INTERNATIONAL STANDARD

ISO/IEC 19763-5

First edition
2015-08-15

Information technology — Metamodel framework for interoperability (MFI) —

Part 5: Metamodel for process model registration

Technologies de l'information — Cadre du métamodèle pour l'interopérabilité (MFI) —

Partie 5: Métamodèle pour l'enregistrement du modèle de procédé

Reference number
ISO/IEC 19763-5:2015(E)

ISO copyright office
Ch. de Blandonnet 8 • CP 401
CH-1214 Vernier, Geneva, Switzerland
Tel. +41 22 749 01 11
Fax +41 22 749 09 47
copyright@iso.org
www.iso.org

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 19763-5 was prepared by Joint Technical Committee ISO/IEC JTC 1, *Information Technology*, Subcommittee SC 32, *Data management and Interchange*.

ISO/IEC 19763 consists of the following parts, under the general title *Information technology — Metamodel framework for interoperability (MFI)*:

— *Part 1: Framework*

— *Part 3: Metamodel for ontology registration*

— *Part 5: Metamodel for process model registration*

— *Part 6: Registry summary*

— *Part 7: Metamodel for service model registration*

— *Part 8: Metamodel for role and goal model registration*

— *Part 9: On demand model selection [Technical Report]*

— *Part 10: Core model and basic mapping*

— *Part 12: Metamodel for information model registration*

— *Part 13: Metamodel for form design registration*

Introduction

Business process collaboration and integration is growing due to worldwide economic pressures to streamline product development and delivery, and reduce operational costs. Enterprises are merging and forming partnerships to address these issues. Providing for the registration of process models in a standard registry so that they can be discovered, understood and compared for use and integration, will help to promote interoperation within and across enterprises.

Business process modelling languages and notations are widely used to represent processes for different purposes. However, the differences in the syntax and semantics of process models hamper sharing and reusing them among enterprises. Therefore, it is necessary to provide a generic mechanism to support registration of administrative information and selected metadata about process models.

This part of ISO/IEC 19763 provides a metamodel to support the registration of selected metadata and semantics of process models for process discovery and reuse. It offers guidance which highlights the common semantics of process models, helps people clarify the structure of a process and the relationship between processes, and aids in discovering processes, regardless of the notation in which they were originally written. Any information related to the details of process modelling languages or the platform for process execution is not taken into account. In particular, although the registration information of process models can be used to support further discovery of web services in terms of the associations between process and web services, the process representing either the execution order within a web service or the orchestration of a set of web services is out of the scope of this part.

NOTE In this part, 'process' is meant to be 'business process', and 'process model' is meant to be 'business process model'.

Information technology — Metamodel framework for interoperability (MFI) —

Part 5: Metamodel for process model registration

1 Scope

The primary purpose of the multipart standard ISO/IEC 19763 is to specify a metamodel framework for interoperability.

This part of ISO/IEC 19763 specifies the metamodel that describes a facility to register administrative information and selected metadata about process models. The metamodel specified in this part of ISO/IEC 19763 is intended to promote semantic discovery and reuse of process models within/across process model repositories. For this purpose, it provides selected metadata and common semantics of process models created with a specific process modelling language, including Business Process Model and Notation (BPMN)[1], UML (Unified Modelling Language) Activity Diagram[5] and EPC (Event-driven Process Chain)[7], etc. The metamodel can help discovery of the function and composition of a process, and promote reuse of its components at different levels of granularity. Figure 1 shows the scope of this part of ISO/IEC 19763.

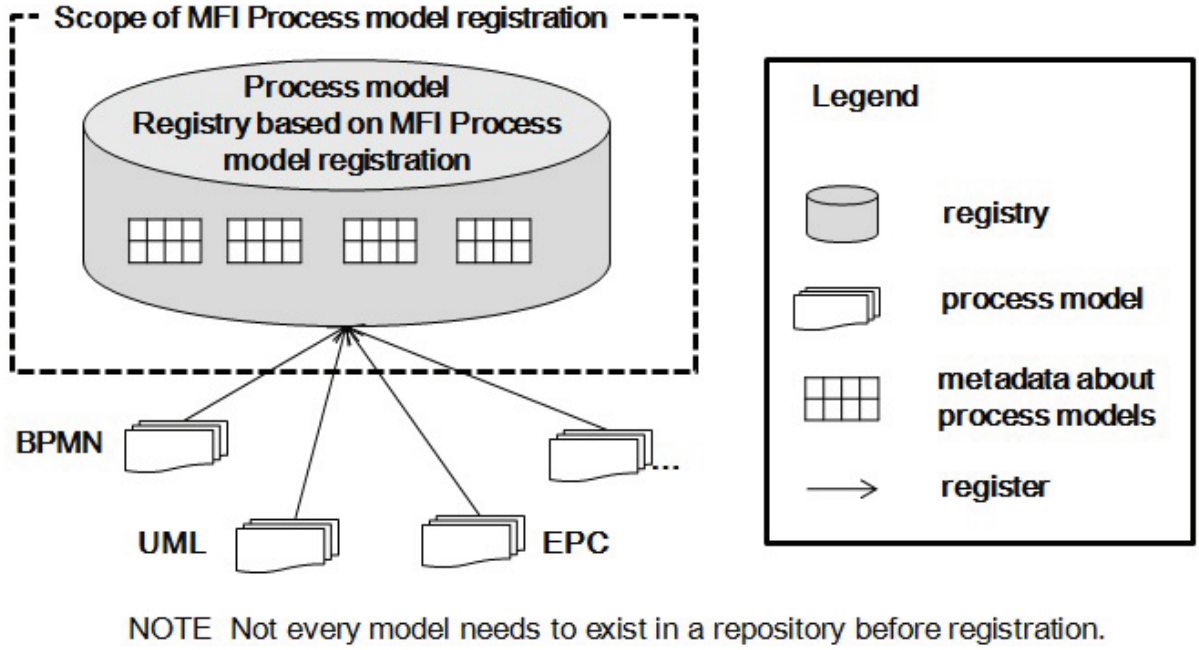


Figure 1 — The scope of MFI Process model registration

The following are outside the scope of this part of ISO/IEC 19763:

— details related to modelling notations or descriptive languages of process models;

— runtime environments or implementation platforms for executing processes.

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 19763-1, *Information technology — Metamodel framework for interoperability (MFI) — Part 1: Framework*

ISO/IEC 19763-7, *Information technology — Metamodel framework for interoperability (MFI)— Part 7: Metamodel for service model registration*

ISO/IEC 19763-8, *Information technology — Metamodel framework for interoperability (MFI) — Part 8: Metamodel for role and goal model registration*

ISO/IEC 19763-10, *Information technology — Metamodel framework for interoperability (MFI) — Part 10: Core model and basic mapping*

3 Terms, definitions and abbreviated terms

3.1 Terms and definitions

For the purposes of this document, the terms and definitions given in ISO/IEC 19763-7, ISO/IEC 19763-8 and ISO/IEC 19763-10, and the following apply.

3.1.1
activity
set of cohesive *tasks* (3.1.21)

[SOURCE: ISO/IEC 12207:2008, 4.3, modified]

3.1.2
control constraint
restriction on the execution order for a given collection of *processes* (3.1.12)

3.1.3
dependency
relationship between *process model elements* (3.1.14), that specifies the *control constraints* (3.1.2)

3.1.4
event
occurrence of a particular set of circumstances

3.1.5
exit condition
constraint that, if true, will cause an operation to terminate before its completion

Note 1 to entry: The operation can be a process or a service operation.

3.1.6
goal
intended outcome of user interaction with a *process* (3.1.12) or *service* (3.1.18)

[SOURCE: ISO/IEC 19763-8:—, 3.1.1]

3.1.7
guard condition
condition that must be satisfied before an associated *process* (3.1.12) can execute

3.1.8
involvement type
statement that indicates the type of involvement of a role with a *process* (3.1.12) or *service* (3.1.18)

EXAMPLE Performer, beneficiary, customer.

[SOURCE: ISO/IEC 19763-8:—, 3.1.4]

3.1.9
join dependency
kind of a dependency, specifying that the following *process model element* (3.1.14) will start when the selected preceding **process model element**s (3.1.14) are completed

3.1.10
postcondition
constraint that must be true at the completion of an operation

[SOURCE: ISO/IEC 14813-5:2010, B.1.116]

Note 1 to entry: The operation can be a process or a service operation.

3.1.11
precondition
constraint that must be true when an operation is invoked

[SOURCE: ISO/IEC 14813-5:2010, B.1.117]

Note 1 to entry: The operation can be a process or a service operation.

3.1.12
process
collection of related, structured *activities* (3.1.1) or *tasks* (3.1.21) that achieve a particular *goal* (3.1.16)

Note 1 to entry: The activities and tasks are represented by the Process metaclass in this part.

