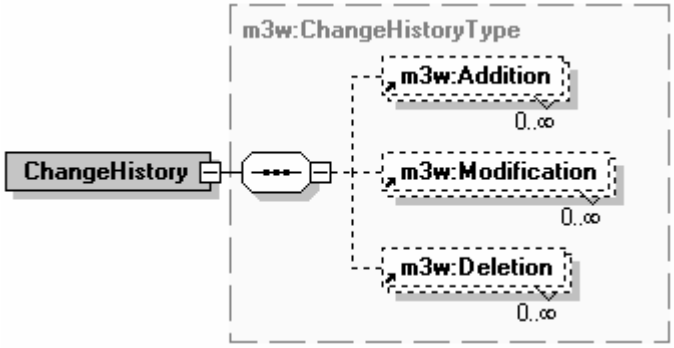
Semantics

Parameter describes parameter of an API.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

11.6.1.2.11 ChangeHistory

Syntax

Diagram				
Used by	M3WLC			
Children	<Addition> <Modification> <Deletion>			
Attributes	**Name**	**Type**	**Use**	**Semantics**
	base_version	String	Required	Describe the previous version of logical component to which this changes are applied
	Date	Date	Optional	Describe the date where these changes are applied and/or published.
Source	<element name="ChangeHistory" type="m3w:ChangeHistoryType"/> <complexType name="ChangeHistoryType"> <sequence> <element ref="m3w:Addition" minOccurs="0" maxOccurs="unbounded"/> <element ref="m3w:Modification" minOccurs="0" maxOccurs="unbounded"/> <element ref="m3w:Deletion" minOccurs="0" maxOccurs="unbounded"/> </sequence> <attribute name="base_version" type="String" use="required"/> <attribute name="date" type="date" use="optional"/> </complexType>			

Semantics

ChangeHistory describes the changes that have been applied. The recorded changes include addition, modification, and deletion of ParamClassType, MMService, and Method.

The changes that are described by ChangeHistory element are changes that are applied to the earlier version of MMAPIs whose version described by “base_version” attribute. The date when changes are applied and published is described by ‘date’ attribute.

© ISO/IEC 2007 – 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

11.6.1.2.12 Addition

Syntax

Diagram				
Used by	ChangeHistory			
Children	-			
Attributes	**Name**	**Type**	**Use**	**Semantics**
	object	DataTypeType	Required	Describe to which object type this addition was applied. The object type includes ParamClassType, LogicalComponent, and Method.
	name	String	Required	Describe the name of the object that was added.
Source	<simpleType name="DataTypeType"> <restriction base="String"> <enumeration value="ParamClassType"/> <enumeration value="LogicalComponent"/> <enumeration value="Method"/> </restriction> </simpleType> <element name="Addition" type="m3w:DataChangeType"/> <complexType name="DataChangeType"> <attribute name="object" type="m3w:DataTypeType" use="required"/> <attribute name="name" type="String" use="required"/> </complexType>			

Semantics

Addition describes an addition of either ParamClassType, MMService, or Method that was applied to the this version of logical service from the previous version.

If the addition (and also other type of change such as modification, deletion) is applied to Method, the value of ‘name’ shall have form as follow: <MMService’s name> + “/” + <Method’s name>

Informative:

Suppose that a Method called “getKey” is added to an MMService called “IPMPService” then the value of ‘name’ attribute is “IPMPService/getKey”.

© ISO/IEC 2007 – 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

11.6.1.2.13 Modification

Syntax

<table>
<tr><td>Diagram</td><td colspan="4">  </td></tr>
<tr><td>Used by</td><td colspan="4">ChangeHistory</td></tr>
<tr><td>Children</td><td colspan="4">-</td></tr>
<tr><td rowspan="3">Attributes</td><td>Name</td><td>Type</td><td>Use</td><td>Semantics</td></tr>
<tr><td>Object</td><td>DataTypeType</td><td>Required</td><td>Describe to which object type this modification was applied. The object type includes ParamClassType, LogicalComponent, and Method.</td></tr>
<tr><td>Name</td><td>String</td><td>Required</td><td>Describe the name of the object that was modified.</td></tr>
<tr><td>Source</td><td colspan="4"><pre>
<simpleType name="DataTypeType">
    <restriction base="String">
        <enumeration value="ParamClassType"/>
        <enumeration value="LogicalComponent"/>
        <enumeration value="Method"/>
    </restriction>
</simpleType>

<element name="Modification" type="m3w:DataChangeType"/>
<complexType name="DataChangeType">
    <attribute name="object" type="m3w:DataTypeType" use="required"/>
    <attribute name="name" type="String" use="required"/>
</complexType>
</pre></td></tr>
</table>

Semantics

Modification describes a modification that is applied to of either ParamClassType, MMService, or Method to the this version of logical service from the previous version.

© ISO/IEC 2007 – 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

11.6.1.2.14 Deletion

Syntax

Diagram				
Used by	ChangeHistory			
Children	-			
Attributes	**Name**	**Type**	**Use**	**Semantics**
	Object	DataTypeType	Required	Describe to which object type this deletion was applied. The object type includes ParamClassType, LogicalComponent, and Method.
	Name	String	Required	Describe the name of the object that was deleted.
Source	<simpleType name="DataTypeType"> <restriction base="String"> <enumeration value="ParamClassType"/> <enumeration value="LogicalComponent"/> <enumeration value="Method"/> </restriction> </simpleType> <element name="Deletion" type="m3w:DataChangeType"/> <complexType name="DataChangeType"> <attribute name="object" type="m3w:DataTypeType" use="required"/> <attribute name="name" type="String" use="required"/> </complexType>			

Semantics

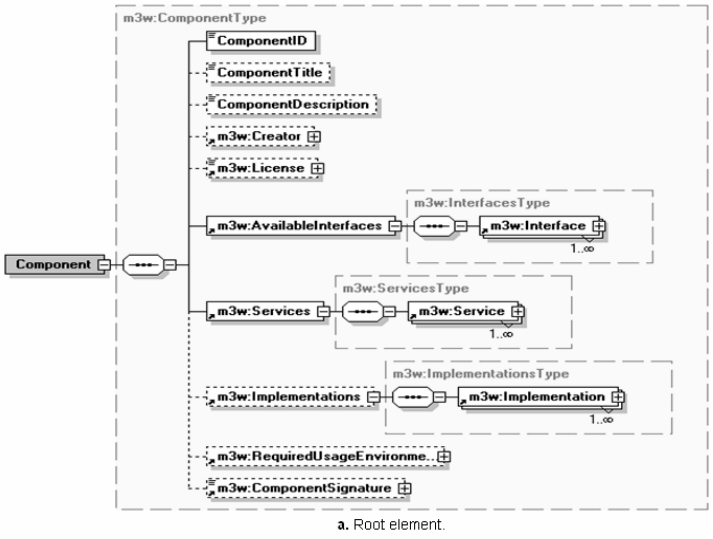
Deletion describes either ParamClassType, MMService, or Method that was deleted in this version of logical service from the previous version.

11.6.2 Service metadata

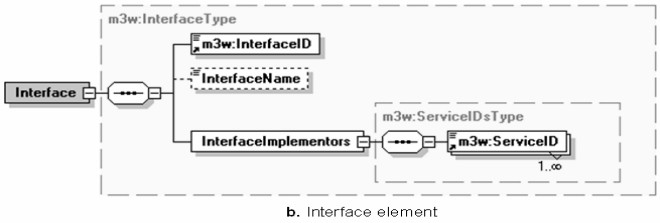
11.6.2.1 Description

Service metadata provides a representation or description for middleware component (container of services) and service. It is a deployment unit which is delivered to the Service Manager to be stored so that Service Manager can draw relationship between the service and the logical component (through interface). Figure 54 — Metadata for component & Service description, shows the structure of the service metadata.

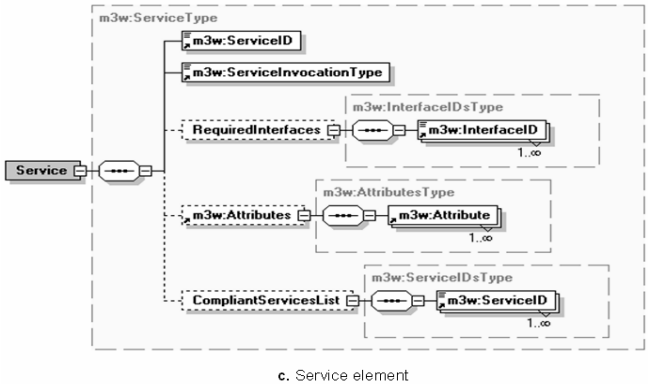
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale



a. Root element.



b. Interface element



c. Service element

Figure 54 — Metadata for component & Service description

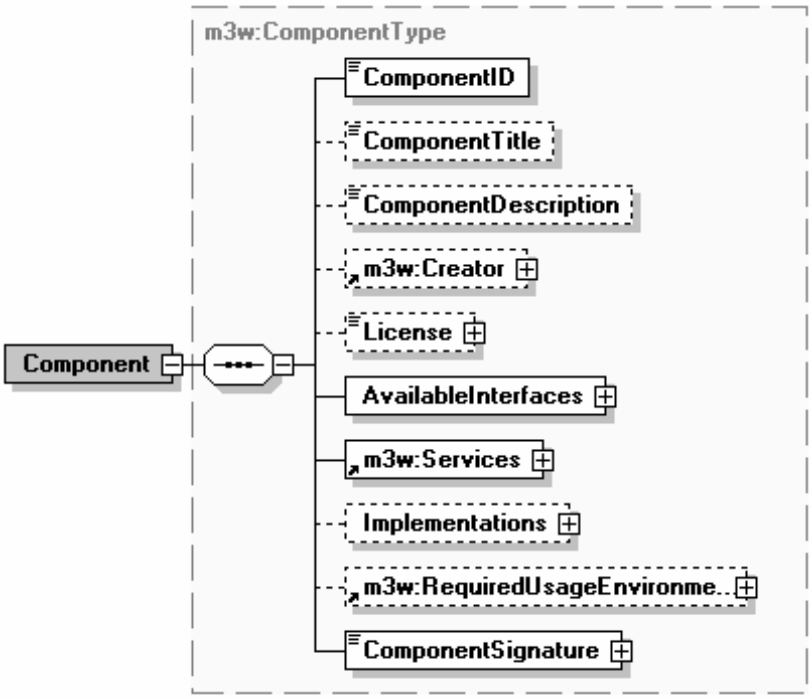
© ISO/IEC 2007 – All rights reserved

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

11.6.2.2 Syntax and Semantics

11.6.2.2.1 Component

Syntax

Diagram	
Children	<ComponentID> <ComponentTitle> <ComponentDescription> <Creator> <License> <AvailableInterfaces> <Services> <Implementations> <RequiredUsageEnvironment> <ComponentSignature>
Source	<element name="Component" type="m3w:ComponentType"/> <complexType name="ComponentType"> <sequence> <element name="ComponentID" type="String"/> <element name="ComponentTitle" type="String" minOccurs="0"/> <element name="ComponentDescription" type="String" minOccurs="0"/> <element ref="m3w:Creator" minOccurs="0"/> <element ref="m3w:License" minOccurs="0"/> <element ref=" m3w:AvailableInterfaces"/> <element ref="m3w:Services"/> <element name="Implementations" type="m3w:ImplementationsType" minOccurs="0"/> <element ref="m3w:RequiredUsageEnvironments" minOccurs="0"/> <element name="ComponentSignature" type="m3w:ComponentSignatureType"/> </sequence> </complexType>

Semantics

The Component element is the container for carrying information to describe a Component.

