INTERNATIONAL STANDARD

ISO/IEC 23004-5

First edition
2008-02-15

Information technology — Multimedia Middleware —

Part 5: Component download

Technologies de l'information — Intergiciel multimédia —

Partie 5: Téléchargement de composant

Reference number
ISO/IEC 23004-5:2008(E)

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 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

Contents

Page

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 23004-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information technology*, Subcommittee SC 29, *Coding of audio, picture, multimedia and hypermedia information*.

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

— *Part 1: Architecture*

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

— *Part 3: Component model*

— *Part 4: Resource and quality management*

— *Part 5: Component download*

— *Part 6: Fault management*

— *Part 7: System integrity management*

Introduction

The Download Framework has been developed as part of M3W to fulfill the requirements concerning upgrading and extension of operational M3W-based systems. During the period that a CE device is owned by a user, there is often a need to improve and/or extend the device's software in order to extend the economic lifetime of the device.

In the Component lifecycle, the Download Framework is responsible for transferring M3W Components from a Repository to a Target (see Figure 1).

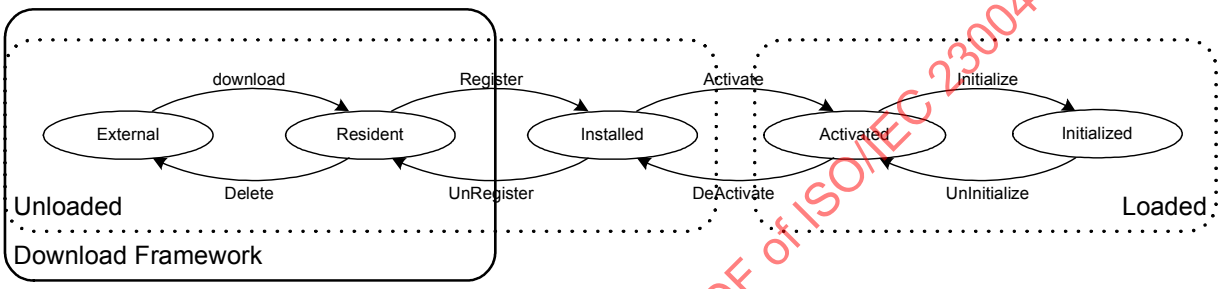


Figure 1 — Scope Download Framework

The download process takes a Component in its "installable Component" stage and downloads it to the Target, which brings the Component in its "resident" state. This is a very macroscopic view on the download process which, when elaborated, reveals the following constituent activities of the download process.

— **Identify need for download:** This activity describes the part in which the need for the presence of a given Component on a given M3W system is identified.

— **Entities are located:** After the need for download is realized, the place from where the Component can be retrieved, as well as its destination, must be identified. Also, the place where the decision about the feasibility of the download will be established must be identified, in case it is an entity other than the sender or the receiver.

— **Connection establishment:** The sender and the receiver in the context of the download for a specific Component establish a connection between them in order to proceed with the download process.

— **Security checks:** Using the connection between them, the server and the receiver run a number of security checks (e.g. mutual authentication, authorization for the download, payment guarantees to cover the costs of the Component as well as the cost of the download process).

— **Feasibility analysis:** This activity includes a number of checks and verifications to ensure that the Component can be installed and activated at its destination while it still satisfies its specifications and without putting in danger the rest of the M3W system at its destination.

— **Transmission properties:** A number of negotiations regarding the security and reliability characteristics of the data transmission as well as providing certain necessary information (e.g. the payment token like credit card number, payment certificate).

— **Component transmission:** This activity comprises the actual transfer of the data that constitute the Component and which includes at least the data corresponding to the representation of the executable model of a Component.

— **Connection termination:** This activity contains all actions necessary for the sender and the receiver to keep consistent information about the final status of the data transfer (and hence the download process so far). If this activity is completed successfully, the remaining parts of the download process necessitate only the participation of the receiver.

The remainder of this part of ISO/IEC 23004 contains the specification of the Download Framework. This specification describes the Download Framework from a number of perspectives.

— The decomposition into roles is described in the Structure View (8.1).

— The interaction between these roles is described in the Behavior View (8.2).

— The deployment of these roles is described in Deployment View (8.3).

Information technology — Multimedia Middleware —

Part 5:
Component download

1 Scope

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

2 Normative references

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

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

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

3 Terms and definitions

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

3.1 Specification terms and definitions

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

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

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

3.1.4
role
abstract class defining behavior only

3.1.5
role instance
object displaying the behavior defined by the role

3.1.6
attribute
instance variable associated with a role

NOTE Attributes are used to associate state information with roles.

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

NOTE For C functions, signature is equivalent to prototype.

3.1.8
specification item
entity defined in a specification

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

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

3.1.10
constraint
restriction that applies to a specification item

3.1.11
execution constraint
constraint on multi-threaded behavior

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

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

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

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

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

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

NOTE Unlike enum sets, these sets may be infinite.

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

NOTE Maps are a kind of generalized array.

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

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

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

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

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

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

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

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

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

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

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

3.1.28
diversity attribute
role attribute that represents a configurable item

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

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

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

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

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

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

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

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

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

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

NOTE In reality, the assertion may temporarily be violated.

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

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

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

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

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

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

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

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

NOTE This is used to specify event-driven behavior.

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

3.1.46
client role
role modeling the users of a logical component

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

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

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

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

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

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

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

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

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

3.1.55
asynchronous function
function with a delayed effect

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

3.2 Realization terms and definitions

3.2.1
appliance
product as seen by the customer

NOTE It consists of the device, an operating system (OS), components and applications.

3.2.2
application
software entity that provides a set of functions to a user

3.2.3
component
unit of trading that conforms to the M3W component model

NOTE In the M3W component model, components are containers for models.

3.2.4
component model
specification of what constitutes a component

NOTE It defines *inter alia* the interaction mechanisms between components and between components and their environment.

3.2.5
device
physical part of the appliance, sometimes also used to denote an identifiable element of that appliance

3.2.6
executable component
model that is executable on a platform

NOTE The executable component is a container for services.

3.2.7
M3W system
system that conforms to the M3W specification

3.2.8
operational
state when an M3W system is fulfilling its required functions for a period of time

3.2.9
platform
operating system (OS) and hardware that executes the OS

3.2.10
system
combination of a platform, a set of executable components, a run-time environment and a set of applications that can provide a user with a set of functions

4 Abbreviated terms

API Application Programming Interface
IDL Interface Definition Language
M3W Multimedia Middleware
OS Operating System
UML Unified Modeling Language

5 Overview of interface suites

This clause is informative and gives an overview of the interface suites in the M3W API for component download. These interface suites can be divided in the following categories:

— Interface suites for receiving M3W Components

— Interfaces for initiating M3W Component transfer

For receiving components this part of ISO/IEC 23004 specifies Target logical component (see 6.1.1). For initiating component transfer this part of ISO/IEC 23004 specifies the Initiator logical component (see 6.2.1). These logical components provide a uniform view (for the applications and other middleware) on the download functionality provided by M3W.

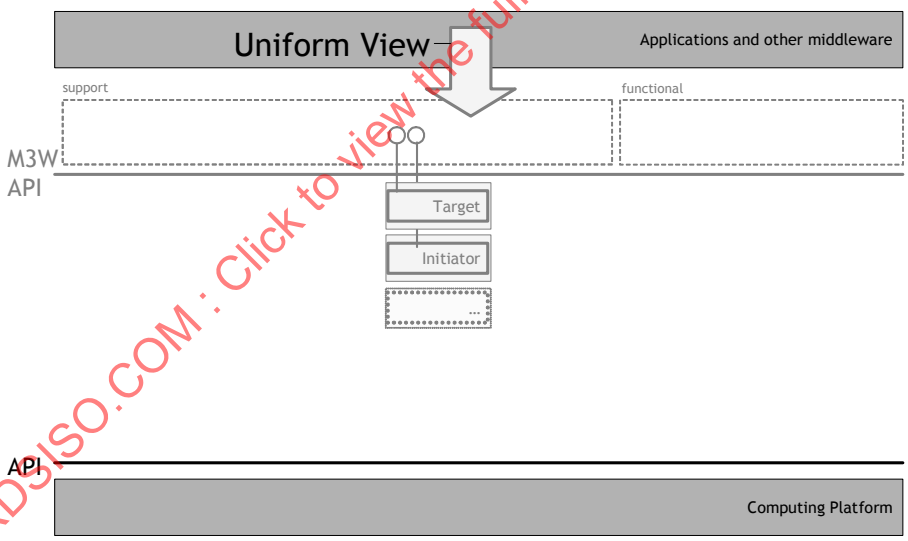


Figure 2 — Overview of interfaces suites (logical components) for component download

The download framework contains more roles than just those used for initiating component transfer and receiving components. The download framework is suitable for transfer of components according to ISO/IEC 23004-3 specification of M3W components. However, these roles only expose interfaces that are internal to the download framework and are therefore not part of the M3W API specification. These other roles (Locator, Decider and Repository) are discussed in the clause on realization of the Download framework (see clause 8). The integration in the overall architecture shall be done according to ISO/IEC 23004-1.

6 Download interface suites

6.1 Interface suites for receiving components

6.1.1 Target

6.1.1.1 Concepts

The Target is one of the roles in the Download framework. The Download framework enables the transfer of components to a device. The Target is responsible for exposing the target profile (used for accessing feasibility of component download), receiving components and storing the received components in non-volatile storage on the device.

6.1.1.2 Types and Constants

6.1.1.2.1 Public Types and Constants

6.1.1.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_TARGET_LOCATOR_NOT_FOUND = 0x00000001
const rcResult RC_ERR_TARGET_CANNOT_STORE_COMPONENT = 0x00000002
```

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_TARGET_LOCATOR_NOT_FOUND	This error is returned when the target is unable to connect to the locator
RC_ERR_TARGET_CANNOT_STORE_COMPONENT	This error is returned when the target is unable to store the downloaded component, for example due to write permissions or low storage space

6.1.1.2.2 Model Types and Constants

None

6.1.1.3 Logical Component

6.1.1.3.1 Interface Role Model

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

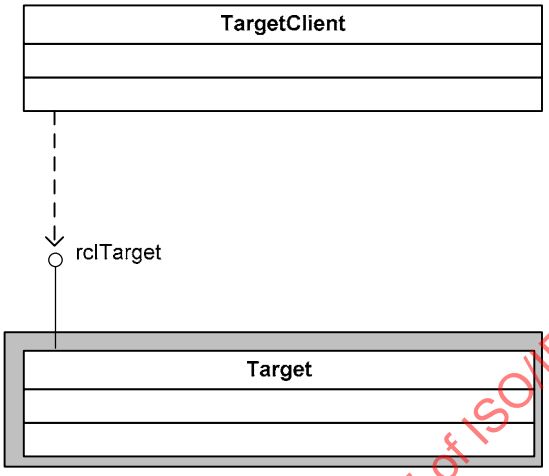


Figure 3 — Interface-Role Model

A Target Logical Component contains the Target role that provides the `rcITarget` interface. This interface can be used by a client to start a download task. The download task is responsible for participating in the interaction between the download roles. More specifically the download task is responsible for receiving and storing components.

6.1.1.3.2 Diversity

6.1.1.3.2.1 Provided Interfaces

Table 2

Role	Interface	Presence
Target	`rcITarget`	`Mandatory`

6.1.1.3.2.2 Configurable Items

The behavior of the Target role only depends on the parameters of the operation `downloadTask`. The operation `downloadTask` has a number of parameters that influence the behavior of the download task created.

6.1.1.3.2.3 Constraints

None.

6.1.1.3.3 Instantiation

6.1.1.3.3.1 Objects Created

When the Target logical component is created an instance of the Target role is created. Multiple instances of the Target logical component can be created on a device. The different targets will need to listen to different ports and have different names.

Table 3

Type	Object	Multiplicity
`Target`	`Target`	`1`

6.1.1.3.3.2 Initial State

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

— No download task has been started.

6.1.1.3.4 Execution Constraints

The Target logical component is thread safe.

6.1.1.4 Roles

6.1.1.4.1 Target

Signature

```
role Target {
}
```

Qualifiers

— Root

Description

A Target enables the reception of components by a device. The target can be used to start a download task, which is a thread responsible for providing the target profile to other download roles and receiving and storing the transferred components.

Independent Attributes

— None

Invariants

— None

Instantiation

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

Active Behavior

The Target role has no active behavior. Upon invocation of the `downloadTask` operation, the client must create a new thread. The active behavior of this task (thread) is providing the target profile to other download roles, receipt of components and storage of these components.

NOTE The transfer of components is initiated by a different role (initiator) which is not part of this logical component.

6.1.1.5 Interfaces

6.1.1.5.1 rclTarget

Qualifiers

— None

Description

This interface is used for activation of the target. Activation means that the target starts listening to a specific port for attempts to transfer components.

Interface ID

```
{bf56a0f9-0d34-4925-88b3-11631628e3bf}
```

6.1.1.5.1.1 DownloadTask

Signature

```
RcResult downloadTask([in] String locatorAddress,
                      [in] String targetName,
                      [in] String targetAddress,
                      [in] Int32 targetPort,
                      [in] String downloadDirectory);
};
```

Qualifiers

— Thread Safe

Description

By invoking this operation the target starts listening to the specified port for attempts to transfer components to this device.

Parameters

Table 4

Name	Description
targetName	Name used to identify the target
targetAddress	Network address of the target. This address is registered at the locator, such that the other roles participating in the download process are able to locate this target.
targetPort	The port to which the target is listening for requests for component transfers.
downloadDirectory	Location to store the downloaded components (absolute path)

Return Values

Only the non-standard error codes are listed.

Table 5

Name	Description
RC_ERR_TARGET_LOCATOR_NOT_FOUND	The target is unable to connect to the specified locator.
RC_ERR_TARGET_CANNOT_STORE_COMPONENT	The target is unable to store received components.

Precondition

True

Action

Start thread listening to the `targetPort` for component transfer attempts.

Postcondition

Thread that is listening to the `targetPort` for component transfer attempts is started.

6.2 Interfaces suites for initiation of component transfer

6.2.1 Initiator

6.2.1.1 Concepts

The Initiator is one of the roles in the Download framework. The Download framework enables the transfer of components to a device. The Initiator is responsible for initiating and coordinating the component transfer process. Initiating means that this role will need to find all the entities that will be involved in the component transfer process and coordinate the interaction between these entities. For the client this logical component provides an operation to request for the transfer of a component to a device.

6.2.1.2 Types and Constants

6.2.1.2.1 Public Types and Constants

6.2.1.2.1.1 Error Codes

Signature