3.1.13
process model
representation of a *process* (3.1.12), using a specific *process modelling language* (3.1.15)

3.1.14
process model element
abstraction of the modelling constructs that constitutes a *process* (3.1.12), including *processes* (3.1.12) and *dependencies* (3.1.3) among them

3.1.15
process modelling language
special language used to represent *processes* (3.1.12)

Note 1 to entry: PSL, BPMN, UML Activities etc. are all process modelling languages.

Note 2 to entry: Special language [ISO 1087-1:2000, 3.1.3].

3.1.16
resource
asset that is utilized, created or consumed by a *process model element* (3.1.14)

Note 1 to entry: The resources can be either physical or virtual.

3.1.17
role
named specific behaviour of an entity participating in a particular context

[SOURCE: ISO/IEC 19763-8:—, 3.1.7]

3.1.18
service
kind of application which encapsulates one or more computing modules and can be accessed through a specified interface

[SOURCE: ISO/IEC 19763-7:—, 3.1.13]

3.1.19
sequence dependency
kind of control constraint between *processes* (3.1.12), specifying that the *processes* (3.1.12) are executed in order

3.1.20
split dependency
kind of control constraint between *process model elements* (3.1.14), specifying that if the preceding *process model element* (3.1.14) is completed, one or more of the following *process model elements* (3.1.14) will execute in parallel

3.1.21
task
specific piece of work to be done

[SOURCE: ISO 16091:2002, 3.1.25]

3.2 Abbreviated terms

BPMN	Business Process Model and Notation [SOURCE: OMG BPMN version 2, formal/2011-01-03]
EPC	Event-driven Process Chain
MFI	Metamodel framework for interoperability [SOURCE: ISO/IEC 19763-1:2007, 4.2]
MFI Core and mapping	ISO/IEC 19763-10, Information technology – Metamodel framework for interoperability (MFI) – Part 10: Core model and basic mapping
MFI Process model registration	ISO/IEC 19763-5, Information technology – Metamodel framework for interoperability (MFI) – Part 5: Metamodel for process model registration
MFI Role and Goal model registration	ISO/IEC 19763-8, Information technology – Metamodel framework for interoperability (MFI) – Part 8: Metamodel for role and goal model registration
MFI Service model registration	ISO/IEC 19763-7, Information technology – Metamodel framework for interoperability (MFI) – Part 7: Metamodel for service model registration
OWL-S	Web ontology language for service [7]
PSL	Process Specification Language [ISO/IEC 18629-1]
UML	Unified Modeling Language [ISO/IEC 19505-2]

4 Conformance

4.1 General

An implementation claiming conformance with this part of ISO/IEC 19763 shall support the metamodel specified in Clause 5, depending on a degree of conformance as described below.

4.2 Degree of conformance

4.2.1 General

The distinction between 'strictly conforming' and 'conforming' implementations is necessary to address the simultaneous needs for interoperability and extensions. This part of ISO/IEC 19763 describes specifications that promote interoperability. Extensions are motivated by needs of users, vendors, institutions and industries, but are not specified by this part of ISO/IEC 19763.

A strictly conforming implementation may be limited in usefulness but is maximally interoperable with respect to this part of ISO/IEC 19763. A conforming implementation may be more useful, but may be less interoperable with respect to this part of ISO/IEC 19763.

4.2.2 Strictly conforming implementation

A strictly conforming implementation

a) shall support the metamodel specified in Clause 5;

b) shall not use, test, access, or probe for any extension features nor extensions to the metamodel specified in Clause 5.

4.2.3 Conforming implementation

A conforming implementation

a) shall support the metamodel specified in Clause 5;

b) as permitted by the implementation, may use, test, access, or probe for any extension features or extensions to the metamodel specified in Clause 5.

NOTE 1 All strictly conforming implementations are also conforming implementations.

NOTE 2 The use of extensions to the metamodel might cause undefined behaviour.

4.3 Implementation Conformance Statement (ICS)

An implementation claiming conformance to this part of ISO/IEC 19763 shall include an Implementation Conformance Statement stating:

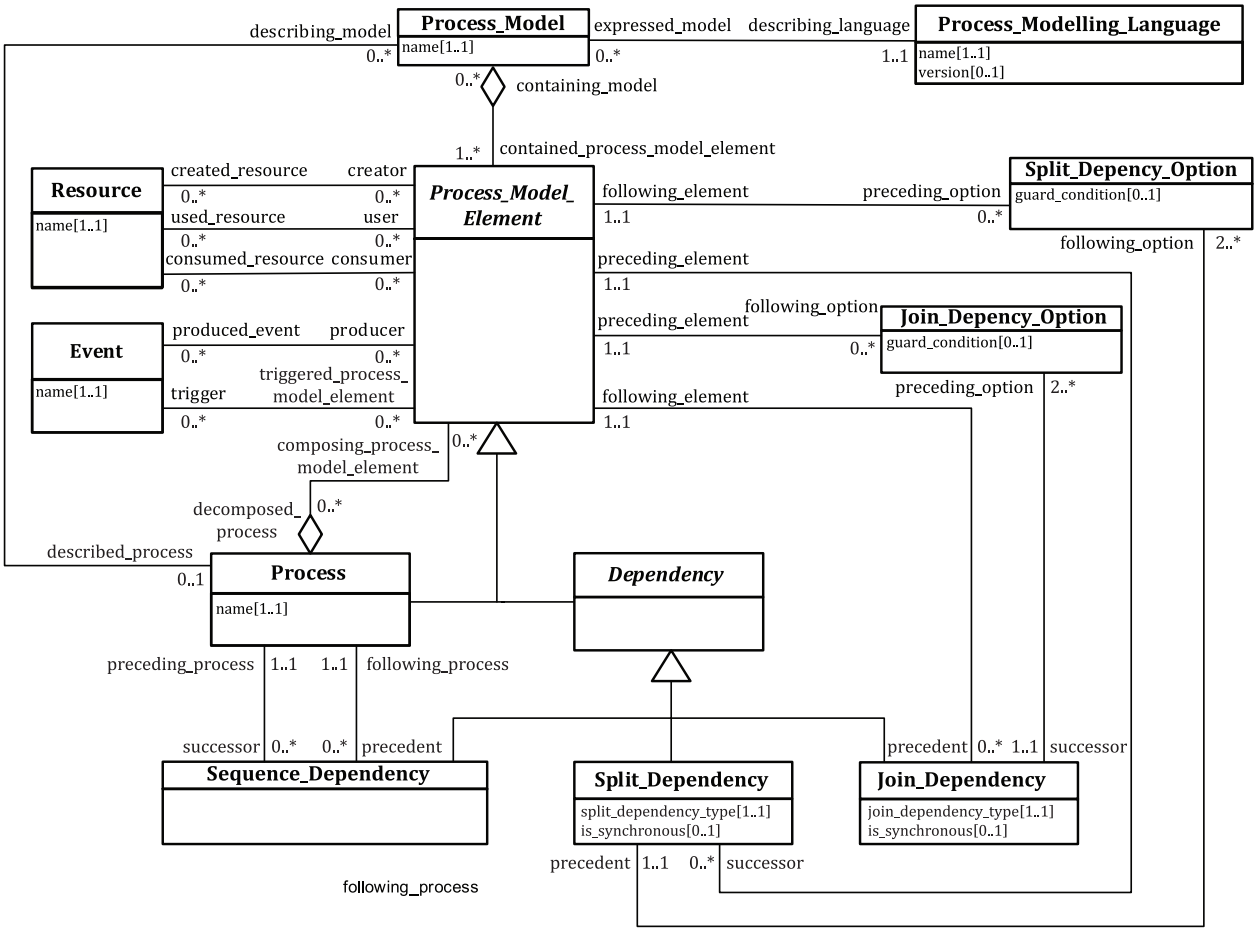
a) whether it is a strictly conforming implementation (see 4.2.2) or a conforming implementation (see 4.2.3);

b) what extensions, if any, are supported or used if it is a conforming implementation.

5 Structure of MFI Process model registration

5.1 Overview of MFI Process model registration

MFI Process model registration provides a generic metamodel to register selected metadata about process models described by a specific modelling language. Figure 2 shows the metamodel for process model registration.



NOTE Metaclasses whose names are italicized are abstract metaclasses.