© ISO/IEC 2007 – 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

11.6.2.2.2 ComponentID

Syntax

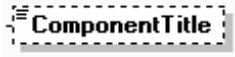
Diagram	
Children	-
Source	<element name="ComponentID" type="String"/>

Semantics

The ComponentID element describes the identification of the Component.

11.6.2.2.3 ComponentTitle

Syntax

Diagram	
Children	-
Source	<element name="ComponentTitle" type="String" minOccurs="0"/>

Semantics

The ComponentTitle element describes the title of the Component. The existence of this element is optional.

11.6.2.2.4 ComponentDescription

Syntax

Diagram	
Children	-
Source	<element name="ComponentDescription" type="String" minOccurs="0"/>

Semantics

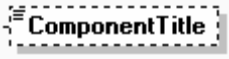
The ComponentDescription element describes the description of the Component. The existence of this element is optional.

© ISO/IEC 2007 – 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

11.6.2.2.5 Creator

Syntax

Diagram	
Children	<CreatorName> <CreatorPostalAddress> <CreatorWebsite> <CreatorEmail>
Source	<element name="Creator" type="m3w:CreatorType"/> <complexType name="CreatorType"> <sequence> <element name="CreatorName" type="String"/> <element name="CreatorPostalAddress" type="String" minOccurs="0"/> <element name="CreatorWebsite" type="String" minOccurs="0"/> <element name="CreatorEmail" type="String" minOccurs="0"/> </sequence> </complexType>

Semantics

The Creator element describes the information about Component's creator.

11.6.2.2.6 CreatorName

Syntax

Diagram	
Children	-
Source	<element name="CreatorName" type="String"/>

Semantics

The CreatorName element describes the name of the creator or developer of the Component.

11.6.2.2.7 CreatorPostalAddress

Syntax

Diagram	
Children	-
Source	<element name="CreatorPostalAddress" type="String" minOccurs="0"/>

© ISO/IEC 2007 – 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

Semantics

The CreatorPostalAddress element describes the postal address of the creator or developer of the Component.

11.6.2.2.8 CreatorWebsite

Syntax

Diagram	
Children	-
Source	<element name="CreatorWebsite" type="String" minOccurs="0"/>

Semantics

The CreatorName element describes the website address of the creator or developer of the Component.

11.6.2.2.9 CreatorEmail

Syntax

Diagram	
Children	-
Source	<element name="CreatorEmail" type="String" minOccurs="0"/>

Semantics

The CreatorName element describes the email address of the creator or developer of the Component.

11.6.2.2.10 License

Syntax

Diagram	
Children	-
Source	<element name="License" type="m3w:ComponentLicenseType"/> <complexType name="ComponentLicenseType" mixed="true"> <sequence> <any namespace="##any" processContents="lax" minOccurs="0"/> </sequence> </complexType>

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

© ISO/IEC 2007 – All rights reserved

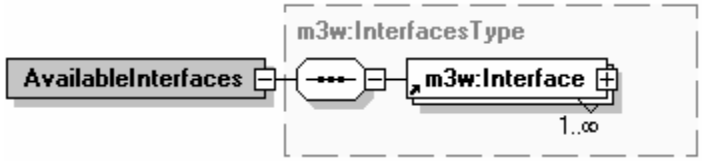
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Semantics

The License element is the container for carrying license information that governs the usage of the Component. The existence of License information is optional in the Component description. The non-existence of License information can be interpreted as Component is license-free.

11.6.2.2.11 AvailableInterfaces

Syntax

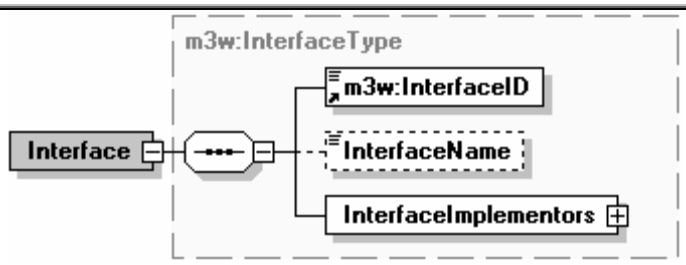
Diagram	
Children	<Interface>
Source	<element name="AvailableInterfaces" type="m3w:InterfacesType"/> <complexType name="InterfacesType"> <sequence> <element ref="m3w:Interface" maxOccurs="unbounded"/> </sequence> </complexType>

Semantics

The AvailableInterfaces element is the container for carrying list of interfaces that the Component carries.

11.6.2.2.12 Interface

Syntax

Diagram	
Children	<InterfaceID> <InterfaceName> <InterfaceImplementors>
Source	<element name="Interface" type="m3w:InterfaceType"/> <complexType name="InterfaceType"> <sequence> <element ref="m3w:InterfaceID"/> <element name="InterfaceName" type="String" minOccurs="0"/> <element name="InterfaceImplementors" type="m3w:ServiceIDsType"/> </sequence> </complexType>

© ISO/IEC 2007 – 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

Semantics

The Interface element describes the interface that is exposed by the Component. The standard technical specification of the interface is represented in the Interface Definition Language (IDL). The Interface element is linked to its IDL representation via InterfaceID.

11.6.2.2.13 InterfaceID

Syntax

Diagram	
Children	-
Source	<element name="InterfaceID" type="String"/>

Semantics

The InterfaceID element describes unique identification of the interface.

11.6.2.2.14 InterfaceName

Syntax

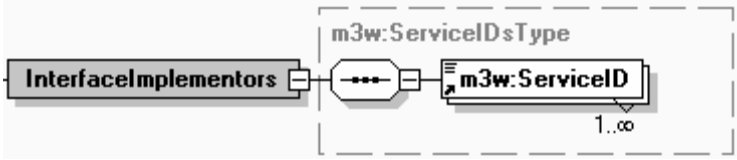
Diagram	
Children	-
Source	<element name="InterfaceName" type="String" minOccurs="0"/>

Semantics

The InterfaceName element describes name of the interface. The existence of interface's name is optional.

11.6.2.2.15 InterfaceImplementors

Syntax

Diagram	
Children	<ServiceID>
Source	<element name="InterfaceImplementors" type="m3w:ServiceIDsType"/>

Semantics

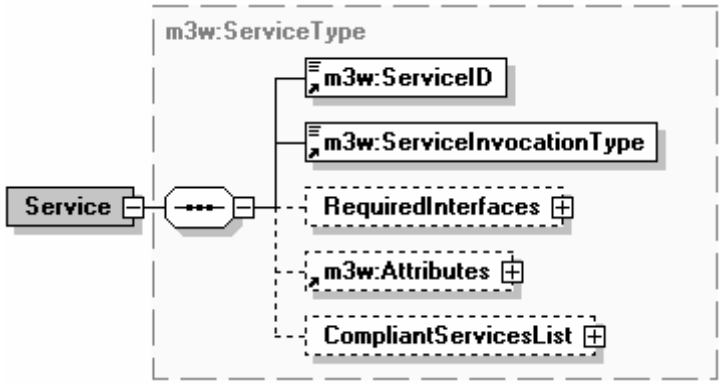
The InterfaceName element describes name of the interface. The existence of interface's name is optional.

© ISO/IEC 2007 – 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

11.6.2.2.16 Service

Syntax

Diagram	
Children	<ServiceID> <ServiceInvocationType> <RequiredInterfaces> <Attributes> <CompliantServicesList>
Source	<element name="Service" type="m3w:ServiceType"/> <complexType name="ServiceType"> <sequence> <element ref="m3w:ServiceID"/> <element ref="m3w:ServiceInvocationType"/> <element name="RequiredInterfaces" type="m3w:InterfaceIDsType" minOccurs="0"/> <element ref="m3w:Attributes" minOccurs="0"/> <element name="CompliantServicesList" type="m3w:ServiceIDsType" minOccurs="0"/> </sequence> </complexType>

Semantics

The Service element describes the service that the Component has.

11.6.2.2.17 ServiceID

Syntax

Diagram	
Children	-
Source	<element name="ServiceID" type="String"/>

Semantics

The ServiceID element describes unique identification of the interface.

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2007 – All rights reserved
Not for Resale

11.6.2.2.18 ServiceInvocationType

Syntax

Diagram	
Children	-
Source	<element name="ServiceInvocationType" type="m3w:ServiceInvocationTypeType"/> <simpleType name="ServiceInvocationTypeType"> <restriction base="String"> <enumeration value="in-process"/> <enumeration value="cross-process"/> <enumeration value="remote"/> </restriction> </simpleType>

Semantics

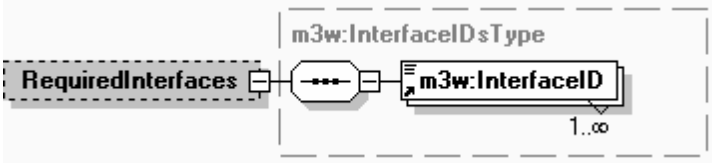
The ServiceInvocationType element describes the location where the process of the service will take place relative to the caller application process. The process of a service can take place only in three places, they are: in-process; cross-process; and remote. A service is said to be in-process if its process runs within the same application space as the caller application. A service is said to be cross-process if its process and caller application's process are in the separated application spaces. If the process of a service is located in different application space and at different machine from that of the caller application then it is called remote.

Informative

The implementation of M3W runtime can have its own consideration to choose which type of service invocation type is more preferred in the case there are two services with different invocation type that implement the requested interface. For example, it can simply decides to return service that is invoked cross-process rather that the one that is remote because the cross-process service may perform faster and consume less network resource.

11.6.2.2.19 RequiredInterfaces

Syntax

Diagram	
Children	<InterfaceID>
Source	<element name="RequiredInterfaces" type="m3w:InterfaceIDsType" minOccurs="0"/>

Semantics

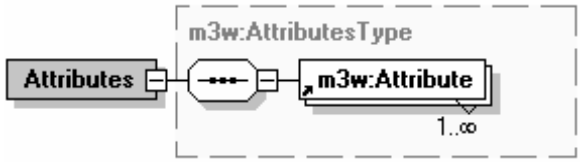
The RequiredInterfaces element is the container for carrying interfaceIDs that is/are required by the service.

© ISO/IEC 2007 – 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

11.6.2.2.20 Attributes

Syntax

Diagram	
Children	<Attribute>
Source	<element name="Attributes" type="m3w:AttributesType"/> <complexType name="AttributesType"> <sequence> <element ref="m3w:Attribute" maxOccurs="unbounded"/> </sequence> </complexType>

Semantics

The Attributes element is the container for carrying attributes information of the service.