```
const rcResult RC_ERR_INITIATOR_LOCATOR_NOT_FOUND = 0x00000001
const rcResult RC_ERR_INITIATOR_TARGET_NOT_FOUND = 0x00000002
const rcResult RC_ERR_INITIATOR_REPOSITORY_NOT_FOUND = 0x00000004
const rcResult RC_ERR_INITIATOR_DECIDER_NOT_FOUND = 0x00000008
const rcResult RC_ERR_INITIATOR_COMPONENT_NOT_FOUND = 0x00000010
const rcResult RC_ERR_INITIATOR_DOWNLOAD_NOT_FEASIBLE = 0x00000020
```

Qualifiers

— Error codes

Description

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

Constants

Table 6

Name	Description
RC_ERR_INITIATOR_LOCATOR_NOT_FOUND	This error is returned when the locator is unable to connect to the locator
RC_ERR_INITIATOR_TARGET_NOT_FOUND	This error is returned when the target cannot be found (using the locator)
RC_ERR_INITIATOR_REPOSITORY_NOT_FOUND	This error is returned when no repository can be found (using the locator)
RC_ERR_INITIATOR_DECIDER_NOT_FOUND	This error is returned when no suitable decider can be found (using the locator)
RC_ERR_INITIATOR_COMPONENT_NOT_FOUND	This error is returned when the requested component is not available in the registered repositories (at the locator)
RC_ERR_INITIATOR_DOWNLOAD_NOT_FEASIBLE	This error is returned when the download is not feasible (decided by the decider)

6.2.1.2.2 Model Types and Constants

None.

6.2.1.3 Logical Component

6.2.1.3.1 Interface Role Model

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

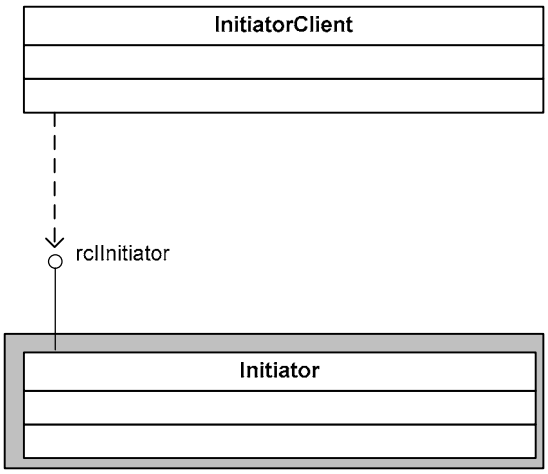


Figure 4 — Interface-Role Model

An Initiator Logical Component contains the Initiator role that provides the `rcIInitiator` interface. This interface can be used by a client to initiate the transfer of a specific component to a specific target. The Initiator role locates all of the involved entities (using the locator, see subclause 8.1.2) and also coordinates the download process.

6.2.1.3.2 Diversity

6.2.1.3.2.1 Provided Interfaces

Table 7

Role	Interface	Presence
Initiator	`rcIInitiator`	`Mandatory`

6.2.1.3.2.2 Configurable Items

The behavior of the Initiator role only depends on the parameters of the operation `getComp`. The operation `getComp` has a number of parameters that influence which component is to be transferred, to which target and the decider that is used.

6.2.1.3.2.3 Constraints

None.

6.2.1.3.3 Instantiation

6.2.1.3.3.1 Objects Created

When the Initiator logical component is created, an instance of the Initiator role is also created. Multiple instances of the Initiator logical component can be created on a device.

Table 8

Type	Object	Multiplicity
`Initiator`	`Initiator`	`1`

6.2.1.3.3.2 Initial State

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

— No component transfers have been initiated from this initiator.

— Initiator is ready to initiate a component transfer

6.2.1.3.4 Execution Constraints

This logical component is Thread-Safe.

6.2.1.4 Roles

6.2.1.4.1 Initiator

Signature

```
role Target {
}
```

Qualifiers

— Root

Description

This role can be used by a client to initiate the transfer of a specific component to a specific target. The Initiator role locates all of the involved entities (using the locator, see subclause 8.1.2) and coordinates the download process.

Independent Attributes

None.

Invariants

None.

Instantiation

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

Active Behavior

None

6.2.1.5 Interfaces

6.2.1.5.1 rclInitiator

Qualifiers

Thread-safe

Description

This interface can be used to initiate the transfer of a specific component to a specific target.

NOTE Part of the transfer test is a feasibility test. This check can verify the technical fit, resource fit as well as the business fit of the component and the target.

Interface ID