Figure 2 — The metamodel of MFI Process model registration

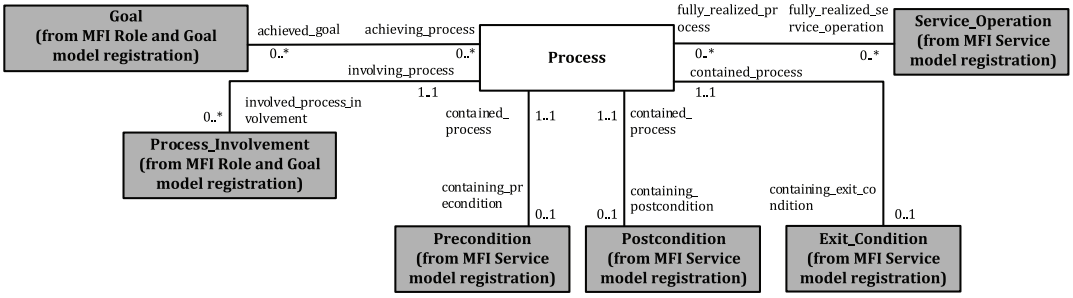
In this part, a process model is used as a representation of a process, and it describes the contained process model elements using a specified process modelling language. The process model elements include processes and dependencies between processes and/or other process model elements. For each process model element, there are some events that can be used to trigger a process model element or to be produced by a process model element. To achieve a particular goal, some resources are used, created or consumed by a process model element.

Dependencies represent the control constraints among processes represented by a process model. In this part, a dependency can be specialized as a sequence dependency, a split dependency, or a join dependency. A sequence dependency specifies that the processes are executed in order. A split dependency specifies that when the preceding process model element is completed, one or more of the following process model elements will execute in parallel. A join dependency specifies that the following process model element will start when the selected preceding process model elements are completed. In a split dependency, a split dependency type is used to specify a logical gate for the following processes. In a join dependency, similarly, a join dependency type is used to specify a logical gate for the preceding processes. In this part, the values of both a split dependency type and a join dependency type can be

XOR, OR and AND. For a split dependency type, XOR means that one and only one of the succeeding process model elements is allowed to execute, OR means that one or more of the succeeding process model elements are allowed to execute, and AND means that all of the succeeding process model elements must execute. For a join dependency type, XOR means that the succeeding process model element executes if one and only one of the preceding process model elements completes successfully, OR means that the succeeding process model element executes if one or more of the preceding process model elements completes successfully, and AND means that the succeeding process model element executes if, and only if, all of preceding process model elements completes successfully. In addition, a split dependency option represents the guard conditions of the following process model elements to be executed after the value of a split dependency type is decided. Similarly, a join dependency option specifies the guard conditions of the preceding process model elements to be executed after the value of a join dependency type is decided.

5.2 Associations between MFI Process model registration and other parts in MFI

Figure 3 shows the associations between MFI Process model registration, MFI Role and Goal model registration, and MFI Service model registration.



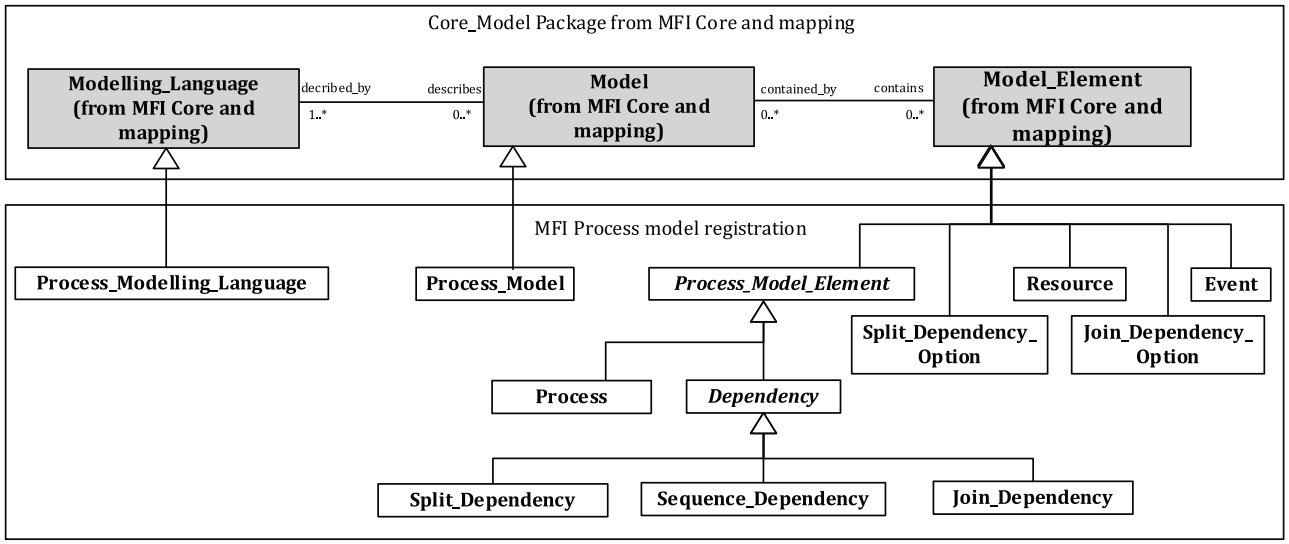
NOTE Metaclasses grey shaded are metaclasses that are defined in the other parts of ISO/IEC 19763.

Figure 3 — The associations between MFI Process model registration, MFI Role and Goal model registration and MFI Service model registration

The association between MFI Process model registration and MFI Role and Goal model registration specifies that each process achieves zero, one or more goals, and each goal is achieved by zero, one or more processes. A goal may exist that is not specified to be achieved by a process, and a process may exist which is not applied to achieve a specific goal. Similarly, each process involves zero, one or more process involvements, where each process involvement is the involvement of a role with a process, such as actor or beneficiary. Each process involvement indicates that a role is involved in the execution of one and only one process. A process involvement shall have exactly one associated process.

The association between MFI Process model registration and MFI Service model registration specifies that each process is fully realized by zero, one or more service operations, and each service operation can fully realize zero, one or more processes. A process may exist that is not specified to be realized by a service, and a service may exist that is not applied to realize a process. Each process may have one precondition and/or one postcondition. A process may exist with no associated precondition or postcondition. Each process has zero or one exit condition to state a set of conditions that will exist to cause a process to terminate before its completion. Each precondition, each postcondition and each exit condition can be defined using either a composite expression or an atomic expression.

The associations between the metaclasses in MFI Process model registration and the metaclasses in MFI Core and mapping are shown in Figure 4.



NOTE 1 Metaclasses whose names are italicized are abstract metaclasses.

NOTE 2 Metaclasses grey shaded are metaclasses that are defined in the other parts of ISO/IEC 19763.

Figure 4 — The associations between MFI Process model registration and MFI Core and mapping

The metaclass Process_Modelling_Language in MFI Process model registration is a subclass of Modelling_Language in MFI Core and mapping. Process_Model in MFI Process model registration is a subclass of Model in MFI Core and mapping. All the remaining metaclasses are subclasses of Model_ Element in MFI Core and mapping.

All subclasses have the association which are inherited from their superclasses. Some inherited associations are specialized in this part of ISO/IEC 19763. The details of specialization are defined in Clause 5.3.

5.3 Metaclasses in MFI Process model registration

5.3.1 Dependency

Dependency is an abstract metaclass each instance of which represents a specific particular dependency.

Superclass

[None]

Attribute	Datatype	Multiplicity	Description

[None]

Reference	Class	Multiplicity	Description	Inverse	Precedence

[None]

Constraints

[None]

5.3.2 Event

Event is a metaclass each instance of which represents a particular event.

Superclass

Model_Element (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
name	string	1..1	The title of an event instance

Reference	Class	Multiplicity	Description	Inverse	Precedence
triggered_process_model_element	Process_Model_Element	0..*	The set of process model elements, each of which is triggered by this event	trigger	Yes
producer	Process_Model_Element	0..*	The set of process model elements, each of which can produce this event	produced_event	No

Constraints

[None]

5.3.3 Join_Dependency

Join_Dependency is a metaclass each instance of which represents a particular join dependency.

Superclass

Dependency

Attribute	Datatype	Multiplicity	Description
join_dependency_type	string	1..1	The statement that specifies whether this join dependency is an 'AND' join dependency, an 'OR' join dependency or an 'XOR' join dependency. 'XOR' means that the succeeding process executes if one and only one of the preceding processes completes successfully, 'OR' means that the succeeding process executes if one or more of the preceding processes completes successfully, and 'AND' means that the succeeding process executes if, and only if, all of preceding processes completes successfully. NOTE Implementers should be aware that there are potential variations in implementations of XOR that might cause incompatibility.
is_synchronous	boolean	0..1	The indication that specifies whether the process model elements to be joined must be synchronous or not. The value of 'TRUE' means synchronous and the value of 'FALSE' means not synchronous.

Reference	Class	Multiplicity	Description	Inverse	Precedence
preceding_option	Join_Dependency_Option	2..*	The set of join dependency options, each of which specifies the guard condition, if any, that is used to determine whether the associated process model element is to be joined with the other process model elements associated with this join dependency through join dependency options	successor	No
following_element	Process_Model_Element	1..1	The process model element that follows this split dependency	precedent	No

Constraints

[None]

5.3.4 Join_Dependency_Option

Join_Dependency_Option is a metaclass each instance of which represents a particular join dependency option.

