ISO

TC 184/SC4

AP 233 Data model Proposal

N 768

SYSTEMS ENGINEERING DATA REPRESENTATION

Introduction

This document represents the third draft of the data model for Capability/2.

The layout of the document is loosely based on the format required for ISO. In Section 4 we have attempted to stay close to ISO/STEP representation.

Section 1 - 3 will be extended with available material, but are not directly relevant to the data model.

Table of Contents

1 SCOPE

2 NORMATIVE REFERENCES

3 DEFINITIONS AND ABBREVIATIONS

This shall be extended with information from proposal made by AS.

4 INFORMATION REQUIREMENTS

This clause specifies the information required for SEDRES Cap/2.

4.1 Units of Functionality

This subclause specifies the units of functionality for SEDRES Cap/2.

This part of ISO XXXX specifies the following units of functionality:

4.1.1 Functional

This unit of functionality provides the capability to

The following application objects are used by the Functional UoF:

- Actual_flow_port
- Aggregation
- Boolean_type_definition
- Causal_chain
- Causal_relationship
- Clock
- Composite_data_type_definition
- Composite_function_defintion
- Context_system_view_relationship
- Control_port
- Data_composition_relationship
- Data_instance
- Data_type_definition
- Enumeration_type_definition
- Event_definition
- Execution_time
- External_entity_context_relationship
- Flow
- Flow_composition_relationship
- Flow_group
- Flow_port
- Flow_port_binding
- Formal_flow_port
- Function_context_definition
- Function_instance
- General_function_definition
- Hierarchical_decomposition_relationship
- Integer_bound
- Integer_type_definition
- Leaf_function_definition
- Null_type_definition
- Number_type_definition
- Real_bound
- Real_type_definition
- Store
- String_type_definition
- System_function_association
- Unit

4.1.2 Requirements

This unit of functionality provides the capability to capture requirements in the system engineering process.

The following application objects are used by the requirement UoF:

– application
– constraint_definition
– data_transfer
– derived_requirement_relationship
– external_entity
– external_to_requirement_association
– functional_requirement_definition
– operational_requirement_definition
– operational_scenario
– physical_requirement_definition
– requirement_allocation_relationship
– requirement_alternative_relationship
– requirement_composition_relationship
– requirement_definition
– requirement_instance
– requirement_system_view_relationship
– scenario_relationship
– system_view
– validation

4.1.3 Configuration management

This unit of functionality provides the capability to represent administrative, process management and configuration information about modeling entities.

The following application objects are used by the configuration management UoF:

– Activity
– Activity_element_assignment
– Activity_relationship
– Address
– Alternative
– Approval
– Approval_assignment
– Approval_date_time
– Approval_person_organization
– Approval_relationship
– Configuration_management_item
– Date
– Date_assignment
– Documentation
– Documentation_reference
– Documentation_relationship
– Element
– Element_version
– Element_version_relationship
– External_document
– General_template
– Justification
– Justification_relationship
– Maturity
– Model_configuration_management_item

- Non_versionable_model_configuration_management_item
- Organization
- Organization_address
- Organization_relationship
- Package
- Package_element_assignment
- Person
- Person_and_organization
- Person_or_organization_assignment
- Personal_address
- Process_configuration_management_item
- Project
- Revision
- Template_element
- Variant
- Versionable_model_configuration_management_item
- Workspace_revision

4.1.4 Behaviour

This unit of functionality provides mechanisms for representing flat and basic hierarchical finite state machines.

The following application objects are used by the behavioral UoF:

- Action_instance
- Action_link
- Action_precedence_relationship
- Actual_name_port
- Actual_transition_port
- AND_state_definition
- Composite_state_definition
- Default_transition
- Formal_name_port
- Formal_transition_port
- FSM_model
- Function_reference
- General_transition
- Leaf_state_definition
- Name_binding
- OR_state_definition
- State_composition_relationship
- State_definition
- State_instance
- Transition
- Transition_composition_relationship
- Transition_expression
- Transition_port
- Transition_port_binding
- Transition_relationship

4.1.5 Graphics

This unit of functionality provides mechanisms for representing simple geometric representation of elements of the functional and behavioral unit of functionalities.

The following application objects are used by the Graphics UoF:

- Actual_port_position

- Formal_port_position
- Graphics_link
- Graphics_node
- Graphics_point
- Graphics_view
- Multi_level_view
- View_relationship
- Visual_elements

4.2 Application Objects

This subclause specifies the application objects for the Functional Unit of Functionality.

4.2.1 Action_instance

An Action_instance is a definition of something that is executed as the Transition the Action_instance is associated with is taken alternatively when the state the Action_instance is associated with is activated.

See 4.3.1 for the Application Assertions.

The data associated with an Action_definition are the following:

- action_for
- description
- language

4.2.1.1action_for

The action_for specifies the Transition (see 4.2.121) or State_definition (see 4.2.114) that this action is associated with.

4.2.1.2 definition

The definition specifies an informal textual description of the action.

4.2.1.3 language

The language specifies the language used to write the action definition.

EXAMPLE: The language may be *structured_text* or *First order logic*.

4.2.2 Action_link

An Action_link is a formal link between the textual description of an Action_instance (see4.2.1) and the Function_instance (see 4.2.64) that is controlled by the action.

The Action_link references the controlled Function_instance object via the element attribute to a Formal_name_port (see 4.2.59) that eventually (through Actual_name_port (see 4.2.8), Name_binding (see 4.2.84), Formal_name_port (see 4.2.59) and Function_reference (see 4.2.65) objects) identifies the controlled Function_instance object.

The type of action performed by the Action_link is described by the command attribute and the Action_instance that activates the Action_link is identified via the action attribute.

The data associated with the Action_link are the following:

- action;
- command;
- element.

4.2.2.1 action

The action specifies the Action_instance (see 4.2.1) object that initiates the Action_link.

4.2.2.2 command

The command specifies how the controlled Function_instance object is affected by the Action_link. There are four types of commands supported:

– resume: The Function_instance is resumed from its last active state. If the last active state is unknown the function is started in its default state. If a resume command is issued to active function the command is discarded by the receiver.
– start: The Function_instance is started in its default state. If a start command is issued to active function the command is discarded by the receiver.
– stop: : The Function_instance is stopped, no record the last state of the Function_instance is kept. If a stop command is issued to stopped function the command is discarded by the receiver.
– suspend: The Function_instance is stopped but a record of its state is kept. If a resume command is issued to activate the function the functions will start is last active state. If the resume command is issued to an active function the command is discarded by the receiver.

It is not possible to create formal links from the behavioral to the functional view for any other commands than those listed above.

4.2.2.3 element

The element specifies the Action_instance object for which the Action_link provides the formal definition.

4.2.3 Action_precedence_relationship

An Action_precedence_relationship indicates that there is some type of temporal relationship between two Action_instance objects (see 4.2.1).

See 4.3.3 for the Application Assertions.

The data associated with an Action_precedence_relationship are the following:

– preceding
– succeeding
– relationship_type

4.2.3.1 preceding

The preceding specifies the first function instance involved in this relationship. The exact semantics of the attribute are dependent on the value of the type attribute.

4.2.3.2 succeeding

The succeeding specifies the second function instance involved in this relationship. The exact semantics of the attribute are dependent on the value of the type attribute.

4.2.3.3 relationship_type

The relationship_type specifies the semantics of the relationship. Where appropriate the following values should be used:

– Sequence: The preceding Action_instance is executed before the succeeding.
– Concurrent: The preceding and succeeding Action_instances are started concurrently.

4.2.4 Activity

An Activity is a part of the process of developing a system. It records the process steps used in either a company or in a standardized process (see IEEE P1220, EIA IS 632). An Activity may receive a set of inputs and may produce a set of information that is recorded a package. An activity has a goal to achieve and is performed by a person within a company facility (see Person_or_organization_assignment item for both the notion of the industrial responsibility and the notion of task accomplishment). Moreover an activity belongs to a process and ultimately by the end of it the system is ready for realization and production. The Activity_relationship (see) item shall be used to describe the structure of such a process. Objects of type Activity are the building blocks used to create a process structure for a Project (see 4.2.102).

Activity_relationship (see 4.2.6) objects can be used to create hierarchies of activities.

Every different type of activity that occurs as part of a project is modeled with an Activity object, Some Activity objects will represent whole phases of the project while others will represent simple design activities. It shall be noticed that each company may have its own standard process (or a process can be agreed on a project basis among partners). Therefore, the present standard does not give any rules or performs any consistency check on this item.

EXAMPLE 1. The requirements analysis phase of the system engineering process would be considered an activity. Performing safety analysis on a certain subsystem would also be considered as an activity.

Industrial responsibility, start and end date etc. can be given to an activity via various assignments that can be made to its super type Configuration_management_item (see 4.2.28).

EXAMPLE 2. An activity can be given a start date via a Date_assignment object with role specifying that it is the start date for the object.

EXAMPLE 3. The responsibility Organization/person for activity can be assigned via a Person_or_organization_assignment object with the role attribute indicating the responsibility held by the Organization/person.

The data associated with an Activity are the following:

- name
- status

4.2.4.1 name

The name specifies the word or set of words by which the Activity is referred to.

The naming convention used for activities and whether or not unique names shall be used shall be decided on a project to project basis.

4.2.4.2 status

The status specifies the progress of the Activity.

Wherever applicable one of the following terms should be used for status:

- 'not_begun': identifies an activity that belongs to the process but which has nor been performed yet,
- 'in_progress': identifies an activity producing design items,
- 'completed': identifies an activity which had produced a baselined output package,
- 'canceled': identifies an activity which has been discarded from the work plan,
- 'delayed': identifies an activity planned in the work plan but which expected start date has been delayed for some reason (note the use of the justification item).

The present standard does not perform any consistency check on the dates provided for an Activity object. This point is clearly left to the tools that handle this information.

In the same way, the set of values that are defined in the present standard can be extended. In that case, both the sending and the receiving tool shall agree upon the value set (data exchange convention documents)

4.2.5 Activity_element_assignment

An Activity_element_assignment specifies a relationship between a Model_configuration_management_item (see 4.2.82) and an Activity (see 4.2.4). The nature of the relationship is indicated by the role attribute.

The semantics of the relationship between the Activity and the element are dependent on the description contained by the role attribute.

Document_reference objects can be assigned to activities to indicate documentation that should be produced during the execution of the activity.

EXAMPLE 4. An Activity_element_assignment object may indicate that an Activity object called *Requirements Analysis* should produce a document, represented by an object of type Documentation_reference called *Agreed Requirements*.

This is the method by which design information is associated with the activities they are relevant in.

EXAMPLE 5. An Activity_element_assignment object may indicate a relationship between an Activity *Testing* and an Model_configuration_management_item for a physical model of a particular hardware component with a role description indicating that the model was *tested* during that Activity. The model may also be related to other Activity, such as Design, with a role of created during.

See 4.3.9 and 4.3.10 for the Application Assertions.

The data associated with an Activity_element_assignment are the following:

- activity
- description
- element
- name

4.2.5.1 activity

The activity specifies the Activity the Activity_element_assignment belongs to.

There shall be exactly one object that defines the activity for an Activity_element_assignment.

4.2.5.2 description

The description specifies more information about a particular association between the element and the activity.

EXAMPLE 6. An Activity_element_assignment entity may have a role_description *for internal use only* or *this documentation will need to be approved by the customer*.

Descriptions may be arbitrarily detailed as required.

4.2.5.3 element

The element specifies Model_configuration_management_item (see 4.2.82) is being assigned to the activity.

4.2.5.4 name

The name specifies the word or set of words by which the Activity_element_assignment is referred to.

EXAMPLE 7. An Activity_element_assignment entity may have a name *initial report* or *manufacturer requirements*.

4.2.6 Activity_relationship

An Activity_relationship is a relationship between two Activity (see4.2.4) objects. Activity_relationship objects can be used either to describe hierarchies, arbitrarily complex sequences of activities or other activity structures.

EXAMPLE: For instance, functional design activities is performed after the requirement elicitation activity, and before the physical design activity, and it splits up into preliminary functional analysis, functional trade-off, detailed functional analysis, etc.

See 4.3.11 for the Application Assertions.

The data associated with an Activity_relationship are the following:

- description
- name
- related_activity
- relating_activity

4.2.6.1 description

The description specifies a textual description of the nature of the relationship between the two Activity (see 4.2.8) objects involved in the relationship.

Where applicable one of the following values shall be used:

- ‘alternative’: Either the relating_activity or the related_activity shall be performed.
- ‘hierarchy’: The relationship between the relating_activity and the related_activity is such that the relating_activity is broken down into the related_activity.
- ‘sequence’: The relating_activity and the related_activity shall be performed in sequence where the relating_activity precedes the related_activity.
- ‘simultaneous’: The relating_activity and the related_activity may be performed simultaneously.

4.2.6.2 name

The name specifies the word or set of words by which the Activity_relationship is referred to.

4.2.6.3 related_activity

The related specifies the second of the two Activity (see 4.2.8) objects related by an Activity_relationship.

NOTE - The semantic of this attribute is defined by the attributes name and description.

4.2.6.4 relating_activity

The relating specifies the first of the two Activity (see 4.2.8) objects related by an Activity_relationship.

NOTE - The semantic of this attribute is defined by the attributes name and description.

4.2.7 Actual_flow_port

An Actual_flow_port is a port associated with a Function_instance (see 4.2.71) , External_entity (see 4.2.50), Store (see 4.2.116), or FSM_model (see 4.2.62). An Actual_flow_port is connected to at least one Flow (see 4.2.53) and bound, via a Flow_port_binding (see 4.2.57) object, to a Formal_flow_port (see 4.2.58) in order to correctly associate formal and actual parameters in a functional hierarchy.

NOTE: Flow ports are classified according to two criterion in the data model: 1. Whether the port is formal (attached to a Composite_function_definition) or actual (attached to a External_entity, Function_instance, FSM_model or Store), and 2. Whether the port is an input or output port. From this classification it can be derived the role the port plays with regard to the Flow objects connected to the port. A port can ‘consume’ flows (the data on the flow is delivered to the port) or it can produce flows (the data on the flow is delivered from the port). For instance, a Formal_flow_port object whose direction attribute is ‘input’ produces data, while an Actual_flow_port whose direction attribute is ‘input’ consumes data. In the data model we capture this information in the derived role attribute. This attribute is used to ensure that Flow object s are connected correctly (a Flow object must have on producer and one consumer), and that Flow_port_binding objects applied only to ports where the actual_port in the binding is a producer of information and the formal_port in the binding is a consumer or vice versa.

See 4.3.6, 4.3.4, 4.3.7 and 4.3.5 for the Application Assertions.

The data associated with Actual_flow_port are the following:

– buffered;
– port_of;
– role.

4.2.7.1 buffered

The buffered specifies, using a Boolean, whether the Actual_flow_port has a FIFO queue of infinite capacity associated with itself. If buffered is TRUE it means that all data values associated with the Actual_flow_port are temporarily stored until read. If buffered is FALSE no track of changed data is kept.

The record of buffering of data is only kept in Actual_flow_port objects. If the Function_instance object the Actual_flow_port is associated with (via the port_of attribute) is decomposed then no record of the fact the data may be buffered is kept.

If the Actual_flow_port is an output port (and buffered) it is implied that a particular data produced by the port can be read by one consuming input port only (Multicast).

If the Actual_flow_port is an input port (and buffered) it is implied that all data received by the port is temporarily stored in the port until the data is read by the object (Function_instance, external_entity, store) the Actual_flow_port is associated with via the port_of attribute. If many Actual_flow_port objects consume the same data (and are buffered) the data values are stored locally at each port it can be consumed individually by each port (Broadcast).

4.2.7.2 port_of

The port_of attribute indicates the Function_instance (see 4.2.64), External_entity (see 4.2.50), Store (see 4.2.116), or FSM_model (see 4.2.62) that an Actual_flow_port is associated with.

At most one Function_instance, External_entity, Store, or FSM_model can be port_of of a particular Actual_flow_port.

4.2.7.3 role

The role attribute specifies whether the Actual_flow_port is consuming or producing data.

NOTE: The value of the role attribute can be derived: If the direction attribute (inherited from Flow_port) is ”input”, then the port consumes data else the port is producing data.

4.2.8 Actual_name_port

An Actual_name_port is an interface of a State_instance (see 4.2.115) such that it imports the knowledge of the existence of a Function_instance (see 4.2.64) known in the context of State_instance so that it can be made available to the elements in the State_definition that the State_instance the Actual_name_port is associated with.

The Actual_name_port is used in cooperation with Formal_name_port (see 4.2.59), Function_reference (see 4.2.65) and Name_binding (see 4.2.84) for performing the formal name mapping from scope of visibility defined in the context of a State_instance and the State_definition (see 4.2.114) that provides the definition of the State_instance.

The data associated with Formal_name_port are the following:

– port_of;
– reference_port

4.2.8.1 port_of

The port_of specifies the State_instance that the Actual_name_port is associated with.

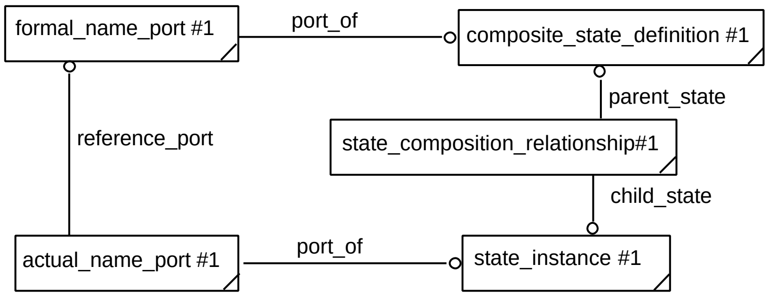
An Actual_name_port is associated with exactly one State_definition object.

4.2.8.2 reference_port

The reference_port specifies the Formal_name_port object that provides the name reference for the Actual_name_port.

The Formal_flow_port specified by reference_port shall be directly associated (via attribute port_of) with the State_definition that the State_instance specified in the port_of attribute of the Actual_name_port is part of the decomposition of.

The figure below illustrates the relationship.



Multiple Actual_name_port objects may have the same Formal_name_port object as their reference_port.