```
5d843ee1-4cb0-494b-a598-1055268b4cb2
```

6.2.1.5.1.1 getComp

Signature

```
Int32 getComp([in] UUID componentUUID,
              [in] String targetName,
              [in] String deciderClass,
              [in] String locatorAddress,
              [in] pVoid statusReport);
```

Qualifiers

Thread-safe

Description

This operation initiates the transfer of component with id `componentUUID`, to the target with name `targetName`, using a decider of type `deciderClass`. To locate the entities involved in this process the locator at address `locatorAddress` is used. Finally the operation provides the possibility of getting a status report by entering a pointer to a callback function.

Parameters

Table 9

Name	Description
`targetName`	Name used to identify the target
`deciderClass`	String indicating the type of decider that should be used. The type of deciders that can be used can be found in the platform instance document.
`locatorAddress`	Network address (URL) of the locator that should be used.
`statusReport`	Pointer to a callback function used for status reporting

Return Values

Only the non-standard error codes are listed.

Table 10

Name	Description
RC_ERR_INITIATOR_LOCATOR_NOT_FOUND	This error is returned when the locator is unable to connect to the locator
RC_ERR_INITIATOR_TARGET_NOT_FOUND	This error is returned when the target cannot be found (using the locator)
RC_ERR_INITIATOR_REPOSITORY_NOT_FOUND	This error is returned when no repository can be found (using the locator)
RC_ERR_INITIATOR_DECIDER_NOT_FOUND	This error is returned when no suitable decider can be found (using the locator)
RC_ERR_INITIATOR_COMPONENT_NOT_FOUND	This error is returned when the requested component is not available in the registered repositories (at the locator)
RC_ERR_INITIATOR_DOWNLOAD_NOT_FEASIBLE	This error is returned when the download is not feasible (decided by the decider)

Precondition

True

Action

Component transfer process is initiated and controlled.

Postcondition

Component identified by `componentUUID` is resident on the non-volatile storage of the target.

7 Realization overview

This clause is informative. The goal of the Download framework is to enable controlled transfer of components and support a large number of different download and upload scenarios. The flexibility to support different scenarios is reached by identification of 5 roles that can be deployed in a very flexible way (different deployment for different scenarios). The transfer is controlled because the Download not only focuses on component transfer but also takes into account aspects such as:

— Identification and Authentication

— Non repudiation

— Feasibility checking

Clause 8 discusses the realization of the download framework. This contains the separation in the different roles:

— Repository

— Target

— Locator

— Decider

— Initiator

For these roles their responsibilities as well their interaction with other roles is discussed. A number of example scenarios and the corresponding deployment of the roles are shown in Annex A. The core realization technology is described in ISO/IEC 23004-3.

8 Download realization

8.1 Structure view

Based on the description of the Component download lifecycle in the introduction, 5 participants in the download process are identified. Two of these participants are concrete entities:

— The Repository where a Component resides that is to be downloaded, and

— The Target which is the actual device where that Component will be downloaded. The Target is characterized by the combination of the Platform, i.e. the hardware and the Operating System, and the Runtime Environment of the M3W device.

In addition to these two entities, three roles are involved in the download process:

— The Initiator, which has responsibility for identifying the need for a download and contacting the involved entities to initiate the download process.

— The Locator, which has responsibility for locating the Target and Repository entities as well as the entity which plays the Decider role for a given download.

— The Decider, which has responsibility for performing feasibility analysis of the download (i.e. whether it is possible for a given Component to be downloaded, registered and activated in a given Target without creating any problems for the Target). This role is also responsible for any negotiations that might take place between a Repository and a Target regarding any requirements and guarantees provided by these two entities.

The remainder of this subclause elaborates on the responsibilities and the behavior of the aforementioned entities and roles in the download process.

8.1.1 Initiator

The entity which plays the role of the Initiator is responsible for:

— Identifying the Component to be downloaded.

— Identifying the Target where a given Component needs to be downloaded to.

— Identifying the Decider which will perform the feasibility analysis for the download as well as the necessary negotiations.

— Contacting the Target and the Repository in order to initiate the download process.

The role of Initiator can be played by:

— The Target. This can be the case when the owner of a M3W system decides to download a new Component. Then, the initiative for performing a download may come from the end-user (who is represented by the Target in the download process). Another case where the Target plays the role of the Initiator is when (due to some runtime conditions such as, for example, reconfiguration, outcome of a Service, etc) the Application or Service Instances of the Target identify a need for downloading new functionality.

— The Repository. This is the case when the developer of Components, the provider of certain Services or the remote administrator of a M3W system identifies that a Component needs to be uploaded to a certain Target. This need can stem from a number of reasons such as contractual agreements, e.g. in the contract between the customer and the service provider it is mentioned that the latter is responsible for automatically updating the software of the M3W system of the former whenever a new version of the service software becomes available. The only constraint is that the entity which identifies the need for Component download also has the Component to be downloaded.

— A 3rd entity, i.e. any Host different from the Target and the Repository. In this case, the entity that identifies the need for download does not possess the Component to be downloaded nor will it receive the Component after the download. This case can be similar with the above one, only the software that identifies the need for download resides on a different Host than the one hosting the Component Repository.

The Initiator role does not impose any constraints on the entity which will play it, i.e. the entity that plays the role of the Initiator does not have to be a M3W system.

8.1.2 Locator

The entity which plays the role of the Locator has a well-known entry point in every involved network in order to be accessible without prior need to search for its address. The Locator is responsible for:

— Providing the location of Repositories containing Components and Services, i.e. given a Component UUID (Universally Unique Identifier) or a Service UUID the Locator returns the information necessary to contact the Repositories that contain the specified Component or the Components containing the specified Service. This contact information can be the physical address of the Host hosting the Repository, or some other type of reference to a Host which allows remote Hosts to connect to it over a given network.

— Providing the location of M3W systems (Targets). This calls for a unique way to identify Targets, e.g. a UUID for Targets. Similar to the above case, when the Locator is provided with a Target ID it returns the contact information of the Target. Since the Target might be accessible over different networks (e.g. a mobile phone can be accessed over the GPRS network but also over an IrDA connection or a Bluetooth link), the Locator may return a list of addresses and the networks where they are valid.

— Providing the location of the Decider.

Let **UUIDC** be the set of all M3W Component UUIDs, **UUIDS** be the set of all M3W Service UUIDs, **UUIDT** be the set of all Target IDs, and **ADDR** be the set of contact information (e.g. physical addresses and network on which these addresses are valid) of all known, M3W Hosts (i.e. Targets and other Download Framework entities). Let also **UUIDD** be the set of all M3W Services that can play the Decider role. Finally, let **POW(X)** denote the powerset of X. Then, the Locator can be defined as the role which provides the following four functions:

$$\mathbf{L_1 \mid UUID_C \rightarrow POW(ADDR)}$$

$$\mathbf{L_2 \mid UUID_S \rightarrow POW(ADDR)}$$

$$\mathbf{L_3 \mid UUID_T \rightarrow POW(ADDR)}$$

$$\mathbf{L_4 \mid UUID_D \rightarrow POW(ADDR)}$$

The Download Framework does not make any assumptions nor does it place any requirements on how the Locator possesses this information. It could e.g. be because it also acts as the registry of Components and Services in a system, or because it can contact that Registry, or due to some other scheme which is beyond the scope of the Download Framework.

It is important to note that the functions which define the Locator role indicate that the Locator may return contact information for more than one M3W. However it does not however state anything about the order in which contact information of these Repositories is placed nor about the completeness of the returned list of Repositories.

The role of Locator can be played by:

— The Target. This is an unlikely case where the Target already knows in which repositories M3W Components and Services reside.

— The Repository. This is also an unlikely case where the Repository knows how to contact all Targets in the system as well as the available Deciders.

— A 3rd entity, a Host different from the above two. This is the most likely case to appear in real life.

The entity that plays the role of the Locator does not need to be a M3W System. It must only be able to recognize UUIDs for M3W Components, Services, Libraries (and Targets if the form of a Target ID is a UUID).

8.1.3 Decider

The entity which plays the role of the Decider is responsible for:

— Deciding whether a given Component can be downloaded on a given Target over a given connection. This decision is taken based on an adequate description of the Component, an adequate description of the Target, and on an adequate description of the connections that the Target and the Repository that contains the aforementioned Component are capable and willing of establishing.

— Negotiating the above 3 types of descriptions regarding the Component, the Target and the connections offered by the Target and the Repository.

The functionality provided by the Decider role is based on 3 types of "adequate descriptions": the description of a Component, the description of a Target and the description of a connection. In the remainder of this part of ISO/IEC 23004 the term "profile" is used to refer to these adequate descriptions.

The role of the Decider can be played by:

— The Target. This is the case where the Target decides for itself whether to accept a given Component from a given Repository. Even if the actual decision about a Component's download to a given Target is eventually taken by some entity other than the Target, it is suggested that the latter possesses a "default", and maybe simple, Decider either as part of its M3W Runtime or as a Service provided by some M3W Component present at the Target. Still, it must be possible to allow the delegation of the download decision to an entity other than the Target which possesses a full-fledged Decider. The default Decider can use Target-proprietary functionality to make a download decision. However, it is probably difficult to let it evolve with new generations of Components (new "profile parameters"). For this reason, it must be allowed to delegate the download decision to another Decider outside the Runtime (i.e. the download decision can be made by a Service Instance or by a remote Host).

— The Repository. Since the Repository is assumed to exist on some machine with virtually unlimited resources, that machine might as well provide the Decider role. A scenario that may occur often is the case where the Repository plays the role of the Initiator and the Decider, e.g. the manufacturer of set-top boxes who also is the provider of the functionality offered by them, may decide to upgrade the software in all the M3W systems it has sold and which are accessible over some network.

— A 3rd entity, a Host different from the above two. This is a case likely to occur in a business model with a great deal of distribution among the M3W systems and the developers of M3W Components. In such cases, there is a need for a 3rd party,(a widely recognized and trusted entity) which can provide reliable information regarding the feasibility of Component download. These 3rd party entities cannot be the Targets or the providers of the M3W Components (i.e. the Repositories).

An entity that plays the Decider role does not need to be a M3W Target. However, the Decider must be able to understand a great deal of M3W related information (i.e. profiles of M3W Components and Targets).

8.1.4 Target

The Target entity is the M3W system where a M3W Component resides after it participates in a successful download process. Among its responsibilities and its capabilities, the most important ones related to download are:

— To provide a description about the resources it is willing and able to offer for the download of a given Component (Target profile).

— To provide a description of the connections it is willing and able to establish over the different networks to which it has access (a connection profile).

— To support a security protocol for Component transfer as described in Subclause 8.2.5.

— To support a transfer protocol for the download with atomic failure semantics (see Subclause 8.2.4).

It is possible that the Target entity plays the role of the Initiator (e.g. user triggered download), the role of the Decider (e.g. built-in, default Decider, see Subclause 8.1.3), and in some extreme cases, also the role of Locator.

8.1.5 Repository

The Repository entity is a Host where a Component resides prior to its participation in a download process. It is possible that the Repository is a M3W system (e.g. in the case of device-to-device download). Among its responsibilities and its capabilities, the most important ones related to download are:

— To provide a description regarding the Component to be downloaded (Component profile).

— To provide a description of the connections it is willing and able and to establish over the different networks to which it has access (connection profile).

— To support a security protocol for Component transfer as described in Subclause 8.2.5.

— To support a transfer protocol for the download with atomic failure semantics (see Subclause 8.2.4).

It is possible that the Repository entity plays the role of the Initiator (e.g. Service provider triggered download), the role of the Decider (e.g. remote administration of M3W systems by the Service provider, see Subclause 8.1.3), and in some cases, also the role of Locator.

8.2 Behavior view

The download process can be divided into three sub-processes:

— The location sub-process (see 8.2.1)

— The decision sub-process (see 8.2.2)

— The transfer sub-process (see 8.2.3)

8.2.1 Location sub-process

Once the need to download a Component has been identified by the Initiator (see Subclause 8.1.1), the Component, the Target, and the Decider must be located. This operation forms a self contained, location sub-which was introduced in Subclause 8.1.2.

In short, the Locator role is played by an entity with well-known entry point in every network on which it is accessible so there is no need to look-up its address. The Locator must locate the Component to be downloaded and the address of a Repository (or a set of addresses of Repositories) where the Component resides. It must also locate the Target and provide the address (or a set of addresses on different networks) through which the Target can be reached. Finally, the Locator must locate the entity that plays the role of the Decider (see subclause 8.1.3) which will make a decision regarding download feasibility for the given Component and Target combination.

The location sub-process starts when the Locator receives from the Initiator a description of the Component to be downloaded, a description of the Target that will receive the specified Component and a description of the Decider which will decide on the feasibility of the download.

The first thing the Initiator must do is to find out in which Repository(-ies) the Component C resides and how to contact the Repository(-ies), the Target and the Decider for the given download in order to let them know that a download must take place.

The Initiator knows how to contact the Locator, and starts by performing a number of mutual security checks and setups (e.g. identification, authentication, communication encryption, etc) with the Locator. The exact security checks and setups depend on security protocol required for the communication between the Initiator and the Locator, this depends on a specific realization of the download framework and is out of the scope of this part of ISO/IEC 23004.

After the secure communication is successfully set up, the Initiator provides to the Locator the UUID of Component C and of Target T as well as a Decider ID in order to receive the contact information of a Repository (or a set of them) where Component C resides, the contact information of the Target T and the contact information of the Decider respectively.

The Locator's involvement in the download process stops when the Initiator receives these three (sets of) addresses.

Once the Initiator receives the aforementioned addresses (contact information), it contacts the Repository (the first available/accessible one if it has received a list of Repositories where Component C resides) and the Target. As a first step, it establishes with each of them a secure communication link in the same way it did with the Locator. Then, the Initiator sends to the Repository a message which initiates the download process for the Repository and which contains the addresses of the Decider and the Target. In a similar way, the Initiator sends to the Target the addresses of the Decider and the Repository.

The Location sub-process is highlighted in Figure 5.

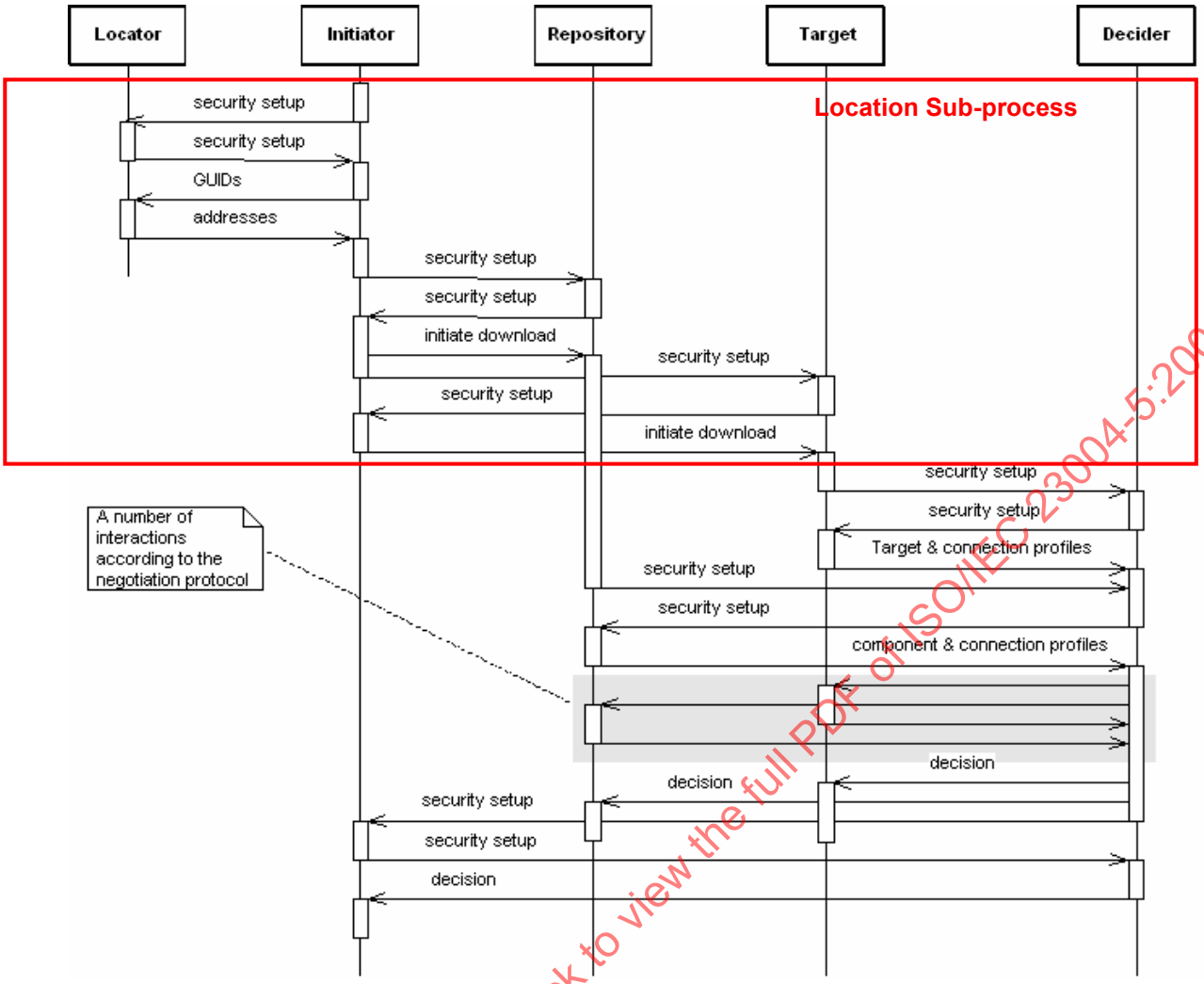


Figure 5 — Sequence diagram highlighting the location sub-process

The remainder of this subclause elaborates on the individual steps of the location sub-process.

8.2.1.1 Locating components

The first part of the location sub-process is the location of a Repository where the Component to be downloaded resides. The Initiator, which has identified the need for the download, has communicated to the Locator a description of the Component to be downloaded. The description of the Component and the associated operation performed by the locator may have one of the two following forms:

— A Component UUID, which means that a specific M3W Component has been identified. An example of this case can be when the remote administrator of some M3W systems has decided that a new piece of software (identified by the specific Component UUID) must be placed on the devices that it administers. For UUIDC being the set of all M3W Component UUIDs, ADDR being the set of contact information (e.g. physical addresses and network on which these addresses are valid) of all known M3W systems, and POW(X) denoting the powerset of X, this operation of the location sub-process is described by the following mathematical function:

$$L_1 \mid UUID_C \rightarrow POW(ADDR)$$

— A Service UUID. This means that the Component to be downloaded is not necessarily a unique M3W Component; rather it can be any M3W Component which contains the identified M3W Service. An example of this is when the owner of a M3W system wishes to obtain (locally on his Device) a given

Service (identified by its specific UUID) without having any information about the M3W Component which actually contains it. For UUIDS being the set of all M3W Service UUIDs, and ADDR and POW() as above, this operation of the location sub-process is described by the following mathematical function:

$$\mathbf{L_2 \mid UUID_S \rightarrow POW(ADDR)}$$

In both cases, the Locator returns a list of addresses of Repositories where the described Component resides. The Download Framework does not specify a special meaning to the order of this list; hence, unless specified elsewhere, this list must not be treated by the Initiator as a priority list.

The contact information described by **ADDR** above can be represented by a pair of data which indicates the physical address where the Repository can be contacted and the network on which this address is valid. This is due to the reason that the Repository might be accessible through more that one network, e.g. over Internet using an IP address or over GPRS using a direct phone call to the Repository (the address in the latter case is the phone number to which the call is made). Each element in the set **ADDR** can is a pair of a physical address and the network on which this address is valid.

Locating M3W Components using their UUIDs or the UUIDs of the Services they provide is the obvious way to define the location sub-process. However, there are other ways of describing a Component. Consider for example the case where a M3W system, after experiencing a failure, needs to download a Component in order to re-establish a consistent configuration (i.e. a set of Components that contain all required Services and Libraries for its Applications). One possibility is to use information kept from a previous consistent configuration and ask to specifically download the missing Component identified by its UUID. However, especially for mobile M3W systems which may find themselves in highly variable environments, downloading a specific Component might not always be possible. Besides, all that is needed might just be a M3W Component that provides at least the same Services and requires at most the same Services as the missing Component.

This example calls for a somewhat more intelligent location sub-process. Instead of providing a Component or Service UUID, the Initiator may provide the Locator with a set of requested Services and a set of available Services (both in terms of Service UUIDs). As a result, the Locator should return the address of a Repository (or a list of such addresses) that contains Components that provide at least all the requested Services and require at most the available Services. In this case, the operation performed by the Locator is described by the following mathematical function:

$$\mathbf{L'_2 \mid POW(UUID_S) \times POW(UUID_S) \rightarrow POW(ADDR)}$$

8.2.1.2 Locating devices