11.6.2.2.21 Attribute

Syntax

Diagram			
Children	-		
Attributes	**Name**	**Type**	**Description**
	attrName	String	Describes the name of the attribute
	attrType	String	Describes the type of the attribute. i.e., integer, String, etc.
	defaultValue	String	An optional attribute that describes the default value of the attribute.
Source	<element name="Attribute" type="m3w:AttributeType"/> <complexType name="AttributeType"> <attribute name="attrName" type="String" use="required"/> <attribute name="attrType" type="String" use="required"/> <attribute name="defaultValue" type="String" use="optional"/> </complexType>		

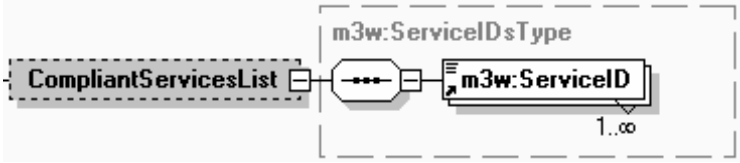
Semantics

The Attribute element describes the attribute of the service. Attribute element is an empty element.

© ISO/IEC 2007 – 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

11.6.2.2.22 CompliantServiceList

Syntax

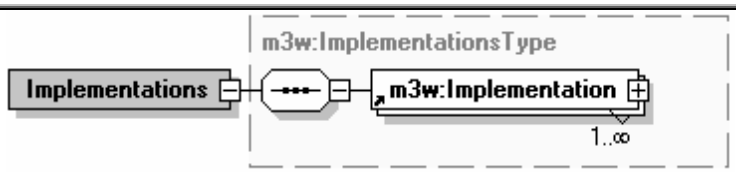
Diagram	
Children	<ServiceID>
Source	<element name="CompliantServicesList" type="m3w:ServiceIDsType" minOccurs="0"/>

Semantics

The CompliantServicesList element is the container for carrying serviceIDs that is/are compliant to the service that carries the list. The compliant service can be used as replacement of this service.

11.6.2.2.23 Implementations

Syntax

Diagram	
Children	<Implementation>
Source	<element name="Implementations" type="m3w:ImplementationsType"/> <complexType name="ImplementationsType"> <sequence> <element ref="m3w:Implementation" maxOccurs="unbounded"/> </sequence> </complexType>

Semantics

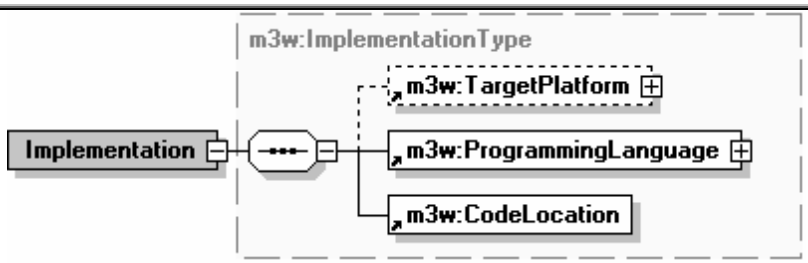
A Component can come to the end-user terminal in a package of multiple implementations for different target environments. The Implementations element is the container for carrying list of implementations of the Component. The RRE can choose which implementation be activate when the service/interface resides in the Component is requested.

© ISO/IEC 2007 – 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

11.6.2.2.24 Implementation

Syntax

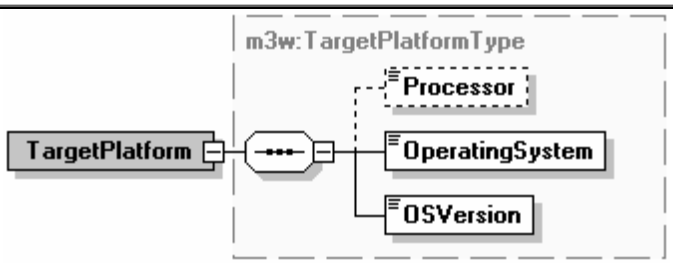
Diagram	
Children	<TargetPlatform> <ProgrammingLanguage> <CodeLocation>
Source	<element name="Implementation" type="m3w:ImplementationType"/> <complexType name="ImplementationType"> <sequence> <element ref="m3w:TargetPlatform" minOccurs="0"/> <element ref="m3w:ProgrammingLanguage"/> <element ref="m3w:CodeLocation"/> </sequence> </complexType>

Semantics

The Implementation element describes the implementation environment of the Component. When a Component needs to be activated, the runtime of M3W can use the implementation description to choose which implementation of the Component to be used.

11.6.2.2.25 TargetPlatform

Syntax

Diagram	
Children	<Processor> <OperatingSystem> <OSVersion>
Source	<element name="TargetPlatform" type="m3w:TargetPlatformType"/> <complexType name="TargetPlatformType"> <sequence> <element name="Processor" type="m3w:ProcessorType" minOccurs="0"/> <element name="OperatingSystem" type="m3w:OSType"/> <element name="OSVersion" type="String"/> </sequence> </complexType>

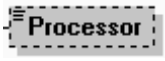
© ISO/IEC 2007 – All rights reserved
Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
Not for Resale

Semantics

The TargetPlatform element describes the target platform of the implementation of the Component.

11.6.2.2.26 Processor

Syntax

Diagram	
Children	-
Source	<element name="Processor" type="m3w:ProcessorType" minOccurs="0"/> <simpleType name="ProcessorType"> <restriction base="String"> <enumeration value="x86"/> <enumeration value="mips"/> <enumeration value="alpha"/> <enumeration value="ppc"/> <enumeration value="sparc"/> <enumeration value="680x0"/> <enumeration value="vax"/> <enumeration value="AS/400"/> <enumeration value="S/390"/> <enumeration value="Other"/> </restriction> </simpleType>

Semantics

The Processor element describes the processor type that is suitable to run the implementation of the Component. The existence of Processor element is optional.

11.6.2.2.27 OperatingSystem

Syntax

Diagram	
Children	-
Source	<element name="OperatingSystem" type="m3w:OSType"/> <simpleType name="OSType"> <restriction base="String"> <enumeration value="AIX"/> <enumeration value="BSDi"/> <enumeration value="VMS"/> <enumeration value="Unix"/> <enumeration value="DOS"/> <enumeration value="HPBLS"/> <enumeration value="HPUX"/> <enumeration value="IRIX"/> <enumeration value="Linux"/> <enumeration value="MacOS"/> <enumeration value="OS/2"/>

© ISO/IEC 2007 – 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

	<enumeration value="AS/400"/> <enumeration value="MVS"/> <enumeration value="Solaris"/> <enumeration value="SunOS"/> <enumeration value="VxWorks"/> <enumeration value="Windows"/> <enumeration value="Other"/> </restriction> </simpleType>

Semantics

The OperatingSystem element describes the operating system that is suitable to run the implementation of the Component.

11.6.2.2.28 OSVersion

Syntax

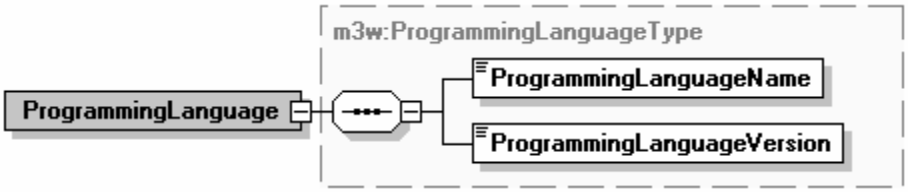
Diagram	
Children	-
Source	<element name="OSVersion" type="String"/>

Semantics

The OSVersion element describes the version of the operating system that is suitable to run the implementation of the Component.

11.6.2.2.29 ProgrammingLanguage

Syntax

Diagram	
Children	<ProgrammingLanguageName> < ProgrammingLanguageVersion >
Source	<element name="ProgrammingLanguage" type="m3w:ProgrammingLanguageType"/> <complexType name="ProgrammingLanguageType"> <sequence> <element name="ProgrammingLanguageName" type="String"/> <element name="ProgrammingLanguageVersion" type="String"/> </sequence> </complexType>

Copyright International Organization for Standardization
Provided by IHS under license with ISO
No reproduction or networking permitted without license from IHS
© ISO/IEC 2007 – All rights reserved
Not for Resale

Semantics

The ProgrammingLanguage element describes the programming language that was used for the implementation.

11.6.2.2.30 ProgrammingLanguageName

Syntax

Diagram	
Children	-
Source	<element name="ProgrammingLanguageName" type="String"/>

Semantics

The ProgrammingLanguageName element describes the programming language name that was used for the implementation.

11.6.2.2.31 ProgrammingLanguageVersion

Syntax

Diagram	
Children	-
Source	<element name="ProgrammingLanguageVersion" type="String"/>

Semantics

The ProgrammingLanguageName element describes the version programming language that was used for the implementation.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

11.6.2.2.32 CodeLocation

Syntax

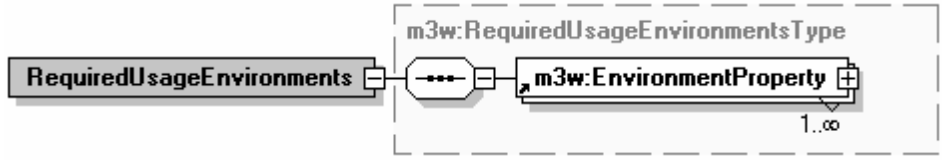
Diagram			
Children	-		
Attributes	**Name**	**Type**	**Description**
	isBuild-In	boolean	Describes whether the implementation is build-in in the Component package or not
	ref	anyURI	Describes location where the implementation can be found.
Source	<element name="CodeLocation" type="m3w:CodeLocationType"/> <complexType name="CodeLocationType"> <attribute name="isBuild-In" type="boolean" use="required"/> <attribute name="ref" type="anyURI" use="optional"/> </complexType>		

Semantics

The CodeLocation element describes the location where the code of the implementation of the Component can be found.

11.6.2.2.33 RequiredUsageEnvironments

Syntax

Diagram	
Children	<EnvironmentProperty>
Source	<element name="RequiredUsageEnvironments"> type="m3w:RequiredUsageEnvironmentsType"/> <complexType name="RequiredUsageEnvironmentsType"> <sequence> <element ref="m3w:EnvironmentProperty" maxOccurs="unbounded"/> </sequence> </complexType>

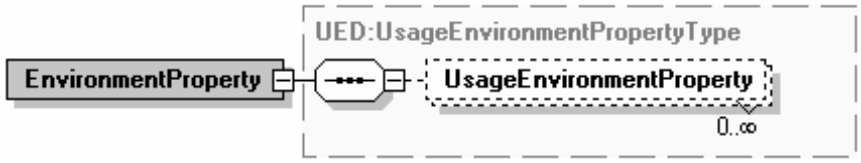
Semantics

The RequiredUsageEnvironments element is container to carry the information about the expected terminal/environment conditions are needed to use this Component.