Superclass

Model_Element (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
guard_condition	string	0..1	The specification of the condition that must be true for the preceding process model element to be joined with the other process model elements associated through the other join dependency options associated with the same join dependency

Reference	Class	Multiplicity	Description	Inverse	Precedence
successor	Join_Dependency	1..1	The join dependency for which this join dependency option specifies the guard condition for one of the associated preceding process model elements	preceding_option	Yes
preceding_element	Process_Model_Element	1..1	The process model element that precedes this join dependency option	following_option	No

Constraints

[None]

5.3.5 Process

Process is a metaclass each instance of which represents a modelling construct which represents a particular process.

Superclass

Process_Model_Element

Attribute	Datatype	Multiplicity	Description
name	string	1..1	The title of a process instance

Reference	Class	Multiplicity	Description	Inverse	Precedence
describing_model	Process_Model	0..*	The set of process models, each of which may describe the composition of this process	described_process	No

composing_ process_ model_ element	Process_ Model_ Element	0..*	The set of process model elements, including processes and dependencies among them, each of which composes a process	decomposed_ process	No
precedent	Sequence_ Dependency	0..*	The set of sequence dependencies, each of which specifies that this process follows the process that is the preceding process for the particular sequence dependency	following_ process	No
successor	Sequence_ Dependency	0..*	The set of sequence dependencies, each of which specifies that this process precedes the process that is the following process for the particular sequence dependency	preceding_ process	No
fully_realizing_ service_operation	Service_ Operation (from MFI Service model registration)	0..*	The set of service operations, each of which may fully realize the process. In the case that a process is fully realized by a set of service operations, the process should be decomposed into a certain level such that each subprocess of the process can be fully realized by a service operation	fully_realized_ process	No
involved_process_ involvement	Process_ Involvement (from MFI Role and Goal model registration)	0..*	The set of process involvements, each of which designates how a particular role is involved in this process	involving_ process	Yes
achieved_goal	Goal (from MFI Role and Goal model registration)	0..*	The set of goals that can be achieved by this process	achieving_ process	Yes
containing_ precondition	Precondition (from MFI Service model registration)	0..1	The constraint that must be true when this process is invoked	contained_ process	Yes
containing_ postcondition	Postcondition (from MFI Service model registration)	0..1	The constraint that must be true at the completion of this process	contained_ process	Yes

containing_exit_ condition	Exit_Condition (from MFI Service model registration)	0..1	The constraint that, if true, will cause this process to terminate before its completion	contained_ process	Yes

Constraints

[None]

5.3.6 Process_Model

Process_Model is a metaclass each instance of which represents a particular process model.

Superclass

Model (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
name	string	1..1	The title of this process model

Reference	Class	Multiplicity	Description	Inverse	Precedence
describing_language	Process_ Modelling_ Language	1..1	The process modelling language that is used to represent this process model. This reference specializes the 'described_by' reference which is inherited from the superclass.	expressed_ model	Yes
contained_process_ model_element	Process_ Model_ Element	1..*	The set of process model elements that are described in this process model. This reference specializes the 'contains' reference which is inherited from the superclass.	containing_ model	Yes
described_process	Process	0..1	The process whose decomposition is described using this process model	describing_ model	Yes

Constraints

[None]

5.3.7 Process_Model_Element

Process_Model_Element is an abstract metaclass each instance of which represents a particular process model element.

Superclass

Model_Element (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
[None]			

Reference	Class	Multiplicity	Description	Inverse	Precedence
containing_model	Process_ Model	0..*	The set of process models which include this process model element. This reference specializes the 'contained_by' reference which is inherited from the superclass.	contained_ process_model_ element	No
decomposed_ process	Process	0..*	The set of processes which decompose into this process model element	composing_ process_model_ element	Yes
created_ resource	Resource	0..*	The set of resources, each of which may be created by this process model element	creator	Yes
consumed_ resource	Resource	0..*	The set of resources, each of which may be consumed by this process model element	consumer	Yes
used_Resource		0..*	The set of resources, each of which may be used in the execution of this process model element	user	Yes
produced_event	Event	0..*	The set of events, each of which may be produced by this process model element	producer	Yes
trigger	Event	0..*	The set of events, each of which may trigger (i.e. activate) this process model element	triggered_ process_model_ element	No

successor	Split_Dependency	0..*	The set of split dependencies each of which, if appropriate, follows this process model element	preceding_element	Yes
preceding_option	Split_Dependency_Option	0..*	The set of split dependency options each of which specifies the guard condition, if any, that is used to determine whether this process model element is to be executed	following_element	Yes
precedent	Join_Dependency	0..*	The set of join dependencies each of which, if appropriate, precedes this process model element	following_element	Yes
following_option	Join_Dependency_Option	0..*	The set of join dependency options each of which specifies the guard condition, if any, that is used to determine whether this process model element is to be joined with another process model element in the associated join dependency	preceding_element	Yes

Constraints

[None]

5.3.8 Process_Modelling_Language

Process_Modelling_Language is a metaclass each instance of which represents a particular process modelling language.

Superclass

Modelling_Language (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
name	string	1..1	The name of a process modelling language instance
version	value	0..1	The version of a process modelling language instance

Reference	Class	Multiplicity	Description	Inverse	Precedence
expressed_model	Process_Model	0..*	The set of process models that are expressed using this process modelling language. This reference specializes the 'describes' reference which is inherited from the superclass.	describing_language	No

Constraints

[None]

5.3.9 Resource

Resource is a metaclass each instance of which represents a particular resource.

Superclass

Model_Element (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
name	string	1..1	The title of a resource instance

Reference	Class	Multiplicity	Description	Inverse	Precedence
consumer	Process_Model_ Element	0..*	The set of processes, each of which can consume this resource in the execution of the process model element	consumed_ resource	No
creator	Process_Model_ Element	0..*	The set of processes, each of which can create this resource in the execution of the process model element	created_ resource	No
user	Process_Model_ Element	0..*	The set of processes, each of which can use this resource in the execution of the process model element	used_resource	No

Constraints

The values of the references ‘consumer’, ‘creator’ and ‘user’ are mutually exclusive for any one process model element; i.e. a process model element can consume a particular resource or create a particular resource or use a particular resource.

5.3.10 Sequence_Dependency

Sequence_Dependency is a metaclass each instance of which represents a particular sequence dependency.

Superclass

Dependency

Attribute	Datatype	Multiplicity	Description
[None]			

Reference	Class	Multiplicity	Description	Inverse	Precedence
following_process	Process	1..1	The process that follows this sequence dependency	precedent	Yes
preceding_process	Process	1..1	The process that precedes this sequence dependency	successor	Yes

Constraints

[None]

5.3.11 Split_Dependency

Split_Dependency is a metaclass each instance of which represents a particular split dependency.

Superclass

Dependency

Attribute	Datatype	Multiplicity	Description
split_dependency_type	string	1..1	The statement that specifies whether this split dependency is an 'AND' split dependency, an 'OR' split dependency or an 'XOR' split dependency. 'XOR' means that one and only one of the succeeding process model elements is allowed to execute, 'OR' means that one or more of the succeeding process model elements are allowed to execute, and 'AND' means that all of the succeeding process model elements must execute.
is_synchronous	boolean	0..1	The indication that specifies whether the process model elements to be split must be synchronous or not. The value of 'TRUE' means synchronous and the value of 'FALSE' means not synchronous.

Reference	Class	Multiplicity	Description	Inverse	Precedence
following_option	Split_Dependency_Option	2..*	The set of split dependency options each of which specifies the guard condition, if any, that is used to determine whether the associated process model element is to be executed	precedent	Yes
preceding_element	Process_Model_Element	1..1	The process model element that precedes this split dependency.	successor	No

Constraints

[None]

5.3.12 Split_Dependency_Option

Split_Dependency_Option is a metaclass each instance of which represents a particular split dependency option.

Superclass

Model_Element (from MFI Core and mapping)

Attribute	Datatype	Multiplicity	Description
guard_condition	string	0..1	The specification of the condition that must be true for the following process model element to be executed. The following process model element will be executed if no guard condition is specified.

Reference	Class	Multiplicity	Description	Inverse	Precedence
precedent	Split_Dependency	1..1	The split dependency for which this split dependency option specifies the guard condition for one of the associated following process model elements	following_option	No
following_element	Process_Model_Element	1..1	The process model element that follows this split dependency option	preceding_option	No

Constraints

[None]

Annex A
(informative)

Examples of MFI Process model registration

In this section, four cases are provided to illustrate how to register various kinds of process models based on MFI Process model registration. In detail, the process models in the four cases use UML Activity Diagram, BPMN, EPC and OWL-S respectively. The corresponding exemplary transformation and registration information are indicated to show that MFI Process model registration can be used to register heterogeneous process models. It also means that MFI Process model registration can harmonize with existing specifications related to process modelling.

A.1 Case A.1: Train_Ticket_Reservation process in UML Activity Diagram

The Train_Ticket_Reservation process, illustrating the process of online train ticket reservation, is expressed as the UML Activity Diagram shown at Figure A.1.