4.2.9 Actual_port_position

An Actual_port_position is the representation of the graphical position of an Actual port (Actual_flow_port (see 4.2.7) or Actual_transition_port (see 4.2.10)) object. The position is represented by a Graphics_point (see 4.2.72) object. Since the actual port may have many different positions (the object the port is a port_of may be present in many contexts.) the Actual_port_position also identifies the decomposition object for which the positions are valid.

The Actual_port_position shall point to a position which is on the border of the Graphics node of the object the port is associated with.

Development NOTE: This rule is not formalized in EXPRESS.

An Actual_port_position is a type of Visual_element (see 4.2.132).

The data associated with Actual_port_position are the following:

- position;
- positioned_port.
- position_in_decomposition

4.2.9.1 position

The position specifies the position of the port.

4.2.9.2 positioned_port

The positioned_port specifies the actual port that is positioned by the Actual_port_position object.

4.2.9.3 position_in_decomposition

The position_in_decomposition identifies the decomposition for which the Actual_port_position is valid.

4.2.10 Actual_transition_port

An Actual_transition_port is a type of Transition_port (see 4.2.124) that is associated with a State_instance via the port_of attribute.

If the role of the Actual_transition_port is producer then the Actual_transition_port represents a place where a Transition begins, else the Actual_transition_port represents the end of a Transition.

> NOTE: Transition ports are classified according to two criterion in the data model: 1. Whether the port is formal (attached to a Composite_state_definition) or actual (attached to a State_instance), and 2. Whether the port is an input or output port. From this classification it can be derived the role the port plays with regard to the Transition objects connected to the port. A port can 'consume' transitions (the transition end in the port) or it can produce transition (the transition start in the port). For instance, a Formal_transition_port object whose direction attribute is 'input' produces Transition objects, while an Actual_transition_port whose direction attribute is 'input' consumes Transition objects. In the data model we capture this information in the derived role attribute. This attribute is used to ensure that Transition objects are connected correctly (a Transition object must have on producer and one consumer), and that Transition_port_binding objects is applied only to ports where the actual_port in the binding is a producer of transitions and the formal_port in the binding is a consumer or vice versa.

An Actual_transition_port is a type of Transition_port (see 4.2.124).

See 4.3.8 for the Application Assertions.

The data associated with an Actual_transition_port are the following:

- port_of;
- role.

4.2.10.1 port_of

A port_of specifies the State_instance for which this port represents an entry or exit point.

4.2.10.2 role

The role specifies whether the Actual_transition_port is a consumer or a producer of a transition.

4.2.11 Address

An Address object specifies an address.

> NOTE - This entity is taken directly from the PDM Schema.

The data associated with an Address are the following:

- country
- electronic_mail_address
- facsimile_number
- internal_location
- postbox_number
- postcode
- region
- street
- street_number
- telephone_number
- telex_number
- town

4.2.11.1 country

The country specifies the country of the address.

The country need not be specified for a particular address.

4.2.11.2 electronic_mail_address

The electronic_mail_address specifies an electronic address associated with this address.

The electronic_mail_address need not be specified for a particular address.

4.2.11.3 facsimile_number

The facsimile_number specifies a fax associated with this number.

The facsimile_number need not be specified for a particular address.

4.2.11.4 internal_location

The internal_location specifies an specific location within an address.

EXAMPLE 8. An internal_location may be a room number or a floor of a building.

The internal_location need not be specified for a particular address.

4.2.11.5 postbox_number

The postbox_number specifies a postbox associated with the address.

The postbox_number need not be specified with a particular address.

4.2.11.6 postcode

The postcode specifies a postcode associated with the address.

The postcode need not be specified for a particular address.

4.2.11.7 region

The region specifies a postcode associated with the address.

The region need not be specified for a particular address.

4.2.11.8 street

The street specifies a postcode associated with the address.

The street need not be specified for a particular address.

4.2.11.9 street_number

The street_number specifies a postcode associated with the address.

The street_number need not be specified for a particular address.

4.2.11.10 telephone_number

The telephone_number specifies a postcode associated with the address.

The telephone_number need not be specified for a particular address.

4.2.11.11 telex_number

The telex_number specifies a postcode associated with the address.

The telex_number need not be specified for a particular address.

4.2.11.12 town

The town specifies a postcode associated with the address.

The town need not be specified for a particular address.

4.2.12 Aggregation

An Aggregation is a definition of a *n*-dimensional data structure where all elements are of the same data_type.

NOTE: Aggregation objects in conjunction with Data_instance objects can be used produce vectors or matrices.

The Aggregation object is associated with a Data_instance (see 4.2.33) object to indicate that the particular Data_instance is aggregated.

The data associated with Aggregation are the following:

– aggregation_for;
– aggregation_list.

4.2.12.1 aggregation_for

The aggregation_for specifies the Data_instance object that the aggregation applies to.

4.2.12.2 aggregation_list

The aggregation_list specifies the list that represents the aggregation bounds for the aggregated Data_instance. The number of elements in the list shall at least be one. Each element in the list represents one dimension in the aggregation. The value of each element in the aggregation_list shall be non-negative.

EXAMPLE: A two dimensional matrix may have an aggregation_list with values [2,3] indicating a 2 by 3 matrix.

If the value of an element in the list is zero it is implied that no upper bound of the aggregation is defined. For all other integer values *n* for an element in the aggregation_list the aggregation (in the dimension) is defined in the range from 1 to *n*.

4.2.13 Alternative

An Alternative represents a relationship between two Element_version (see 4.2.44) objects where the two Element_versions are alternative implementations of the requirements. This means that one can either chose one or the other solution (criteria have to be given as a justification of the choice - This point depends upon the project organization). Alternative item can be exchanged without any impact from an external point of view. For instance, a function has a set of inputs and outputs, and two decomposition are proposed. Then designers can either chose one or the other (according to criteria) without making any changes in the environment in which the function will take place.

An Alternative is a type of Element_version_relationship (see 4.2.45).

EXAMPLE 9. In preliminary design the need for a subsystem to measure airspeed in a plane might be identified. Designs involving different types of measuring devices may be considered. These different designs should be considered Alternatives.

NOTE - This entity has no extra attributes associated with it. It is included solely to improve the semantics of the model.

4.2.14 AND_state_definition

An AND_state_definition is the definition of a state, which has two or more parallel state machines as children.

An AND_state_definition is a type of Composite_state_definition (see4.2.27).

EXAMPLE : A single state *operate* may be hierarchically decomposed into to child states *operate console* and *operate machine*, via two State_composition_relationship objects. When the fsm is in the high level state*operate* it will be in both *operate machine* AND *operate console.*

Note: An AND_state_definition must be composed of at least two State_instance objects.

4.2.15 Application

An Application represent the view of the complete system under development. The Application is a collector object to which Operational_scenario (see 4.2.89) objects, Requirement_instance (see 4.2.109) objects and Function_context_definition (see 4.2.63) objects pertinent to the system under development can be assigned.

An Application is a type of System_view (see 4.2.119).

4.2.16 Approval

An Approval is a formal statement from a suitably qualified person indicating acceptance of the Configuration_management_item (see 4.2.28).

This entity can have temporal information assigned to it via an Approval_date_time object.

The data associated with an Approval are the following:

- level
- status

4.2.16.1 level

The level provides some indication of the approval level.

4.2.16.2 status

The status specifies a label which identifies the status of the Approval.

EXAMPLE 10. A status for an approval to a model_configuration_management_item object might be *preliminary_approval.*

4.2.17 Approval_assignment

An Approval_assignment entity is an entity that is used to associate an Approval (see4.2.14) with an Configuration_management_item (see 4.2.28).

See 4.3.12 and 4.3.13 for the Application Assertions.

The data associated with an Approval_assignment are the following:

- assigned_approval;
- description;
- item;
- status.

4.2.17.1 assigned_approval

The assigned_approval specifies the Approval object that is being assigned.

4.2.17.2 description

The role_description specifies the role the Approval plays for the item.

The description is optional and need not be specified for a particular Approval_assignment.

4.2.17.3 item

The item specifies the Configuration_management_item objects to which the Approval is being assigned.

4.2.17.4 status

The status specifies the status of the approval_assignment.

EXAMPLE 11. A status for an approval_assignment to a model_configuration_management_item object might be *preliminary_approval_assignment*.

4.2.18 Approval_date_time

An Approval_date_time is an object for associating a date and/or time with an Approval (see 4.2.14).

NOTE - This entity is taken directly from the PDM Schema.

NOTE - This entity is effectively a type of Approval with associated temporal information.

See 4.3.14 for the Application Assertions.

The data associated with an Approval_date_time are the following:

- date
- dated_approval
- role

4.2.18.1 date

The date specifies the date and/or time which is being associated with the Approval.

4.2.18.2 dated_approval

The dated_approval specifies the Approval with which the date and/or time is being associated.

4.2.18.3 role

The role specifies the meaning of the date with regards to this approval.

EXAMPLE: The role may indicate that the date is the preliminary time of the approval or that it was when the approval was made by a customer. If the date is a future date the role may indicate that the date is the expected date of approval.

4.2.19 Approval_person_organization

An Approval_person_organization is the entity via which a person and/or organization can be associated with an Approval (see 4.2.14).

NOTE - This entity is taken directly from the PDM Schema.

The data associated with an Approval_person_organization are the following:

- authorized_approval
- person_organization
- role

4.2.19.1 authorized_approval

The authorized_approval specifies the Approval which this person and/or organization is being associated.

4.2.19.2 person_organization

The person_organization specifies the person and/or organization associated with the Approval.

4.2.19.3 role

The role specifies the role that the person and/or organization plays for the Approval.

4.2.20 Approval_relationship

An Approval_relationship is the entity by which two arbitrary Approval objects (see4.2.14) can be related to each other. The nature of the relation is defined by the description attribute.

See 4.3.14 for the Application Assertions.

The data associated with an Approval_relationship are the following:

- description
- name
- related_approval
- relating_approval

4.2.20.1 description

The description specifies a textual description of the relationship between the two Approval objects.

4.2.20.2 name

The name specifies a name associated with the two Approval objects.

4.2.20.3 related_approval

The related_approval specifies the second of the two Approval objects involved in the relationship.

NOTE - The exact semantics of this attribute depends on the value of the description attribute.

4.2.20.4 relating_approval

The relating_approval specifies the first of the two Approval objects involved in the relationship.

NOTE - The exact semantics of this attribute depends on the value of the description attribute.

4.2.21 Boolean_type_definition

A Boolean_type_definition is the definition of a Boolean data type.

The value set of a Boolean_type_definition is {true, false}.

A Boolean_type_definition is a type of Data_type_definition (see4.2.35).

The data associated with a Boolean_type_definition are the following:

– default_value;
– initial_value.

4.2.21.1 default_value

The default_value specifies the value to be used if the value assigned a given Data_instance is out of range.

The default_value is optional and need not be specified for a particular Boolean_type_definition.

4.2.21.2 initial_value

The initial_value specifies the value to be assigned the object when it is created, for instance at system power-up.

The initial_value is optional and need not be specified for a particular Boolean_type_definition.

4.2.22 Causal_chain

A Causal_chain is a simple mechanism for representing sequences of function execution traces through a system. Any number of Causal_relationship (see 4.2.23) objects can be associated with a single Causal_chain object. From the Causal_relationship objects information such as threads or causal chains through a system can be derived.

A Causal_chain is a type of Versioned_configuration_management_item (see 4.2.130).

Causal_chains exist for the purpose of examining the performance of the system. Typically Causal_chains would be used to examine the start to end timing for a sequence of functions or the accuracy of a calculation performed by a sequence of functions.

The data associated with a Causal_chain are the following:

– description;
– name.

4.2.22.1 description

The description specifies a textual description of the intended semantics of the grouping of the Causal_relationship objects.

EXAMPLE : A description may be This causal chain represents the sequence of functions that will be executed when the system is first turned on.

4.2.22.2 name

The name specifies the word or set of words to which the Causal_chain is referred to.

4.2.23 Causal_relationship

A Causal_relationship is the causal relation between two Function_instance (see 4.2.71) objects such that the caused Function_instance is affected by a status change in the causing Function_instance. The semantics of the relation is defined by the type attribute. Causal_relationship objects that represent the same chain of function are grouped by a Causal_chain (see 4.2.22) object.

The Function_instance objects joined by a Causal_relationship will also be joined by data flows. Individually the functions may also receive data flows from other functions in the chain and sources external to the functional chain or system. The dynamics of data flows received by a function will determine when it is performed.

The causing and caused Function_instance shall be part of the composition of the same Composite_function_definition.

See 4.3.18 and 4.3.19 for the Application Assertions.

The data associated with a Function_instance are the following:

– caused;
– causing;
– chain;
– description;
– type.

4.2.23.1 chain

The chain specifies the Functional_chain the Causal_relationship is part of.

4.2.23.2 causing

The causing specifies the first of the two Function_instance (see 4.2.71) objects involved in the relationship.

4.2.23.3 caused

The caused specifies the second of the two Function_instance (see 4.2.71) objects involved in the relationship.

4.2.23.4 description

The description specifies the text describing some aspect of the causal relationship.

The description need not be specified for a particular Causal_relationship.

4.2.23.5 type

The type specifies the relation between the causing and caused Function_instance. Where appropriate one of the following values shall be used for the type.

– ‘Concurrent’: The causing and caused Function_item objects are activated at the same instance in time.
– ‘Mutually_exclusive’: The causing and caused Function_item objects are mutually exclusive. Only one of the Function_item objects may be active at any time.
– ‘Sequential’: The relation between causing and caused Function_item objects is such that the caused Function_item object is activated by the termination of the causing Function_item object.

4.2.24 Clock

A clock is a logical device that emit events (clock ticks) at periodic intervals.

A clock is associated with a Control_port (see 4.2.31) and is used to define the synchronous behavior of Function_instance (see 4.2.64) objects.

Development NOTE: The Clock entity is a replacement for the special function_definition type Synchronous_process. Using Clocks (associated with Control_ports) periodic activation of functions can be described without creating special function types (Data_flow_function_definition and Process_definition removed from model). Instead the following semantics is used in the updated model:
A Function_instance is considered to be continuously active unless it has one or more control_ports associated with itself. (A control_port is a port whose data part is used solely for activation/deactivation of functions). If a Function_instance has control_ports attached the function is considered active when the data of the control_port is non-zero (numeric case), TRUE (Boolean case) or when an event is received if the type of the data is event_definition. Using Control_ports an output signal from a function can be used to control another function, or a clock can be associated with one Control_port and create periodic activations of the Function.
An alternative to model the clock as an object directly associated with the control_port would have been to use Flows to connect Clocks to Functions. The advantage of this is that the model is more directly usable in typical SA tools, but the cost is that quite a lot of work must be performed to deduct that a function is synchronous based on the information in the data model.
In the alternative described we can have situations where a ”long” Flow spanning several hierarchy levels has as its source a Clock and as its destination the Control_port. In such cases it would be very difficult to detect that the function the Control_port is associated is in fact a synchronous process.
This modeling decision can be disputed, but we think the current model allows for easy implementation and is quite easy to understand.

The data associated with Clock are the following:

– control_signal;
– description;
– name;
– period;
– trigger_for.

4.2.24.1 control_signal

The control_signal specifies the Data_instance (whose definition shall be of type Event_definition) that the Clock produces at periodic intervals (given by the period value).

The Clock shall have exactly one Data_instance in the role of control_signal.

4.2.24.2 description

The description specifies a text describing the Clock.

4.2.24.3 name

The name specifies the word or set of words by which the Clock is referred to.

4.2.24.4 period

The period specifies the time intervals in seconds (or fractions there of) by which the Clock emits events.

4.2.24.5 trigger_for

The trigger_for specifies the Control_port that the Clock is associated with.

The Clock and Control_port shall reference the same Data_instance object in the attributes control_signal and data respectively.

4.2.25 Composite_data_type_definition

A Composite_data_type_definition is a specification for a Data_type_definition (see4.2.35) that is further decomposed. This extends the concept of working with data at different levels of abstraction, to having data types which can be represented at different levels of abstraction.

The data associated with Composite_data_type_definition are the following:

– composition_type

4.2.25.1 composition_type

The composition_type specifies the type of data composition that is defined for the Composite_data_type_definition. Where appropriate the following values shall be used:

– ‘Abstraction’: The Composite_data_type is made up of any combination of the child Data_instance objects.
– ‘Composition’: The Composite_data_type is made up of the collection of all child Data_instance objects.
– ‘Mutual_exclusion’: The Composite_data_type is made up of one of the child Data_instance objects.

4.2.26 Composite_function_definition

A Composite_function_definition represents a function definition that is decomposed in this model.

A Composite_function_definition is a type of General_function_definition (see4.2.67).

A Composite_function_definition shall be composed of at least one Function_instance (see 4.2.64), FSM_model (see 4.2.62) or Store (see 4.2.116).

4.2.27 Composite_state_definition

A Composite_state_definition is the representation of a state that has a breakdown structure which is composed of states (either leaf states or composed states) and of transitions. A

Composite_state_definition is either an AND_state_definition (see4.2.14) or an OR_state_definition (see 4.2.90). A Composite_state_definition shall have at least one State_instance (see4.2.115) object in its decomposition.

A Composite_state_definition is a type of State_definition (see4.2.115).

A Composite_state_definition is an abstract type, hence there will be no objects of type Composite_state_definition.

4.2.28 Configuration_management_item

A Configuration_management_item is the top of the hierarchy which allows the representation and association of general information to all the entities in the hierarchy. The Configuration_management_item is a super-type of Process_configuration_management_item (see 4.2.101) which deals with the management of information related to processes and Model_configuration_management_item (see 4.2.82) which deals with system engineering design item.

EXAMPLE: A Date_assignment object is associated with a Configuration_management_item object which means that zero or more dates, and associated roles, can be assigned to any model definition or activity or any other entity type in the Configuration_management_item inheritance graph.

A Configuration_management_item is an abstract type that shall never be instanciated.

4.2.29 Constraint_definition

A Constraint_definition is the definition of a design constraint. The Constraint_definition represents a requirement which gives a specific feature the system has to fulfill. For instance, the fact that a product shall be compliant with a given electromagnetic environment is recorded in this kind of requirement. The whole set of constraint requirement represents the « performances » of the system within its environment as well as its ability to be included in its environment. Therefore, a constraint requirement might result in a specific architecture (functional or physical) and the design of interfaces.