© ISO/IEC 2007 – 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

11.6.2.2.34 EnvironmentProperty

Syntax

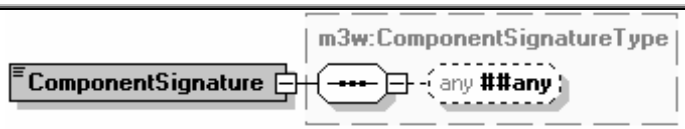
Diagram	
Children	
Source	<element name="EnvironmentProperty" type="UED:UsageEnvironmentPropertyType"/>

Semantics

The EnvironmentProperty element describes the required terminal/environment property to use the Component. It adopts the Usage Environment Description data type from ISO/IEC 21000-7 (MPEG-21 Part 7: Digital Item Adaptation).

11.6.2.2.35 ComponentSignature

Syntax

Diagram	
Children	
Source	<element name="ComponentSignature" type="m3w:ComponentSignatureType"/> <complexType name="ComponentSignatureType" mixed="true"> <sequence> <any namespace="##any" processContents="lax" minOccurs="0"/> </sequence> </complexType>

Semantics

The ComponentSignature element is container to carry the signature that guarantees the integrity of the Component description.

12 Remote Method Invoker

12.1 Overview

M3W manages the execution of services deployed in components. These services are used by means of late binding, i.e. a service and a requester are bound only when the latter actually needs to execute the former.

By default configuration this mechanism uses only local components, i.e. those deployed on the device and previously registered to the local runtime process.

© ISO/IEC 2007 – 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

Inside the M3W two logical components have been included as optional. The benefit is to have the possibility to issue request to service instance that are located in other M3W Systems by means of network messages. This opens to sharing of functionalities, to distributed computing for fault tolerance/load balancing purposes.

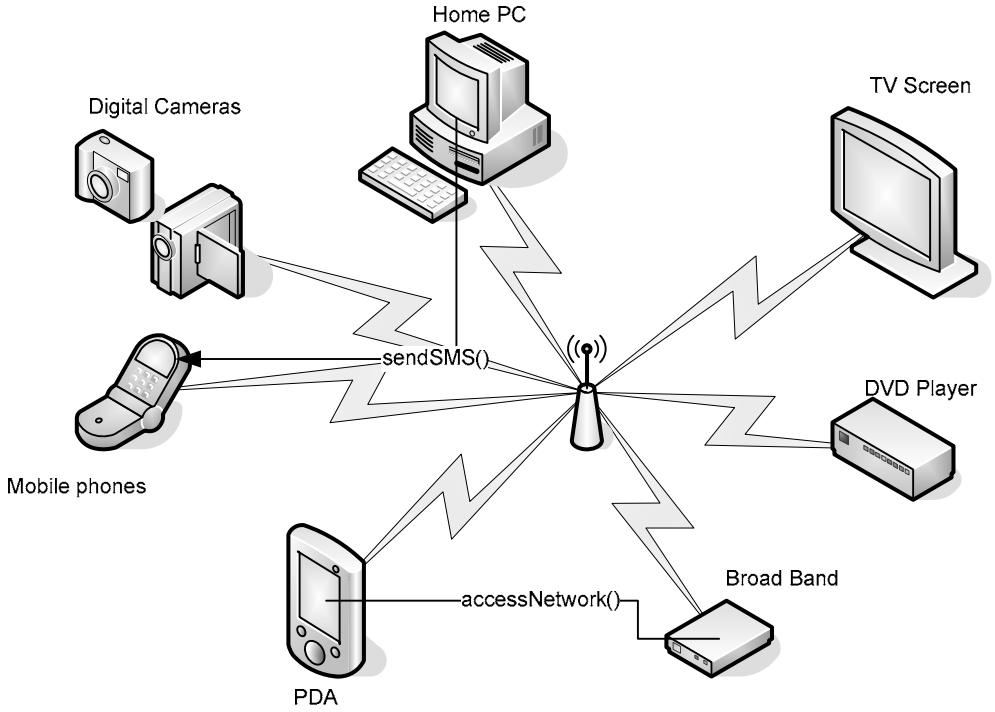


Figure 55 — Sharing of M3W functionalities among devices at home

New generation of consumer electronic applications can be realized in order to exploit/manage the functionality that are discovered inside the distributed environment (e.g. home wi-fi network).

The high-level exploitation is based on a low-level control of the services which are exposed by any M3W System in a distributed environment. A logical view of applications and functionalities interacting with other functionalities not necessarily located in the local system is depicted in the following figure.

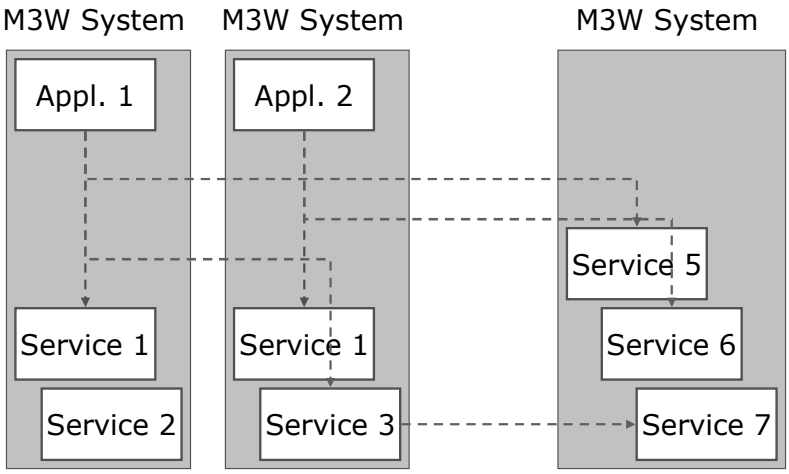


Figure 56 — Use of service by client code: local ore remote instances

© ISO/IEC 2007 – 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

In order to achieve this objective, a stack of communication layers is needed. The logical components which have to be at disposal to allow enabling the transparent execution of remote component services are resumed in the following:

— a remote requester (REMI-R) that allows client code to issue method execution requests to instances (new or existing) to the other M3W Systems connected to the network.

— a remote provider (REMI-P) that is able to accept and to handle method requests by redirecting them to the proper “active” instances. It is also capable to respond about service availability, and with services/interfaces metadata.

— server and client for Remote Procedure Call (RPC).

Please note, that since these logical components are not mandatory, several device configurations for Remote Capabilities are allowed (e.g. “only-client” devices can be configured by providing an implementation for the REMI-R and the RPC client logical components). In order to allow a “complete” solution for Remote Capabilities exploitation both REMI-R and REMI-P, with the corresponding RPC components, are recommended. The REMI-P is needed when client code needs to use a remote service passing as argument an interface reference to a local instance; in this case the REMI-P has to be configured in order to handle the method requests addressing the passed reference.

12.2 Proxies and wrappers

An additional layer in order to allow the maximum reuse of existing client code exploiting local service instance can be introduced. The proxy concept can reproduce the local instance behaviour by implementing the required interfaces, while it actually uses REMI-R interfaces in order to produce remote instance method invocations.

On the other side the existing implementation code needs to be reused as well, by allowing REMI-P request handler implementation exploiting existing interface reference method calls. With this objective we defined a wrapper concept in order to intercept and to decode “by name” interface method invocations directed to a specific instance. This wrapper is responsible for redirecting such requests to the proper method calls on the “real” interface reference implementing the required functionality.

It has to be clarified that Proxy and Wrapper concept are optional for the implementation, anyway we consider a good way in order to realize REMI-R and REMI-P functionalities and to allow maximum reuse of the existing client code which has been developed for local usage of services.

Figure 57 — Proxy and Wrapper concepts, depicts the main relations among proxies, wrappers and services.

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

© ISO/IEC 2007 – 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

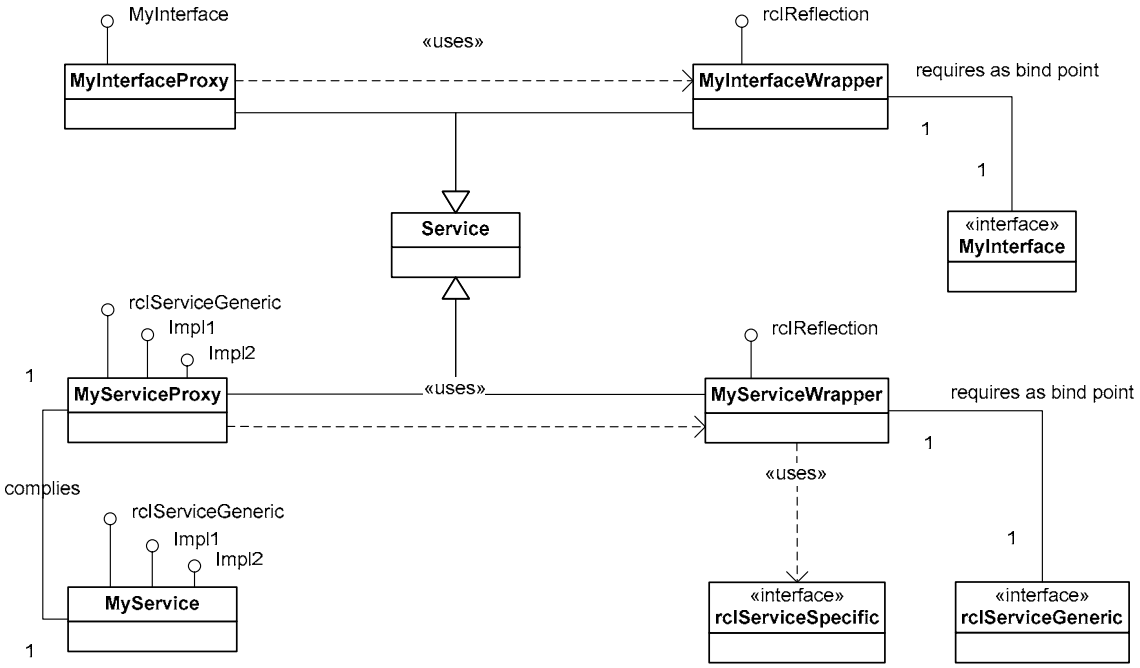


Figure 57 — Proxy and Wrapper concepts

Please note that both Proxy and Wrapper are Services for the M3W. The different roles that are played by proxy and wrapper have been separated for interface and service since they can model a generic interface or an interface which is well-known by the M3W component model: rclServiceGeneric interface. They have to be Services in order to be registered to the M3W, associated to related entities and instantiated by the client code at runtime.

MyInterfaceProxy can be used by client code using MyInterface without any change, the implementation has to hide the low-level method calls issued to a REMI-R service instance.

MyInterfaceWrapper is a dual entity, even if it is not mandatory for implementing REMI-P service, we consider it as a good implementation choice. It has to expose a generic interface that is known by the REMI-P for invoking “by name” the methods of MyInterface. The implementation has to decode the generic invocation redirecting to the wrapped interface reference. A generic interface to expose this capability has been called rclReflection.

Proxies and Wrappers for services have the same semantic meaning of those defined for interfaces, while they model a more complex concept in the M3W: the service. Basically they are put directly in relation with the modelled service. MyServiceProxy has been defined as a compliant service of MyService: this means that it has to implement all the required interface, including rclServiceGeneric and has to provide the same. The “complies” concept is helpful for allowing transparent activation of proxies instead of real service instances when compliance has been registered to the M3W. MyServiceProxy also implements all the known relations among interfaces and it can implements locally *queryInterface* invocations, while using the REMI-R only for functional calls to a MyService instance which is active in a remote M3W System. MyServiceWrapper can be considered as a rclServiceGenericWrapper, while it is implemented to use the specific interface set of MyService. Indeed, it requires as bind point to “a generic service” interface reference (rclServiceGeneric).