This process consists of a set of sub-processes. The process will start if the customer successfully completes login. The customer can then select the train information that meets the requirements of his or her travel itinerary. After this, the customer needs to complete and save the desired reservation details. The system will then automatically notify the customer of the reservation details, either by Email or SMS (Short Message Service) as chosen by the customer, as confirmation of his or her reservation.

For some purposes, such as improving reusability of process models, it will be helpful to model the details of a sub-process separately. This is shown in this example where a separate model is used to describe the decomposition of the Notify_Customer. Thus, there are two process models, one for the complete Train_Ticket_Reservation process and the other for the Notify_Customer sub-process, to be registered.

Using the facilities of the metamodel of MFI Process model registration, the process model for Notify_Customer is declared as describing decomposition of the sub-process Notify_Customer of the Train_Ticket_Reservation process. The process model Notify_Customer is named after sub-process Notify_Customer so that two-way searching is possible.

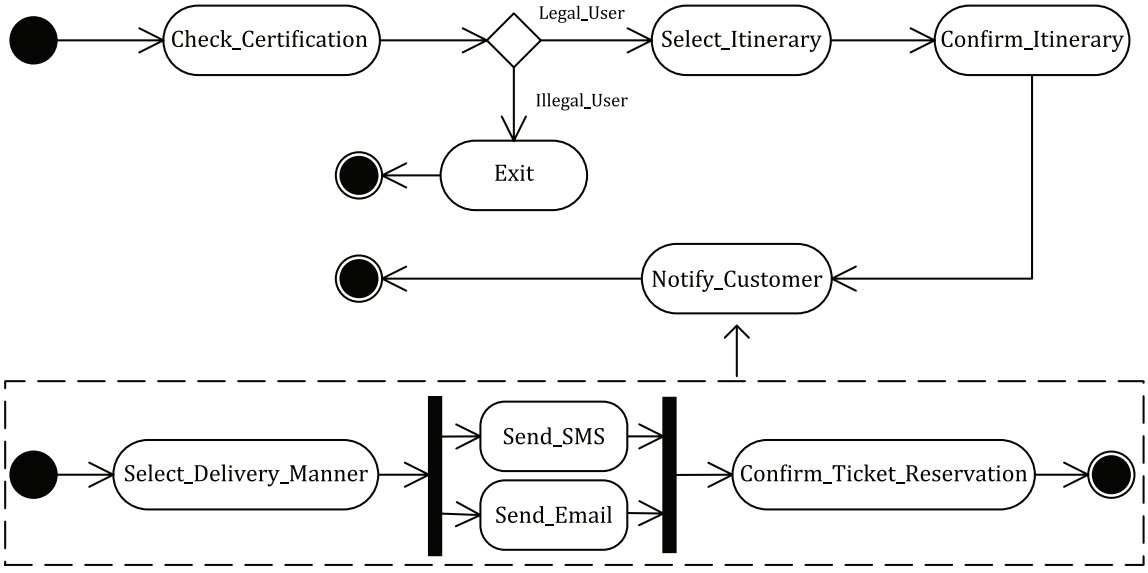


Figure A.1 — The process model of Train_Ticket_Reservation in UML Activity Diagram notation

Table A.1 shows the MFI Process model registration metaclasses that are used to represent the elements in the Train_Ticket_Reservation process model in Figure A.1. The use of these metaclasses supports the registration of process models expressed as a UML Activity Diagram, such as that for Train_Ticket_Reservation, in a registry based on MFI Process model registration.

Table A.1 — Exemplary transformation for Case A.1

Notation of UML Activity Diagram	Metaclass in MFI Process model registration	Notation of UML Activity Diagram	Metaclass in MFI Process model registration
	Event		Split_Dependency
			
	Process		Join_Dependency
			Sequence_Dependency

Figure A.2 and Figure A.3 provide the object instances to illustrate the registration of the Train_Ticket_Reservation process model (the sub-processes that constitute the parent process) and the Notify_Customer process model (the sub-sub-process that decompose the sub-process) respectively.

<Process_Model>
Object101

Attribute/Reference	Literal/Instance
name	"Train_Ticket_Reservation_Model"
describing_language	Object102
contained_process_model_element	Object104, Object105, Object108, Object110, Object111, Object112, Object113, Object114

<Process_Modelling_Language>
Object102

Attribute/Reference	Literal/Instance
name	"UML"
version	"2.1.2"
expressed_model	Object101, Object121

<Event>
Object103

Attribute/Reference	Literal/Instance
name	"Ticket_Reservation_Request"
triggered_process_model_element	Object104

<Process>
Object104

Attribute/Reference	Literal/Instance
name	"Check_Certification"
containing_model	Object101
trigger	Object103
successor	Object105

<Split_Dependency>
Object105

Attribute/Reference	Literal/Instance
split_dependency_type	"XOR"
is_synchonous	"FALSE"
containing_model	Object101
preceding_element	Object104
following_option	Object106,Object107

<Split_Dependency_Option>
Object106

Attribute/Reference	Literal/Instance
guard_condition	"Illegal_User"
following_element	Object108
precedent	Object105

<Split_Dependency_Option>
Object107

Attribute/Reference	Literal/Instance
guard_condition	"Legal_User"
following_element	Object110
precedent	Object105

<Process>
Object108

Attribute/Reference	Literal/Instance
name	"Exit"
containing_model	Object101
preceding_option	Object106
produced_event	Object109

<Event>
Object109

Attribute/Reference	Literal/Instance
name	"User_Exit"
producer	Object108

<Process>
Object110

Attribute/Reference	Literal/Instance
name	"Select_Itinerary"
containing_model	Object101
preceding_option	Object107
successor	Object111

Figure A.2 — *(continued)*

<Sequence_Dependency>
Object111

Attribute/Reference	Literal/Instance
containing_model	Object101
preceding_process	Object110
following_process	Object112

<Process>
Object112

Attribute/Reference	Literal/Instance
name	"Confirm_Itinerary"
containing_model	Object101
successor	Object113
precedent	Object111

<Sequence_Dependency>
Object113

Attribute/Reference	Literal/Instance
containing_model	Object101
preceding_process	Object112
following_process	Object114

<Process>
Object114

Attribute/Reference	Literal/Instance
name	"Notify_Customer"
containing_model	Object101
precedent	Object113
produced_event	Object115
describing_model	Object121

<Event>
Object115

Attribute/Reference	Literal/Instance
name	"Ticket_Reservation_Closed"
producer	Object114

Figure A.2 — Registration of the Train_Ticket_Reservation example (for the parent process Train_Ticket_Reservation)

<Process_Model>
Object121

Attribute/Reference	Literal/Instance
name	"Notify_Customer_Model"
describing_language	Object102
described_process	Object114
contained_process_model_element	Object123, Object124, Object127, Object128, Object129, Object132

<Event>
Object122

Attribute/Reference	Literal/Instance
name	"Notify_Itinerary_Started"
triggered_process_model_element	Object123

<Process>
Object123

Attribute/Reference	Literal/Instance
name	"Select_Delivery_Manner"
containing_model	Object121
trigger	Object122
successor	Object124

<Split_Dependency>
Object124

Attribute/Reference	Literal/Instance
split_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object121
preceding_element	Object123
following_option	Object125, Object126

<Split_Dependency_Option>
Object125

Attribute/Reference	Literal/Instance
guard_condition	"Deliver_By_Email"
following_element	Object127
precedent	Object124

<Split_Dependency_Option>
Object126

Attribute/Reference	Literal/Instance
guard_condition	"Deliver_By_SMS"
following_element	Object128
precedent	Object124

<Process>
Object127

Attribute/Reference	Literal/Instance
name	"Send_Email"
containing_model	Object121
preceding_option	Object125
following_option	Object130

<Process>
Object128

Attribute/Reference	Literal/Instance
name	"Send_SMS"
containing_model	Object121
preceding_option	Object126
following_option	Object131

<Join_Dependency>
Object129

Attribute/Reference	Literal/Instance
join_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object121
preceding_option	Object130, Object131
following_element	Object132

Figure A.3 — (*continued*)

<Process_Model>
Object121

Attribute/Reference	Literal/Instance
name	"Notify_Customer_Model"
describing_language	Object102
described_process	Object114
contained_process_model_element	Object123, Object124, Object127, Object128, Object129, Object132

<Event>
Object122

Attribute/Reference	Literal/Instance
name	"Notify_Itinerary_Started"
triggered_process_model_element	Object123

<Process>
Object123

Attribute/Reference	Literal/Instance
name	"Select_Delivery_Manner"
containing_model	Object121
trigger	Object122
successor	Object124

<Split_Dependency>
Object124

Attribute/Reference	Literal/Instance
split_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object121
preceding_element	Object123
following_option	Object125, Object126

<Split_Dependency_Option>
Object125

Attribute/Reference	Literal/Instance
guard_condition	"Deliver_By_Email"
following_element	Object127
precedent	Object124

<Split_Dependency_Option>
Object126

Attribute/Reference	Literal/Instance
guard_condition	"Deliver_By_SMS"
following_element	Object128
precedent	Object124

<Process>
Object127

Attribute/Reference	Literal/Instance
name	"Send_Email"
containing_model	Object121
preceding_option	Object125
following_option	Object130

<Process>
Object128

Attribute/Reference	Literal/Instance
name	"Send_SMS"
containing_model	Object121
preceding_option	Object126
following_option	Object131

<Join_Dependency>
Object129

Attribute/Reference	Literal/Instance
join_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object121
preceding_option	Object130, Object131
following_element	Object132

Figure A.3 — Registration of the Notify_Customer example (for the sub-process Notify_Customer which can be further decomposed and have separated registration)

A.2 Case A.2: Handle_Online_Car_Repair process in BPMN

The Handle_Online_Car_Repair process, illustrating the process handling an online order for the repair of a car, is expressed as the BPMN model shown at Figure A.4.