Constraint_definition is a type of Requirement_definition (see4.2.108).

4.2.30 Context_system_view_relationship

A Context_system_view_relationship is a relation between a Function_context_definition (see4.2.63) object and a System_view (see 4.2.119) object such that the Function_context_definition defines the context view for the particular System_view object.

The data associated with Context_system_view_relationship are the following:

- function_context;
- system_view.

4.2.30.1 function_context

The function_context specifies the Function_context_definition that is associated with the System_view

4.2.30.2 system_view

The system_view specifies the System_view object that is associated with the Function_context_definition.

4.2.31 Control_port

A Control_port is a definition of a behavioral interface for a Function_instance (see4.2.64) such that the Function_instance object can not be activated unless all Control_port objects associated with the Function_instance object are enabled. When a Control_port is enabled is decided by the value and type of the data attribute (inherited from Flow_port (see 4.2.56)) of the Control_port.

The Data_instance object identified by the data attribute shall be have any of these types: Number_type_definition (see 4.2.87), Event_definition (see 4.2.47) and Boolean_type_definition (see 4.2.21) as its definition value.

NOTE 1: The Data_type_definition object of the Data_instance object referenced by the Data attribute is of type Event_definition then the Control_port is enabled by the arrival of an event. Detected events are not buffered. The detection of the event activates the Function_instance and the function is active until it deactivates itself by its own account. If an event occurs when the Function_instance of the Control_port is active then the event is discarded and no further action is taken. If the Data_type_definition object of the Data_instance object referenced by the data attribute is of type Boolean_type_definition then the Control_port is enabled as long as the value of the data is TRUE. The Function_instance that the Control_port is a port_of is active as long as the Control_port is enabled. If the Data_type_definition object of the Data_instance object referenced by the data attribute is of type Number_type_definition then the Control_port is enabled as long as the value of the data is non zero. The Function_instance that the Control_port is a port_of is active as long as the Control_port is enabled.

NOTE 2: If there are several Control_port objects that have a single Function_instance object as their port_of attribute then the Function_instance object is activated only if all Control_port objects are enabled (AND relationship).

The value of the direction attribute (inherited from Flow_port) shall be ‘input’ for all Control_port objects.

A Control_port is a type of Flow_port.

The data associated with a Control_port are the following:

– offset;
– port_of.

4.2.31.1 offset

The offset specifies the time in seconds the receiving data values are delayed until they are evaluated by the Control_port.

The offset shall not be a negative value.

4.2.31.2 port_of

The port_of specifies the Function_instance object that the Control_port is associated with.

4.2.32 Data_composition_relationship

A Data_composition_relationship represents a hierarchical relationship between a composite parent data type and one of it’s constituent parts. It is a fundamentally important relationship to support the concept of data abstraction, i.e., being able to work with complex data at different level of abstraction. The semantics of the relationship are determined by the value of the role attribute.

EXAMPLE 1. A Composite_data_type_definition object named *3D Point* may have three Data_composition_relationship objects, with roles *x*, *y*, *z* respectively, each connected to a Data_instance object whose name may not be connected to the role the Data_instance plays in the composition.

See 4.3.20 and 4.3.23 for the Application Assertions.

The data associated with a Data_composition_relationship are the following:

- child;
- parent;
- role.

4.2.32.1 child

The child specifies one part of the parent data type.

4.2.32.2 parent

The parent specifies the data type being decomposed.

4.2.32.3 role

The role specifies in a textual format the role the child object of the relationship plays in the context of the parent object.

The role is optional and need not be specified for a particular Data_composition_relationship.

4.2.33 Data_instance

A Data_instance represents the instantiation of a Data_type_definition (see4.2.35).

NOTE: In the terminology used in computer science a Data_instance corresponds to a variable. Although in the systems engineering domain the Data_instance shall be interpreted in a wider sense.

See 4.3.24 for the Application Assertions.

The data associated with a Data_instance are the following:

- definition;
- is_constant;
- name.

4.2.33.1 definition

The definition specifies the Data_type_definition object that defines this instance.

4.2.33.2 is_constant

The is_constant specifies whether the Data_instance has a constant value or not. If is_constant is TRUE the value of the Data_instance must not change from the initial_value of the Data_type_definition that the Data_instance has as its definition value.

If the Data_instance is a constant (is_constant = TRUE) then there shall exist an initial_value for the Data_type_definition object that is indicated by the definition attribute of the Data_instance.

4.2.33.3 name

The name specifies the word or set of words by which the Data_instance is referred to.

EXAMPLE 1. The name for a Data_instance might be *engine_weight* or *rpm*.

4.2.34 Data_transfer

A Data_transfer is the definition of the data flow and its direction between an External_entity (see 4.2.50) and the system under specification.

A Data_transfer item allow the association of a data to the expression of a requirement. It defines the data that is transmitted between the external_entity and the system. There can be an arbitrary number of data associated with a requirement.

Since the data model makes no assumption on the template used in company using it, there is no consistency check encoded in the data model that would ensure that the data associated with a data_transfer item is consistent with the description of the requirement (this kind of check shall be ensured by the tools that implement the company rules).

See 4.3.25 for the Application Assertions.

The data associated with a Data_transfer are the following:

– data;
– direction;

– transfer.

4.2.34.1 data

The data is the Data_instance flowing between the External_entity and the system.

4.2.34.2 direction

The direction is the specification of the data flow direction between the External_entity and the system. The direction is an enumeration that can be either of: ‘to_system’ or ‘from_system’.

4.2.34.3 transfer

The transfer is the identification of the External_to_requirement_association (see4.2.52) object that the Data_transfer applies to.

4.2.35 Data_type_definition

A Data_type_definition is the specification for a data type.

NOTE 1: In the terminology used in computer science a data definition corresponds to a data type (basic or structured).

A Data_type_definition is a type of Versionable_model_configuration_management_item (see 4.2.130).

The data associated with a Data_type_definition are the following:

– description;
– name.

4.2.35.1 description

The description specifies additional information on the Data_type_definition.

The description is optional and need not be specified for a particular Data_type_definition.

4.2.35.2 name

The name specifies the word or set of words by which the Data_type_definition is referred to.

The name is optional and need not be specified for a particular Data_type_definition.

4.2.36 Date

A Date is the specification of a date and time of day.

Development NOTE: The capability 1 model has been reused since the Part 41 model uses quite a large number of entities that allows for alternate date and time representations. In the final proposal the part 41 definitions will be used.

The data associated with a Date are the following:

- day_component;
- hour_component;
- minute_component;
- month_component;
- second_component.
- year_component;

4.2.36.1 day_component

The day_component specifies the day number of a month. The value of the day_component must not exceed the number of days in the current month.

Development NOTE: The constraint in the model is just that the number of days may not exceed 31. Complete specifications are part of part 41.

4.2.36.2 hour_component

The hour_component specifies the time in a day in hours.

The value of the day_component shall be in the range 0-23.

4.2.36.3 minute_component

The minute_component specifies the time within an hour in minutes.

The value of the minute_component shall be in the range 0-59.

4.2.36.4 month_component

The month_component specifies the date within a year in months.

The value of the month_component must be in the range 1-12.

4.2.36.5 second_component

The second_component specifies the time within a minute in seconds.

The value of the second_component shall be in the range 0-59.

4.2.36.6 year_component

The year_component specifies the year part of a date.

The year_component shall be given with all valid digits.

EXAMPLE: The year 1999 shall be represented as the integer ‘1999’.

4.2.37 Date_assignment

A Date_assignment is the assignment of a date to a Configuration_management_item (see 4.2.28) object. The exact semantics of the assignment depend on the role attribute given to the object.

EXAMPLE 12. A Date_assignment object with a role *start_date* has quite a different meaning to a Date_assignment object with a role *end_date*.

The use of Date_assignment objects allows any number of dates to be assigned to any item in the Configuration_management_item hierarchy. By the use of this relationship entity we can assign to entities such as Activity (see4.2.4) information like ‘start date’, ‘end date’ and ‘review date’ and we can assign to modeling entities information like ‘creation date’, ‘modification date’ etc.

The data associated with a Date_assignment are the following:

- date
- item
- role

See 4.3.26 for the Application Assertions.

4.2.37.1 date

The date specifies the date that is being assigned.

4.2.37.2 item

The item specifies the Configuration_management_item to which the date is being assigned.

4.2.37.3 role

The role specifies the semantics of the date assigned for the item. Where applicable the following values shall be used:

– ‘creation’: The assignment specifies that the referenced object was created at the given date.
– ‘update’: The assignment specifies that the referenced object was altered at the given date.
– ‘start_date’: The assignment specifies that work on the referenced object was started at the given date.
– ‘end_date’: The assignment specifies that work on the referenced object was ended at the given date.

4.2.38 Default_transition

A Default_transition is the state transition object that identifies the transition to the state that will be the initial state of a Composite_state_definition (see4.2.27).

If there exist Transition (see4.2.121) objects from Formal_transition_port (see 4.2.61) objects of the Composite_state_definition then these supersedes the Default_transition if the state is entered through one if these transitions.

The Default_transition is a type of General_transition.

The shall be at most one Default_transition of a Composite_state_definition. If there are no Default_transition objects in a Composite_state_definition then the initial state of the state machine that is represented by the Composite_state_definition is not explicitly defined.

4.2.39 Derived_requirement_relationship

A Derived_requirement_relationship specifies the relation between a set of existing Requirement_instance (see 4.2.109) objects and a new Requirement_instance that is derived from the required properties given by the original set of Requirement_instance objects.

Derived_requirement_relationship can also be used to record requirements that have been identified during later design stages (for instance functional design and physical design activities). However the use of the relation implies that a reference to existing requirements shall be made.

The original and derived requirements involved in this relation shall not be considered to form a hierarchy.

NOTE: This entity shall be used when the derived Requirement_definition which provides the definition for the derived Requirement_instance contains new information compared with the Requirement_definition objects of the existing Requirement_instance objects. (Refer to 4.2.107 Requirement_composition_relationship if the new requirement is a strict subset of an existing Requirement_definition)

The data associated with a Derived_requirement_relationship are the following:

– derived_requirement;
– description;
– original_requirement.

NOTE: The Derived_requirement data must not refer a Requirement_instance which is in the set of original_requirement.

4.2.39.1 derived_requirement

The derived_requirement specifies the Requirement_instance that is derived from a set of existing requirements

4.2.39.2 description

The description specifies the textual information on how the Derived_requirement was derived from the set of original_requirement.

The description shall only capture information on the relation. It shall not be used for capturing information on the properties of the derived or original requirement set.

4.2.39.3 original_requirement

The original_requirement is a set of Requirement_instance objects used to derive the Derived_requirement

4.2.40 Documentation

A Documentation is a representation of some information intended to facilitate user communications.

A Documentation is a type of Document_reference (see 4.2.41).

The data associated with a Documentation are the following:

- text

4.2.40.1 text

The text specifies the body of the Documentation.

The text need not be specified for a particular Documentation.

4.2.41 Documentation_reference

A Documentation_reference is a reference to a document. The document referenced by the Documentation_reference can be already in existence or be proposed. This may be useful when creating process structures for a project as it allows required documentation and expected due dates to specified. The term documentation shall be understood widely since it can represents either paper based documentation or any multimedia documentation (pictures, drawings, sounds, diagrams...).

A Document_reference is only a reference to a document by name and description, the body of the document is not included. If the document is text a subtype of Documentation_reference, namely Documentation (see 4.2.39), can be used to represent the document completely.

The data associated with a Documentation_reference are the following:

- description
- name

4.2.41.1 description

The description specifies information about the contents of the document, whether that be the proposed contents or the existing contents of the document.

4.2.41.2 name

The name specifies the word or set of words by which the document is referred to.

4.2.42 Documentation_relationship

A Documentation_relationship represents an arbitrary relationship between two Documentation (see 4.2.39) objects.

The data associated with a Documentation_relationship are the following:

- description
- name
- related_documentation
- relating_documentation

See 4.3.28 for the Application Assertions.

4.2.42.1 description

The description specifies a textual description of the nature of the relationship between the two Documentation objects.

4.2.42.2 name

The name specifies a name given to the relationship between the two documentation objects.

4.2.42.3 related_documentation

The related_documentation specifies the second of the two documentation objects involved in the relationship.

NOTE - The semantics of this attributes are determined by the description attribute.

4.2.42.4 relating_documentation

The relating_documentation specifies the first of the two documentation objects involved in the relationship.

NOTE - The semantics of this attributes are determined by the description attribute.

4.2.43 Element

An element is either a single object or a unit in a group of objects. It collects information that is common to all versions of the object.

NOTE - Effectively an Element is the logical container for all versions of the same thing. For example three revisions of the same functional model should be collected together by an Element with a name and description of the functional model.

The data associated with an Element are the following:

- description
- name

4.2.43.1 description

The description specifies some text describing the Element.

4.2.43.2 name

The name specifies an text string by which to identify the particular Element.

4.2.44 Element_version

An Element_version is a snapshot of a design item for which a change history is maintained. A change history is kept from an initial item release and is recorded by Element_version. Therefore, the picture is composed of an Element and with a set of Element_version which record the various evolution of the given Element. In the present standard, design elements can come from the requirement UoF, function UoF, behavior UoF, documentation, data type. Within the data model, Element_version is only applied to entities that are subtypes of Versionable_model_configuration_management_item. This then leads to situation where a design is made up element instances referring to a particular version of the element represented. Therefore, a design can be made up of a set of element standing in different version. Then a new version of the whole design can be made of a single change in the design. For instance, design version 1 can be made of function 1 in version 1 and function 2 in version 3 and function 3 in version1. Then the next design release (version 2) can feature function 1 in version 2 and function 2 in version 3 and function 3 in version 1. This example shows that there are 2 level of version management since one might want to manage the whole design version whereas others might want to manage version of single items or even on both.

NOTE - The set of Element_version objects of an Element represent the history of an Element within a particular life-cycle stage or possibly over the whole life-cycle.

The data associated with an Element_version are the following:

- description
- version_indentifier
- version_of

4.2.44.1 description

The description specifies information unique to this version of the Element.

4.2.44.2 version_identifier

The version_identifier specifies the identifier of the Element_version. The version_identifier shall be unique within the scope of the associated Element (see 4.2.43).

4.2.44.3 version_of

The version_of specifies the Element (see 4.2.43) which the Element_version object represents a version of.

4.2.45 Element_version_relationship

An Element_version_relationship is the pairing of one Element_version (see4.2.44) with another Element_version.

See 4.3.49 for the application assertions.

The data associated with a Element_version_relationship are the following:

- change_description
- related_version
- relating_version

4.2.45.1 change_description

The change_description specifies a description of the difference between two Element_version objects.

NOTE - The contents of the change_description will vary depending on the nature of relationship that exists between versions.

EXAMPLE 13. The change_description for alternative versions would indicate the difference between the alternatives. The change_description for revision versions would indicate the change that was made which lead to the creation of a new version.

4.2.45.2 related_version

The related_version specifies the second of the two Element_version objects related by the Element_version_relationship.

NOTE - The semantic of this attribute varies depending on the value of the change description attribute.

4.2.45.3 relating_version

The relating_version specifies the first of the two Element_version objects related by the Element_version_relationship.

NOTE - The semantic of this attribute varies depending on the value of the change description attribute.

4.2.46 Enumeration_type_definition

An Enumeration_type_definition is a data type whose value range is defined by an enumeration of elements.

EXAMPLE: We may have an instantiated an Enumeration_type_definition Display_type with enumeration_elements = {comms_display, nav_display, messages_display}, initial_value = comms_display and default_value = nav_display.

An Enumeration_type_definition is a type of Data_type_definition (see4.2.35).

The data associated with an Enumeration_type_definition are the following:

- default_value
- enumeration_elements
- initial_value

4.2.46.1 default_value

The default_value specifies the value to be used if the value available is out of range.

NOTE: The value of the default_value shall be in the set of enumeration_elements.

The default_value is optional and need not be specified for a particular Enumeration_type_definition.

4.2.46.2 enumeration_elements

The enumeration_elements specifies the set of possible values for the Enumeration_type_definition.

NOTE: There must be at least two enumeration elements specified for a particular Enumeration_type_definition.

4.2.46.3 initial_value

The initial_value specifies the value to be assigned the object when it is created.

NOTE: The value of the initial_value shall be in the set of enumeration_elements.

The initial_value is optional and need not be specified for a particular Enumeration_type_definition.

4.2.47 Event_definition

An Event is a data type which does not carry any value or persistence. However an event instance carries meaning, in indicating when a significant things happens at a point in time. The occurrence of the event communicates both: that a significant thing has happened, and when ('just now').

4.2.48 Execution_time

An Execution_time is an estimation of the time is takes for a function to produce the outputs when either it has been activated or as soon as the whole input data set is available. The execution time shall not be confused with the time for a physical component to perform the function which would be related to the physical architecture definition in the physical UoF. The execution time given here records for instance the time that is necessary within a process industry to perform a specific action (for instance in a process industry it may be necessary to have a piece of work in a given acid bath for a given period to achieved specific properties)..

The exact semantics of this entity is defined by the role attribute.

NOTE 1: An Execution_time can be associated with all types of General_function_definition objects.

NOTE 2: The Execution_time specifies the time it takes for a function to be executed, not counting interference from other functions in the system.

The data associated with an Execution_time are the following:

– role
– time
– timing
– unit

See 4.3.29 and 4.3.30 for the Application Assertions.

4.2.48.1 role

The role specifies how the time attribute should be interpreted.

NOTE: Possible values for role are {worst_case, best_case, nominal_case}.

EXAMPLE: For instance, the function *calculate_time_to_destination* may have a *best_case execution_time* of 0 seconds, a *worst_case execution_time* of 0.020 seconds, and no value expressed for *nominal_case execution_time.*

4.2.48.2 time

The time specifies the real number that specifies the execution time.

4.2.48.3 timing

The timing specifies the General_function_definition (see 4.2.67) to which the Execution_time object applies to.

4.2.48.4 unit

The unit specifies the time unit for the time attribute.

4.2.49 External_document

An External_document is a document that is referenced in the model but not included in the system design. An External_document shall already be in existence (i.e. it is not a document that will be created in the future.)