A general implementation issue which is shared by all proxies and wrappers is related to the parameter marshalling and un-marshalling, since they use REMI-R (the proxies) and they are used by REMI-P (the wrappers) and both of these logical components deal only with a general purpose marshal type (rclParam) which is able to read/write any type of signature argument. In the parameter marshalling/un-marshalling task, proxies and wrappers behave like dual entities. Figure 58 — Dealing with parameters in a Proxy and Figure 59 — Dealing with parameters in a Wrapper show the basic steps in transforming a specific method call in a generic request (by the suitable proxy) and then obtaining back a specific method call from a generic request (by the suitable wrapper).

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 2007 – All rights reserved

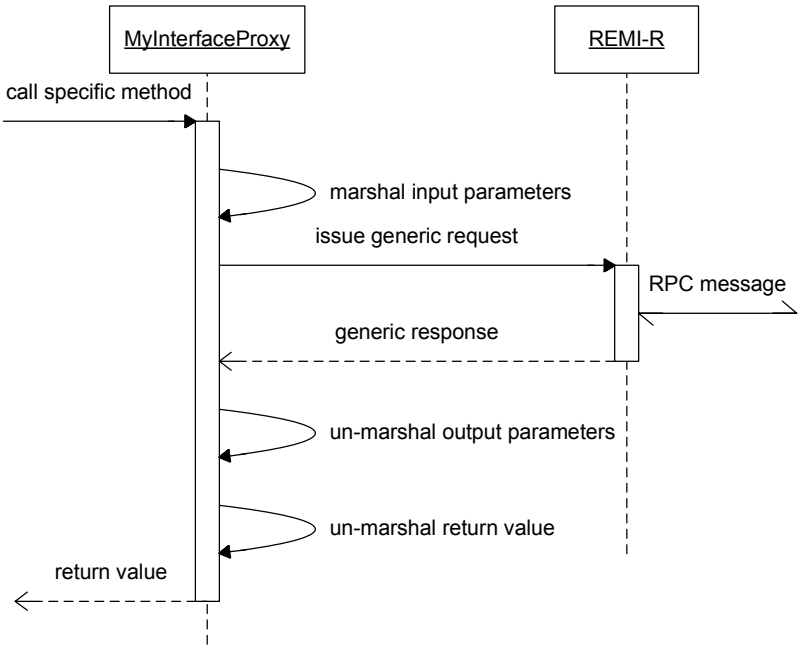


Figure 58 — Dealing with parameters in a Proxy

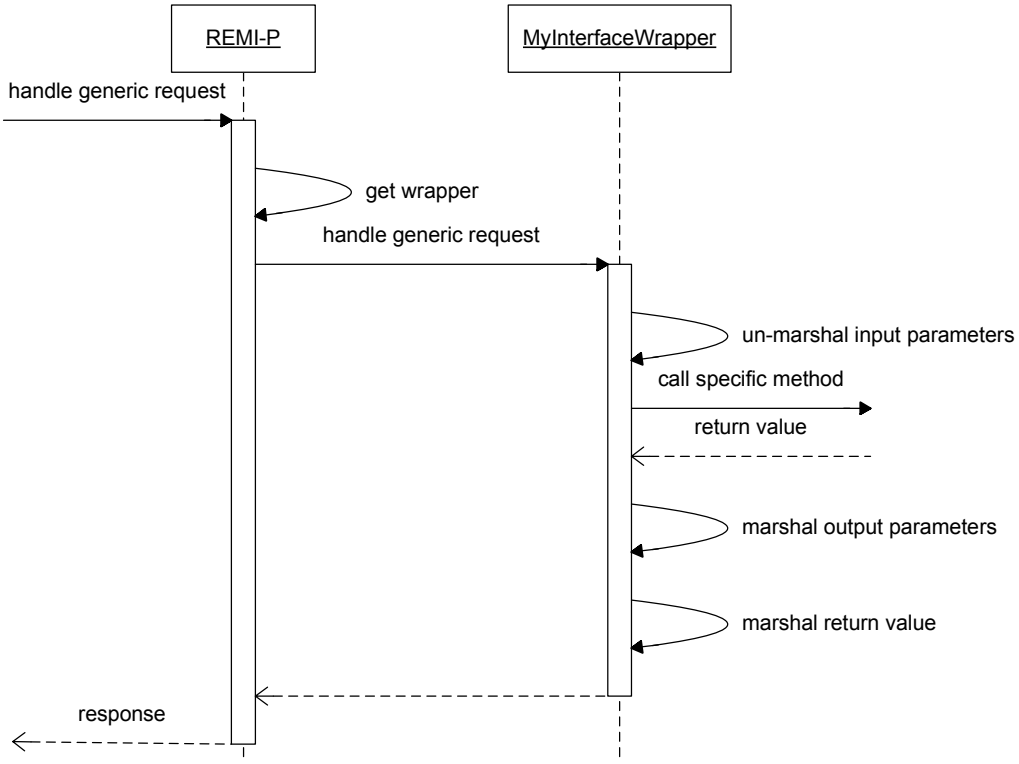


Figure 59 — Dealing with parameters in a Wrapper

Please note that common marshalling actions have to be taken by both intermediate entities, while they occurs in the opposite order. This sequence diagram put also in evidence the duality between REMI-R and REMI-P.

© ISO/IEC 2007 – 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

12.3 Marshalling parameters

In transporting a specific method call to a unique communication channel (the RPC infrastructure) a common marshalling process is needed. It has to support several kinds of argument types including basic/standard types, arrays and user-defined types. Furthermore, the proposed objective of reusing existing client code imposes that also interface references should be considered by the marshalling technology.

An interface hierarchy has been designed in order to cope with any parameter type. Figure 60 — Interface for marshalling parameters depicts all the proposed interfaces related to parameter marshalling.

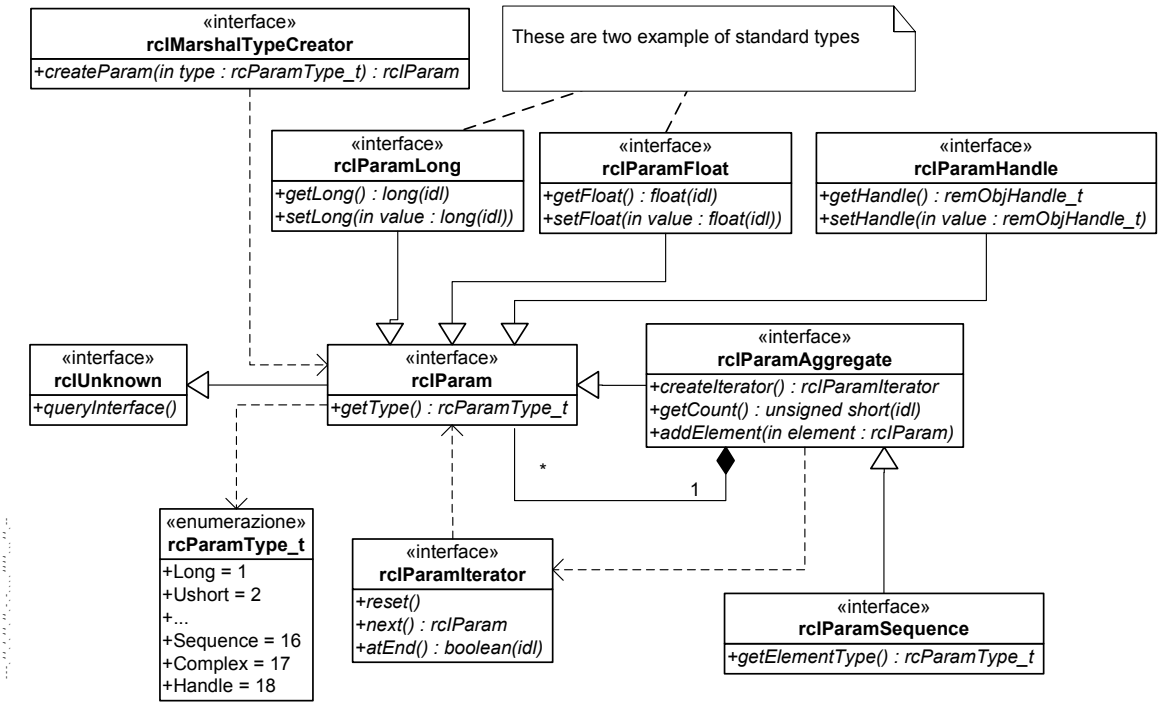


Figure 60 — Interface for marshalling parameters

Please note that all the basic/standard types defined by M3W have been included.

User-defined types and arrays can be marshalled and un-marshalled by using rclParamAggregate and rclParamSequence interfaces. To have an idea about the marshalling support a client code example has been summarized in Figure 61 — Marshalling example. This sequence represents the required steps in order to marshal three parameters (reported in C-like notation):

```
long mylong;
float myfloats[10];
struct astruct { short myshort; string mystring } s;
```