The second part of the location sub-process is to locate the Target where the Component must be downloaded. As mentioned in Subclause 8.1.2, this calls for a unique way to identify Targets. The M3W architecture does not explicitly define an identification schema for Targets. It does allow Targets to be identified by UUID, however. In the remainder of thi the term "Target UUID" is used to denote the means to uniquely describe a Target in a M3W system.

Provided that the Initiator communicated to the Locator a valid Target UUID, the Locator must provide a description of how the described Target can be accessed. For **UUIDT** being the set of all M3W system UUIDs, **ADDR** being the set of contact information (e.g. physical addresses and network on which these addresses are valid) of all known M3W Systems, and **POW(X)** denoting the power set of X, this operation of the location sub-process is described in the following mathematical function:

$$\mathbf{L_3 \mid UUID_T \rightarrow POW(ADDR)}$$

8.2.1.3 Locating the Decider

The last part of the location sub-process is the location of the entity that plays the Decider role described in Subclause 8.1.3. As mentioned already in Subclause 8.1.3 and further elaborated on in Subclause 8.1.4, the Target may have a "default" Decider, but the entity which will eventually play the role of the Decider can be a different one form the Target and the Repository. In this case, the address of the 3rd entity must be provided

by the Initiator to the Repository and the Target, in order for them to know where to send the Component and Target profiles respectively.

The entity which plays the Decider role is not necessarily a M3W system. To locate this entity, the process is somewhat similar to using a Service UUID to locate the Component to be downloaded; only in this case the search is not launched for a Repository where Components offering the specific Service reside but for M3W system where a Component that provides the given Service is installed and activated or for a Host that implements the Decider role. For **UUIDD** being the set of UUIDs of all M3W Services or of Hosts which offer the functionality described by the Decider role, **ADDR** being the set of contact information (e.g. physical addresses and network on which these addresses are valid) of all known systems in the Download Framework, and **POW(X)** denoting the power set of X, this operation of the location sub-process is described by the following mathematical function:

$$\mathbf{L_4 \mid UUID_D \rightarrow POW(ADDR)}$$

The above formula provides the widest possible selection of Deciders in the M3W systems where the download will take place. In practice, the choice of the Decider might be restricted by a richer set of constraints. For example, both the Target and the Repository may require the Decider to have certain credentials in order to be allowed to decide on the feasibility of a download that relates to both of them. However, it is not the Locator's responsibility to choose which of the possible Deciders will be used for a given download. Rather, the Locator is only responsible for returning the widest possible selection of potential Deciders, leaving the choice of the one to be used to other entities (e.g. to the entity that plays the Initiator role).

8.2.2 Decision sub-process

Key aspects in the download process are the decision whether the download can be performed or not, and where to make this decision. This issue was introduced in Subclause 8.1.3. It is assumed that a single Component is to be downloaded on a Target from a remote Repository. The Component has been announced, identified, and located in earlier stages. It will be transferred in a later stage.

In short, the proposed decision process takes properties from the downloadable Component, its Target, and the communication link between the Target and the Component's source, a Repository. These properties are captured in profiles. A Decider evaluates and matches the related profiles to judge whether a download can be achieved. If not, it may manage a negotiation between the Repository and the Target to see if some "best fit" can be achieved. The entity to be downloaded should at least be an Executable Model of the Component; although it may be more than that (this can be parameterized).

When the Repository receives the addresses of the Decider and the Target in the Initiator's message, it contacts the Decider to establish a secure connection with mutual security checks and setups and then it sends to the Decider the Component profile C and the profile of the connection it is capable and willing to establish with Target T. Similarly, when Target T receives the addresses of the Decider and the Repository in the Initiator's message, it contacts the Decider to establish a secure connection with mutual security checks and setups and then it sends to the Decider its own Target profile and the profile of the connection it is capable and willing to establish with the Repository.

The Decider analyzes the Component and Target profiles it received as well as the two connection profiles according to the decision logic as described in subclause 8.1.3. If there are any mismatches and there is a possibility for negotiation, the Decider communicates with the Target and the Repository anew and as many time as compelled by the negotiation protocol.

Once the negotiation procedure is over, a decision about the feasibility of the download is taken by the Decider. If the decision is negative, i.e. download is not feasible, a simple message indicating the termination of the download process is sent by the Decider to the Repository and to the Target. Otherwise, the Decider communicates the positive decision for the download to the Repository and the Target. The same message indicates the decided/agreed parameters of the download (i.e. Component profile, Target profile and connection profile).

Then, the Decider establishes a secure connection with the Initiator in the same way that secure connections were established above and sends to it the final decision regarding the download.

In case of a positive decision for the download, the Decider's involvement in the download process ends after this step. Otherwise, the Decider might be used by the Initiator if the latter tries to initiate a download of Component C to Target T from another Repository. The latter depends on the policies implemented by the Initiator and falls beyond the scope of the Download Framework. In any case, the Initiator's involvement continues until the Component is successfully transferred to the Target or the attempts for transfer fail. These cases are discussed in the remainder of this subclause.

The sequence diagram of the decision sub-process of the download process is given in Figure 6. This figure shows that the Locator's involvement in the download process ends after the Initiator has the addresses of the Repository, the Target and the Decider. What happens after the interactions described by this diagram; depends on the decision communicated by the Decider to the Target, the Repository and the Initiator.

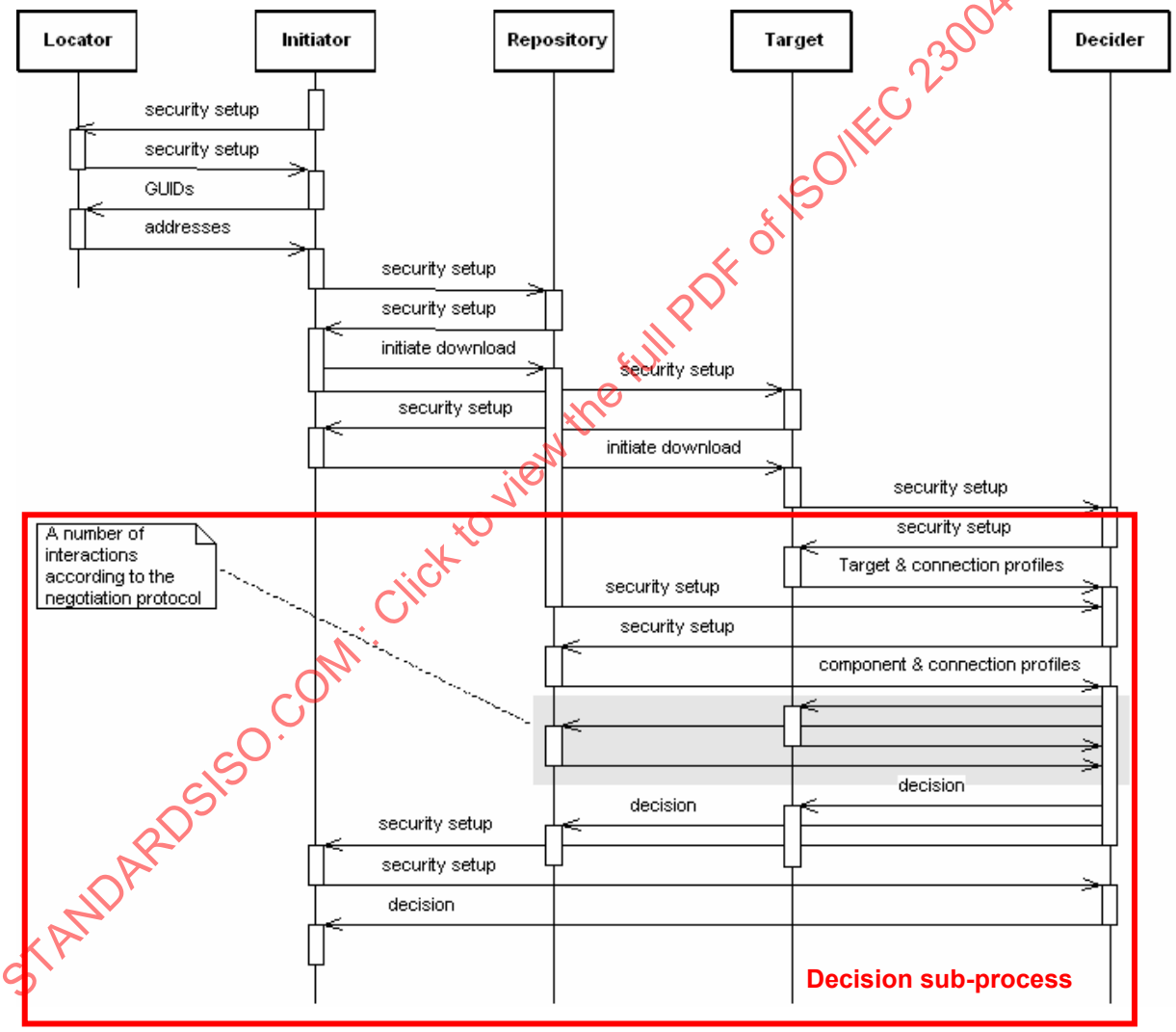


Figure 6 — Sequence diagram highlighting the decision sub-process

If required, a secure link must be constructed between Repository and Target before a download decision takes place. The decision process and its triggered communication protocol may be protected as well as the actual Component data to be downloaded, as it may contain confidential data, possibly susceptible to tempering.

8.2.2.1.1 Profiles

The concept of a Component profile has already been introduced. A Component profile captures the properties of a Component as far as they relate to deciding whether or not the Component can be downloaded onto a given Target.

The Component profile must be available separately from its executable code. It is attractive to realize it as another model in the overall Component definition, next to the executable code. At least these two models should be made available for downloading a Component from a Repository to a Target. The profile flows to the Decider, while the executable code flows to the Target.

The Download Framework does not define the actual contents of profiles, nor their syntax (this depends on the specific realization of the download framework). But to give an idea of the type of properties that may play a role in the decision process we give an informal list of candidate properties for a Component profile:

— the number of bytes to be transferred to the Target,

— stored Component size (it might be difficult to map the Component size to the actual size taken by the Component once downloaded),

— installed Component size,

— instantiated Component size,

— upper limit of the dynamic memory required by a Component instance

— CPU family, version, etc.

— the number of bytes of persistent memory needed by the Component

— Target operating system for which the Component is suited,

— Operating system versions acceptable for the Component,

— the format in which the executable code is transferred (e.g., zip, tar, ...),

— executable file format (e.g. ELF),

— the language of the installed Component code (and maybe the version of the language)

— specific hardware requirements (e.g., clock, display, MPEG decoder),

— specific platform capabilities (not specifically linked to M3W), e.g. a network system required by the Component (e.g., TCP/IP on Ethernet, UDP/IP on UMTS; there can be more than one of these)

— CPU / thread usage, probably expressed in terms of required QoS.

— trading information, like a manufacturer identification,

Target and connection profiles will capture similar data (not discussed here). A Target profile mirrors a Component profile in order for a Decider to match resource "supply" and "demand". An agreement has to be made about interpreting non-specified properties for either profile.

8.2.3 Transfer sub-process

The starting point of the transfer sub-process is when the Decider has communicated the positive download decision to the Repository and to the Target along with each other's contact information. In brief, the transfer sub-process consists of the preparation of the transmission and the actual transfer of the Component data.

The preparation consists of the negotiation between the Target and the Repository regarding the parameters of the connection for the Component transmission.

The transfer may follow either a "pull" scenario or a "push" scenario. In the “pull” scenario the Target initiates the connection with the Repository in order to pull the given Component from the Repository. In the “push” scenario, the Repository initiates the connection to the Target in order to push the given Component to the Target.

A failure may occur at any point during the download process but the only point where a failure concerns the atomic failure semantics of the download process (see 8.2.4) is when it happens during the transfer of the Component from the Repository to the Target.

8.2.3.1 Pull transfer

Assuming a positive decision regarding the feasibility of the download and considering the case where the Target "pulls" the Component from the Repository, the following steps occur in the download process.

First the Target allocates the necessary resources for the download. These resources conform to the outcome of the Decider's decision about the download, i.e. following the results of the negotiation phase, the agreed Runtime resources and QoS guarantees required by the download are allocated.

Then, the Target establishes a secure connection with the Repository by mutual security checks and setup. After that, the Target initiates the configuration of the other parameters of the connection. These parameters include the transfer bandwidth, timeout settings, and any other necessary information. The configuration parameters for the connection and their values are those decided/agreed by the Decider while processing the connection profiles of the Target and the Repository.