An External_document is a type of Documentation_reference (see 4.2.41)

The data associated with an External_document are the following:

- location

4.2.49.1 location

The location specifies informally where the body of the document can be found.

4.2.50 External_entity

An external entity is a representation of an object that is part of the environment of the under design. The system to be designed usually does not exists yet (or at least has to be updated), but when operational it will be operate in the context of the existing entities. The External_entity objects represents objects with which the system will interact with and to which it shall provides services (or at least be compliant with). The set of External_entity objects for a system defines the system boundaries.

An External_entity may represents an arbitrary complex object, but designers are only interested in the macroscopic view of it. Therefore, they may be described by their inputs and outputs, by their behaviors (macroscopic on most of the cases), and by timelines expressing the inputs and outputs dynamic.

Since the description of an external entities shall remain at a coarse grain, the data model provides users with a basic description composed of a set of relationship that can reference this entity to describe the inputs and outputs (see description of Flow (4.2.53)), and some free text which structure is left open. The description of external entities has been made minimum since tools handling such entities have non uniform features.

The data model also does not provide a definition object of the external_entity item since it is assumed that an external entity shall not be decomposed (as opposed to the system description).

An External_entity may only be referenced in Function_context_definition objects (see 4.2.63) through the use of External_entity_context_relationship (see 4.2.51) objects.

An External_entity may be referenced by an arbitrary number of External_entity_context_relationship objects.

An External_entity is a type of Non_versionable_model_configuration_management_item (see 4.2.85).

The data associated with an External_entity are the following:

- description
- name

4.2.50.1 description

The description is a text that provides additional information on the external_entity. The description can hold information about the real nature of the entity or further information that may be necessary for full understanding of the role the entity plays in the application context.

The description is optional and need not be specified for a particular External_entity.

4.2.50.2 name

The name is the word or set of words to which the External_entity is referred to.

4.2.51 External_entity_context_relationship

An External_entity_context_relationship is a relation between an External_entity (see 4.2.50) and the Functional_context_definition (see 4.2.63) with which the External_entity is associated.

See 4.3.31 and 4.3.32 for the Application Assertions.

The data associated with an External_entity_context_relationship are the following:

- associated_context;
- description;
- external.

4.2.51.1 associated_context

The context specifies the Functional_context_definition in which the External_entity has some role.

4.2.51.2 description

The description specifies the role that the External_entity specified by attribute external plays in the Function_context_definition specified by the attribute associated_context.

NOTE: As External_entity objects may be part of many Function_context_definition objects the description attribute allows for a description of the role of the External_entity in a particular Function_context_definition.

The description is optional and need not be specified for a particular External_entity_context_relationship.

4.2.51.3 external

The external specifies the External_entity involved in the context.

4.2.52 External_to_requirement_association

An External_to_requirement_association is a relation from a Requirement_instance (see 4.2.109) object to an External_entity (see 4.2.50) object. The relation shall be interpreted as the requirement implies the interaction with the External_entity.

Since the template used to describe a requirement may differ from one company to another, the present data model does not describe any constraints on how many time this relationship is instanciated. For instance in a company where rules are that a requirement expression shall contain references to at least two external entities when dealing with functional requirements, this relationship will be instanciated twice.

The data associated External_to_requirement_association are the following:

- associated_requirement;
- implied_external

See 4.3.33 for the Application Assertions.

4.2.52.1 associated_requirement

The associated_requirement specifies the Requirement_instance the External_entity shall be associated with. It would typically mean that the text which is the definition of the requirement refer to the external entity object. There is no control whether the textual description is in line with the number of relationship instances.

Exactly one Requirement_definition is identified via the associated_requirement

4.2.52.2 implied_external

The implied_external specifies the External_entity that shall be associated with the Requirement_definition.

Exactly one External_entity is identified via the implied_external.

4.2.53 Flow

A Flow is a path between two Flow_port objects along which data flows.

The direction of data is not defined for the flow but can be derived from the Flow_port objects connected to the flow.

Within the information model flows are connected to both formal and actual flow ports. Formal_flow_port objects define the interfaces to General_function_definition objects. Actual ports define the interfaces to instances. Flows can flow between formal ports and actual ports; and actual ports and actual ports. A flow from one formal port to another would mean that is data on the Flow is not affected or referenced within the function. This would be suspicious though not necessarily disallowed.

A Flow shall always be consumed by a Flow_port object and produced by a Flow_port object.

The Data_instance (see 4.2.33) objects carried by the Flow can be derived by performing a set intersection operation of the data carried by the consuming port and the producing port.

NOTE: The set intersection of the data_instance objects on the source_port and destination_port shall not be empty.

If the Flow attributes destination_port and source_port both points to objects of type Actual_flow_port (see 4.2.7) then both ports objects may not have their buffered attribute set to TRUE.

See 4.3.34 for the Application Assertions.

The data associated with a Flow are the following:

- control_flow;
- destination_port;
- name;
- source_port.

4.2.53.1 control_flow

The control_port specifies whether the Flow is a control flow or not.

NOTE: A control flow is a Flow whose data component is solely used for activation or deactivation of a Function_instance object.

If control_flow is TRUE the Flow is a control flow.

4.2.53.2 destination_port

The destination_port specifies the Flow_port that serves as the destination for the Flow.

NOTE: The role of the destination_port shall be 'consumer'.

4.2.53.3 name

The name specifies the word or set of words by which the Flow is referred to.

4.2.53.4 source_port

The source_port specifies the Flow_port that serves as the source of the Flow.

NOTE: The role of the source_port shall be 'producer'.

4.2.54 Flow_composition_relationship

A Flow_composition_relationship is the relation between a Flow (see 4.2.53) or a Flow_group (see 4.2.55) and the Function_context_definition (see 4.2.63) or Composite_function_definition (see 4.2.26) that the Flow or Flow_group are part of.

See 4.3.35, 4.3.36, 4.3.37, 4.3.38 for the application assertions.

The data associated with Flow_composition_relationship are the following:

– Flow_component;
– Flow_context.

4.2.54.1 Flow_component

The flow_component specifies the Flow or Flow_group which is part of the relation.

4.2.54.2 Flow_context

The flow_context specifies the Function_context_definition or Composite_function_definition which is part of the relation.

4.2.55 Flow_group

A Flow_group is a container for grouping related Flow (see 4.2.53) objects.

The main reason for using a Flow_group is to assign the same graphical properties to the group Flow objects that are part of the Flow_group.

Note. Many design notations do not specifically include ports, ports can be synthesized from other connection types in the notations. The main driver for ports in the information model is to support a well defined interface for the objects the Flow_port is associated with, which as a consequence result in the possibility for many Function_instance objects to use a General_function_definition objects.

A Flow_group shall consist of at least two Flow objects.

The data associated with Flow_group are the following:

- elements;
- name.

4.2.55.1 elements

The elements is the set of Flow objects that make up the Flow_group.

NOTE: All Flow objects in the Flow_group shall be connected to consuming Flow_port objects who all reference the same object via the port_of attribute, and the producing Flow_port objects shall all be connected Flow_port objects that reference the same object via the port_of attribute.

4.2.55.2 Name

The name specifies the word or set of words by which the Flow_group is referred to.

4.2.56 Flow_port

A Flow _port is an abstract super type for formal and actual ports which define the interfaces to definitions and instances respectively. A Flow_port indicates that some data is either entering or leaving the object indicated by the port_of attribute.

A Flow_port is an abstract entity that is never instanciated.

Note. Many design notations do not specifically include ports, ports can be synthesized from other connection types in the notations. The main driver for ports in the information model is to support a well defined interface for the objects the Flow_port is associated with.

See 4.3.39 for the Application Assertions.

The data associated with a Flow_port are the following:

- data
- direction

4.2.56.1 data

The data specifies the Data_instance associated with the Flow_port. A port need only produce or consume part of the data carried by an associated flow. The data specifies which part of the data this is.

4.2.56.2 direction

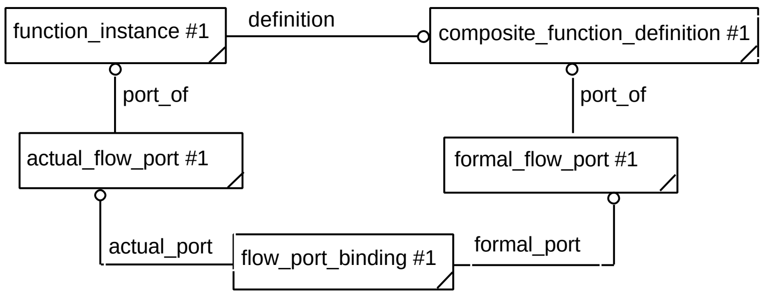
The direction indicates whether the Flow_port is an input or output port.

NOTE: Flow_port objects are not allowed to both input and output data. An input and output port would map to two flow ports, an input port and an output port.

4.2.57 Flow_port_binding

A Flow_port_binding indicates a relationship between an Actual_flow_port (see 4.2.7) and a Formal_flow_port (see 4.2.58). The Flow_port_binding ensures that the parameter binding involving a Function_instance and General_function_definition is correct. The following must be true for a Flow_port_binding to be correct:

The General_function_definition object the Formal_flow_port specified by the formal_port attribute is the same object as the General_function_definition object the Function_instance specified by the actual port has as its definition. The Required structure is illustrated in the diagram below.



The Data_type_definition object of the Data_instance given by the data attribute must be identical for the formal_port and the actual port.

The Data_instance object of the data attribute must not be identical for the formal_port and the actual port attributes.

Development NOTE: This last rule is necessary for reuse!

See for the Application Assertions.

The data associated with a Flow_port_binding are the following:

- actual port
- formal_port

4.2.57.1 actual port

The actual port specifies the Actual_flow_port on the Function_instance involved in the Flow_port_binding.

4.2.57.2 formal_port

The formal_port specifies the Formal_flow_port (see 4.2.58) on the General_function_definition involved in the Flow_port_binding.

4.2.58 Formal_flow_port

A Formal_flow_port is the definition of a data entry (interface definition) to a General_function_definition (see 4.2.67). If a General_function_definition has an interface the interface will be specified by one or more Formal_flow_port objects.

NOTE: Flow ports are classified according to two criterion in the data model: 1. Whether the port is formal (attached to a General_function_definition) or actual (attached to a External_entity, Function_instance, FSM_model or Store), and 2. Whether the port is an input or output port. From this classification it can be derived the role the port plays with regard to the Flow objects connected to the port. A port can 'consume' flows (the data on the flow is delivered to the port) or it can produce flows (the data on the flow is delivered from the port). For instance, a Formal_flow_port object whose direction attribute is 'input' produces data, while an Actual_flow_port whose direction attribute is 'input' consumes data. In the data model we capture this information in the derived role attribute. This attribute is used to ensure that Flow object s are connected correctly (a Flow object must have on producer and one consumer), and that Flow_port_binding objects applied only to ports where the actual_port in the binding is a producer of information and the formal_port in the binding is a consumer or vice versa.

NOTE: When a Function_instance of the General_function_definition is created an Actual_flow_port object corresponding to each Formal_flow_port is created if (and only if) there is a flow connected to the actual port (and the actual port is related to the corresponding Formal_flow_port object via a Flow_port_binding object.)

The data associated with a Formal_flow_port are the following

- port_of;
- role

See 4.3.40 for the Application Assertions.

4.2.58.1 port_of

The port_of specifies the General_function_definition for which this is a port.

4.2.58.2 role

The role attribute specifies whether the Formal_flow_port is consuming or producing data.

NOTE: The value of the role attribute can be derived: If the direction attribute (inherited from Flow_port) is ”input”, then the port consumes data else the port is producing data.

4.2.59 Formal_name_port

A Formal_name_port is an interface of a State_definition (see 4.2.114) such that it imports the knowledge of the existence of a Function_instance (see 4.2.64) known in the context of State_instance that has the State_definition as its definition to the elements of the State_definition.

A Formal_name_port is used in cooperation with Actual_name_port (see 4.2.8), Function_reference (see 4.2.65) and Name_binding (see 4.2.84) for performing the formal mapping from the object visibility space defined in the context of a State_instance (see 4.2.115) and the State_definition (see 4.2.114) that provides the definition of the State_instance.

The data associated with Formal_name_port are the following:

- port_of

4.2.59.1 port_of

The port_of specifies the State_definition that the Formal_name_port is associated with

A Formal_name_port is associated with exactly one State_definition object.

4.2.60 Formal_port_position

An Formal_port_position is the representation of the graphical position of a Formal port (Formal_flow_port (see 4.2.58) or Formal_transition_port (see 4.2.61)) object. The position is represented by a Graphics_point (see 4.2.72) object.

A Formal_port_position is a type of Visual_element.

The data associated with Formal_port position are the following:

- position;
- positioned_port.

4.2.60.1 position

The position specifies the position of the port.

4.2.60.2 positioned_port

The positioned_port specifies the formal port that is positioned by the Formal_port_position object.

4.2.61 Formal_transition_port

A Formal_transition_port is a definition of the entry or exit point of a transition from a Composite_state_definition (see 4.2.27). It indicates that instances of the corresponding State_definition will have corresponding Actual_transition_port objects where transitions begin or end (depending on the direction attribute of the port).

NOTE: Transition ports are classified according to two criterion in the data model: 1. Whether the port is formal (attached to a Composite_state_definition) or actual (attached to a State_instance), and 2. Whether the port is an input or output port. From this classification it can be derived the role the port plays with regard to the Transition objects connected to the port. A port can 'consume' transitions (the transition end in the port) or it can produce transition (the transition start in the port). For instance, a Formal_transition_port object whose direction attribute is 'input' produces Transition objects, while an Actual_transition_port whose direction attribute is 'input' consumes Transition objects. In the data model we capture this information in the derived role attribute. This attribute is used to ensure that Transition objects are connected correctly (a Transition object must have on producer and one consumer), and that Transition_port_binding objects is applied only to ports where the actual_port in the binding is a producer of transitions and the formal_port in the binding is a consumer or vice versa.

NOTE: Since a transition does not carry any data by itself the use of Formal_transition_port objects in Leaf_state_definition objects is limited. Thus Formal_transition_port objects are only linked with Composite_state_definition objects and only when the transition that is connected to the corresponding actual port is extended in the context of the State_definition the formal_transition_port is associated with via the port_of attribute.

The data associated with a Formal_transition_port are the following:

- port_of.
- role

4.2.61.1 port_of

The port_of specifies the Composite_state_definition for which this Formal_transition_port defines a entry or exit point.

4.2.61.2 role

The role specifies whether the Actual_transition_port is a consumer or a producer of a transition.

4.2.62 FSM_model

A FSM_model is an entity representing an entire finite state machine. It has a single, attribute which specifies the top of the (possibly hierarchical) state machine which it is encapsulating.

NOTE: It has been chosen to describe behaviors using hierarchic finite state machine as described in the statechart notation but avoiding too specific features such as condition connectors and history connector for instance (those features can usually be avoided to describe a behavior even if they provide short hand. Moreover, only a limited set of methodology and tool can handle such constructs).

With a hierarchic FSM, two kind of states of used: composed state which contains a state breakdown structure, and leaf_state which appear at the lowest level of detail for a given FSM. Note that a FSM may be updated by tool so that leaf states becomes composed states, but this is handled by the management of the design versions.

FSM_model objects may be referred to within the function breakdown structure and represents the activation and de-activation logic of functions in the functional breakdown structure (see also description of states for further indication on how the functional UoF and the behavior UoF are linked).

This entity is included in the model to allow a FSM to be included in a General_function_definition (see 4.2.67) and provide the link between the functional view and state view of a system.

See 4.3.42 for the Application Assertions.

The data associated with a FSM_model are the following:

– definition;
– name.

4.2.62.1 definition

The definition specifies the Composite_state_definition (see4.2.27) that represents the top level of the finite state machine.

NOTE 1: The definition of a FSM model is always a Composite_state_definition since we require there are at least some states defined for the FSM.

NOTE 2: Since the FSM_model indicates the a self-contained state machine the Composite_state_definition object that acts as definition of an FSM_model object must not have any formal_transition_port objects attached to it. In other words there shall not be any transitions leading to any external states in the Composite_state_definition that is the definition of a FSM_model.

4.2.62.2 name

The name is the word or set of words by which the FSM_model is referred to.

4.2.63 Function_context_definition

A Function_context_definition specifies the environment (from a functional view) that the function playing the role of the system under specification operates in.

NOTE: There can only be one Function_instance (the Function_instance representing the system function) in a Function_context_definition.

Function_context_definition objects shall be used to define the functional environment of Application (see 4.2.15)and Operational_scenario objects (see 4.2.89).

See (???) for the Application Assertions.

The data associated with Function_context_definition are the following:

– description;
– name.

4.2.63.1 description

The description specifies additional information on the Function_context_definition.

The description is optional and need not be specified for a particular Function_context_definition.

4.2.63.2 name

The name specifies the word or set of words by which the Function_context_definition is referred to.

The name is optional and need not be specified for a particular Function_context_definition.

4.2.64 Function_instance

A Function_instance is a representation of an entity that performs an activity within a system. The functional domain describes what a system does. The key to describing what a system does is the function. Within the systems engineering domain a function fulfills a requirement of the system; it could provide a service to users of the system, process materials or change the state of the world the system operates in.

Systems engineering also needs to deal with products which normally (simply due to their size and complexity) make use of many technologies in their solution. As such our concept of function must accommodate *function* in the loosest sense, with a wider interpretation than simply*processing information*. Examples of such functions would be:

- *Produce electric power* (the function of an electric generator);
- *Follow way-point to destination* (function of an aircraft, in a particular mode, rather than simply a controlling auto-pilot);
- *Defend this (particular) airspace* (function of a defence system).

The first of these functions above may be solely a mechanical function (no information processing elements), while the second and third may be represented by a refined view of a set of interacting sub-functions, some of which may be mechanical, some information processing, some hybrid (both). We may call these *types of functions*.

Within a system functions form a hierarchy, with the top level being the system itself and the bottom level being individual actions. The intermediate levels of the functional hierarchy form levels of abstraction in the functional description of the system.

This is the concept that a given function can be expressed in a more explicit way, as a set of interacting sub-functions.