This process consists of a set of sub-processes. The process will start when a repair request is registered. The credit card will then be charged for the payment for the car repair. If the credit card is declined, the Handle_Fault process will be invoked to return an error and terminate the whole process prematurely; if the credit card is accepted, the customer will be able to schedule a garage appointment.

The customer may now select one or both of two options. One of these uses the Order_Tow_Truck process, if required, to order a tow truck to move the broken-down car. The other option uses the Rent_Temporary_Car process to order a replacement car while the broken-down car is repaired. Rent_Temporary_Car is decomposed into three sub-processes: Select_Temporary_Car, Evaluate_Temporary_Car and Order_Temporary_Car. Select_Temporary_Car and Evaluate_Temporary_Car, which evaluates whether the selected car meets the customer's needs or not, will be executed repeatedly till an appropriate car is chosen. Order_Temporary_Car then executes.

The Handle_Online_Car_Repair process completes when the customer confirms the order.

To model the Rent_Temporary_Car sub-process (which is decomposed into process for selecting, evaluating and ordering a temporary car) the BPMN Expanded Sub-Process is used. Within BPMN, an Expanded Sub-Process is used to expose the decomposed flow details of the sub-process within the context of its parent process. The Expanded Sub-Process is displayed as a rounded rectangle that is enlarged to display the whole of the decomposed sub-process.

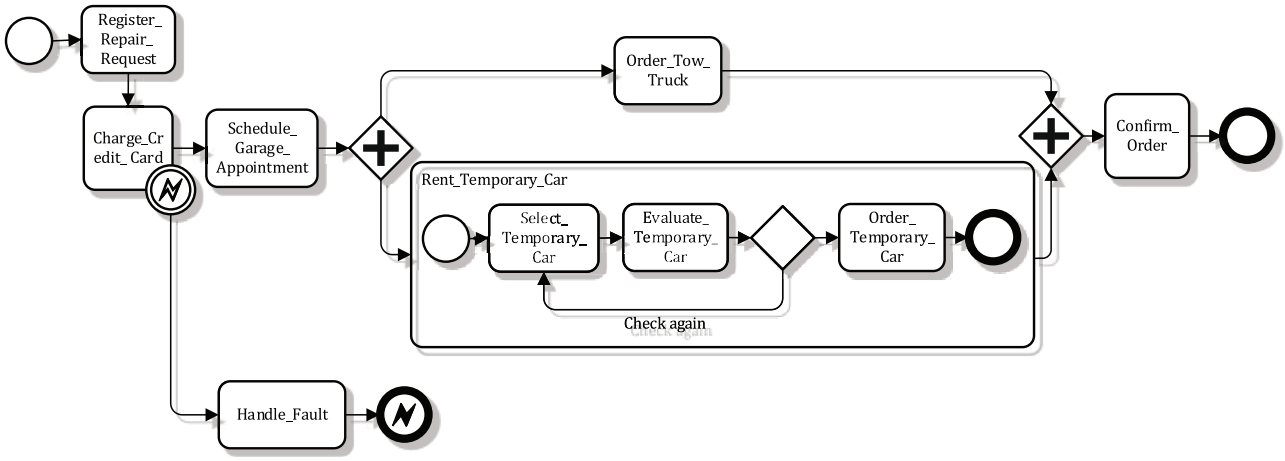


Figure A.4 — The process model of Handle_Online_Car_Repair in BPMN notation

Table A.2 shows the MFI Process model registration metaclasses that are used to represent the elements in the Handle_Online_Car_Repair process model in Figure A.4. The use of these metaclasses supports the registration of process models expressed using BPMN, such as that for Handle_Online_Car_Repair, in a registry based on MFI Process model registration.

Figure A.5 provides the object instances to illustrate the registration of the Handle_Online_Car_Repair process model.

Table A.2 — Exemplary transformation for Case A.2

Notation of BPMN	Metaclass in MFI Process model registration	Notation of BPMN	Metaclass in MFI Process model registration
	Event		Split_Dependency
			
			
	Process		Join_Dependency
			Sequence_Dependency

<Process_Model>
Object201

Attribute/Reference	Literal/Instance
name	"Handle_Online_Car_Repair_Model"
describing_language	Object202
contained_process_model_element	Object204, Object205, Object206, Object207, Object210, Object212, Object213, Object216, Object217, Object219, Object220, Object221, Object222, Object225, Object229, Object230

<Process_Modelling_Language>
Object202

Attribute/Reference	Literal/Instance
name	"BPMN"
version	"2.0"
expressed_model	Object201

<Event>
Object203

Attribute/Reference	Literal/Instance
name	"Repair_Request"
triggered_process_model_element	Object204

<Process>
Object204

Attribute/Reference	Literal/Instance
name	"Register_Repair_Request"
containing_model	Object201
trigger	Object203
successor	Object205

<Sequence_Dependency>
Object205

Attribute/Reference	Literal/Instance
containing_model	Object201
preceding_process	Object204
following_process	Object206

<Process>
Object206

Attribute/Reference	Literal/Instance
name	"Charge_Credit_Card"
containing_model	Object201
precedent	Object205
successor	Object207

<Split_Dependency>
Object207

Attribute/Reference	Literal/Instance
containing_model	Object201
split_dependency_type	"XOR"
is_synchonous	"FALSE"
preceding_element	Object206
following_option	Object208,Object209

<Split_Dependency_Option>
Object208

Attribute/Reference	Literal/Instance
guard_condition	Charge_Credit_Card_Successful
following_element	Object212
precedent	Object207

<Split_Dependency_Option>
Object209

Attribute/Reference	Literal/Instance
guard_condition	Charge_Credit_Card_Successful
following_element	Object210
precedent	Object207

<Process>
Object210

Attribute/Reference	Literal/Instance
name	"Handle_Fault"
containing_model	Object201
preceding_option	Object209
produced_event	Object211

<Event>
Object211

Attribute/Reference	Literal/Instance
name	"Credit_Card_Fault"
producer	Object210

<Process>
Object212

Attribute/Reference	Literal/Instance
name	"Schedule_Garage_Apointment"
containing_model	Object201
preceding_option	Object208
successor	Object213

Figure A.5 — (*continued*)

<Split_Dependency>
Object213

Attribute/Reference	Literal/Instance
containing_model	Object201
split_dependency_type	"OR"
is_synchonous	"FALSE"
preceding_element	Object212
following_option	Object214,Object215

<Split_Dependency_Option>
Object214

Attribute/Reference	Literal/Instance
guard_condition	"Need_Tow_Truck"
following_element	Object216
precedent	Object213

<Split_Dependency_Option>
Object215

Attribute/Reference	Literal/Instance
guard_condition	"Need_Temporary_Car"
following_element	Object217
precedent	Object213

<Process>
Object216

Attribute/Reference	Literal/Instance
name	"Order_Tow_Truck"
containing_model	Object201
preceding_option	Object214
following_option	Object227

<Process>
Object217

Attribute/Reference	Literal/Instance
name	"Rent_Temporary_Car"
containing_model	Object201
preceding_option	Object215
following_option	Object228
composing_process_model_element	Object219, Object220, Object221, Object222, Object225

<Event>
Object218

Attribute/Reference	Literal/Instance
name	"Rent_Temporary_Car_Request"
triggered_process_model_element	Object219

<Process>
Object219

Attribute/Reference	Literal/Instance
name	"Select_Temporary_Car"
containing_model	Object201
decomposed_process	Object217
trigger	Object218
successor	Object220
preceding_option	Object223

<Sequence_Dependency>
Object220

Attribute/Reference	Literal/Instance
containing_model	Object201
decomposed_process	Object217
preceding_process	Object219
following_process	Object221

<Process>
Object221

Attribute/Reference	Literal/Instance
name	"Evaluate_Temporary_Car"
containing_model	Object201
decomposed_process	Object217
precedent	Object220
successor	Object222