© ISO/IEC 2007 – 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

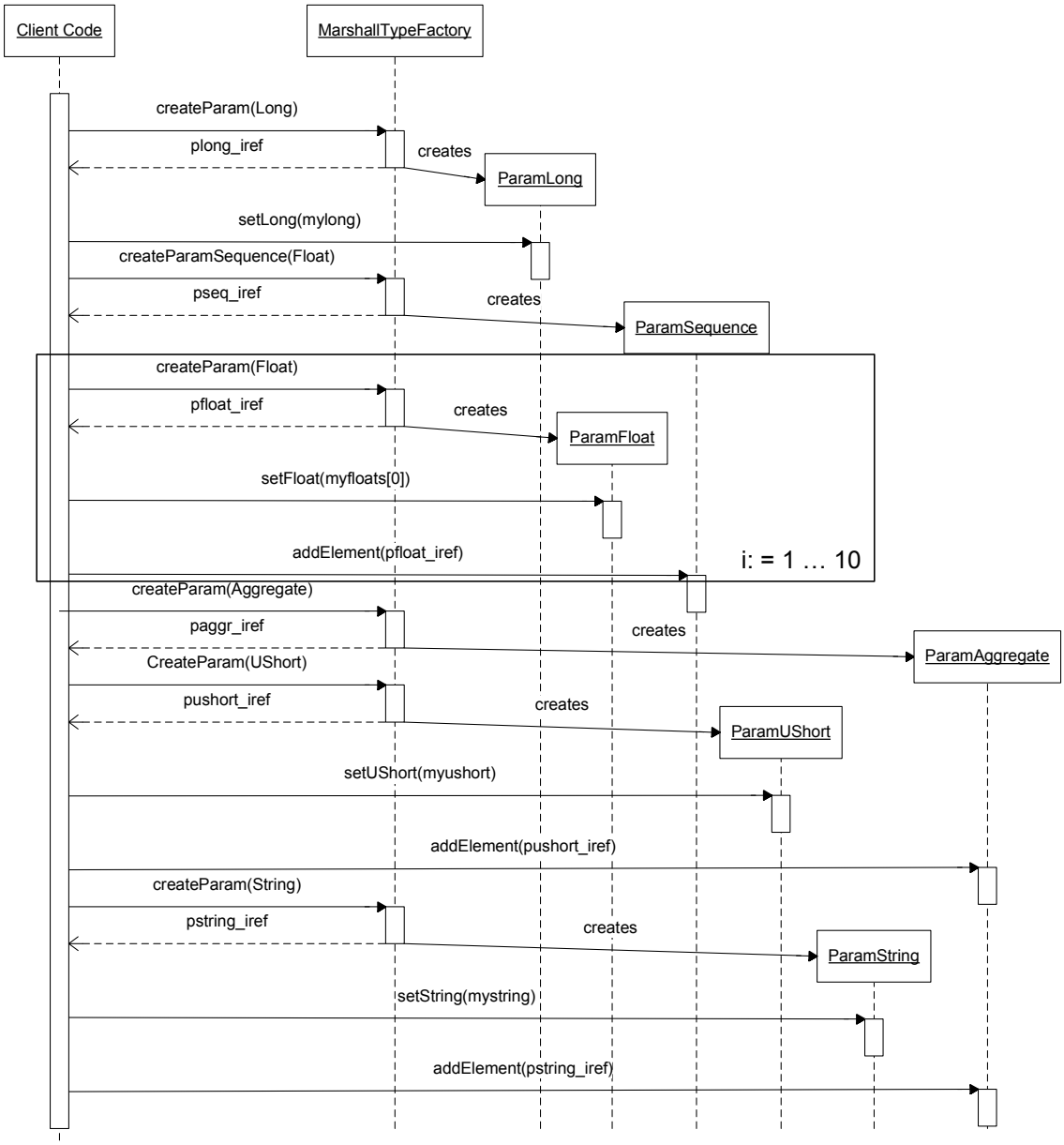


Figure 61 — Marshalling example

Marshalling of interface reference is realized by performing different actions w.r.t. the standard marshalling process. It has to allow a correct and transparent usage of an interface reference which came from a remote method invocation. In this process Proxy and Wrapper becomes much more important.

On the invocation side, passing an interface reference means to marshal all the information that is required by the remote entity to use the local interface reference. Besides this interface reference has to be ready to accept method invocation requests from a remote M3W System.

On the handling side, receiving an interface reference, means to un-marshal the information regarding how to reach the remote instance. On the basis of this information a proxy has to be created and passed to the method execution on the target instance.

The previous example has to be generalized even if interface references are expected as output parameters (or return value) of a method execution. In Figure 62 — Marshalling an interface reference and Figure 63 — Un-marshalling an interface reference, the generic marshalling and un-marshalling process for interface references are respectively shown.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

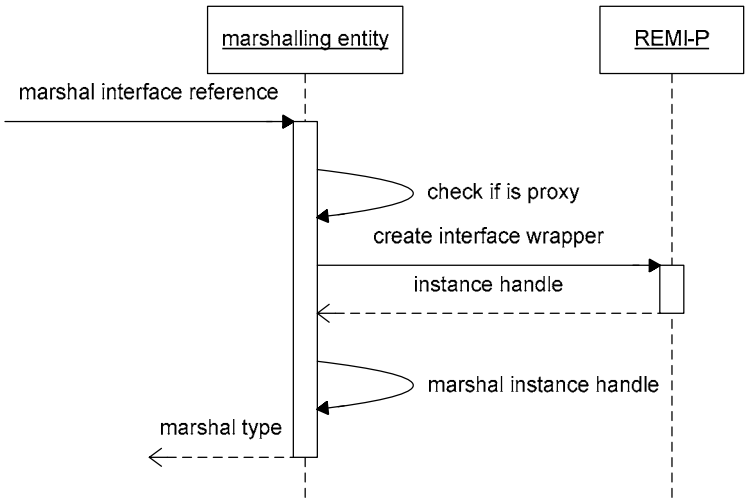


Figure 62 — Marshalling an interface reference

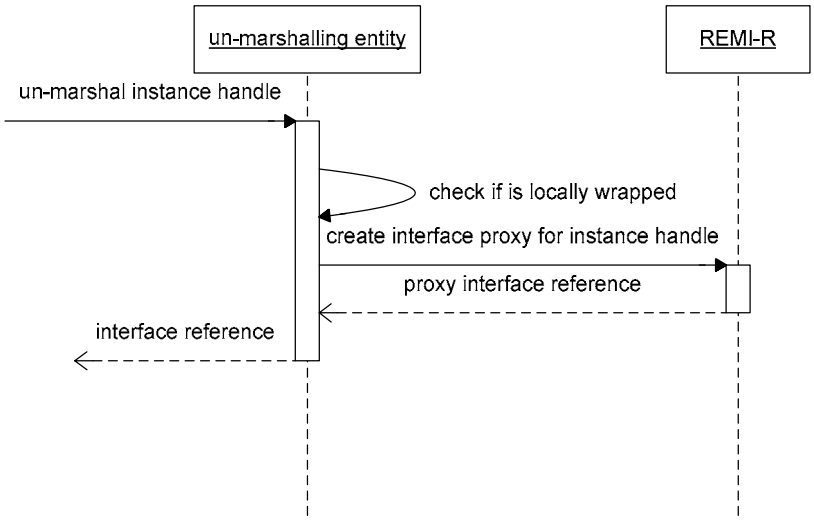


Figure 63 — Un-marshalling an interface reference

During the marshalling process, it has to be checked if the interface reference identifies an interface proxy. If this is the case, it means the reference is to a remote instance, so no local wrapper is created, rather information needed to determine location and instance (i.e. the handle) is directly marshalled (please note that in this case the handle will straight identify the remote M3W System offering the system, in order to avoid useless network communications). Otherwise, it means that the interface reference identifies a local instance, so a wrapper for that instance has to be created before marshalling the handle.

During the un-marshalling process, a check is performed to verify if the handle received identifies a local instance wrapped to make it available to a remote M3W System. If this is the case, no proxy is created and the instance is directly used, in order to involve the minimum number of entities, thus optimizing performances. Otherwise, a proxy has to be created in order to let the local M3W system transparently exploit the remote instance identified by the handle received.

In case of marshalling an instance handle, which contains the information to retrieve the instance in the distributed environment, a specific interface has been defined because such a parameter is not considered as a user-defined, but a built-in type for exploiting Remote Capabilities.

© ISO/IEC 2007 – 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

In order to be implementation independent for marshalling technology, a logical component MarshalTypeFactory has been conceived. This implements rclMarshalTypeCreator which is able to return interface references for all the defined interfaces (i.e. rclParamLong, rclParamInt, rclParamAggregate, etc).

12.4 REMI-R logical component

The REMI-R logical component is the base building block for enabling remote method invocation. It exposes four interfaces as depicted in Figure 64 — REMI-R logical component:

— rclMethodExecutionInvoker: it provides functionalities to invoke method execution for existing or new instances located in the distributed environment; it provides means to request the creation such remote instance;

— rclProxyManager: it provides functionalities to generate/manage proxies for interfaces (i.e. existing implementations provided by services) or for services (i.e. new instances to be activated by other M3W Systems);

— rclProxyServiceManager: (optional) it provides functionality to control the associations among service/interface identifiers and related proxy services identifiers (e.g. UUID of MyService → UUID of MyServiceProxy);

— rclPeerManager: (optional) it provide functionalities to control the list of currently known M3W System in the network (in terms of URIs).

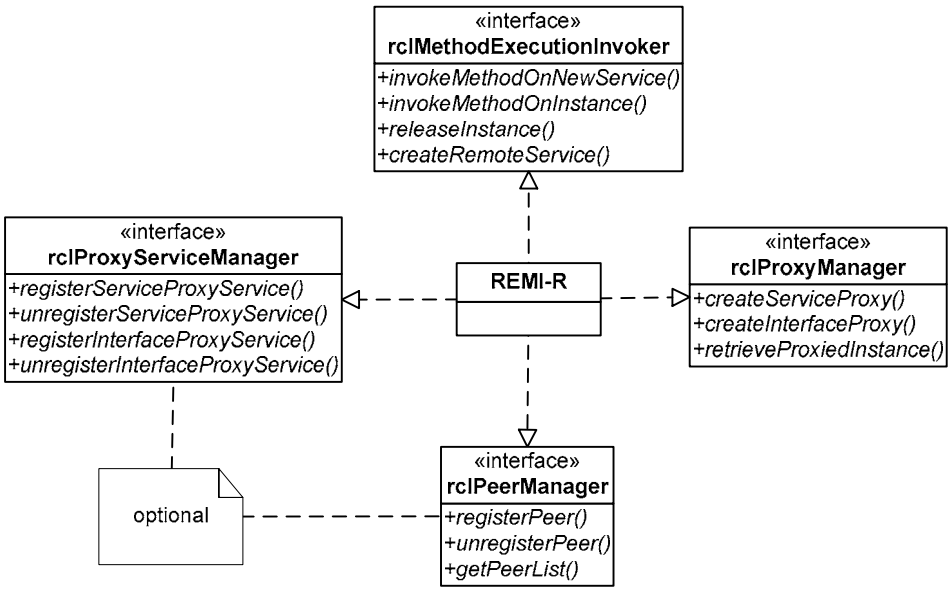


Figure 64 — REMI-R logical component

The main aspect to be considered is the capability of invoking method execution on remote instances. To realize this functionality the REMI-R is responsible for the following actions:

— retrieving the location of the requested instance;

— serialization of the input parameters (use of rclParam interface for reading each parameter type and value);

— use of the RPC protocol to transport the method invocation request;

© ISO/IEC 2007 – 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

— deserialization of the output parameters and the return value (use of rcIParam interface for writing each parameter type and value)

When client code uses *invokeMethodOnInstance*, these steps are required, while a variation is considered when a request is issued by using a service identifier by using *invokeMethodOnNewService*: in this case it requires the activation of a new service instance in the distributed environment, so requests for service availability and service information precede the invocation request. An instance handle is returned at the end of remote execution which enables to perform further calls to the same instance. Another use strategy could be to use *createRemoteService* in order to obtain an instance handle of an new active instance in the remote environment. Please note that both "activation" methods (i.e. the ones which are required to command an instance creation) can optionally restrict their invocation to only one M3W System. This helps in having more control than transparency when exploiting remote services.

Since instance life-cycle has to be managed, a release functionality has been provided: it commands to release the related instance in the remote M3W System.

Please note that a clinet code can choose among three different programming style of Remote Capabilities:

— by using directly the invoking methods exposes directly by the REMI-R: in that case it has all the resposibilities regarding marshalling and un-marshalling of parameters;

— by using the rcIProxyManager interface of REMI-R for creating a ServiceProxy for the required service and then using transparently proxy interface reference;

— by using the Runtime Environement service creation mechanism if a "complies" record has been registered to the M3W which associates a service identifier with its corresponding proxy service identifier: in this case the common service creation will create a proxy instance in place of the service instance, so the transparency is total.