Once the connection is configured the Repository starts the transmission of the Component.

When the transmission is completed, the Target notifies the Repository about the success or failure of the Component transmission. Then, the Target notifies the Initiator. The Repository also notifies the Initiator about the outcome of the Component transfer after it receives the transmission notification from the Target. Hence, the Initiator is aware of whether the download was successful or not. If the two notifications from the Target and the Repository to the Initiator do not agree (one claims success while the other claims failure), then the Initiator could take a number of steps for restoring the consistency of the entities that have participated in the download. This is beyond the scope of the Download Framework. The remainder of this subclause only deals with the case of a transmission success, leaving the failure case for consideration in subclause 8.2.3.3.

Right after the emission of the notification of a successful Component transmission (and hence reception by the Target) the Component is found on the Target and it is in the "resident Component" lifecycle stage. At this point, any open connections from the Target to the Repository and to the Initiator are closed, and any resources allocated for the download by the Target may be released.

Before Services or Libraries contained in the Component can be instantiated, the Component and the Services/Libraries must be registered with the RRE. This is outside the scope of the Download Framework, however.

Figure 7 illustrates graphically the action and message sequence in the case of the Component being "pulled" from the Repository by the Target.

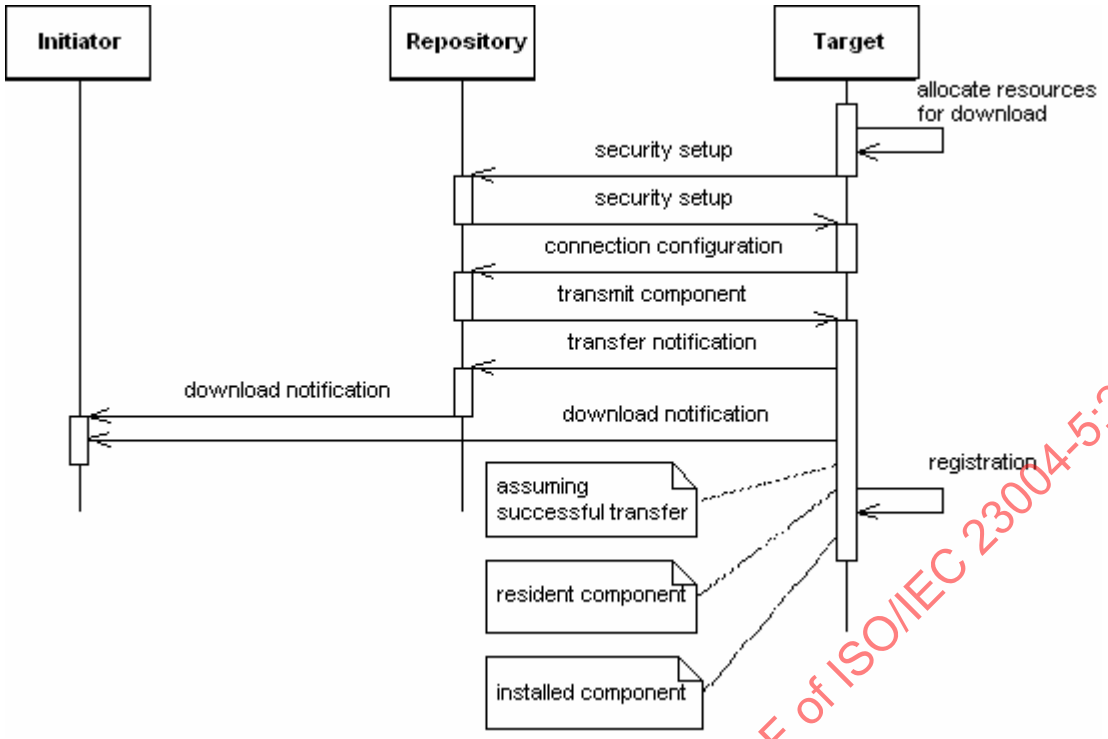


Figure 7 — Sequence diagram of Pull Transfer

8.2.3.2 Push transfer

The push transfer, where the Repository "pushes" the Component to the Target, is similar to the pull case, except from the first few steps.

In the push transfer the Repository takes the first step and establishes a secure connection with the Target by mutual security checks and setup.

After that, the Repository initiates the configuration of the other parameters of the connection. Similarly to the pull transfer, these parameters include the transfer bandwidth, timeout settings, and any other necessary information. The configuration parameters for the connection and their values are those decided/agreed by the Decider while processing the connection profiles of both the Target and the Repository.

When the connection configuration is completed, the Target allocates the necessary resources for the download. Similarly to the pull transfer, these resources conform to the outcome of the Decider's decision about the download, i.e. following the results of the negotiation phase; the agreed Runtime resources and QoS guarantees required by the download are allocated. Once these resources are allocated, the Target informs the Repository that it is ready to receive the Component.

After this point, the same actions take place and the same messages are exchanged as in the pull transfer. In other words, the Repository starts the transmission of the Component, the Target notifies the Repository about the success or failure of the Component transmission, the Target notifies the Initiator, and the Repository also notifies the Initiator. Right after the emission of the notification of a successful Component transmission (and hence reception by the Target) the Component is found on the Target and it is in the "resident" lifecycle stage. At this point, any open connections from the Target to the Repository and to the Initiator are closed.

Figure 8 graphically illustrates the action and message sequence in the case of the Component being pushed from the Repository to the Target.

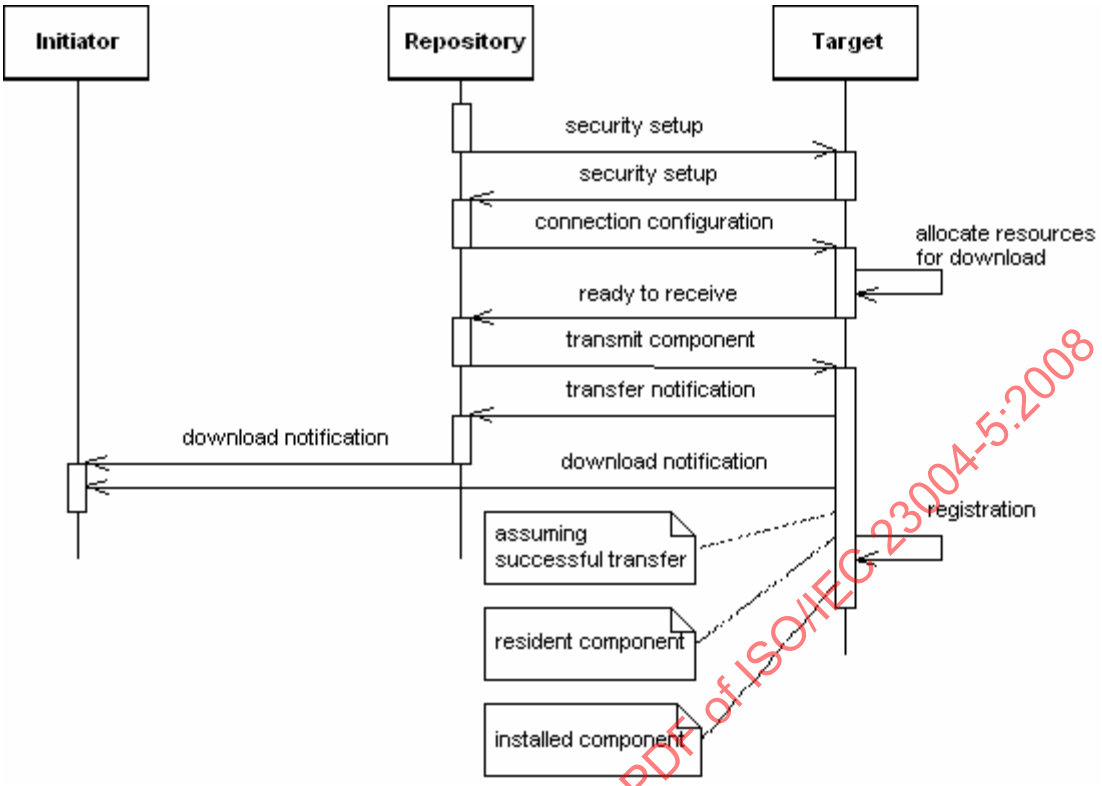


Figure 8 — Sequence diagram of push transfer

8.2.3.3 Transmission failure

Failures can occur at any point of time during the download process, e.g. when the Initiator interacts with the Locator, the Target or the Repository, when the Decider interacts with the Target or the Repository, or even when the Target interacts with the Repository. Most of these failures do not have an impact on the consistency of the Target. For example, a failure that occurs right after the allocation of the download resources by the Target may necessitate just the release of these allocated resources as the recovery measure. However, if a failure occurs during the transmission of the Component from the Repository to the Target, then intermediate download results may exists at the Target. This violates the atomic semantics of the download process and requires recovery measures which will restore the consistency of the Target. The remainder of this subclause focuses on this failure case.

As a starting point, we assume that the download process has proceeded until the point of Component transmission (whether it is a "pull" or a "push" case is irrelevant). During the Component transmission a failure occurs. Then, the Target notifies the Repository about the failure of the transmission. Depending on the fault tolerance protocol in use, a number of retransmission attempts may follow. Assuming that during all the retransmission attempts a failure occurs, the Target keeps on notifying the Repository about the transmission failure. After the reception of the last (according to the fault tolerance protocol) notification of transmission failure, the Repository may take a number of "house keeping" steps, e.g. record in some log-file the failure of the attempted download.

At the Target side, after the emission of the last notification regarding the failure of the Component transmission, the Target proceeds by releasing the resources allocated for the given Component download.

Then, the Target removes all intermediate products of the unsuccessful Component transmission. This action includes the release of the memory and/or storage space where the incompletely received Component data resides.

Finally, the Target takes appropriate actions for restoring itself to a consistent state. These actions of consistent state restoration depend on the specific characteristics of the download. For example, in the case

where due to memory limitations the memory space used by the data of the unsuccessfully downloaded Component was previously occupied by a Component which was to be replaced by the newly downloaded one, the Target might have to initiate the download of the old Component. Alternatively, the Target might initiate the download of the same Component that failed previously, only this time request to download from a different Repository. The exact actions taken for the restoration of a consistent state at the Target are outside the scope of the Download Framework.

Figure 9 graphically illustrates the action and message sequence in the case of a failure occurrence during the transmission of the Component from the Repository to the Target.

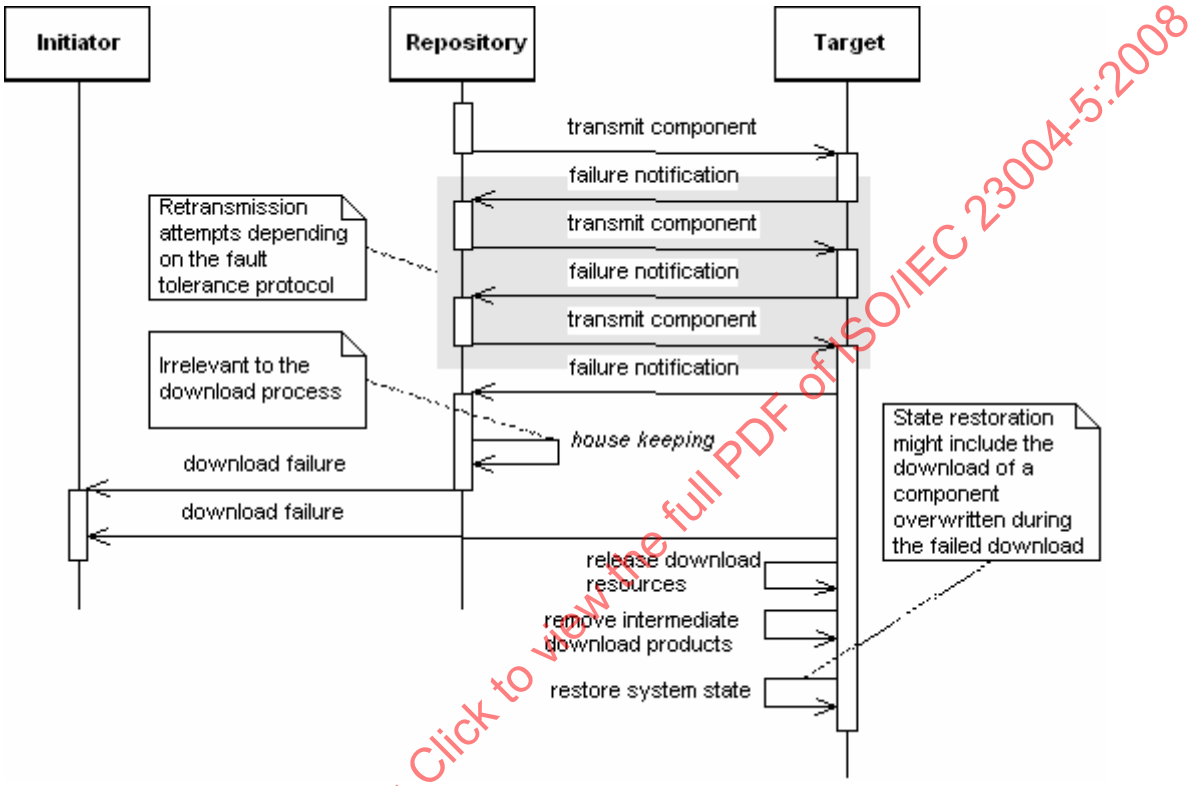


Figure 9 — Sequence diagram of transmission failure

8.2.4 Failure semantics

In order to be practically useful, the Component download process must satisfy certain failure related properties regarding the behavior of the Target that is involved in the download of a Component. To state the obvious, the outcome of a download must not crash the Target or bring it in an otherwise inconsistent state. Also, the occurrence of failures during the download process must not have a long term impact on the Target. In general terms, the download process and the failures that may occur during the course of it must not cause a failure of the Target.

8.2.4.1 Component download seen as a transaction

From the Target viewpoint, a download process must either happen to completion and leave the device in a consistent state or appear as if it was never attempted. In other words, the download process must be an atomic transaction for the Target. Atomic transactions (or just transactions for brevity) are actions which appear to occur indivisibly with respect to failures and to other concurrent transactions.

Transactions satisfy the ACID properties, i.e. they are Atomic, Consistent, Isolated and Durable. Atomicity means that a transaction either reaches successful completion or all the side products are removed and it appears as it was never attempted. Consistency means that a transaction preserves system invariants, i.e. it

does not bring the system to an inconsistent state. Isolation (sometimes called serialize-ability) means that the outcome of a transaction is not affected by other transactions that happen concurrently, i.e. the overall result of the concurrent transaction is the same as if they would have been executed sequentially in some order. Finally, durability means that the effects of a completed transaction are very likely to survive subsequent failures (except maybe from catastrophic system failures) and system resets.

Component download in the Download Framework must satisfy the ACID properties too. The download process must be atomic since, unless the download is successfully completed, all side products of the download attempt must be removed from the Target (which includes the release of the resources allocated for the download). The download process, whether successfully completed or not, must leave the Target in a consistent state. Moreover, if more than one download process occurs at the same time, there must be no interference among the resources allocated for each of them. Finally, the results of a successful download process must be durable, i.e. the downloaded Component must still be resident[1)] at the Target after transient Target failures or resets.

8.2.4.2 Failure model

In order to deal with the behavior of a system in the presence of failures, the types of failure that are taken into consideration must be specified, i.e. the failure model of the system must be described. For the Component download in the context of the Download Framework, the failure model considers only crash and omission failures:

— Component crash. The unit of failure in a M3W system is the Service Instance, which means that a Service Instance is considered to have failed when any part of its behavior indicates a failure. The failure model specifies Service Instance failures with crash semantics, i.e. a Service Instance fails by ceasing its interactions with its environment without directly contaminating it with failure products. In other words, when a Service Instance fails, there are no messages sent to any other Service Instancess inside the given Target. The failure can be detected by the other Service Instances that are bound to the failed one by means of timeouts.

— Target crash. An entire Target may fail by crashing. This is equivalent to a simultaneous crash failure of all of the Service Instances on the given Target. The failure can be detected by the Hosts in the surrounding environment of the failed one by means of timeouts.

— Host crash. In the download process, not all participants need to be M3W systems (e.g. the Repository). Any non-M3W system may experience crash failures, which means that it instantly ceases any communication with other Hosts and Targets in its surrounding environment. The failure can be detected by the other Hosts and Targets in the surrounding environment of the failed Host by means of timeouts.