<Split_Dependency>
Object222

Attribute/Reference	Literal/Instance
containing_model	Object201
decomposed_process	Object217
split_dependency_type	"XOR"
is_synchonous	"FALSE"
preceding_element	Object221
following_option	Object223, Object224

<Split_Dependency_Option>
Object223

Attribute/Reference	Literal/Instance
guard_condition	"Unsatisfied_Evaluation"
precedent	Object222
following_element	Object219

<Split_Dependency_Option>
Object224

Attribute/Reference	Literal/Instance
guard_condition	"Satisfied_Evaluation"
precedent	Object222
following_element	Object225

<Process>
Object225

Attribute/Reference	Literal/Instance
name	"Order_Temporary_Car"
containing_model	Object201
decomposed_process	Object217
preceding_option	Object224
produced_event	Object226

<Event>
Object226

Attribute/Reference	Literal/Instance
name	"Rent_Temporary_Car_Closed"
producer	Object225

<Join_Dependency_Option>
Object227

Attribute/Reference	Literal/Instance
guard_condition	"Order_Tow_Truck_Successful"
preceding_element	Object216
successor	Object229

<Join_Dependency_Option>
Object228

Attribute/Reference	Literal/Instance
guard_condition	"Order_Temporary_Car_Successful"
preceding_element	Object217
successor	Object229

Figure A.5 — (*continued*)

<Join_Dependency>
Object229

Attribute/Reference	Literal/Instance
containing_model	Object201
join_dependency_type	"OR"
is_synchonous	"FALSE"
following_element	Object230
preceding_option	Object227, Object228

<Process>
Object230

Attribute/Reference	Literal/Instance
name	"Confirm_Order"
containing_model	Object201
precedent	Object229
produced_event	Object231

<Event>
Object231

Attribute/Reference	Literal/Instance
name	"Car_Repaired_Closed"
producer	Object230

Figure A.5 — Registration of the Handle_Online_Car_Repair example

A.3 Case A.3: Make_Record process in EPC

The Make_Record process, illustrating the process of song recording, is expressed as the EPC Diagram shown at Figure A.6.

This process consists of a set of sub-processes. The process will be triggered by the event Start. After the corresponding equipments are ready, a song is chosen to be recorded. Having chosen the song, that song is recorded. If there is a requirement to record more songs (such as for an album that needs more than one song) it is necessary to choose and record the extra songs repeatedly until all the required songs have been recorded; if not, the record will be sent for marketing, and the Make_Record process is completed.

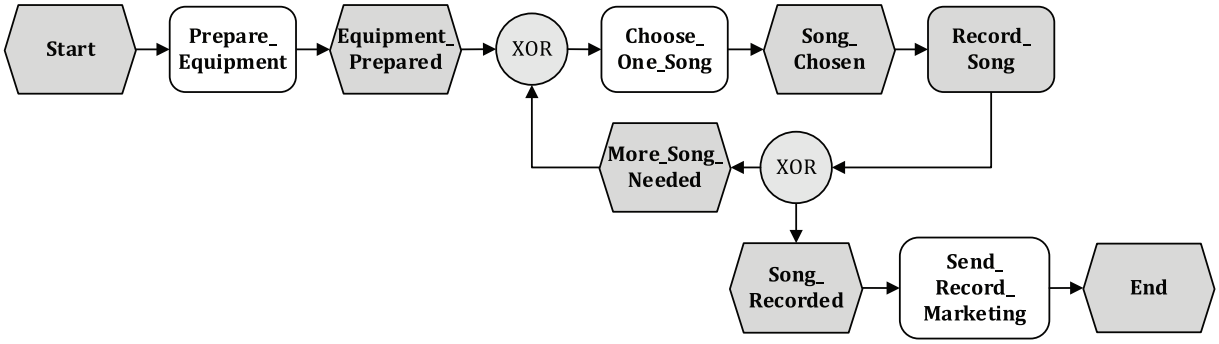
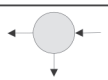


Figure A.6 — The process model of Make_Record in EPC notation

Table A.3 lists the exemplary transformation from the elements in EPC process model to the corresponding metaclasses specified in MFI Process model registration.

Table A.3 — Exemplary transformation for Case A.3

Srl	EPC Notation	Metaclass in MFI Process model registration	Srl	EPC Notation	Metaclass in MFI Process model registration
1		Event	4		Split_Dependency
2		Sequence_Dependency	5		Join_Dependency
3		Process	6		Guard_Condition

The start event and end event in an EPC model are registered as instances of Event in MFI Process model registration (see Table A.3, Srl 1).

Functions are registered as Processes (see Table A.3, Srl 3).

Operators with one input and multiple outputs are registered as instances of Split_Dependency (see Table A.3, Srl 4).

Events after and directly connected to an operator (XOR in this case) are registered as the guard condition in an instance of Split_Dependency_Option (see Table A.3, Srl 6).

Operators with multiple inputs and one output are registered as instances of Join_Dependency (see Table A.3, Srl 5).

Events after and directly connected to an operator (XOR in this case) are registered as the guard condition in an instance of Join_Dependency_Option (see Table A.3, Srl 6).

Events between two functions are registered as instances of Sequence_Dependency (see Table A.3, Srl 2).

The detailed registration information for the Make_Record example is given in Figure A.7.

<Process_Model>
Object301

Attribute/Reference	Literal/Instance
name	"Make_Record_Model"
describing_language	Object302
contained_process_model_element	Object304, Object306, Object307, Object308, Object309, Object310, Object314

<Process_Modelling_Language>
Object302

Attribute/Reference	Literal/Instance
name	"EPC"
expressed_model	Object301

<Event>
Object303

Attribute/Reference	Literal/Instance
name	"Start"
triggered_process_model_element	Object304

<Process>
Object304

Attribute/Reference	Literal/Instance
name	"Prepare_Equipment"
containing_model	Object301
trigger	Object303
successor	Object305

<Join_Dependency_Option>
Object305

Attribute/Reference	Literal/Instance
guard_condition	"Equipment_Prepared"
preceding_element	Object304
successor	Object306

<Join_Dependency>
Object306

Attribute/Reference	Literal/Instance
join_dependency_type	"XOR"
is_synchonous	"FALSE"
containing_model	Object301
following_element	Object307
preceding_option	Object305, Object313

<Process>
Object307

Attribute/Reference	Literal/Instance
name	"Choose_One_Song"
containing_model	Object301
precedent	Object306
following_element	Object308

<Sequence_Dependency>
Object308

Attribute/Reference	Literal/Instance
containing_model	Object301
preceding_process	Object307
following_element	Object309

<Process>
Object309

Attribute/Reference	Literal/Instance
name	"Record_Song"
containing_model	Object301
precedent	Object308
successor	Object310

<Split_Dependency>
Object310

Attribute/Reference	Literal/Instance
split_dependency_type	"XOR"
is_synchonous	"FALSE"
containing_model	Object301
preceding_element	Object309
following_option	Object311, Object312

<Split_Dependency_Option>
Object311

Attribute/Reference	Literal/Instance
guard_condition	"Need_More_Song"
precedent	Object310
following_element	Object313

<Split_Dependency_Option>
Object312

Attribute/Reference	Literal/Instance
guard_condition	"Record_Completed"
precedent	Object310
following_element	Object314

<Join_Dependency_Option>
Object313

Attribute/Reference	Literal/Instance
guard_condition	"Need_More_Song"
preceding_element	Object311
successor	Object306

<Process>
Object314

Attribute/Reference	Literal/Instance
name	"Send_Record_Marketing"
containing_model	Object301
preceding_option	Object312
produced_event	Object315

<Event>
Object315

Attribute/Reference	Literal/Instance
name	"End"
producer	Object314

Figure A.7 — Registration of the Make_Record example

A.4 Case A.4: Query_Bus_Information process in OWL-S

The Query_Bus_Information process, illustrating the process of bus information query, is expressed as the OWL-S shown at Figure A.8.

This process consists of a set of sub-processes. The process starts from selecting query port, then user can query the bus information by location name or by route number, in the final step, visualized bus information is provided to user.