In addition to the hierarchical relationship, we have the notion that functions*interact in various ways.*

NOTE: It is important to bear in mind the distinction between the concept of a hierarchical set of functions, and the process by which that set of functions has been derived. We may call the hierarchical set of functions a*functional decomposition*; this can be misleading, since it implies a top-down process of development, nevertheless the phrase is frequently used. There are at least three different approaches for creating a functional hierarchy:

Top-down:

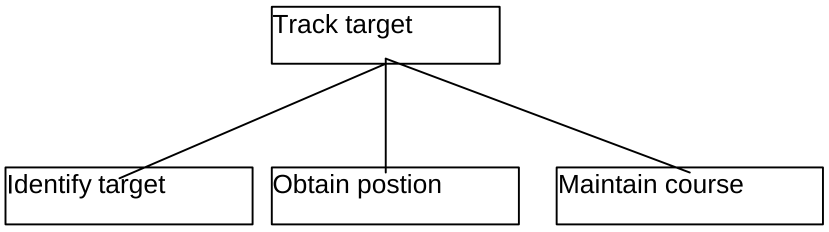
Bottom-up;

Outside-in (or stimulus-driven).

The functional hierarchy that these approaches would drive out is unlikely to be identical, although if they are each applied against the same set of requirements, the three alternative models ought to be functional equivalent.

Our description of a functional model focuses on the network of functions, rather than on how it was produced. As such the *meta-model* underlying the functional hierarchy derived from the three approaches will be the same (assuming events from 3 are outside the scope of the functional model.

Eg. A track target function might be an abstraction of the functions to identify the target, obtain current position and maintain a course to the target.



The realization of the above functions may involve diverse technologies such as radio communications, radar tracking, interaction with an external system such as a GPS satellite and computing equipment to calculate a course.

Should functions also deal with the realization of the system? There may not be a user requirement for a car to have a fuel pump or an engine cooling system but most of them do.

A Function_instance represents the instantiation of a General_function_definition. The split between instance and definition allows the reuse of definitions both within and between systems. A function definition may be explicitly defined in terms of a mini-specification, pre-condition / post-condition specification, finite state table, pseudo-code, etc. Alternately a function definition may consist of a further hierarchy of functions. A General_function_definition may in turn be (partly) described by the Function_instance objects it contains.

By default a Function_instance is active and continuous. However of Function_instance may be controlled by Control_port (see 4.2.31), Causal_relationships (see 4.2.23) and or FSM_model (see 4.2.62) objects

There may be several Function_instance objects associated with the same definition.

The data associated with a Function_instance are the following:

- definition;
- name.

4.2.64.1 definition

The definition specifies the General_function_definition (see4.2.67) that provides the definition for the Function_instance.

4.2.64.2 name

The name is the word or set of words by which the Function_instance is referred to.

4.2.65 Function_reference

A Function_reference is a link between a Function_instance (see 4.2.64) and a FSM_model (see 4.2.62) such that the knowledge of the existence of the Function_instance is imported into the FSM_model.

The Function_reference shall be used to create formal links between actions defined in a state machine and the Function_instance object the action affects.

The data associated with Function_reference are the following:’

- function_link;
- port_of.

4.2.65.1 function_link

The function_link specifies the Function_instance that is controlled from within the FSM_model. The Function_instance referred shall exist at the same level of decomposition or at a lower level as the FSM_model.

Development NOTE: This rule has not been formalized in the EXPRESS model for capability-2.

4.2.65.2 port_of

The port_of specifies the FSM_model that acts as a controller for the Function_instance specified by the function_link attribute.

4.2.66 Functional_requirement_definition

A Functional_requirement_definition is the definition of a functional requirement. A functional requirement describes the services the system has to provide external entities with. A functional requirement is a textual description which can obey certain rules depending upon company internal know-how. It expresses what the system has to do using an external point of view since the system to be designed does not yet exist (or at least shall be updated).

EXAMPLE: A functional requirement can express that the system shall give external entity E1 the ability to initiate controls on Entity E2.

The data model as presented in the standard allows for the use of templates that are internal to any company. Therefore, the data model does not expresses any constraints on the format that shall be used to describe such fucntionalities.

Functional_requirement_definition is a type of Requirement_definition (see 4.2.108).

4.2.67 General_function_definition

A General_function_definition provides a structured definition of a function (or a process) at some level of abstraction in a system.

See 4.2.64 for a discussion of functions.

The General_function_definition is abstract and shall not be instanciated.

The data associated with General_function_definition are the following:

- name
- description.

4.2.67.1 description

The description is a text describing the Composite_function_definition. The text shall be considered as an informal description of the function.

The description is optional and need not be specified for a particular General_function_definition.

4.2.67.2 name

The name is the word or set of words by which the General_function_definition is referred to.

The name is optional and need not be presents for a particular General_function_definition.

4.2.68 General_template

A General_template is a text whose constituent elements (paragraphs) are named.

Development NOTE: The General_template is provided for situations where the nature of the textual information need to be defined further.

EXAMPLE: In some cases there may exist the need to exchange information between tools and this information is not captured by the entities in the data model, in such cases General_template objects can be instanciated and each element of the template may hold information that informs the reader about the nature of encoded text.

The data associated with General_template are the following:

- elements

4.2.68.1 elements

The elements specifies the list of template elements that make up the General_template.

4.2.69 General_transition

A General_transition is a State transition which has only a destination State defined for the transition. The General_transition is a super-type of Default_transition (see 4.2.38) and Transition (see 4.2.121).

The General_transition is abstract and shall never be instanciated.

The data associated with General_transition are the following:

- destination_port

4.2.69.1 destination_port

The destination_port specifies the Transition_port (see 4.2.124) of the State_instance (see 4.2.115) or State_definition (see 4.2.114) that will be entered when the General_transition is taken.

There shall be exactly one destination_port for a General_transition

4.2.70 Graphics_link

A Graphics_link is a representation of a open graphical figure spanning a list of Graphics_point (see) objects for an occurrence of a General_transition (see) via the Transition_composition_relationship (see), Flow (see) or Flow_group (see) via the Flow_composition_relationship (see). The first element in the list represents the originating point of the first vertex in the link and the last element represents the terminating point of the last vertex.

NOTE: The graphical information is not directly associated with the objects that are represented by the Graphics_link object as the objects may have totally different representations if they appear in more than one context.

A Graphics_link is a type of Visual_element (see).

The data associated with Graphics_link are the following:

- associated_with;
- has_points.

4.2.70.1 associated_with

The associated_with specifies the Transition_composition_relationship, the Flow_composition_relationship object for which the Graphics_link provides the positioning representation.

4.2.70.2 has_points

The has_points is the list of Graphics_point objects that defines the connection points for the Graphics_link. There shall be at least two Graphics_point objects in the list.

4.2.71 Graphics_node

A Graphics_node is a representation of a closed rectangle used for presenting the position of a Function_instance (see) via the Hierarchical_composition_relationship (see) or System_function_association (see), or a Store (see) or FSM_model (see) via the Hierarchical_composition_relationship, an External_entity (see) via the External_entity_context_relationship (see), or a State_instance (see) via the State_composition_relationship (see).

A Graphics_node is a type of Visual_element (see).

The data associated with Graphics_node are the following:

- associated_with;
- bottom_right;
- top_left.

4.2.71.1 associated_with

The associated_with specifies the Hierarchical_composition_relationship, System_function_association or External_entity_context_relationship for which the graphical information is provided.

4.2.71.2 bottom_right

The bottom_left specifies the Graphics_point object (see) that provides positioning information for the lower left corner of the node.

4.2.71.3 top_left

The top_left specifies the Graphics_point object (see) that provides positioning information for the upper right corner of the node.

4.2.72 Graphics_point

A Graphics_point is a point in the coordinate plane. The coordinate plane is valid in the range {0<= x, y <= 1}. No information on the aspect ratio is given.

NOTE: This model is chosen since the dimensions of elements in a graphical view does not have any meaning in the domain. If the aspect ratio is set up in such a way the image is percieved to be distorted it up the viewer to adjust the aspect ratio.

The data associated with Graphics_point are the following:

- x_coordinate;
- y_coordinate.

4.2.72.1 x_coordinate

The x_coordinate specifies the horizontal position in the coordinate plane.

4.2.72.2 y_coordinate

The y_coordinate specifies the vertical position in the coordinate plane.

4.2.73 Graphics_view

A Graphics_view is the collection of all items carrying graphical information that applies to the contents of a Function_context_definition (see), General_function_definition (see) or Composite_state_definition (see).

The data associated with Graphics_view are the following:

- definition_for.

4.2.73.1 definition_for

The definition_for specifies the General_function_definition (see) or Composite_state_definition (see) that the Graphics_view provides the visual information for.

4.2.74 Hierarchical_decomposition_relationship

A Hierarchical_decomposition_relationship indicates that the Function_instance (see 4.2.71) , Store (see 4.2.116) or FSM_model (see 4.2.62) on the child attribute is part of a hierarchical decomposition of the General_function_definition (see 4.2.67) on the parent attribute.

NOTE: Each child that is part of the decomposition of a parent definition will be associated with the parent via a separate Hierarchical_decomposition_relationship object.

The hierarchies described by Hierarchical_decomposition_relationship objects in combination with Composite_function_definition and Function_instance objects shall not contain any cycles. This rule shall be applied recursively over all Hierarchical_decomposition_relationship, Composite_function_definition and Function_instance objects in any given function hierarchy.

Development NOTE: This important rule is not yet formalized into EXPRESS code.

See 4.3.44, 4.3.45, 4.3.46 and 4.3.47 for the Application Assertions.

The data associated with an Hierarchical_decomposition_relationship are the following:

- child;
- description;

– parent.

4.2.74.1 child

The child specifies the Function_instance (see 4.2.71), Store (see) or FSM_model (see) that is part of the decomposition of the parent.

4.2.74.2 description

The description specifies the role that the child plays in the Composite_function_definition specified by the parent attribute.

NOTE: As a child object may be part of many Composite_function_definition objects the description attribute allows for a description of the role of the child in a particular Composite_function_definition.

The description is optional and need not be specified for a particular Hierarchical_decomposition_relationship.

4.2.74.3 parent

The parent specifies the General_function_definition that is partially decomposed by the child.

4.2.75 Integer_bound

An Integer_bound is the upper or lower bound defined for Integer_type_definition (see) objects. The semantics of the Integer_bound is decided by the bound_type attribute.

NOTE: The bound value of Integer_bound indicates either the upper or lower bound. The bound value is included in the valid value range. For instance, an integer_bound object with bound_type = upper_bound and bound = 3 indicates that the upper bound of the integer type is 3 is 3, i.e. it can have values up to and including 3.

NOTE: Open ended intervals for an integer_type_definition is specified using a single integer_bound object.

NOTE: The valid value set specified using integer_bounds shall not be overlapping.

The data associated with Integer_bound are the following:

– bound;
– bound_type.

4.2.75.1 bound

The bound specifies the integer that is the upper or lower bound.

4.2.75.2 bound_type

The bound_type specifies whether the integer_bound is an upper or lower bound.

4.2.76 Integer_type_definition

An Integer_type_definition represents the definition of Integer type data type. An Integer_type_definition is a type of Number_definition.

NOTE: By the use of Integer_bound objects the valid range of an Integer_type_definition may be defined. If bounds are specified the default_value and the initial_value must be within the valid bounds.

The data associated with integer_type_definition are the following:

– accuracy;
– bounds;
– default_value;
– initial_value;
– resolution.

4.2.76.1 accuracy

The accuracy specifies the extent to which a value has high fidelity, i.e. is faithful to reality.

EXAMPLE: The measured airspeed is accurate to 2km/hour.

4.2.76.2 bounds

The bounds specifies a list of Integer_bound objects that defines the valid value range for the Integer_type_definition.

The bounds list may be of any size as long as the role or two consequtive Integer_bounds elements is not the same. In other words, if list element *N* is an upper_bound then element *N+1* (if it exists) shall be a lower_bound.

The values of the Integer_bound objects in the list shall be strictly increasing.

Development NOTE: An unbounded set is used here as it appears to be the natural natural to order the bounds object in a list. It will also make interface implementation easier as there is no need to analyse the Integer_bound on a 1 to 1 basis.

4.2.76.3 default_value

The default_value specifies the value to be assigned the object if the assigned value is illegal, for instance, out of the bounded range, or of the wrong type.

NOTE: The value of the default_value shall be within the valid value range if one is defined.

The default_value is optional and need not be specified for a particular integer_type_definition.

4.2.76.4 initial_value

The initial_value specifies the value to be assigned the object when it is created. , which would typically be either on power-up, or within a run-time system, for instance on creation of a *new_navigation_point* or *new_target*.

NOTE: The value of the initial_value shall be within the valid value range if one is defined.

The initial_value is optional and need not be specified for a particular integer_type_definition.

4.2.76.5 resolution

The resolution specifies the extent to which the storage mechanism for a variable or value is capable of fine granularity in distinguishing between distinct values.

EXAMPLE: The resolution of an integer value is normally 1. The resolution of altitude may be 1 in 65535.

4.2.77 Justification

A Justification is a textual description of the reasons for the existence of, comment, or design rationale for an Model_configuration_management_item (see 4.2.82), FSM_model (see 4.2.62), Function_instance (see 4.2.71), Requirement_instance (see), state_instance or physical_instance object.

Justification items records the « know-how » of the company performing the design. It can records information on the choice for a specific design compared to other alternatives. Therefore, a justification field is put on the alternative that will be further used, and justification will also be put on the alternatives that will not be used (criteria, evaluation and result of the evaluation against the criteria).

Depending upon the relationship among partners or teams working on a project, there might be some cases where the « know-how » shall not be broadcasted. Therefore the justification may or may not appear in the data exchanges.

A Justification is a type of Process_configuration_management_item.

The data associated with a Justification are the following:

– applies_to;
– justification_text
– role.

4.2.77.1 applies_to

The applies_to identifies the object the Justification is associated with.

A Justification is associated with exactly one object.

4.2.77.2 justification_text

The justification_text is the textual comment on the object referenced by the applies_to attribute.

4.2.77.3 role

The role is the word or set of words by which the nature of the justification is referred to.

EXAMPLE: The role for a justification may be comment, rationale, justifcation etc.

4.2.78 Justification_relationship

A Justification_relationship represents some relationship between two Justification (see4.2.77) objects. The exact semantics of the relationship depend on the value of the description attribute.

EXAMPLE : The description attribute for a Justification_relationship might be*Disagrees with* in the case where there are two conflicting Justifications assigned to a single object.

See 4.3.50 for the Application Assertions.

The data associated with a Justification_relationship are the following:

- description
- related
- relating

4.2.78.1 description

The description is a text that specifies the nature of the relationship.

4.2.78.2 related

The related specifies the second of the two Justification objects involved in the relationship. The exact semantics of the attribute depend on the value of the description attribute.

4.2.78.3 relating

The relating specifies the first of the two Justification objects involved in the relationship. The exact semantics of the attribute depend on the value of the description attribute.

4.2.79 Leaf_function_definition

A Leaf_function_definition is a General_function_definition (see4.2.67) that is not decomposed further. A Leaf_function_definition is a type of General_function_definition.

The data associated with a Leaf_funciton_definition are the following:

– function_type;

– predefined;
– specification;
– specification_language.

4.2.79.1 function_type

The function_type specifies the word or set of words by which the type of the function is referred to. The function_type shall be interpreted in the generic sense as they specify a class of functions

NOTE: Possible values for Function_Type could be “mathematical”, “transformational”, “Sine”, ”Lift lever” etc.

4.2.79.2 predefined

The predefined specifies whether the Leaf_function_definition object has a formally defined semantics in the exporting tool.

Development NOTE: This attribute is a simple method for allowing for tools having large libraries of predefined functions to exchange information without having to map too much information onto user defined functions. This shall be regarded as a temporary solution for capability 2 only.

4.2.79.3 specification

The specification specifies the text that describes what (the service, transformation or state change etc.) is performed by the function.

4.2.79.4 specification_language.

The specification_language specifies with a word or set of words the language the specification is written in.

NOTE: Possible values for specification_language could be “ANSI C”, “structured English” etc.

4.2.80 Leaf_state_definition

A Leaf_state_definition is the definition of a state that is not further decomposed.

NOTE: A state can be known as leaf state in one system version and be marked as composed state when the system has been further refined. This kind of situation is handled by the use of version items. Nevertheless, a state recorded as leaf state in one scenario can not be known a composed state in the an other scenario which belongs to the same application version.

A Leaf_state_definition is a type of State_definition (see**Erreur! Source du renvoi introuvable.**).

Development NOTE: The Leaf_state_definition appears to be redundant since all State_definition objects can hold actions except for the fact that Leaf_state_definition objects shall not have any Formal_transition_port objects associated. Leaf_states are not analogous to leaf functions. For the final proposal the Leaf_state_definition may well be merged into the State_definition entity.

4.2.81 Maturity

A Maturity is a piece of information about the maturity of a model object. The definition of the level of maturity assigned to a piece of design data is left to the project organization. The data model as it stands does not make any assumption on the consistency that could be required on the maturity allocated to item in terms of hierarchic description.

For instance, if a function has a breakdown structure we can have situations where a parent function is considered to be mature while the child functions are considered to be relatively immature. A Maturity object assigned to a Non_versionable_model_configuration_management_item (see) represent the maturity of the object in the context it is used in, while a Maturity object assigned to a Versionable_model_configuration_management_item (see) represent the maturity of the definition of the item regardless of what context it may appear in.

In the same way, the data model does not give any constraints on how the maturity field relates to approvals and validation items.

A Maturity is a type of Process_configuration_management_item (see 4.2.101).

Development NOTE: The justification was made a subtype of Process_configuration_management_item to allow convenient attachment of author and date.

See 4.3.52 for the Application Assertions.

The data associated with a Maturity are the following:

- element_maturity
- maturity_description

4.2.81.1 element_maturity

The element_maturity specifies the Model_configuration_management_itemobject that this maturity is being applied to.

4.2.81.2 maturity_description

The maturity_description specifies the word or set of used to describe the maturity of the object.

4.2.82 Model_configuration_management_item

A Model_configuration_management_item is a place holder for management and organization information associated with an modeling entity. This entity is the super type for all modeling definition and instance types.