Please rememeber that remote execution of service functionality is affected by network system availability so a successful invocation is not as much granted as in local environement direct calls. So, the third choice of remote exploitation is recommended to be used only in stable conditions of network, system and service reliability.

12.5 REMI-P logical component

The REMI-P logical component is the counter part of the REMI-R for handling method invocation request issued by the latter. The REMI-P can register itself to a RPC protocol server implementation, then the latter notifies when a given request has to be handled by the REMI-P. It exposes four interfaces as depicted in Figure 65 — REMI-P logical component:

— rcIMethodExecutionHandler: it provides functionalities to handle method execution for existing or new instances which are local to the M3W;

— rcIWrapperManager: it provides functionalities to generate/manage wrapper for interfaces or for services;

— rcIRemotableServiceRetriever: it provides functionality to inspect the list of service that are available for remote usage and to gather the related information;

— rcIRemotableServiceManager: (optional) it provides functionality to control the associations among service/interface identifiers and related wrapper services identifiers (e.g. UUID of MyService → UUID of MyServiceWrapper);

© ISO/IEC 2007 – 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

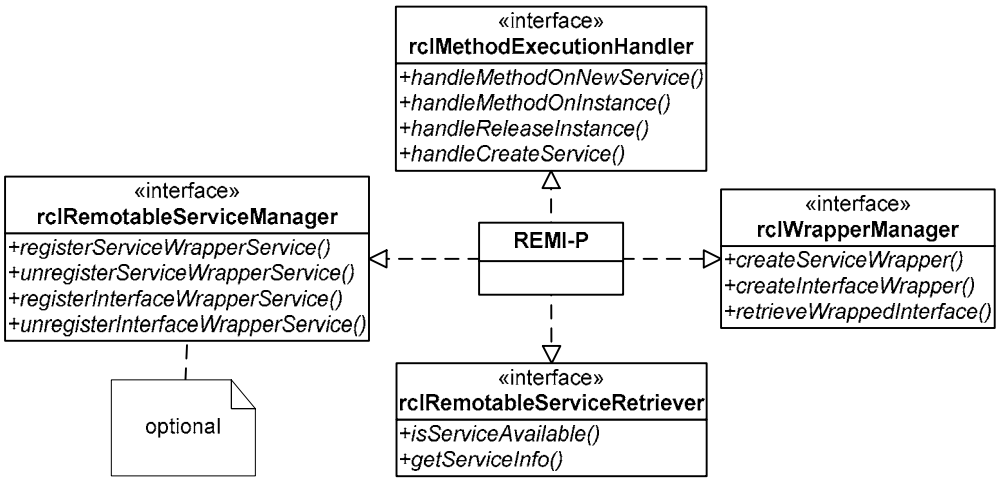


Figure 65 — REMI-P logical component

In a symmetric way w.r.t. REMI-R, the most important feature of REMI-P is to handle method invocation requests. To realize this functionality an action list is proposed:

— retrieving the Wrapper related to the requested instance;

— deserialization of the input parameters;

— “by name” invocation by using the rclReflection interface that is exposed by the Wrapper;

— serialization of the output parameters and the return value

Also in the REMI-P side, a request which targets a new instance, specifying service identifier, has to be handled. In this case the generation of a new Wrapper which has been associated to the service identifier is required; the instantiation of the service by the Runtime Environment and the binding are included in the Wrapper creation. In the same manner a REMI-P can serve a request that only requires for an instantiation of a service. Even in this case a wrapper has to be created and indexed in the local wrapper list in order to accept further request to the created instance.

The release handling uses the usual releasing method of local interface references.

12.6 Remotable service development support

From the development side, proxy and wrapper should to be produced together with the service definition. In the service definition all the required information to produce proxies and wrappers for the service and the related interfaces are present.

In Figure 66 — Required services for the deployment of a “remotable” service, all the required services to be implemented are depicted. Please note that all the implementation code for proxies and wrappers can be automatically generated by inspecting the IDL of service definition (and the related interfaces). This code solve issues like specific parameter marshalling and un-marshalling (including interface references) and “by name” request decoding.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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

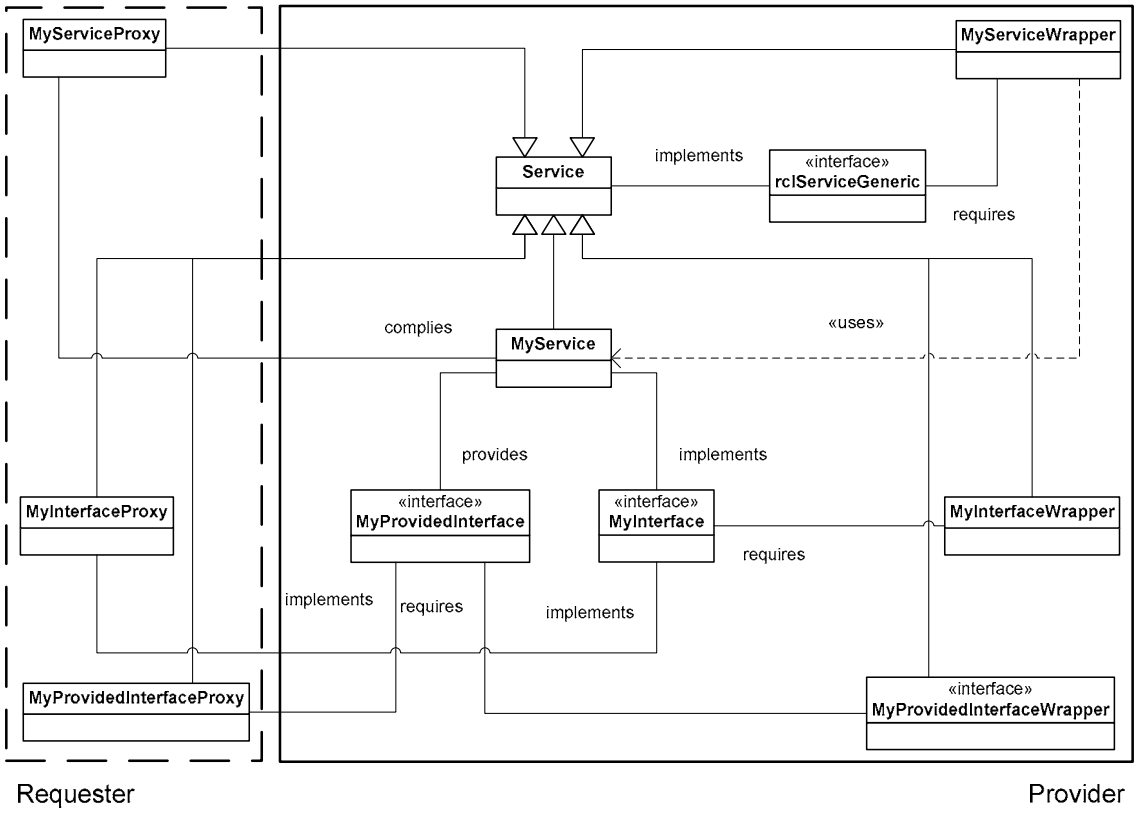


Figure 66 — Required services for the deployment of a “remotable” service

It has to be noted that, since this implementation choice is used, a service is ready for remote invocations if:

— The service identifier has been regitered to the M3W and it has been deployed in a registered local component;

— The related wrapper service has been registered to the M3W and it has been deployed in a registered local component;

— The service identifier and the related wrapper service identifier have been associated inside the REMI-P wrapperServiceRecord.

On the side of application developer exploiting remote service instance, a remote service can be used if:

— The related proxy service identifier has been registered to the M3W and it has been deployed in a registered local component;

— The service identifier and the related proxy service identifier have been associated inside the REMI-R proxyServiceRecord.

Additional requirements about which wrapper and proxy services have to be registered are stated by the interface reference input/output passing between requester code and provider implementation. For any interface expected to be issued to (received by) a remote invocation an interface wrapper service (an interface proxy service) has to be registered and associated on the client side. On the server side the dual condition is true, so if an interface reference is received as input parameter (is issued as output parameter) when handling a remote invocation request an interface proxy service (an interface wrapper service) has to be registered and associated to the interface identifier.

--`,,```,,,,````-`-`,,`,,`,`,,`---
© ISO/IEC 2007 – 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
(normative)

Service Manager interface definition

```
interface rcISMControl {4cb0ab70-1cf4-43ac-9697-a4e80fbb2d34} {
    /*
* path -> path where the metadata file exist
     * The metadata is in the form of XML representation
     *
     * Return true if the metadata is successfully registered, false otherwise
     *
     */
    rcResult registerServiceMetadata (in String path, out boolean retValue);


    /*
*
     * srvID -> The id of the service
     * Return true if the metadata is successfully unregistered, false otherwise
     */
    rcResult unregisterServiceMetadata (in pUUID srvID, out boolean retValue);

/*
* path -> path where the metadata file exist
     * The metadata is in the form of XML representation
     *
     * Return true if the metadata is successfully registered, false otherwise
     *
     */
    rcResult  registerLogicalComponentMetadata  (in  String  path,  out  boolean
retValue);

    /*
*
     * lcID -> The UUID of the service
     * Return true if the metadata is successfully unregistered, false otherwise
     */
    rcResult unregisterLogicalService (in pUUID lcID, out boolean retValue);
}

interface rcISMClient {4cb0ab70-1cf4-43ac-9697-a4e80fbb2d35} {
    /*
     * Check if there is service implementation for requested lcID.
     *
     * lcID -> The id of requested logical component
     *
     * Return true if there is installed service that implement
     * interfaces of this requested logical component, false otherwise.
     */
    rcResult isServiceAvailable (in pUUID lcID, out boolean retValue);

    /*
     * Get the handler to the service that implements a logical component.
```

© ISO/IEC 2007 – 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

```
     *
     * lcID -> The id of requested logical component
     *
     * Return the rcIService, NULL if there is no such service.
     */
    rcResult  getInstanceForLogicalComponent  (in  pUUID  lcID,  out  prcIService
*retValue);
}
```

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

© ISO/IEC 2007 – 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
(normative)