— Communication link breakdown. When a Target communicates with a Host (e.g. the Target communicates with the Repository), the communication link may fail by stopping the transmission of messages. However, the communication link cannot fail by delivering corrupted (or incomplete) messages, or by delivering multiple copies of the same message or delivering messages that were never sent.

Byzantine failures which are failures where the failed entity exhibits arbitrary (possibly malicious) behavior are left outside the scope of the failure model.

8.2.4.3 Atomicity

The atomicity property of the download process must be ensured by a fault tolerance protocol that governs the "Component transfer" phase in the Component download lifecycle. The outline of a minimum behavior that such a fault tolerance protocol should offer on the Target is given below:

— If a failure occurs on the Target, on the Repository, or on the communication link after the connection between the Repository and the Target is established but before the Repository has sent any Component

1) This assumes a Target with some form of persistent storage.

data, then the Target (after recovering from its own failure if necessary) has only to release the resources allocated for the download.

— If a failure occurs on the Target, on the Repository, or on the communication link during the transfer of the Component from the Repository to the Target, then the Target (after recovering from its own failure if necessary) must release the resources allocated for the download and remove from its memory any intermediate Component data that have already been transferred. Optionally, in case of a transient failure, the Target may notify the Repository about the failure during the transfer and the Repository may attempt again to transfer the Component. In this case, the Target must use exactly the same resources that were initially allocated for the download.

8.2.4.4 Consistency

The consistency property of the download process is ensured by the Decider. For the download of a Component C to a Target T, the entity which plays the role of the Decider is responsible for detecting potential problems that C might cause to the consistency of T and subsequently taking a negative decision for the download of C to T. If the Decider takes a positive decision for the download of C to T and after the successful download of C the Target T reaches an inconsistent state, this will be considered as a failure which occurred to the Target T. Hence, the Component and Target profiles must contain sufficient information for the Decider to make the necessary consistency analysis and take the decision regarding the download accordingly.

8.2.4.5 Serialize-ability

The serialize-ability property of the download process is partially ensured by the resource manager at the Target. The resource manager is responsible for controlling the Target resources and allocating the necessary resources for a Component download. Hence, it is its responsibility to ensure that there are no race conditions or other interference among the resources allocated for different download processes that happen to occur concurrently.

8.2.4.6 Durability

The durability property of the download process is partially ensured by the persistent storage facilities at the Target. The persistent storage must guarantee that its contents remain unaffected by transient Target failures and by reset actions on the Target.

8.2.5 Security

The download process includes the respect of certain security properties for the communication channels established between the download roles.

The roles involved in the download process have been described in the subclause 8.1.

The roles communicate in order to achieve the successful Component download. The communication may have to be secured and then must respect the security properties decided a priori. Every path may have different security requirements and different levels of strength. For example, some communication path may need the parties to be mutually authenticated while in others cases only a secure encrypted communication channel may be established. Specifying a channel as encrypted or authenticated is not enough. The parameters, i.e. the algorithm used and the key length, for each security property desired must also be described. All of these properties must be addressed in the setup.

The basic security requirements for the download process in M3W are:

— Authentication: It should be possible for the receiver of a message to ascertain its origin; an intruder should not be able to masquerade as someone else.

— Integrity: It should be possible for the receiver of a message to verify that it has not been modified in transit; an intruder should not be able to substitute a false message for a legitimate one.

— Non-repudiation: A sender should not be able to falsely deny later that it sent a message. A receiver should not be able to falsely deny later it received a message.

— Confidentiality: A sender should be able to send an encrypted message; only the legitimate receiver can read it.

Confidentiality is usually provided through encryption. Authentication, data integrity and non-repudiation are usually provided through digital signature and public key certificates. This does not exclude that these features cannot be provided by other means as long as they adhere to the M3W specifications.

8.2.5.1 Security setup

Every time a communication channel needs to be established between different entities, a security setup is carried out before any data is exchanged. The security setup is the procedure in which the parties agree upon the security level and in which all necessary steps are performed and messages exchanged to create the secure channel for the data exchange.

The security setup can be void. In this case, no security mechanism is required. It happens, for instance, when the two roles interacting are played by the same entity. In other cases, the security setup includes the achievement of one or more of the security requirements.

For the security setup the different roles need to agreement on a common representation that describe the individual security capabilities of the different roles. A possible representation is given in Annex B.

Every communication path between two entities participating in the Download Framework has a set of the security requirements assigned. Table 11 shows the dependencies between the roles and the requirements. Every communication path between the roles in the download process is present in the table. For each path the mandatory features are indicated. The features not mandatory are optional.

Path	Authentication	Integrity	Encryption	Non-repudiation
Locator-Initiator	√	√		
Initiator-Repository	√	√		
Initiator-Target	√	√		
Target-Decider	√	√	√	
Repository-Decider	√	√	√	
Target-Repository	√	√	√	√

Table 11 — Path Security Feature dependency

Using the common secure features as criteria, the interactions between the roles can be grouped in three different sets. The first set needs the mutual authentication of the endpoints and integrity, but not confidentiality and not non-repudiation. The second set has as requirements the mutual authentication of the endpoints, integrity and confidentiality, but not non-repudiation. The third set has the all the security features.

The end-points must agree on the security properties of their communication path. The parameters on the requirements could be negotiated to some extend. For example, one party could ask for an encrypted link and the other could offer a choice between different encryption algorithms available. An agreement on the security properties should be reached before proceeding to the next phases, otherwise the communication aborts.

The security level of each feature depends on:

— The roles involved in the communication

— The transport protocol stack where the roles reside

The basic requirements in terms of secure algorithms, key lengths and how the security features are achieved are system specific and outside the scope of the Download Framework. This specification only addresses the setup phase and which security features are involved in each communication path.

Three different conceptual sequence diagrams are drawn to illustrate the security setup phases.

8.2.5.1.1 Authentication and integrity

In this case, the security setup purpose is to establish a communication channel with the endpoints mutually authenticated. Moreover, the integrity feature assures nobody can tamper with the data. The sequence diagram in Figure 10 graphically illustrates the actions and the messages sequence of this phase. An explanation of the diagram follows below. In this example the parties are the Initiator and the Locator. The procedure will be the same for the communication between different parties that have the same security requirements.

The parties exchange hello messages to identify themselves. It is a simple notification that the negotiation process is started. Included in the hello messages there are some parameters to be agreed upon for securing the connection.

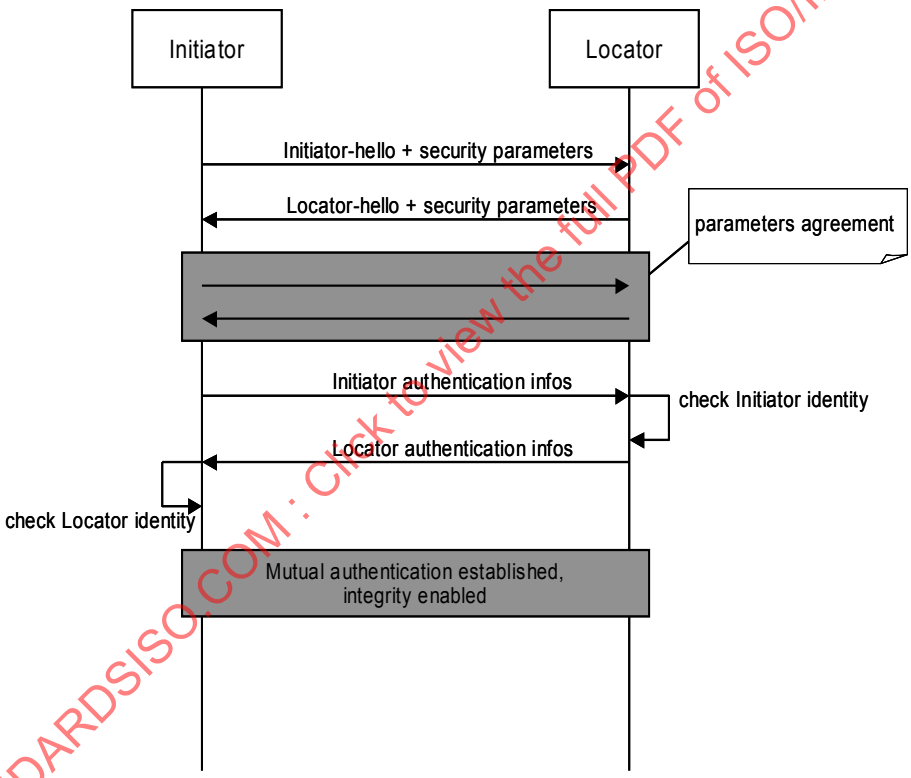


Figure 10 — Establishment of Authentication and Integrity

Examples of parameters could be: protocol version, algorithms for authentication, algorithms for integrity check, keys length. Messages are exchanged to negotiate the parameters. If and agreement is reached the setup continues.

The Initiator sends its authentication info to the Locator. For instance a digital-signed certificate is sent. The Locator checks the identity of the Initiator verifying the information received. Because we are dealing with mutual authentication, the Locator sends its credentials to the Initiator. If the checks are passed, the mutual authentication is achieved. All data packets exchanged include message integrity checks. Cryptographic hashes, which include a secret key as a part of the calculation, may be used to verify that no changes to data have been made. At this point, the security setup phase is ended. An authenticated and tampering-proof channel has been created.

8.2.5.1.2 Authentication, integrity and encryption

The case of security setup with authentication, integrity and encryption is similar to the previous diagram except for the last few steps. In this case, after the parties are mutually authenticated, it is necessary to set up the encryption of the channel. The block highlighted in gray in Figure 11 as “key exchange” has the purpose to establish a common encryption key to secure the channel. The authentication phase has already been carried out and the information derived from it may be used to create the common secret key.

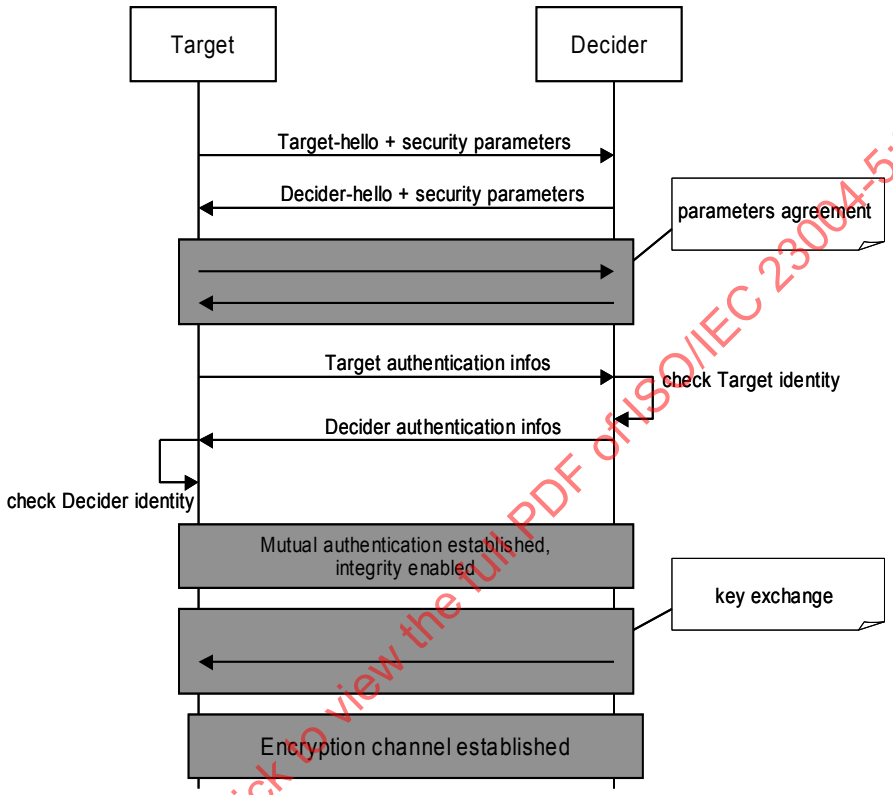


Figure 11 — Establishment of Authentication, Integrity and Encryption

The common secret key may be established using a key exchange algorithm. Key exchange algorithms are a special class of public key algorithms that do not encrypt data but do allow two sides of a transaction to figure out the same unpredictable number. In this part of ISO/IEC 23004 we are not dealing with the choice of the algorithm used. The gray “key exchange” box contains the messages necessary to create the common secret key used to encrypt and decrypt the data. The sequence is specific to the particular key agreement algorithm chosen.

8.2.5.1.3 Authentication, integrity, encryption and non-repudiation

This case applies when a communication channel is established between the Target and Repository. The non-repudiation phase is not part of the setup performed in the beginning of the communication, but it is still part of the security requirements between the endpoints under discussion.