NOTE - This element can be thought of as a collection point for all the configuration management information associated with a model element.

Model_configuration_management_item is an abstract entity and shall never be instanciated.

See 4.3.53 for the Application Assertions.

The data associated with Model_configuration_management_item are the following:

– id.

4.2.82.1 id

The id specifies a unique identifier for the Model_configuration_management_item object. The id shall be unique across all Model_configuration_management_item objects.

The id is not intended to carry any meaning for a human reader.

4.2.83 Multi_level_view

A Multi_level_view is the top_most element in a hierarchy of Graphics_view (see) objects.

NOTE: The Multi_level_view shall not be instanciated if no hierarchy of Graphics_view objects exist for a particular Graphics_view object.

The data associated with a Multi_level_view are the following:

– description;
– name;
– top_view.

4.2.83.1 description

The description specifies a textual description of the view.

4.2.83.2 name

The name specifies the word or set of words by which the Multi_level_view is referred to.

4.2.83.3 top_view

The top_view specifies the Graphics_view object that the Multi_level_view is the top_view for.

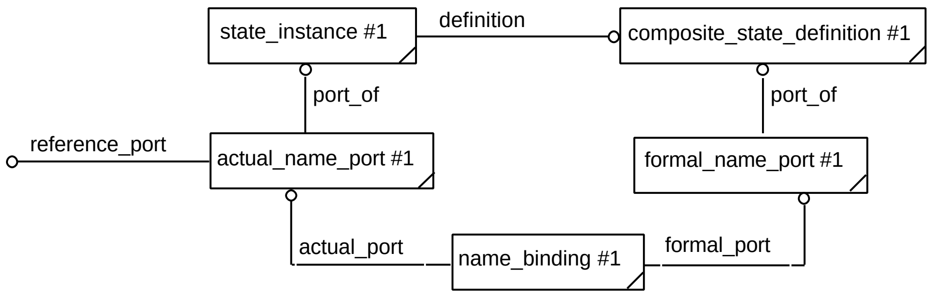
4.2.84 Name_binding

A Name_binding is the relation between a Formal_name_port (see) and and an Actual_name_port (see) or a Function_reference (see). The name_port makes a Function_instance which is known in the context of the Actual_name_port known also to the context of the Formal_name_port. The Name_port's purpose is to create a formal link between actions associated with a State or Transition in a State_definition and the Function_instance object that is affected by the action.

If such a formal link is desired Name_binding objects (in conjunction with Formal_name_port and Actual_name_port objects) for each hierarchical level of the state machine the actions are performed in.

NOTE: Name_port and Name_binding are required as any given State_definition may be used in several contexts. Thus the actions generated within the State_definition can not reference the affected Function_instance objects directly, as the affected Function_instance objects are bound to be different instances for each case where the State_definition is used.

The Actual_name_port specified by actual_port attribute shall be a port_of the State_instance that has the State_definition that the Formal_name_port is a port_of specified by formal_port attribute as its definition. The figure below illustrates the rule:



NOTE 1: The Name_binding is not valid unless the same Composite_state_definition object is identified by both the "actual_port.port_of.edfinition" and the "formal_port.port_of" sequences.

NOTE 2: The reference_port attribute of the object actual_name_port #1 references the Formal_name_port of the parent State_definition of the object State_instance #1. This State_definition is not shown in the example, but each Actual_name_port object must refernce Formal_name_port via the reference_port attribute, or no knowledge of Function_instances are imported into the state.

The data associated with Name_binding are the following:

– actual_port;
– formal_port.

4.2.84.1 actual_port

The actual_port specifies the Actual_name_port that takes part in the binding

4.2.84.2 formal_port

The formal_port specifies the Formal_name_port that takes part in the binding

4.2.85 Non_Versionable_model_configuration_management_item

A Non_Versionable_model_configuration_management_item is the super type of all design instance elements that may appear in a design. A Non_Versionable_model_configuration_management_item is a type of Model_configuration_management_item.

Development NOTE: We are not convinced the CMI, MCMI, NVMCMI hierarchy of entities is the most appropriate.

4.2.86 Null_type_definition

A Null_type_definition is the definition of an unspecified or undefined data type.

NOTE: The Null_type_definition shall only be used in cases where the data type of a Data_instance is unspecified or unknown.

4.2.87 Number_type_definition

A Number_type_definition represents the definition of numeric data type.

A Number _type_definition is a type of Data_type_definition (see4.2.35).

NOTE: The Number_Type_Definition shall only be used in cases where it can not be defined if a particular numeric data type is a real or an integer.

The data associated with Number_type_definition are the following:

– accuracy;
– default_value;
– initial_value;
– resolution.

4.2.87.1 accuracy

The accuracy specifies the extent to which a value has high fidelity, i.e. is faithful to reality.

EXAMPLE: The measured airspeed is accurate to 2 km/hour.

4.2.87.2 default_value

The default_value specifies the value to be assigned the object when an invalid value is provided, for instance, when the value is out of the bounded range, or of the wrong type.

NOTE: The absense of valid values typically indicates the element supplying the value is out of order.

NOTE: The value of the default_value shall be the valid value range if one is defined.

The default_value is optional and need not be specified for a particular Number_type_definition.

4.2.87.3 initial_value

The initial_value specifies the value to be assigned the object when it is created.

NOTE: The value of the initial_value shall be the valid value range if one is defined.

The initial_value is optional and need not be specified for a particular Number_type_definition.

4.2.87.4 resolution

The resolution specifies the extent to which the storage mechanism for a variable or value is capable of fine granularity in distinguishing between distinct values.

EXAMPLE: The resolution of a Number_type_definition value could be 0.5.

Development NOTE: If the resolution of a number_type_definition is known then the specific type may also be known.

4.2.88 Operational_requirement_definition

An Operational_requirement_definition is the definition of a requirement that is expressed during the use of the system. Such requirements are used to capture dynamic constraints that will apply when the system is under operation.

For instance an operational requirement can record facts such as the system has to be handled by a human being, therefore, it shall have properties such as weight, shape... In the same way, it can records constraints that apply when the system has to be set-up for operation (for instance for a craft, there may be some operation that shall be performed before flight).

Operational_requirement_definition is a type of Requirement_definition (see4.2.108).

4.2.89 Operational_scenario

An Operational_scenario is a description of a specific phase of a system's life-cycle. The Operational_scenario collects requirements and functional information that applies to a particular life cycle phase of the system.

The system under design may be used in several contexts: when in maintenance mode, the system may have to provide users with different funtionalities to even different external entities.

EXAMPLE: Operational_scenarios of a system (represented by an Application object) could be stand-by or operational.

The user of the data model shall be aware that the system function which is described in the various operational_scenario represent the same entity but can have different decomposition according to the operational_scenario considered. Therefore, there may be several different Function_definition objects that represents the system function if the system is described as a set of Operational_scenario objects.

An Operational_scenario is a type of System_view (see4.2.119)

See 4.3.54 for the application assertion.

The data associated with Operational_scenario are the following:

– scenario_of.

4.2.89.1 scenario_of

The scenario_of specifies the Application (see) that the Operational_scenario describes a life cycle phase for.

4.2.90 OR_state_definition

An OR_state_definition is the definition of a composite state definition where only one child state at a time is active.

EXAMPLE : A single state *operate* may be hierarchically decomposed (using objects of type State_composition_relationship) into two child states *calculate* and *display*, via two hierarchical_decomposition objects. When the fsm is in the high level state *operate* it will be in ONE OF *display* OR *calculate*.

An OR_state_definition is a type of State_definition (see**Erreur! Source du renvoi introuvable.**).

4.2.91 Organization

An Organization is a group of people involved in a particular business process.

NOTE - This entity is taken directly from the PDM Schema.

The data associated with an Organization are the following:

- description;
- id;
- name.

4.2.91.1 description

The description specifies a textual string which provides some additional information about the Organization.

4.2.91.2 id

The id specified an identifier by which the organization is identified.

The id is optional and need not be specified for a particular Organization.

4.2.91.3 name

The name specifies the word or set of words used to identify a particular Organization.

4.2.92 Organization_address

An Organization_address is the location information that can be associated with one or more Organization (see 4.2.91) objects.

NOTE - This entity is taken directly from the PDM Schema.

An Organization_address is a type of Address (see 4.2.11).

See 4.3.54 for the Application Assertions.

The data associated with an Organization_address are the following:

- organizations

4.2.92.1 organizations

The organizations specifies which organizations are at that at the specified address.

4.2.93 Organization_relationship

The Organization_relationship is used to represent an arbitrary relationship between two Organization (see 4.2.91) objects.

NOTE - This entity is taken directly from the PDM Schema.

EXAMPLE 14. The relationship between two organizations may be *contractor* or *partner*.

See 4.3.56 for the Application Assertions.

The data associated with an Organization_relationship are the following:

- description
- name
- related_organization
- relating_organization

4.2.93.1 description

The description specifies a textual string the describes the nature of the relationship between the two Organization objects.

4.2.93.2 name

The name specifies a name that is associated with the relationship between the two Organization objects.

4.2.93.3 related_organization

The related_organization specifies the second of the two Organization involved in the relationship.

NOTE - The exact semantics of this attribute depend of the value of the description attribute.

4.2.93.4 relating_organization

The relating_organization specifies the first of the two Organization involved in the relationship.

NOTE - The exact semantics of this attribute depend of the value of the description attribute.

4.2.94 Package

A Package is a collection of related objects that do not interact directly with each other. Packages can be used to group information on any level of design abstraction, from a small set of Requirement_instance objects that somehow are related to the set of design information that is the outcome of a design activity.

EXAMPLE: The requirement elicitation phase will produce a set of requirement items that will feed the functional design activity. This example also shows that the set (possibly a subset) of information that comes out of an activity will act as input to another activity.

The status of a package can be expressed by assignment of Approval (see) and/or Maturity (see) objects to a Package object.

EXAMPLE: A package can be created to group the elements that serves as input to, or output from an Activity (see 4.2.8). A package can also be used for grouping related Requirement_instance objects

The data associated with a Package are the following:

- description
- name

4.2.94.1 description

The description specifies additional information about this package.

EXAMPLE 15. The description for a package might be *The set of models sent to subcontractor*.

4.2.94.2 name

The name specifies the word or set of words by which the Package is referred to.

4.2.95 Package_element_assignment

A Package_element_assignment is a specification of the relation between a Model_configuration_management_item(see) object and a Package (see 4.2.94) such that the object identified by the element attribute is a part of the package object specified by the package attribute.

If an object is assigned to a package then all objects directly related to the object is also considered to be part of the package.

EXAMPLE: If a Function_instance object is assigned to a package then the General_function_definition that provides the definition for the Function_instance is also considered to be in the package.

Development NOTE: The above assignment rule is very weak and shall be improved for the final proposal.

No specific rules on the allowed contents of a Package has been formulated other than a specific element in a package may not directly or indirectly be added more than once to the package.

Development NOTE: This rule is not encoded in the EXPRESS code for the capability 2 model.

See 4.3.57 and 4.3.58 for the Application Assertions.

The data associated with a Package_element_assignment are the following:

– description
– element
– name
– package

4.2.95.1 description

The description specifies a description of the relationship between the package element and the package. The description provides any information about the relationship that the name attribute can not represent.

4.2.95.2 element

The element specifies Model_configuration_management_item (see) that belongs to the Package.

See for the application assertion.

4.2.95.3 name

The name specifies a name given to the relationship between the package_element and the package.

EXAMPLE 16. A package may have a document assigned to it with role name *Overall package description*.

4.2.95.4 package

The package specifies the Package the Model_configuration_management_item belongs to.

4.2.96 Person

A Person is an individual human being who has some relationship to the product data. The person shall always be identified in the context of one or more organizations.

NOTE - This entity is taken directly from the PDM Schema.

The data associated with a Person are the following:

- first_name
- last_name
- middle_names
- prefix_titles
- suffix_titles

4.2.96.1 first_name

The first_name specifies the first name of the Person.

The first_name need not be specified for a particular Person.

4.2.96.2 last_name

The last_name specifies the surname of the Person.

The last_name need not be specified for a particular Person.

4.2.96.3 middle_names

The middle_names specifies any middle names of the Person.

The middle_names need not be specified for a particular Person.

4.2.96.4 prefix_titles

The prefix_titles specifies any prefix titles of the Person.

The prefix_titles need not be specified for a particular Person.

4.2.96.5 suffix_titles

The suffix_titles specifies the suffix titles of the Person.

The suffix_titles need not be specified for a particular Person.

4.2.97 Person_and_organization

A Person_and_organization is an entity that combines a Person (see 4.2.96) and an Organization (see 4.2.91).

NOTE - This entity is taken directly from the PDM Schema.

EXAMPLE 17. The Person *Bill Gates* and the Organization *Netscape* are unlikely to be joined via a Person_and_orgnization entity.

See 4.3.59 and 4.3.60 for the Application Assertions.

The data associated with a Person_and_organization are the following:

- description
- name
- the_organization
- the_person

4.2.97.1 description

The description specifies information about the relationship between the person and the organization that is not captured by the name attribute.

4.2.97.2 name

The name specifies a title that is given to the relationship between the person and the organization.

EXAMPLE 18. The name for the relationship between a person and an organization might be *Managing Director* or *Contact Person*.

4.2.97.3 the_organization

The the_organization specifies the Organization that makes up part of this coupling.

4.2.97.4 the_person

The the_person specifies the Person that makes up part of this coupling.

4.2.98 Person_or_organization_assignment

A Person_and_organization_assignment is an entity used to associate a Person_and_organization with a Configuration_management_item (see).

NOTE - This entity is taken directly from the PDM Schema.

Development NOTE: It might be useful to add Element as a possible select type to allow a person_or_organization_assignment to make assignment that span over all versions of an object instead of definitions only.

EXAMPLE 19. An object of type Person_and_organization_assignment might be used to associate the Person_and_organization *Bill Gates and Microsoft* with the Activity *World Domination.*

The data associated with a Person_and_organization_assignment are the following:

- assigned_person_and_organization
- item
- role

4.2.98.1 assigned_person_or_organization

The assigned_person_and_organization specifies the Person and/or organization object that is being assigned to the item.

4.2.98.2 item

The item specifies the Configuration_management_information (see) object that is being assigned the the person and organization.

4.2.98.3 role

The role specifies a Person_and_organization_role object that describes the nature of the relationship between the Person_and_organization and the item. Where appropriate the following values shall be used for the role attribute:

'Author': The Assigned_person_or_organization has been responsible for creating and formulation of the item.

4.2.99 Personal_address

A Personal_address is the location information that can be associated with one or more Person (see) objects.

NOTE - This entity is taken directly from the PDM Schema.

An Personal_address is a type of Address.

The data associated with an Personal_address are the following:

- people

4.2.99.1 people

The people specifies which Persons are associated with the specified address.

4.2.100 Physical_requirement_definition

A Physical_requirement_definition is a representation of constraints that are put on the system shapes, volume and processing performance. This type of constraints are encountered when the system has to fit into a limited cabinet or when it has to perform an operation is a given time.

In these cases, this kind of requirements will may imply the choice of a given technology when designer will have to map functional design onto real components.

Physical_requirement_definition is a type of Requirement_definition.

4.2.101 Process_configuration_management_item

A Process_configuration_management_information is the management and organizational information associated with process that is represented in the data model.

NOTE 20. - The processes related to Process_configuration_management_information objects are processes from a configuration management point of view. The Activity entity is a sub-type of Process_configuration_management_information.

Development NOTE: The Process_configuration_management_information may dissapear if there are few attributes that are common to configuration management entities.

4.2.102 Project

A Project is a unique process with a time limit, with a defined goal, with a defined budget, and with defined resources. A project is defined as a set of activities that are performed to achieve a given objective (this set of activity is known as the work_program in this standard). Therefore, each activity is defined a goal and is given a set of inputs. It then has to produce the necessary output.

The data associated with a Project are the following:

- work_program

4.2.102.1 work_program

The work_program specifies the Activity (see 4.2.8) objects that are carried out within the project.

EXAMPLE 21. A project for designing a new system may involve Activities such as ”requirements analysis” and ”verifying design for sub system 1”.

See 4.3.65 for the application assertions.

4.2.103 Real_bound

A Real_bound specifies the upper or lower bound defined for a Real_type_definition. The semantics of the Real_bound is decided by the bound_type attribute.

NOTE: The bound value of Real_bound indicates either the upper or lower bound. The bound value is included in the valid value range. For instance, a Real_bound object with bound_type = “upper_bound” and bound = 3.14 indicates that the upper bound of the real type is 3.14.

NOTE: Open ended intervals for an Real_type_definition is specified using a single Real_bound object.

NOTE: The valid value set specified using Real_bounds shall not be overlapping.

The data associated with Real_bound are the following:

– bound;
– bound_type.

4.2.103.1 bound

The bound specifies the real value that is the upper or lower bound.

4.2.103.2 bound_type

The bound_type specifies whether the Real_bound is an upper or lower bound.

4.2.104 Real_type_definition

A Real_type_definition represents the definition of real type data type. A Real_type_definition is a type of Number_definition.

NOTE: By the use of real_bound objects the valid range of a Real_type_definition may be defined. If bounds are specified the default_value and the initial_value must be within the valid bounds.

The data associated with Real_type_definition are the following:

- accuracy;
- bounds;
- default_value;
- initial_value;
- resolution.

See for the Application Assertions.

4.2.104.1 accuracy

The accuracy specifies the extent to which a value has high fidelity, i.e. is faithful to reality.

EXAMPLE: The measured airspeed is accurate to 2.5 km/hour.

4.2.104.2 bounds

The bounds specifies a list of Real_bound objects that defines the valid value range for the Real_type_definition.

The bounds list may be of any size as long as the role or two consequtive Real_bounds elements is not the same. In other words, if list element *N* is an upper_bound then element *N+1* (if it exists) shall be a lower_bound.

The values of the Real_bound objects in the list shall be strictly increasing.

Development NOTE: An unbounded set is used here as it appears to be the natural to order the bounds object in a list. It will also make interface implementation easier as there is no need to analyze and order the Real_bound objects.

4.2.104.3 default_value

The default_value specifies the value to be assigned the object if the assigned value is illegal. , for instance, out of the bounded range, or of the wrong type.