In this case, those processes are expressed in the form of text; they are lengthy and have almost the same tagged structure, such as input, output, and execution between them. So Figure A.8 only illustrates part of the OWL-S code.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<!-- Instance Definition of Query Bus Information Process Model -->
 -<process:ProcessModel rdf:ID="Query_Bus_Information_Model"> <process:hasProcess
rdf:resource="#Query_Bus_Information_Process"/> <service:describeS rdf:resource="http://www.daml.org/services/owl-
s/1.0/QueryBusInformationService.owl#Query_Bus_Information_Agent"/> </process:ProcessModel>

<process:AtomicProcess rdf:ID="Select_Query_Port">
  <process:comment>No Comments</process:comment>
  <process:hasInput>
    <process:Input rdf:ID="Port_Name">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="Port_Select_Succeed">
      <process:parameterType rdf:datatype="http://www.owl-ontologies.com/UrbanTransportation.owl#Bus_Information"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>

<process:AtomicProcess rdf:ID="Query_With_Location_AP">
  <process:comment>No Comments</process:comment>
  <process:hasInput>
    <process:Input rdf:ID="Location_Name">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="Raw_Bus_Information_Data_With_Location">
      <process:parameterType rdf:datatype="http://www.owl-ontologies.com/UrbanTransportation.owl#Bus_Information"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>

<process:AtomicProcess rdf:ID="Query_With_Route_AP">
  <process:comment>No Comments</process:comment>
  <process:hasInput>
    <process:Input rdf:ID="Route_Number">
      <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
    </process:Input>
  </process:hasInput>
  <process:hasOutput>
    <process:Output rdf:ID="Raw_Bus_Information_Query_With_Route">
      <process:parameterType rdf:datatype="http://www.owl-ontologies.com/UrbanTransportation.owl#Bus_Information"/>
    </process:Output>
  </process:hasOutput>
</process:AtomicProcess>
```

Figure A.8 — (*continued*)

```
<process:AtomicProcess rdf:ID="Provide_Bus_Information">
 <process:comment>No Comments</process:comment>
 <process:hasInput>
  <process:Input rdf:ID="Raw_Bus_Information_Data">
   <process:parameterType rdf:datatype="http://www.w3.org/2001/XMLSchema#string"/>
  </process:Input>
 </process:hasInput>
 <process:hasOutput>
  <process:Output rdf:ID="Graphic_Bus_Information_Data">
   <process:parameterType rdf:datatype="http://www.owl-ontologies.com/UrbanTransportation.owl#Bus_Information"/>
  </process:Output>
 </process:hasOutput>
</process:AtomicProcess>

<process:CompositeProcess rdf:ID="Query_Bus_Information_CP">
 <process:composedof><process:Choice><process:components><process:ControlConstructList>
   <list:first>
    <process:Perform rdf:ID="Perform_Query_With_Route_AP">
     <process:process rdf:resource="Query_With_Route_AP"/>
    </process:Perform>
   </list:first>
   <list:rest><process:ControlConstructList>
    <list:first>
     <process:Perform rdf:ID="Perform_Query_With_Location_AP">
      <process:process rdf:resource="Query_Bus_With_Location_AP"/>
     </process:Perform>
    </list:first>
    <list:rest rdf:resource="http://www.daml.org/services/owl-s/1.1/generic/ObjectList.owl#nil"/>
    </list:rest>
   </process:ControlConstructList></list:rest>
 </process:ControlConstructList></process:components></process:Choice></process:composedof>
</process:CompositeProcess>
</rdf:RDF>
```

Figure A.8 — Query_Bus_Information process model in OWL-S notation

While an OWL-S based process model is transformed for MFI Process model registration, at least one start and multiple ends will be automatically added in. In case 4, only one start and one end are added.

Table A.4 lists the exemplary transformation from the elements in case 4 to the corresponding metaclasses in MFI Process model registration. The OWL-S is not graph-oriented, and emphases on the functional closure for describing properties and capabilities, rather than structuring fragments used in most graph notations. And the typical and simple transformation (such as one-to-one match) used in the previous 3 cases is not the suitable representation for the transformation between OWL-S and metaclasses of MFI Process model registration. So we use a compound structures of metaclasses of MFI Process model registration to interpret the control structures of OWL-S, such as Choice closure (in this case, it is interpreted as a block of alternative branches between a split dependency with XOR type and a join dependency with XOR type). And only atomic process in OWL-S is considered as Process in MFI registration, for all connection (input and output) are happened among atomic processes, rather than composite processes. The detailed registration information of the Query_Bus_Information example is indicated in Figure A.9.

Table A.4 — Exemplary transformation for Case A.4

Notation of OWL-S	Metaclass in MFI Process model registration		Notation of OWL-S	Metaclass in MFI Process model registration
<process:**AtomicProcess** rdf:ID="…"> … </process:AtomicProcess>	Process		<process:composedof> <process:**Choice**> <process:components> <process:ControlConstructList> … </process:composedof>	Split_Dependency (start) + Join_Dependency (end) (same type: XOR)

<Process_Model>
Object401

Attribute/Reference	Literal/Instance
name	"Query_Bus_Information_Model"
describing_language	Object402
contained_process_model_element	Object404, Object405, Object408, Object409, Object410, Object413

<Process_Modelling_Language>
Object402

Attribute/Reference	Literal/Instance
name	"OWL-S"
version	"1.2"
expressed_model	Object401

<Event>
Object403

Attribute/Reference	Literal/Instance
name	"Query_Bus_Information_Started"
triggered_process	Object404

<Process>
Object404

Attribute/Reference	Literal/Instance
name	"Select_Query_Port"
containing_model	Object401
trigger	Object403
successor	Object405

<Split_Dependency>
Object405

Attribute/Reference	Literal/Instance
split_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object401
preceding_element	Object404
following_option	Object406,Object407

<Split_Dependency_Option>
Object406

Attribute/Reference	Literal/Instance
guard_condition	"Known_Location_Address"
following_element	Objec408
precedent	Object405

<Split_Dependency_Option>
Object407

Attribute/Reference	Literal/Instance
guard_condition	"Known_Route_Number"
following_element	Object409
precedent	Object405

<Process>
Objec408

Attribute/Reference	Literal/Instance
name	"Query_With_Location"
containing_model	Object401
preceding_option	Object406
successor	Object411

<Process>
Object409

Attribute/Reference	Literal/Instance
name	"Query_With_Route"
containing_model	Object401
preceding_option	Object407
successor	Object412

<Join_Dependency>
Object410

Attribute/Reference	Literal/Instance
join_dependency_type	"OR"
is_synchonous	"FALSE"
containing_model	Object401
following_element	Object413
preceding_option	Object411, Object412

<Join_Dependency_Option>
Object411

Attribute/Reference	Literal/Instance
guard_condition	"Get_Bus_Information"
preceding_element	Object408
following_element	Object410

<Join_Dependency_Option>
Object412

Attribute/Reference	Literal/Instance
guard_condition	"Get_Bus_Information"
preceding_element	Object409
successor	Object410

<Process>
Object413

Attribute/Reference	Literal/Instance
name	"Provide_Bus_Information"
containing_model	Object401
precedent	Object410
successor	Object414

<Event>
Object414

Attribute/Reference	Literal/Instance
name	"Bus_Informaiton_Query_Closed"
producer	Object413

Figure A.9 — Registration of the Query_Bus_Information example

Annex B
(informative)

List of process modelling languages

It is advisable that the instance of 'Process_Modelling_Language' be one of the values in column 'name' of Table B.1.

Table B.1 — List of Process_Modelling_Languages

Name	Description
BPMN	Business Process Model and Notation, Object Management Group, 2011[1]
BPEL	Business Process Execution Language for Web Service (BPEL/BPEL4WS), 2003–05–03, Version 1.1[2]
UML	A language that conforms to ISO/IEC 19505-2 Information technology – OMG Unified Modeling Language (OMG UML) Version 2.1.2 – Part 2: Superstructure.
PSL	A language that conforms to ISO/IEC 18629 Process Specification Language[3].
IDEF3	IDEF3 (Integrated DEFinition for Process Description Capture Method) is a business process modelling method. It is a scenario-driven process flow description capture method intended to capture the knowledge about how a particular system works[4].
EPC	Event-driven Process Chain (EPC) is a type of flowchart used for business process modelling. It was developed in the early nineties in a joint effort between researchers at the University of Saarland and SAP[6].
OWL-S	Ontology Web Language for Services supplies Web service providers with a core set of markup language constructs for describing the properties and capabilities of their Web services in unambiguous, computer-interpretable form[7].

Bibliography

[1] Business Process Model and Notation (BPMN) Version 2.0, OMG Document Number: formal/2011-01-03, January, 2011. Available at: http://www.omg.org/spec/BPMN/2.0.

[2] Business Process Execution Language for Web Services. (BPEL 1.1), 2003-05-05. Available at: http://xml.coverpages.org/BPELv11-May052003Final.pdf.

[3] ISO 18629-1:2004, *Industrial automation systems and integration — Process specification language — Part 1: Overview and basic principles*

[4] IDEF3 Process Description Capture Method Report, September 1995. Available at: http://www.idef.com/pdf/Idef3_fn.pdf.

[5] ISO/IEC 19505-2, *Information technology — Object Management Group Unified Modeling Language (OMG UML) — Part 2: Superstructure*

[6] Keller G., Nüttgens M., Scheer A.-W. Semantische Prozebmodellierung auf der Grundlage", publication of the Institut für Wirtschaftsinformatik, paper 89, Saarbrucken, 1992

[7] Ontology Web Language for Service. OWL-S1.2 2008-12, available at: http://www.daml.org/services/owl-s/.

ICS 35.040

Price based on 35 pages