The messages exchanged for the security setup between the Target and the Repository is exactly the same as described in Figure 11. The non-repudiation phase starts when the Repository transmits the Component to the Target. The feature is intended to protect against the originator of the message denying having sent the message, and to protect against the receiver denying having received the message. For instance, the Target may deny having received the Component and gains advantage by not paying for it.

The following assumes that the communication channel is reliable, and that authentication and encryption between the Target and the Repository has already been established.

The non-repudiation protocol can be different depending on the assumptions:

— If the communicating parties play fair, the non-repudiation protocol is simple. From Figure 7 and Figure 8 we only need to add a digital signature to the “Transmit Component” and “Transfer notification” messages. The signature may contain information about the downloaded Component, the time of downloading it, and the identification of the entities. After the two messages have been exchanged, the parties cannot deny having received or sent the message.

— If the communication parties do not necessarily play fair, the above protocol leaves the Target or the Repository in an advantageous situation. For instance, the Repository sends a message to the Target; the Target simply refuses to send an acknowledgment. The problems could be addressed by invoking Services of a trusted third party, but that is beyond the scope of the Download Framework.

With as basic assumption that the parties play fair, the non-repudiation phase is described by the first bullet and by the Figure 12.

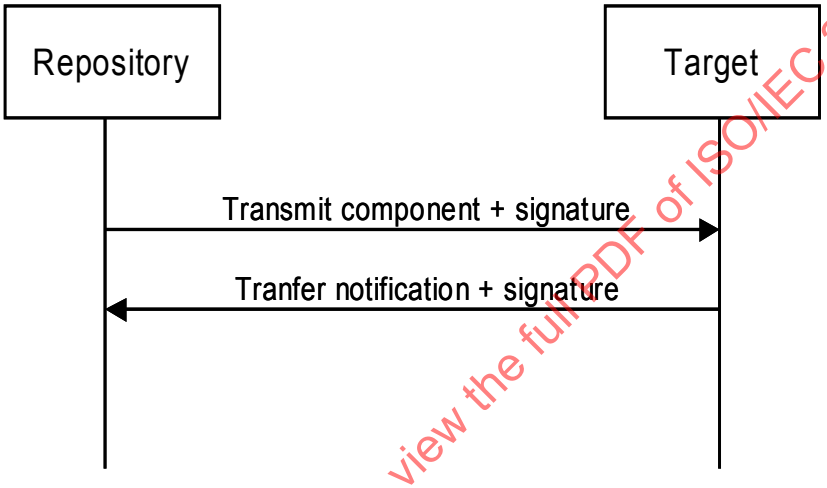


Figure 12 — Sequence diagram with non-repudiation

8.3 Deployment

Deployment of the roles in the download framework is very flexible in order to support different download (and upload) scenarios, point to point as well as broadcast. A number of examples of deployment of the individual roles are discussed in Annex A.

Annex A
(informative)

Deployment example

A.1 Introduction

This Annex presents some different possibilities for deployment of the download roles. Two different examples (each one with two different alternatives) will be described with different deployment of roles (Initiator, Locator, Decider) over the Target, Repository, and third party Hosts. After the overall description, an example of the concrete sequence of the downloading process will be showed. In each example the actual transfer process can be started by the Repository or by the Target (push or pull transfer).

A.2 Single manufacturer example

A.2.1 Initiator outside terminal

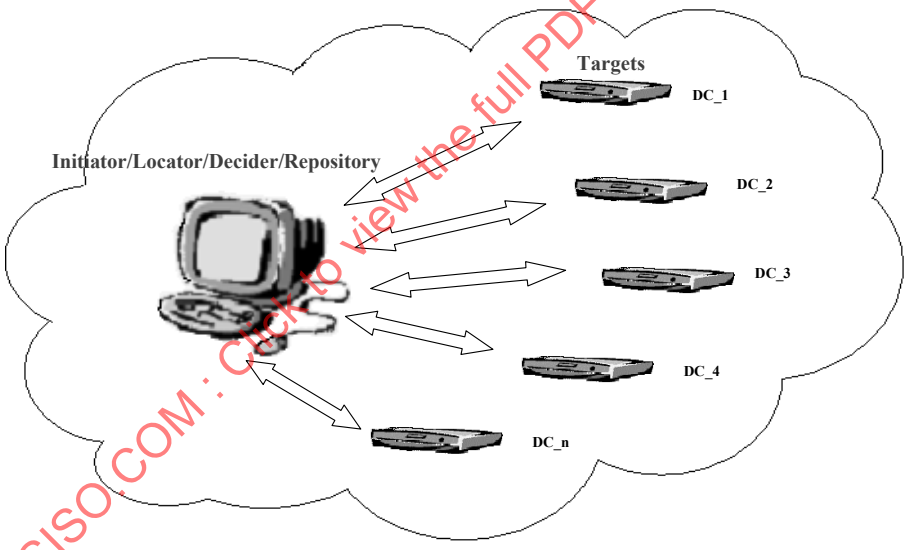


Figure A.1 — Initiator outside the terminal

In this scenario there are many devices connected through a private network or through the global Internet. The manufacturer of the device has the Repository Host assigned to the task of downloading new Components to the Targets. Each device is built on the same platform (hardware & software) and is physically located at different places.

The manufacturer decides to upgrade a Component in each device. It's a multicasting scenario. The manufacturer's Repository plays the roles: Initiator/Locator/Decider. This scenario is depicted by the following sequence diagram.

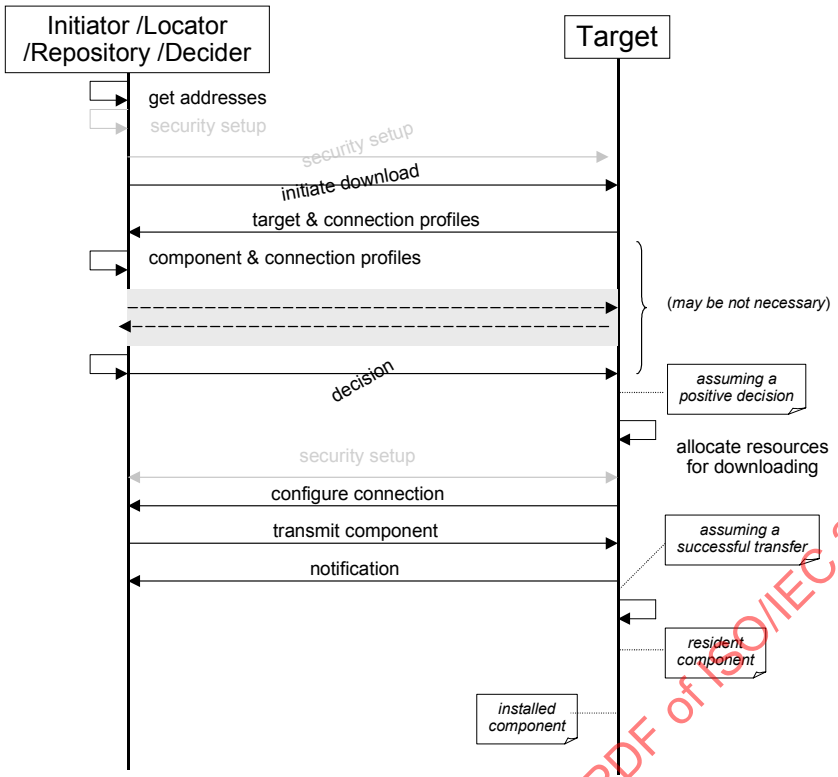


Figure A.2 — Sequence diagram initiator outside target

A.2.2 Initiator on terminal

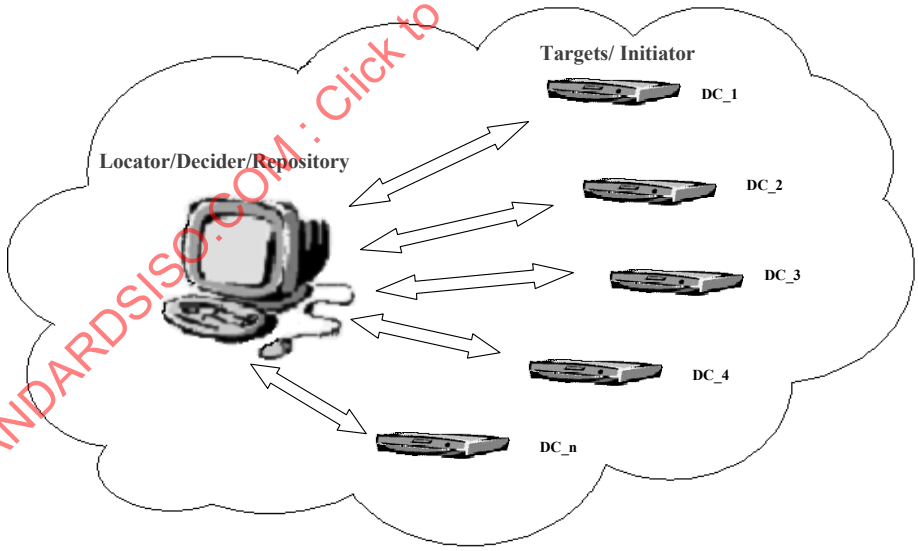


Figure A.3 — Initiator on target (single manufacturer)

This scenario is similar to the previous one but, in this case, the Initiator is the Target itself. The manufacturer has the Repository Host assigned to the task of downloading new Components in the Targets. A Target has detected a problem and decides to download a Component in order to analyze and repair the problem. The Target plays the Initiator role, and connects by default with the manufacturer's Repository which assumes the remaining roles: Locator/Decider. This scenario is depicted by the following sequence diagram.

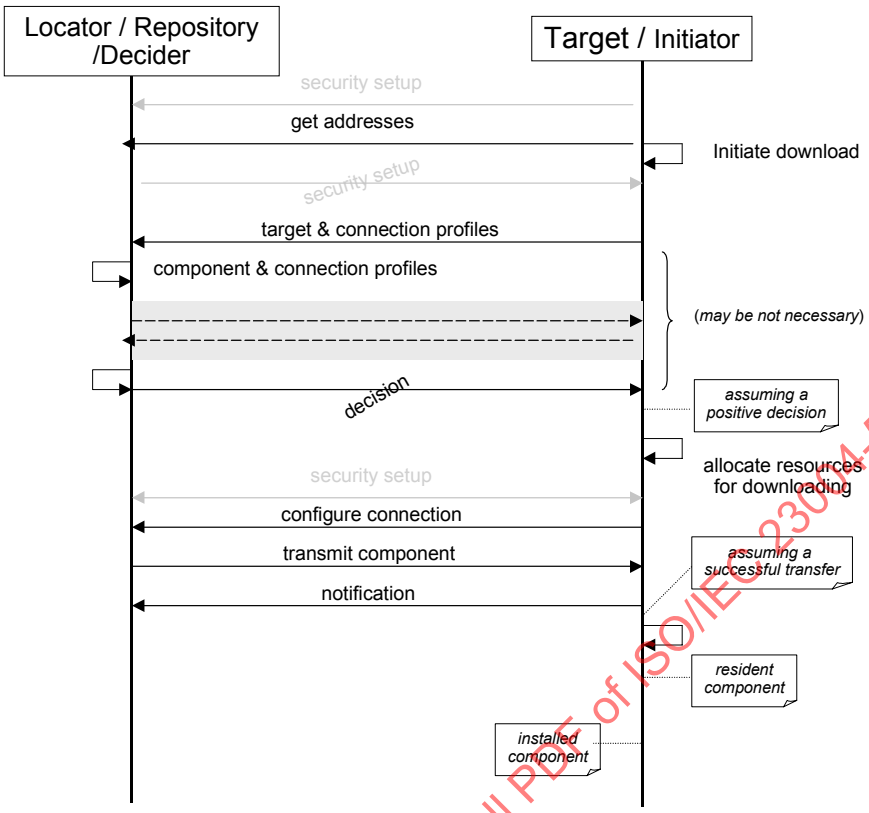


Figure A.4 — Sequence diagram initiator on target (single manufacturer)

A.2.3 Broadcast reception

This scenario describes the broadcast reception of components. DSM-CC object or data carousels technology [13] can be used to transfer components as part of a broadcast stream. The download framework can be used to get components from the broadcast stream. In this case all the roles that are part of the download framework are deployed on the M3W device. The Repository role is responsible for the reception of the components that are part of the broadcast stream. The Repository can cache these components if this is needed. The components that are part of the stream are in the repository. When an initiator initiates "download" this will result in transfer of the component from the repository to the target (copying the executable component).

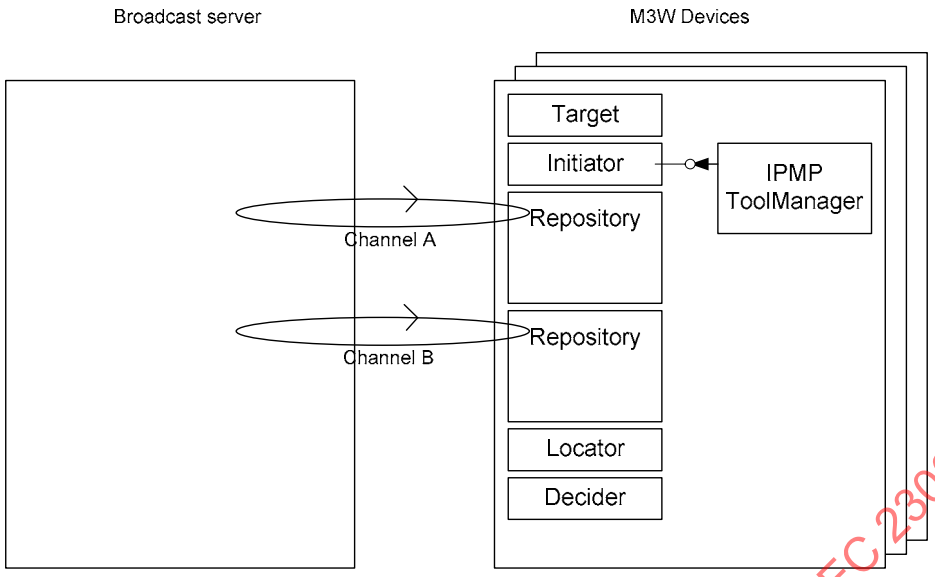


Figure A.5 — Deployment: Single manufacturer broadcast

The scenario described and illustrated above is often used to get IPMP Tool from a broadcast stream. An IPMP Tool Manager identifies the need for an IPMP Tool and uses the initiator to start the retrieval of this IPMP Tool from the broadcast.

A.2.4 Mixed broadcast and point to point

This scenario is an extension on the scenario described in A.2.3. The deployment depicted in Figure A.11 — Deployment: Multiple manufacturer broadcast could support retrieval of components from the broadcast stream if the requested components as well as download from a manufacturer repository (point to point) in case that the requested component is not part of the broadcast.

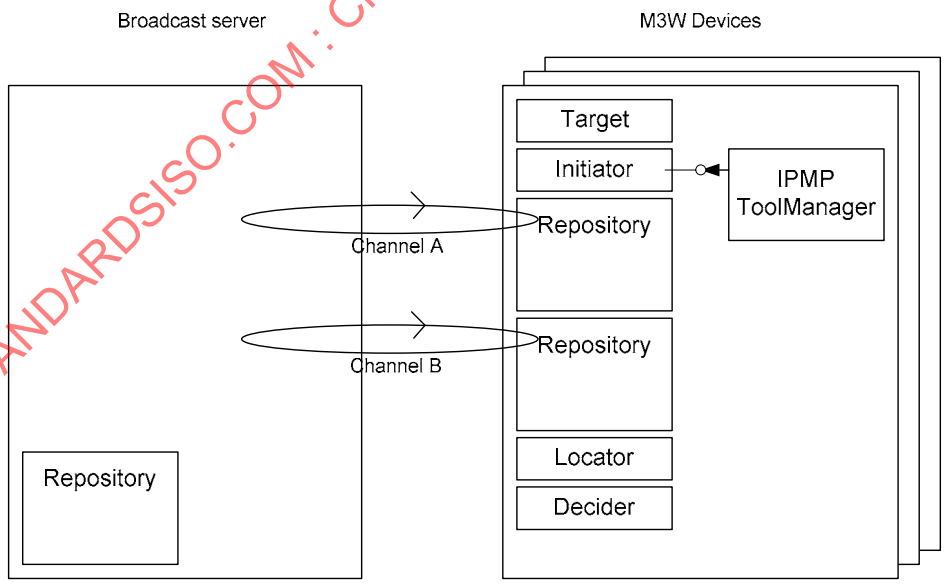


Figure A.6 — Deployment: Mixed broadcast and point to point

A.3 Multiple manufacturers example

A.3.1 Initiator outside target

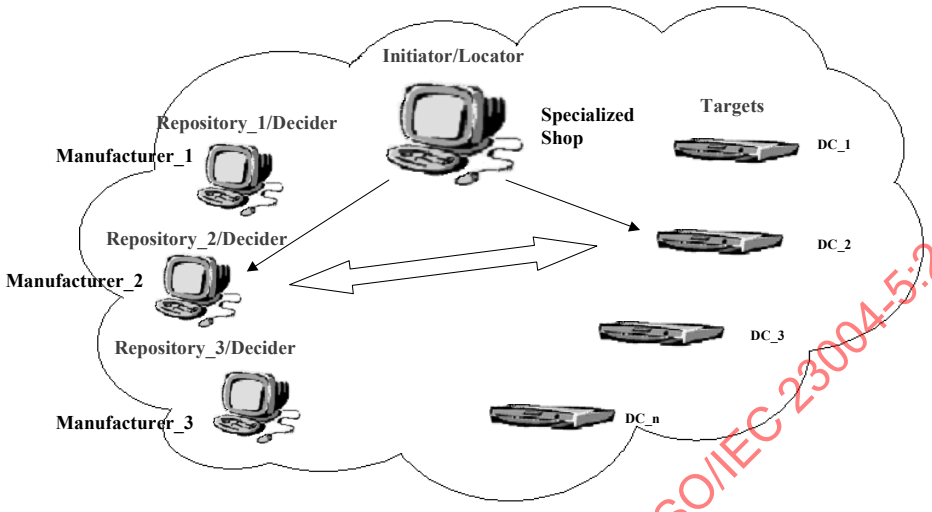


Figure A.7 — Initiator outside target (multiple manufacturers)

In this scenario there are many manufacturers of different devices. Each has developed Components for their devices. The user goes to a specialized shop and, after having bought a new device, the shop's seller starts the process of installing the most recent Component(s) for that device.

The shop's Host has the roles of Initiator and Locator. There could be many Repositories from a specific manufacturer. The Locator will choose the appropriated one following various criteria like: geographical proximity, computer load and so on. The Repository will play the Decider role.

The scenario mentioned above is illustrated using the following sequence diagram.

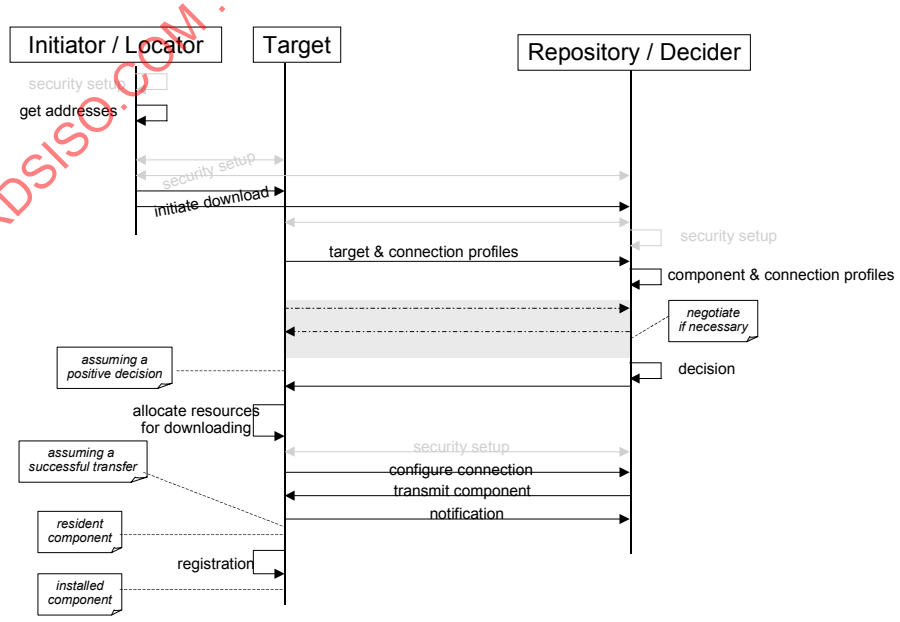


Figure A.8 — Sequence diagram initiator outside the target (multiple manufacturers)

A.3.2 Initiator on target

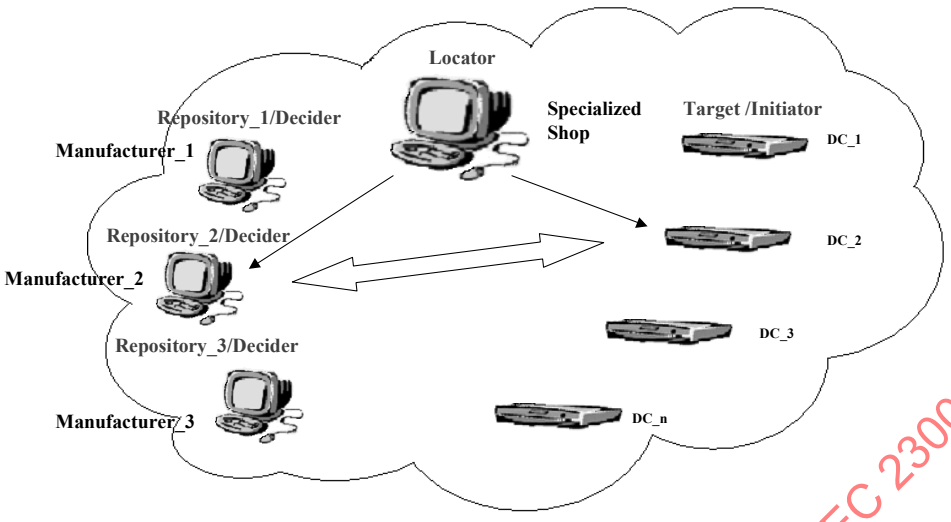


Figure A.9 — Initiator on target (multiple manufacturers)

This scenario is very similar to the previous one, but now the Target will play the role of the Initiator. It still uses the specialized shop, however, where the shop's Host plays the Locator role. The Decider role is still played by the manufacturer's Repository. This scenario is illustrated by the following sequence diagram:

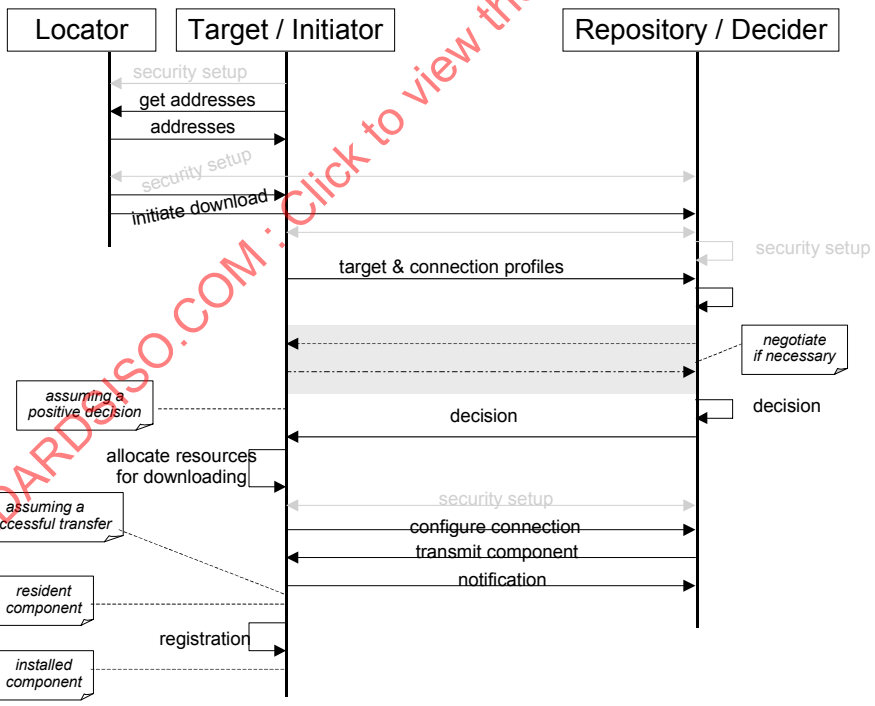


Figure A.10 — Sequence diagram initiator on target (multiple manufacturers)

A.3.3 Broadcast reception

This scenario describes the broadcast reception of components from multiple manufacturers. This example is similar to the example described in A.2.3. The difference is that multiple broadcast servers are involved.

DSM-CC object or data carousels technology [13] can be used to transfer components as part of a broadcast stream. The download framework can be used to get components from the broadcast stream. In this case all the roles that are part of the download framework are deployed on the M3W device. The Repository role is responsible for the reception of the components that are part of the broadcast stream. The Repository can cache these components if this is needed. The components that are part of the stream are in the repository. When an initiator initiates "download" this will result in transfer of the component from the repository to the target (copying the executable component).

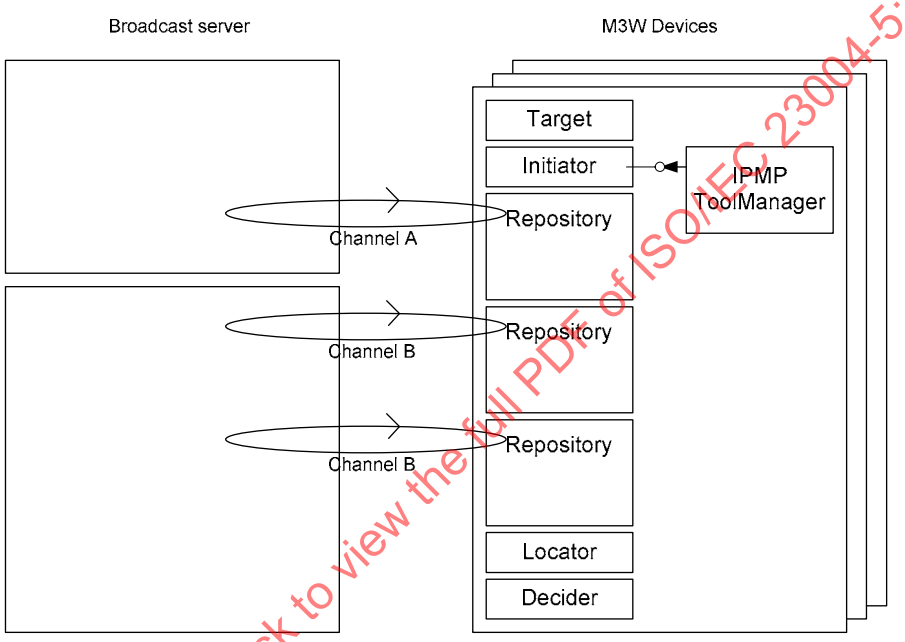


Figure A.11 — Deployment: Multiple manufacturer broadcast

The scenario described and illustrated above is often used to get IPMP Tool from a broadcast stream. An IPMP Tool Manager identifies the need for an IPMP Tool and uses the initiator to start the retrieval of this IPMP Tool from the broadcast.

Annex B
(informative)

Meta data for security parameters

B.1 Example metadata schema for security parameters

This annex describes a metadata schema for modeling the supported security mechanism in M3W in XML representation. Figure B.1 — metadata structure for modeling M3W security mechanism, shows the structure of the proposed metadata schema for security parameters. It contains four major elements to carry information such as the protocols for data transfer, authentication, signature and encryption. All the elements declared and their data types defined in the security metadata schema is based on the M3W requirements for secure download (which we think also applicable to other areas such as for remote service invocation).

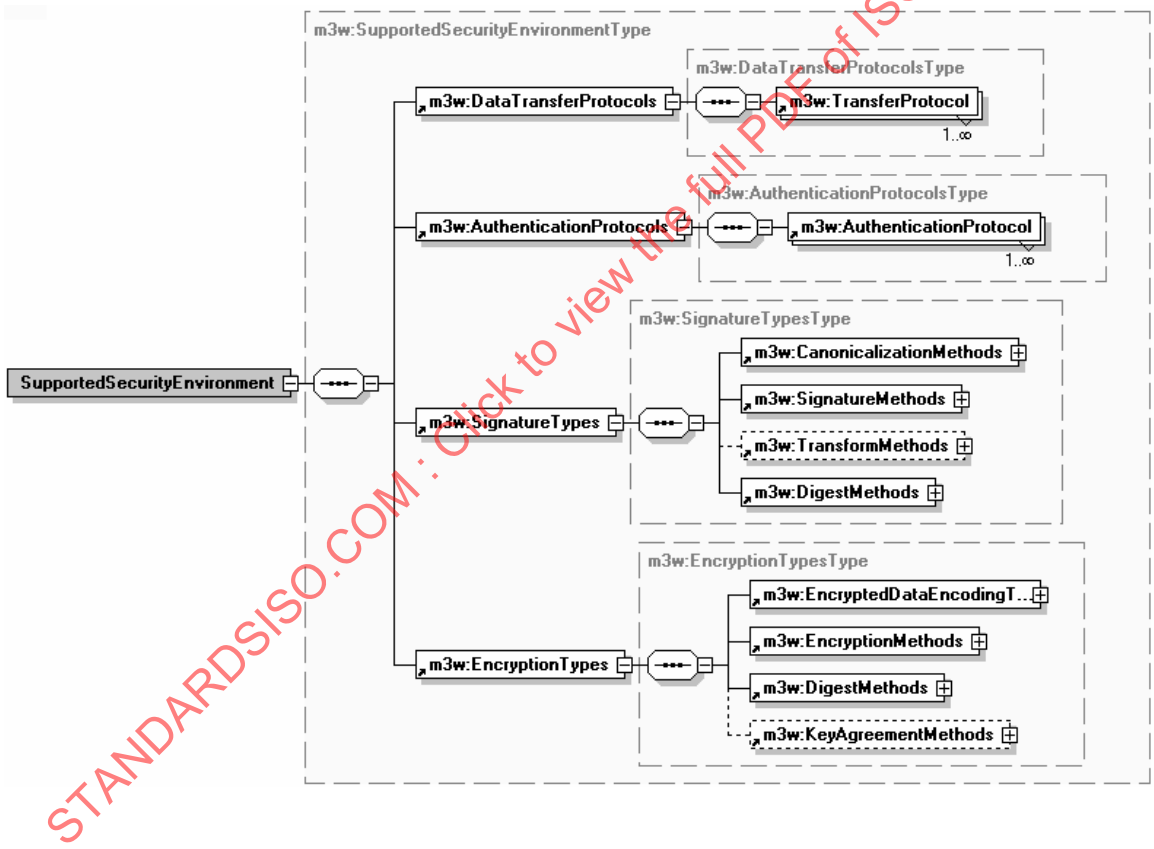
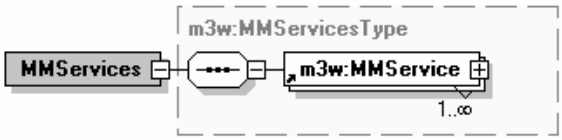


Figure B.1 — metadata structure for modeling M3W security mechanism

B.2 Syntax and semantics

B.2.1 SupportedSecurityEnvironment

Syntax

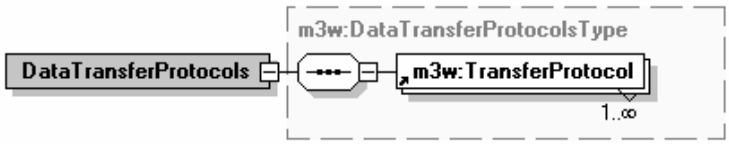
Diagram	
Used by	-
Children	<DataTransferProtocols><AuthenticationProtocols><SignatureTypes><EncryptionTypes>
Source	<element name="SupportedSecurityEnvironment" type="m3w:SupportedSecurityEnvironmentType"/> <complexType name="SupportedSecurityEnvironmentType"> <sequence> <element ref="m3w:DataTransferProtocols"/> <element ref="m3w:AuthenticationProtocols"/> <element ref="m3w:SignatureTypes"/> <element ref="m3w:EncryptionTypes"/> </sequence> </complexType>

Semantics

The `SupportedSecurityEnvironment` is a top level element that carries information of security environment supported by an M3W implementation. It provides a container to carry data supported transfer protocols, supported authentication protocols, supported digital signature, and supported encryption methods.

B.2.2 DataTransferProtocols

Syntax

Diagram	
Used by	SupportedSecurityEnvironment
Children	<TransferProtocol>
Source	<element name="DataTransferProtocols" type="m3w:DataTransferProtocolsType"/> <complexType name="DataTransferProtocolsType"> <sequence> <element ref="m3w:TransferProtocol" maxOccurs="unbounded"/> </sequence> </complexType>

Semantics

DataTransferProtocols is a container that carries a list of supported transfer protocols.

B.2.3 TransferProtocol

Syntax

Diagram				
Used by	DataTransferProtocols			
Attributes	**Name**	**Type**	**Use**	**Semantics**
	algorithm	anyURI	Required	Name of the supported protocol algorithm. The value of algorithm attribute follows the URI convention recommended by W3C
	preference	int	Required	Preference level of the algorithm usage. The higher the value the more preferred the algorithm is.
Source	<element name="TransferProtocol" type="m3w:MethodType"/> <element name="Method" type="m3w:MethodType"/> <complexType name="MethodType"> <attribute name="algorithm" type="anyURI" use="required"/> <attribute name="preference" type="int" use="required"/> </complexType>			