NOTE: The value of the default_value shall be within the valid value range if one is defined.

The default_value is optional and need not be specified for a particular Real_type_definition.

4.2.104.4 initial_value

The initial_value specifies the value to be assigned the object when it is created, , which would typically be either on power-up, or within a run-time system, for instance on creation of a *transmit_frequency* or *power_setting*.

NOTE: The value of the initial_value shall be within the valid value range if one is defined.

The initial_value is optional and need not be specified for a particular Real_type_definition.

4.2.104.5 resolution

The resolution specifies the extent to which the storage mechanism for a variable or value is capable of fine granularity in distinguishing between distinct values.

EXAMPLE: The resolution of a real number, for instance a measurement of air speed could be 0.5 km/hour.

4.2.105 Requirement_allocation_relationship

A Requirement_allocation_relationship is a relation between a Requirement_instance object and a Function_instance (see 4.2.71), Causal_chain (see), FSM_model (see), State_instance (see) or Store (see). The semantics of the relation is defined by the role attribute.

This item ensures the traceability of requirements in later design stages. This item shall not be confused with the allocation table that exists between the functional and the physical level.

The data associated with a Requirement_allocation_relationship are the following:

- description;
- relation_to;
- requirement;
- role.

4.2.105.1 description

The description specifies a textual comment on the relation. This text captures how the requirements are fulfilled in the designs and participate in the justification of the overall design and in the traceability process.

The description need not be specified for a particular Requirement_allocation_relationship.

4.2.105.2 relation_to

The relation_to specifies the object that the Requirement_instance object, given by the requirement attribute, has a relation to.

4.2.105.3 requirement

The requirement specifies the Requirement_instance that is part of the relation.

4.2.105.4 role

The role specifies the semantics of the relation. Where applicable the following values shall be used:

– 'allocation': The application_object specifies an allocation relationship where the requirement object is allocated to the relation_to object. The allocation relation means that the relation_to objcet shall be designed in such a way that the requirement is fulfilled.
– 'fulfils': The application_object specifies an fulfils relationship where the requirement stated in the requirement object is fulfilled by relation_to object. The fulfils relation means that the relation_to object has been evaluated and found to be compliant with requirement object.

4.2.106 Requirement_alternative_relationship

A Requirement_alternative_relationship is the relation between two Requirement_instances whichare considered to be alternatives of each other. The relationship is commutative and transitive. If Requirement_instance A is an alternative to Requirement_instance B then B is also an alternative to A. If Requirement_instance A is an alternative to Requirement_instance B and Requirement_instance B is an alternative to Requirement_instance C then Requirement_instance A is an alternative to Requirement_instance C.

An alternative represents several technical solution that can potentially be used within the design of the system. Alternatives can be dropped later on during the design after trade-off analysis for instance. Nevertheless, alternatives items remains in the design data base. Moreover, the justification attribute inherited from configuration management UoF records the choice of one or the other alternative. This attribute ensure the capture of the « know-how » of the organizations using the standard.

NOTE: The relating and related attributes shall relate to alternate requirements at the same level.

The data associated with a Requirement_alternative_relationship are the following:

- description;
- related;
- relating.

4.2.106.1 description

The description is a text that defines why the related Requirement_instance is considered to be an alternative to the relating Requirement_instance. This kind of attribute is filled-in after trade-off analysis performed with the customer. Alternatives can also appear at further design stages such as functional and physical design stages (this relation is captured by the use of the Alternative (see) entity).

This value is optional and need not be specified.

4.2.106.2 related

The related specifies the second Requirement_instance in the relation.

4.2.106.3 relating

The related specifies the first Requirement_instance

4.2.107 Requirement_composition_relationship

A Requirement_composition_relationship is the relation between a Requirement_definition (see 4.2.108) and a Requirement_instance (see 4.2.109) representing a child requirement of a Requirement_definition. The Requirement_composition_relationship is a symmetrical relation.

This relationship shall be used to describe a hierarchy of requirements whereas relationships on the same level are described using Derived_requirement_relationship (see) objects. The hierarchy of the model is achieved by the combined use of the definition attribute on the requirement_instance item and the parent and child attributes on this relationship item.

Note 1: The Requirement_composition_relationship should be used in two situations:

1. When the description of a requirement_definition is such that it is more appropriate to split the definition into two or more requirements derived from the first, or

2. There are two or more requirement_instances that apply to the same system whose description value of the defining requirement_definitions are so similar so they can be better defined by a single requirement.

Note 2: The Requirement_composition_relationship shall only be used when combining basic requirements or splitting complex requirements. If a requirement is derived from other requirement objects of type Derived_requirement_relationship (see) that serves as a more detailed definition of the requirement.

4.2.107.1 child_instance

The child_instance specifies the Requirement_instance that serves as the specialized instance in the relationship.

There shall be exactly one Requirement_instance defined in the role as child_instance.

4.2.107.2 description

The description specifies a textual description of the nature of the composition relationship.

4.2.107.3 parent_requirement

The parent_requirement specifies the Requirement_definition (see) that serves as the less detailed definition of a requirement.

There shall be exactly one Requirement_definition defined in the role as parent_requirement.

4.2.108 Requirement_definition

A Requirement_definition is the definition of a requirement. It represents the generic description of a requirement that is identified during the requirement elicitation and capture phase. This definition is provided to allow re-use of requirement instances when several systems are build using common roots.

The Requirement_definition is abstract and shall not be instanciated. Instead a Requirement_definition shall be of one of the following: a Constraint_definition (see 4.2.29), a Functional_requirement_definition (see 4.2.66), an Operation_requirement_definition (see 4.2.88), or a Physical_requirement_definition (see 4.2.100).

Other requirements classes can exist in industry but the data model only provides a limited set of categories to avoid adding too much complexity. Moreover, the classification of item against one or the other category may vary upon the definition used in companies. Therefore, it has been chosen to focus only on items that have a usually recognized definition.

Development NOTE: The above statement suggests that the classification of requirements shall be dropped since the number of requirement types appears to have no upper bound.

A Requirement_definition is a type of Versionable_model_configuration_management_item.

The data associated with a requirement_definition are the following:

– description;
– name.

4.2.108.1 description

The description specifies the definition of the requirement in textual form. The format of the description is given by the text_select type which supports the use of templates that might be used in some organizations. Therefore no specific constraint is expressed in this standard on the format that can be used. It belongs to the users of the standard to define a specific structure if required.

There shall be exactly one object that defines the description for a Requirement_definition.

4.2.108.2 name

The name specifies the word or set of words by which the Requirement_definition is referred to.

The name is optional and need not be specified for a particular Requirement_definition.

4.2.109 Requirement_instance

A Requirement_instance is the occurrence of a requirement that is defined in a Requirement_definition (see).

A requirement_definition may have zero or more instances in the schema. Requirement_instance shall refer exactly one Requirement_definition.

The data associated with a requirement_instance are the following:

– definition;
– name.

4.2.109.1 definition

The definition specifies the Requirement_definition that serve as the definition of the particular requirement_instance.

See for the application assertion.

There shall be exactly one object that defines the definition for a Requirement_instance.

4.2.109.2 name

The name is the word or set of words by which the Requirement_instance is referred to.

4.2.110 Requirement_system_view_relationship

A Requirement_system_view_relationship is the association of a Requirement_instance to an Operational_scenario. A Requirement_instance may be associated with any number of Operational_scenarios (by the use of several Requirement_system_view_relationship objects), but a Requirement_instance object may only be associated once to a specific System_view object.

The data associated with Requirement_system_view_relationship are the following:

– description;
– requirement;
– system_view.

4.2.110.1 description

The description specifies a text describing why the Requirement_instance identified via the attribute requirement is associated with the System_view identified via system_view.

The Description is optional and need not be specified. If present there shall be exactly one object that defines the description for the Requirement_system_view_relationship.

4.2.110.2 requirement

The requirement specifies the Requirement_instance that is associated to the Operational_Scenario specified by system_view.

The requirement shall reference exactly one Requirement_instance.

4.2.110.3 system_view

The system_view specifies the Operational_scenario that the Requirement_instance specified by requirement is associated to.

The system_view shall reference exactly one System_view.

4.2.111 Revision

A Revision represents a relationship between two Element_versions (see 4.2.87) where the related Element_version (see 4.2.87) is created by modifying the relating Element_version.

A Revision is a type of Element_version_relationship (see 4.2.86).

EXAMPLE 22. A version of a functional model may be updated to fulfill revised requirements. The updated version of the model would be a revision of the original version.

NOTE - This entity has no extra attributes associated with it. It is included solely to improve the semantics of the model.

4.2.112 Scenario_relationship

A Scenario_relationship is a specification of the relation between two Operational_scenario objects.

NOTE: Since a system can be described in several point in its life-cycle using operational_scenario items, it shall also be possible to record how and under which conditions the system goes from one operational_scenario to another.

The relationship is asymmetrical: If there exist a Scenario_relationship between two Operational_scenario objects - relationship from A and B - then the relation between B and A does not exist.

The Scenario_relationship relation is limited in the sense that it is only defined for Operational_scenario objects on the same decomposition level. The hierarchical relation is not supported.

The data associated with a Scenario_relationship are the following:

- description;
- related;
- relating;
- transition_condition.

4.2.112.1 description

The description is a text that specifies the relationship between two Operational_scenarios

The description is optional and need not be specified.

4.2.112.2 related

The related specifies the second of the Operational_Scenario objects related by the Scenario_relationship.

4.2.112.3 relating

The relating specifies the first of the Operational_Scenario objects related by the Scenario_relationship.

4.2.112.4 transition_condition

The transition_condition is a text that specifies under which conditions the system may change mode, i.e. when the transition from the relating Operational_scenario to the related may take place.

EXAMPLE: For an aircraft system with operational scenarios ”airborne” and ”on ground” the transistion_condition between the model could be ”Weight on wheels”.

4.2.113 State_composition_relationship

A State_composition_relationship is a parent-child relation for a hierarchical breakdown of states in a state machine.

The exact semantics of the relationship depend on whether the parent state is an OR_state_definition or a AND_state_definition. If the parent_state is of type OR_state_definition then only one of the child_states may be active when the parent_state is active. If the parent_state is of type AND_state_definition then all child_states are concurrently active when the parent_state is active.

The hierarchies described by State_composition_relationship objects in combination with Composite_state_definition and State_instance objects shall not contain any cycles. This rule shall be applied recursively over all State_composition_relationship Composite_state_definition and State_instance objects in a hierarchy of states.

Development NOTE: This important rule is not yet formalized into EXPRESS code.

The data associated with a State_composition_relationship are the following:

- child_state;
- parent_state.

4.2.113.1child_state

The child_state specifies the child state in the relationship.

4.2.113.2 parent_state

The parent_state represents the parent state in the relationship.

4.2.114 State_definition

A State_definition is the definition of a state. A state represents a point in time of the evolution of the system. States may have a hierarchy. The hierarchy may be of two types depending upon the state type. Two kind or states have been identified:

- AND states which indicates that at least two branches of the hierarchy of state are active at the same time (and branches may be synchronised or not - this point is not handled in the data model) - See AND_state_definition () for further details,
- OR states which indicates that only one branch in the state hierarchy is active (when two branches can be chosen from a transition, the two conditions are supposed to be exclusive). See OR_state_definition for further details.

A State_definition is an abstract entity and shall never be instantiated.

The semantics associated with a state are intentionally kept to a minimum to allow maximum transfer between tools.

A State_definition is a type of Versionable_model_configuration_management_item.

The data that is associated with a state_instance are the following:

– description;
– name.

4.2.114.1 description

The description specifies specifies additional information about the State_definition.

The description is optional and need not be specified for a particular State_definition.

4.2.114.2 name

The name specifies the word or set of words by which the State_definition is referred to.

The name is optional and need not be specified for a particular State_definition.

4.2.115 State_instance

A State_instance is an instance of a state in a finite state machine. A State_instance shall be unique in the description of a FSM.

The data that is associated with a state_instance are the following:

– definition;
– name.

4.2.115.1 definition

The definition specifies the State_definition that provides the definition of the state.

See for the application assertion.

4.2.115.2 name

The name is the word or set of words by which the State_instance is referred to.

4.2.116 Store

A Store is a place where Data_instance objects can accumulate. The scope of the Store can either be local or global

A Store contains data of a particular type. A Store consumes and produces data instances. There shall be a correspsondance between the types of data instances and the type of data stored by the Store.

Once inside a Store a data instance no longer has a recognisable instance name.

The data associated with Store are the following:

– global_scope;
– name;
– permanent;
– read_only;
– store_data_type;
– store_size.

4.2.116.1 global_scope

The global_Scope specifies whether the data stored in the Store is accessible globally in the function hierarchy or locally only. If the global_scope is TRUE then multiple occurences of a Store object within a function hierarchy means that data entered in one context can be retrieved in all context the store is defined in (provided that the contexts the Store are defined in all share a common parent function_definition). If the global_scope is FALSE then the data entered into the store from one context can only be accessed from within that context.

4.2.116.2 name

The name is the word or set of words by which the Store is referred to.

4.2.116.3 permanent

The permanent specifies whether the Store is permanent or not i.e. whether the store retains its contents between executions of the system (e.g. In a computing example is the data stored on the hard disk or in volatile memory)

NOTE: The possible values for the permanent are: {true, false, unknown}.

The permanency of a store could be related to operational models (eg. on most personal computers the date and time need to be re-entered when the battery is changed but not on each power down / power up cycle).

4.2.116.4 read_only

The read_only specifies whether it is possible to write to the Store or not.

NOTE: The possible values for the read_only are: {true, false, unknown}.

How a read only store gets values is not specified. It is possible that a store could have different values for read_only in different operating modes (ie. Operational data may be loaded during ground warm up but is read only during flight).

4.2.116.5 store_data_type

The store_data_type specifies the data_type_definition of the elements in the store. A store will only store elements of one type.

4.2.116.6 store_size.

The store_size specifies the maximum number of data elements of the type given by store_data_type that can be held in the store.

What happens to elements once this limit has been reached is not defined (eg. New values could be ignored or oldest values could be discarded).

The store_size shall be greater than zero.

4.2.117 String_type_definition

A String_type_definition is the definition of a textual data type. This would typically be used to in giving a type to a descriptive attribute. String_type_definition is a type of data_type_definition

The data associated with String_type_definition are the following:

– default_value;
– initial_value;
– maximum_size.

4.2.117.1 default_value

4.2.117.2initial_value

The initial_value is the value given to the String_type_definition object when it is created.

The initial_value is optional and need not be specified for a particular String_type_definition object.

If a maximum_size value has been assigned, the length of the initial_value string must be equal to or shorter than the maximum_size.

4.2.117.3 maximum_size

The maximum_size is maximum possible size of the String_type_definition object.

The maximum_size is optional and need not be specified for a particular String_type_definition object.

4.2.118 System_function_association

A System_function_association represents a relationship between a Functional_context and the top level function instance that represents the entire system under specification.

NOTE: The system function indicated by the attribute system_function of System_function_association is the function_instance that represents the full system and can thus not have any parent functions. Thus there can be no hierarchical_decomposition_relationships whose child attribute is pointiing at the system function.

See for the Application Assertions.

The data associated with a System_function_association are the following:

– associated_context;
– system_function.

4.2.118.1 associated_context

The associated_context specifies the Functional_context for the system.

4.2.118.2 system_function

The system_function specifies the top level Function_instance (see 4.2.71) .

4.2.119 System_view

A System_view is a representation of the system under specification.

The System_view is a super type of Application (see) and Operational_scenario (see).

The System_view is a type of Versionable_model_configuration_management_item (see).

The System_view is an abstract entity and shall never be instanciated.

The data associated with System_view are the following:

– description
– name.

4.2.119.1 description

The description specifies a textual description of the System_view.

4.2.119.2 name

The name is the word or set of words by which the System_view is referred to.

4.2.120 Template_element

A Template_element is a name-value tuple for textual or numerical information. The Template element has an element_value attribute identifying the value of the element and a name attribute describing the name of the Template_element.

NOTE: By using Template_elements it is possible to create simple templates for textual information. A General_template defines the structure of the template.

Development NOTE: The template structure shall be viewed as a proposal only. It may be to complex for capability 2 and indeed SEDRES.

The data associated with Template_element are the following:

– element_value;
– name.

4.2.120.1 element_value

The element_value is the value that is associated with a template element

4.2.120.2 name

The name is the word or set of words by which the Template_element is referred to.

4.2.121 Transition

A Transition is a path between two Transition_port objects indicating that there exists a direct transition between the states that the Transition_ports are associated with. A Transition may be guarded by at most one Transition_expression (see) object. If there are no Transition_expression object associated with a Transition the existence of a Transition_expression object who always evaluates to TRUE shall be assumed. If a state is active and any of the Transition_expressions of the outgoing transitions from the state evaluates to TRUE that transition may be executed. The execution of a transition is performed in zero time.

Any number of Action_instance objects may be associated with a Transition objects. When a Transition is taken all associated actions are executed. The execution of actions are assumed to be instantaneous but the execution sequence of the actions may still be of importance.

A Transition is a type of General_transition (see 4.2.69).

If there are several outgoing transitions from a state whose Transition_expression object evaluates to TRUE it is not defined which transition will be executed.

In a flat finite state machine a transition is always between two Actual_transtion_port objects, but if the state machine is hierarchical a transition may have a source_port or destination_port (inherited from General_transition) which is connected to a State_definition object.

NOTE 1: A Transitions shall not have as its source_port and destination_port objects of type formal_transition_port. This mean that either the source_port or the destination_port of a Transition is associated with a State_instance object.

NOTE 2: There shall not be a Transition_expression object associated with Transition objects that does not have as its source_port an object of type Actual_transition_port.

NOTE 3: The State_instance (see) the source_port and destination_port is defined for (via the port_of attribute) shall be related to the same Composite_state_definition via the State_composition_relationship (see), or if the type of source_port or destination_port is a Formal_transition_port (see) object then the Formal_transition_port object shall be the port_of the same Composite_state_definition object as the State_instance the other port is a port of.

The data associated with a Transition are the following:

– source_port.

4.2.121.1 source_port

The source_port specifies the Transition_port that is the destiniation of the transition.