Logical Service metadata schema

```
<?xml version="1.0" encoding="UTF-8"?>
<schema targetNamespace="http://mpeg-m3w/MM_APIs"
xmlns="http://www.w3.org/2001/XMLSchema" xmlns:m3w="http://mpeg-m3w/MM_APIs"
elementFormDefault="qualified" attributeFormDefault="unqualified">
    <element name="M3WLC" type="m3w:M3WLCType"/>
    <complexType name="M3WLCType">
        <sequence>
            <element ref="m3w:LogicalComponents"/>
            <element ref="m3w:ChangeHistory" minOccurs="0"/>
        </sequence>
        <attribute name="version" type="String" use="required"/>
        <attribute name="date" type="date" use="optional"/>
        <attribute name="documentationRef" type="anyURI" use="optional"/>
    </complexType>
    <element name="LogicalComponents" type="m3w:LogicalComponentsType"/>
    <complexType name="LogicalComponentsType">
        <sequence>
            <element ref="m3w:ParamClassTypes" minOccurs="0"/>
            <element ref="m3w:LogicalComponent" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="ParamClassTypes" type="m3w:ParamClassTypesType"/>
    <complexType name="ParamClassTypesType">
        <sequence>
            <element ref="m3w:ParamClassType" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="ParamClassType" type="m3w:ParamClassTypeType"/>
    <complexType name="ParamClassTypeType">
        <sequence>
            <element ref="m3w:Description" minOccurs="0"/>
        </sequence>
        <attribute name="name" type="String" use="required"/>
    </complexType>
    <element name="Description" type="String"/>
    <element name="LogicalComponent" type="m3w:LogicalComponentType"/>
    <complexType name="LogicalComponentType">
        <sequence>
            <element ref="m3w:Description" minOccurs="0"/>
            <element ref="m3w:Role" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="String" use="required"/>
        <attribute name="name" type="String" use="required"/>
        <attribute      name="isStandard"      type="boolean"      use="optional"
default="true"/>
    </complexType>
    <element name="Role" type="m3w:RoleType"/>
    <complexType name="RoleType">
        <sequence>
            <element ref="m3w:Description" minOccurs="0"/>
            <element ref="m3w:Interface" maxOccurs="unbounded"/>
```

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

© ISO/IEC 2007 – 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

```
        </sequence>
        <attribute name="id" type="String" use="required"/>
        <attribute name="name" type="String" use="required"/>
    </complexType>
    <element name="Interface" type="m3w:InterfaceType"/>
    <complexType name="InterfaceType">
        <sequence>
            <element ref="m3w:Method" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="id" type="String" use="required"/>
        <attribute name="name" type="String" use="required"/>
    </complexType>
    <element name="Method" type="m3w:MethodType"/>
    <complexType name="MethodType">
        <sequence>
            <element ref="m3w:Parameter" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="name" type="String" use="required"/>
        <attribute name="output" type="String" use="required"/>
    </complexType>
    <element name="Parameter" type="m3w:ParameterType"/>
    <complexType name="ParameterType">
        <attribute name="name" type="String" use="required"/>
        <attribute name="type" type="String" use="required"/>
        <attribute name="optional" type="boolean" use="optional" default="false"/>
    </complexType>
    <element name="ChangeHistory" type="m3w:ChangeHistoryType"/>
    <complexType name="ChangeHistoryType">
        <sequence>
            <element ref="m3w:Addition" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="m3w:Modification" minOccurs="0" maxOccurs="unbounded"/>
            <element ref="m3w:Deletion" minOccurs="0" maxOccurs="unbounded"/>
        </sequence>
        <attribute name="base_version" type="String" use="required"/>
        <attribute name="date" type="date" use="optional"/>
    </complexType>
    <simpleType name="DataTypeType">
        <restriction base="String">
            <enumeration value="ParamClassType"/>
            <enumeration value="LogicalComponent "/>
            <enumeration value="Method"/>
        </restriction>
    </simpleType>
    <element name="Addition" type="m3w:DataChangeType"/>
    <element name="Modification" type="m3w:DataChangeType"/>
    <element name="Deletion" type="m3w:DataChangeType"/>
    <complexType name="DataChangeType">
        <attribute name="object" type="m3w:DataTypeType" use="required"/>
        <attribute name="name" type="String" use="required"/>
    </complexType>
</schema>
```


© ISO/IEC 2007 – 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 C
(normative)

Service metadata schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by Park Sung Jun
(Information and Communications University) -->
<schema                                targetNamespace="http://mccb.icu.ac.kr/m3w"
xmlns:m3w="http://mccb.icu.ac.kr/m3w"  xmlns:UED="urn:mpeg:mpeg21:2003:01-DIA-NS"
xmlns="http://www.w3.org/2001/XMLSchema"          elementFormDefault="qualified"
attributeFormDefault="unqualified">
    <import namespace="urn:mpeg:mpeg21:2003:01-DIA-NS" schemaLocation="UED.xsd"/>
    <element name="Component" type="m3w:ComponentType"/>
    <complexType name="ComponentType">
        <sequence>
            <element name="ComponentID" type="String"/>
            <element name="ComponentTitle" type="String" minOccurs="0"/>
            <element name="ComponentDescription" type="String" minOccurs="0"/>
            <element ref="m3w:Creator" minOccurs="0"/>
            <element ref="m3w:License" minOccurs="0"/>
            <element ref="m3w:AvailableInterfaces"/>
            <element ref="m3w:Services"/>
            <element ref="m3w:Implementations" minOccurs="0"/>
            <element ref="m3w:RequiredUsageEnvironments" minOccurs="0"/>
            <element ref="m3w:ComponentSignature" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="Creator" type="m3w:CreatorType"/>
    <complexType name="CreatorType">
        <sequence>
            <element name="CreatorName" type="String"/>
            <element name="CreatorPostalAddress" type="String" minOccurs="0"/>
            <element name="CreatorWebsite" type="String" minOccurs="0"/>
            <element name="CreatorEmail" type="String" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="License" type="m3w:ComponentLicenseType"/>
    <complexType name="ComponentLicenseType" mixed="true">
        <sequence>
            <any namespace="##any" processContents="lax" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="ComponentSignature" type="m3w:ComponentSignatureType"/>
    <complexType name="ComponentSignatureType" mixed="true">
        <sequence>
            <any namespace="##any" processContents="lax" minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="AvailableInterfaces" type="m3w:InterfacesType"/>
    <complexType name="InterfacesType">
        <sequence>
            <element ref="m3w:Interface" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <complexType name="InterfaceIDsType">
```

© ISO/IEC 2007 – 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

```
        <sequence>
            <element ref="m3w:InterfaceID" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="InterfaceID" type="String"/>
    <element name="Interface" type="m3w:InterfaceType"/>
    <complexType name="InterfaceType">
        <sequence>
            <element ref="m3w:InterfaceID"/>
            <element name="InterfaceName" type="String" minOccurs="0"/>
            <element name="InterfaceImplementors" type="m3w:ServiceIDsType"/>
        </sequence>
    </complexType>
    <complexType name="ServiceIDsType">
        <sequence>
            <element ref="m3w:ServiceID" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="ServiceID" type="String"/>
    <element name="Services" type="m3w:ServicesType"/>
    <complexType name="ServicesType">
        <sequence>
            <element ref="m3w:Service" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="Service" type="m3w:ServiceType"/>
    <complexType name="ServiceType">
        <sequence>
            <element ref="m3w:ServiceID"/>
            <element ref="m3w:ServiceInvocationType"/>
            <element    name="RequiredInterfaces"    type="m3w:InterfaceIDsType"
minOccurs="0"/>
            <element ref="m3w:Attributes" minOccurs="0"/>
            <element    name="CompliantServicesList"    type="m3w:ServiceIDsType"
minOccurs="0"/>
        </sequence>
    </complexType>
    <element name="ServiceInvocationType" type="m3w:ServiceInvocationTypeType"/>
    <simpleType name="ServiceInvocationTypeType">
        <restriction base="String">
            <enumeration value="in-process"/>
            <enumeration value="cross-process"/>
            <enumeration value="remote"/>
        </restriction>
    </simpleType>
    <element name="Attributes" type="m3w:AttributesType"/>
    <complexType name="AttributesType">
        <sequence>
            <element ref="m3w:Attribute" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="Attribute" type="m3w:AttributeType"/>
    <complexType name="AttributeType">
        <attribute name="attrName" type="String" use="required"/>
        <attribute name="attrType" type="String" use="required"/>
        <attribute name="defaultValue" type="String" use="optional"/>
    </complexType>
    <element name="Implementations" type="m3w:ImplementationsType"/>
    <complexType name="ImplementationsType">
        <sequence>
```


© ISO/IEC 2007 – 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

```
        <element ref="m3w:Implementation" maxOccurs="unbounded"/>
    </sequence>
</complexType>
<element name="Implementation" type="m3w:ImplementationType"/>
<complexType name="ImplementationType">
    <sequence>
        <element ref="m3w:TargetPlatform" minOccurs="0"/>
        <element ref="m3w:ProgrammingLanguage"/>
        <element ref="m3w:CodeLocation"/>
    </sequence>
</complexType>
<element name="TargetPlatform" type="m3w:TargetPlatformType"/>
<complexType name="TargetPlatformType">
    <sequence>
        <element name="Processor" type="m3w:ProcessorType" minOccurs="0"/>
        <element name="OperatingSystem" type="m3w:OSType"/>
        <element name="OSVersion" type="String"/>
    </sequence>
</complexType>
<simpleType name="ProcessorType">
    <restriction base="String">
        <enumeration value="x86"/>
        <enumeration value="mips"/>
        <enumeration value="alpha"/>
        <enumeration value="ppc"/>
        <enumeration value="sparc"/>
        <enumeration value="680x0"/>
        <enumeration value="vax"/>
        <enumeration value="AS/400"/>
        <enumeration value="S/390"/>
        <enumeration value="Other"/>
    </restriction>
</simpleType>
<simpleType name="OSType">
    <restriction base="String">
        <enumeration value="AIX"/>
        <enumeration value="BSDi"/>
        <enumeration value="VMS"/>
        <enumeration value="Unix"/>
        <enumeration value="DOS"/>
        <enumeration value="HPBLS"/>
        <enumeration value="HPUX"/>
        <enumeration value="IRIX"/>
        <enumeration value="Linux"/>
        <enumeration value="MacOS"/>
        <enumeration value="OS/2"/>
        <enumeration value="AS/400"/>
        <enumeration value="MVS"/>
        <enumeration value="Solaris"/>
        <enumeration value="SunOS"/>
        <enumeration value="VxWorks"/>
        <enumeration value="Windows"/>
        <enumeration value="Other"/>
    </restriction>
</simpleType>
<element name="ProgrammingLanguage" type="m3w:ProgrammingLanguageType"/>
<complexType name="ProgrammingLanguageType">
    <sequence>
        <element name="ProgrammingLanguageName" type="String"/>
        <element name="ProgrammingLanguageVersion" type="String"/>
```

© ISO/IEC 2007 – 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

```
        </sequence>
    </complexType>
    <element name="CodeLocation" type="m3w:CodeLocationType"/>
    <complexType name="CodeLocationType">
        <attribute name="isBuild-In" type="boolean" use="required"/>
        <attribute name="ref" type="anyURI" use="optional"/>
    </complexType>
    <element                                    name="RequiredUsageEnvironments"
type="m3w:RequiredUsageEnvironmentsType"/>
    <complexType name="RequiredUsageEnvironmentsType">
        <sequence>
            <element ref="m3w:EnvironmentProperty" maxOccurs="unbounded"/>
        </sequence>
    </complexType>
    <element name="EnvironmentProperty" type="UED:UsageEnvironmentPropertyType"/>
</schema>
```


© ISO/IEC 2007 – 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, 2004

[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] Dale Rogerson, Inside COM - Microsoft's Component Object Model, Microsoft Press, ISBN: 1-57231-349-8.

© ISO/IEC 2007 – 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

ICS 35.040

Price based on 211 pages

© ISO/IEC 2007 – 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