There shall be exactly one source_port for a Transition

4.2.122 Transition_composition_relationship

A Transition_composition_relationship is the relation between a General_transition (see4.2.69) and the Composite_state_definition (see 4.2.27) that the Transition resides in. The purpose of the relation is to relate a Transition to the context it is defined for..

The data associated with a Transition_composition_relationship are the following:

– state_definition
– transition

4.2.122.1 state_definition

The state_definition specifies the Composite_state_definition that the General_transition defined by the transition attribute resides in.

4.2.122.2 transition

The transition specifies the General_transition that is part of the context defined by the Composite_state_definition identified by the state_definition attribute.

4.2.123 Transition_expression

A Transition_expression is some text describing the condition that shall be fulfilled for a transition to be taken.

At this stage of the modeling process Transition_expression objects are used capture events, conditions and any other information that can be associated with the firing of a transition.

EXAMPLE: A Transition_expression might be*Launch_missile (if uid == 1)*.

The data associated with a Transition_expression is the following:

- definition
- language

4.2.123.1 definition

The definition specifies the text that forms the transition expression.

4.2.123.2 language

The language specifies the language in which the definition attribute is written. The data model makes no assumption of the language that can be used to formally describe a transition and therefore, no specific rule is given to check this attributes against any semantic meaning. Therefore, it can be any formal language, structured natural language, programming language such as C, ADA etc. The language is merely provided to allow for a human reader to understand the constructs specified in the definition.

4.2.124 Transition_port

A Transition_port is the origin or destination of a transition in a finite state machine. According to the situation the Transition_port is used in (either with a state_instance or a state_definition) the Formal_transition_port (see 4.2.61) or Actual_transition_port (see 4.2.10) items shall be used.

A Transition_port is an abstract entity and will thus never be instanciated.

The data associated with a Transition_port are the following:

- direction

4.2.124.1 direction

The direction indicates whether the Transition_port is an input or output port.

4.2.125 Transition_port_binding

A Transition_port_binding indicates a relationship between an Actual_transition_port (see4.2.10) and a Formal_transition_port (see 4.2.61). The Transition_port_binding ensures that the parameter binding involving a State_instance and State_definition is correct on a one to on basis. The following must be true for a Transition_port_binding to be correct:

The State_definition object the Formal_flow_port specified by the formal_port attribute is the same object as the State_definition object the State_instance specified by the actual_port has as its definition.

See 4.3.82 and 4.3.83 for the Application Assertions.

The data associated with a Transition_port_binding are the following:

- actual_port
- formal_port

4.2.125.1 actual_port

The actual_port specifies the Actual_transition_port on the State_instance involved in the Transition_port_binding.

4.2.125.2 formal_port

The formal_port specifies the formal_transition_port on the State_definition involved in the Transition_port_binding.

4.2.126 Transition_relationship

A Transition_relationship indicates some relationship between two Transition objects. The semantics of the relationship depend on the role attribute.

The data associated with a Transition_relationship are the following:

– related
– relating
– role

4.2.126.1 related

The related attribute indicates the second of the two Transition objects involved in the relationship. The exact semantics of the relationship depend on the value of the role attribute.

4.2.126.2 relating

The relating attribute indicates the first of the two Transition objects involved in the relationship. The exact semantics of the relationship depend on the value of the role attribute.

4.2.126.3 role

The role attribute describes the nature of the relationship between the Transition objects. Where appropriate the following values should be used:

'Condition': The related and relating transitions share common expressions in their respective guard such that the common elements can be represented by inserting a pseudo state. (This corresponds to the Condition marker in Statemate).

4.2.127 Unit

A Unit is the occurrence of a unit of measurement. A unit is always associated with a Number_type_definition.

The data associated with Unit are the following:

4.2.127.1 associated_with

The associated_with specifies the Number_type_definition the unit is associated with.

4.2.127.2 unit_name

The unit_name specifies the name of the unit of measurement, using (ideally) an appropriate ISO standard of Unit terms. Examples would be: s (seconds), m (meters); kg (kilograms), etc.

NOTE: The unit_name shall correspond with standard measurement types.

Development NOTE: Part 41 provides models in this area that can be used for the final proposal.

4.2.128 Validation

A Validation is a confirmation of the correct implementation of a required feature in a system specification. A validated Requirement_instance means that the system the requirement applies to has been judged to be compliant with the requirement.

A Validation is a type of Process_configuration_management_item.

Development NOTE: The justification was made a subtype of Process_configuration_management_item to allow convenient attachment of author and date.

The data associated with a Validation are the following:

– validated_requirement;
– validation_comment.

4.2.128.1 validated_requirement

The validated_requirement specifies the Requirement_instance that has been validated.

Exactly one Requirement_instance is validated via the validated_requirement.

4.2.128.2 validation_comment

The validation comment is a text that holds the comments raised during validation

The validation_comment is optional and need not be specified.

4.2.129 Variant

A Variant represents a relationship between two Element_version objects (see 4.2.87) where the two Element_versions are variants of each other.

A Variant is a type of Element_version_relationship (see 4.2.86).

EXAMPLE 23. Two versions of a model for a kettle may be developed in parallel with the only difference between the kettle models being the plugs are designed for different continents. The two different versions should be considered variants of each other.

NOTE - This entity has no extra attributes associated with it. It is included solely to improve the semantics of the model.

4.2.130 Versionable_model_configuration_management_item

A Versionable_model_configuration_management_item is a definition of a item which is versionable.

The Versionable_model_configuration_management_item is abstract and shall thus never be instanciated.

NOTE: All definition_type entities are subtypes of Versionable_model_configuration_management_item.

The data associated with Versionable_model_configuration_management_item are the following:

– associated_version.

4.2.130.1 associated_version

The associated_version specifies the Element_version that contains versioning information about this entity.

The associated_version need not to be specified for a particular Versionable_model_configuration_management_item.

If no associated_version is supplied it shall be assumed that only one version of the Versionable_model_configuration_management_item exists.

4.2.131 View_relationship

A view_relationship is a hierarchical relation between two Visual_element (see) objects such that the child object is a part of the Graphics_view (see) of the parent object.

NOTE: If a View_relationship exists between two Graphics_view objects the same hierarchical relationship shall exist between the Function_context_definition (see), General_function_definition (see) or State_definition (see) that are involved in the hierarchy.

Development NOTE: This rule is not encoded in the EXPRESS for capability-2.

Each Graphics_view object is potentially part of many Multi_level_views. To allow for overlapping views each View_relationship shall, via the valid_in attribute define the Multi_level_view for which the relationship is valid.

The data associated with View_relationship are the following:

- child;
- parent;
- valid_in.

4.2.131.1 child

The child specifies the Graphics_view which is superimposed in the Graphics_view of the parent.

4.2.131.2 parent

The parent specifies the Graphics_view which acts as a context of the child Graphics_view.

4.2.131.3 valid_in

The valid_in specifies the Multi_level_view for which the relation is valid.

4.2.132 Visual_element

A Visual_element is the super-type for all entities carrying graphical information for an object. The Visual_element is a super-type of Actual_port_position (see 4.2.9), Formal_port_position (see 4.2.60), Graphics_link (see 4.2.70) or Graphics_node (see 4.2.71).

The Visual_element is abstract and shall not be instanciated.

The data associated with Visual_element are the following:

- view.

4.2.132.1 view

The view specifies the Graphics_view for which the Visual_element is defined for.

4.2.133 Workspace_revision

A Workspace_revision represents a relationship between two Element_version objects (see 4.2.87) where the related Element_version is created by modifying the relating Element_version and the new version does not go into the global repository.

NOTE - This relationship type is very similar to Revision (see117th). The difference is that a workspace revision is not released into the global repository.

A Workspace_revision is a type of Element_version_relationship (see 4.2.86).

EXAMPLE 24. An engineer may correct some minor typographical errors from a previous version. Subsequently more changes are made and the model is re-released. The version created when fixing typographical errors would be considered a Workspace_revision and the version created and released would be considered a Revision.

NOTE - This entity has no extra attributes associated with it. It is included solely to improve the semantics of the model.

4.3 Application Assertions

This subclause specifies the application assertions for the blah application protocol. Application assertions specify the relationships between application objects, the cardinality of the relationships, and the rules required for the integrity and validity of the application objects and UoF’s. The application assertions and their definitions are given below.

4.3.1 Action_instance to Leaf_state_definition
Each Action_instance refers to exactly one Leaf_state_definition in the role of action_for. Each Leaf_state_definition acts as action_for zero or one Action_instance.
This relationship is established via the action_association_select select.

4.3.2 Action_instance to Transition
Each Action_instance refers to exactly one Transition in the role of action_for. Each Transition acts as action_for zero or one Action_instance.
This relationship is established via the action_association_select select.

4.3.3 Action_precedence_relationship to Action_instance
Each Action_precedence_relationship refers to exactly one Action_instance in the role of relating.
Each Action_instance acts as relating for zero or more Action_precedence_relationship.
Each Action_precedence_relationship refers to exactly one Action_instance in the role of related.
Each Action_instance acts as related for zero or more Action_precedence_relationship.

4.3.4 Actual_flow_port to External_entity
Each Actual_flow_port refers to exactly one External_entity in the role of port_of. Each External_entity acts as port_of for zero or more Actual_flow_port.
This relationship is established via the port_usage_select select.

4.3.5 Actual_flow_port to FSM_model
Each Actual_flow_port refers to exactly one FSM_model in the role of port_of. Each FSM_model acts as port_of for zero or more Actual_flow_port.
This relationship is established via the port_usage_select select.

4.3.6 Actual_flow_port to Function_instance
Each Actual_flow_port refers to exactly one Function_instance in the role of port_of. Each Function_instance acts as port_of for zero or more Actual_flow_port.
This relationship is established via the port_usage_select select.

4.3.7 Actual_flow_port to Store
Each Actual_flow_port refers to exactly one Store in the role of port_of. Each Store acts as port_of for zero or more Actual_flow_port.
This relationship is established via the port_usage_select select.

4.3.8 Actual_transition_port to State_instance
Each Actual_transition_port refers to exactly one State_instance in the role of port_of. Each State_instance acts as port_of for zero or more Actual_transition_port.

4.3.9 Activity_element_assignment to Activity
Each Activity_element_assignment refers to exactly one Activity in the role of activity. Each Activity acts as activity for zero or more Activity_element_assignment.

4.3.10 Activity_element_assignment to Model_configuration_management_item
Each Activity_element_assignment refers to exactly one Model_configuration_management_item in the role of element. Each Model_configuration_management_item acts as element for zero or more Model_configuration_management_information.

4.3.11 Activity_relationship to Activity
Each Activity_relationship refers to exactly one Activity in the role of relating_activity. Each Activity acts as relating_activity for zero or more Activity_relationship.
Each Activity_relationship refers to exactly one Activity in the role of related_activity. Each Activity acts as related_activity for zero or more Activity_relationship.

4.3.12 Approval_assignment to Approval
Each Approval_assignment refers to exactly one Approval in the role of assigned_approval. Each Approval acts as assigned_approval for one or more Approval_assignment.

4.3.13 Approval_assignment to Configuration_management_information
Each Approval_assignment refers to exactly one Configuration_management_information in the role of item. Each Configuration_management_information acts as item for zero or more Approval_assignment.

4.3.14 Approval_date_time to Approval_assignment
Each Approval_date_time refers to exactly one Approval_assignment in the role of dated_approval. Each Approval_assignment acts as dated_approval for zero or one Approval_date_time.

4.3.15 Approval_date_time to Date
Each Approval_date_time refers to exactly one Date in the role of date. Each Date acts as date for zero or more Approval_date_time.

4.3.16 Approval_person_organization to Approval
Each Approval_person_organization refers to exactly one Approval in the role of authorized_approval. Each Approval acts as authorized_approval for zero or more Approval_person_organization.

4.3.17 Approval_relationship to Approval
Each Approval_relationship refers to exactly one Approval in the role of relating_approval. Each Approval acts as relating_approval for zero or more Approval_relationship.
Each Approval_relationship refers to exactly one Approval in the role of related_approval. Each Approval acts as related_approval for zero or more Approval_relationship.

4.3.18 Causal_relationship to Functional_chain
Each Causal_relationship refers to exactly one Functional_chain in the role of chain. Each Functional_chain acts as chain for one or more Causal_relationship.

4.3.19 Causal_relationship to Function_instance
Each Causal_relationship refers to exactly one Function_instance in the role of preceding. Each Function_instance acts as causing for zero or more Causal_relationship.
Each Causal_relationship refers to exactly one Function_instance in the role of succeeding. Each Function_instance acts as caused for zero or more Causal_relationship.

4.3.20 Context_system_view_relationship to Function_context_definition

Each Context_system_view_relationship refers to exactly one Function_context_definition in the role of function_context. Each Function_context_definition acts as function_context for zero or one Context_system_view_relationship.

4.3.21 Context_system_view_relationship to System_view

Each Context_system_view_relationship refers to exactly one System_view in the role of system_view. Each System_view acts as system_view for zero or one Context_system_view_relationship.

4.3.22 Data_composition_relationship to Composite_data_type_definition
Each Data_composition_relationship refers to exactly one Composite_data_type_definition in the role of parent. Each Composite_data_type_definition acts as parent for one or more Data_composition_relationship.

4.3.23 Data_composition_relationship to Data_instance
Each Data_composition_relationship refers to exactly one Data_instance in the role of child. Each Data_instance acts as child for zero or more Data_composition_relationship.

4.3.24 Data_instance to Data_type_definition
Each Data_instance refers to exactly one Data_type_definition in the role of definition. Each Data_type_definition acts as definition for one or more Data_instance.

4.3.25 Data_transfer to External_to_requirement_association
Each Data_transfer refers to exactly one External_to_requirement_association in the role of transfer. Each External_to_requirement_association acts as transfer for zero or one Data_transfer.

4.3.26 Date_assignment to Configuration_management_information
Each Date_assignment refers to exactly one Configuration_management_information in the role of item. Each Configuration_management_information acts as item for zero or more Date_assignment.

4.3.27 Derived_requirement_relationship to Requirement_instance
Each Derived_requirement_relationship refers to one or more Requirement_instance in the role of orginal_requirement. Each Requirement_instance acts as orginal_requirement for zero or more Derived_requirement_relationship.
Each Derived_requirement_relationship refers to exactly one Requirement_instance in the role of derived_requirement. Each Requirement_instance acts as derived_requirement for zero or one Derived_requirement_relationship.

4.3.28 Documentation_relationship to Documentation_reference
Each Documentation_relationship refers to exactly one Documentation_reference in the role of relating_documentation. Each Documentation_reference acts as relating_documentation for zero or more Documentation_relationship.
Each Documentation_relationship refers to exactly one Documentation_reference in the role of related_documentation. Each Documentation_reference acts as related_documentation for zero or more Documentation_relationship.

4.3.29 Execution_time to General_function_definition
Each Execution_time refers to exactly one General_function_definition in the role of timing. Each General_function_definition acts as timing for zero to three Execution_time.

4.3.30 Execution_time to Number_type_definition
Each Execution_time refers to exactly one Number_type_definition in the role of time. Each Number_type_definition acts as time for zero or more Execution_time.

4.3.31 External_entity_context_relationship to External_entity
Each External_entity_context_relationship refers to exactly one External_entity in the role of external. Each External_entity acts as external for zero or more External_entity_context_relationship.

4.3.32 External_entity_context_relationship to Function_context_definition
Each External_entity_context_relationship refers to Function_context_definition in the role of associated_context. Each Function_context_definition acts as associated_context for External_entity_context_relationship.

4.3.33 External_to_requirement_association to Requirement_instance
Each External_to_requirement_association refers to exactly one Requirement_instance in the role associated_requirement. Each Requirement_instance acts as associated_requirement for zero or more External_to_requirement_association.

4.3.34 Flow to Flow_port
Each Flow refers to two or more Flow_port in the role of connected_to. Each Flow_port acts as connected_to for one or more Flow .

4.3.35 Flow_composition_relationship to Composite_function_definition

Each Flow_composition_relationship refers to exactly one Composite_function_definition in the role of flow_context. Each Composite_function_definition acts as flow_context for zero or more Flow_composition_relationship.

NOTE: This is established through the type flow_context_select.

4.3.36 Flow_composition_relationship to Flow

Each Flow_composition_relationship refers to exactly one Flow in the role of flow_component. Each Flow acts as flow_component for zero or more Flow_composition_relationship.

4.3.37 Flow_composition_relationship to Flow_group

Each Flow_composition_relationship refers to exactly one Flow_group in the role of flow_component. Each Flow_group acts as flow_component for one or more Flow_composition_relationship.

4.3.38 Flow_composition_relationship to Function_context_definition

Each Flow_composition_relationship refers to exactly one Function_context_definition in the role of flow_context. Each Function_context_definition acts as flow_context for zero or more Flow_composition_relationship.

NOTE: This is established through the type flow_context_select.

4.3.39 Flow_port to Data_instance
Each Flow_port refers to exactly one Data_instance in the role of data. Each Data_instance acts as data for zero or more Flow_port.

4.3.40 Formal_flow_port to Composite_function_definition
Each Formal_flow_port refers to exactly one Composite_function_definition in the role of port_of. Each Composite_function_definition acts as port_of for zero or more Formal_flow_port.
This relationship is established via the port_specification_select select.

4.3.41 Formal_transition_port to State_definition
Each Formal_transition_port refers to exactly one State_definition in the role of port_of. Each State_definition acts as port_of for zero or more Formal_transition_port.

4.3.42 FSM_model to Composite_state_definition
Each FSM_model refers to exactly one Composite_state_definition in the role of definition. Each Composite_state_definition acts as definition for zero or more FSM_model.

4.3.43 Function_instance to General_function_definition
Each Function_instance refers to exactly one General_function_definition in the role of definition. Each General_function_definition acts as definition for one or more Function_instance.

4.3.44 Hierarchical_decomposition_relationship to Composite_function_definition
Each Hierarchical_decomposition_relationship refers to exactly one Composite_function_definition in the role of parent. Each Composite_function_definition acts as parent for one or more Hierarchical_decomposition_relationship.

4.3.45 Hierarchical_decomposition_relationship to FSM_model
Each Hierarchical_decomposition_relationship refers to exactly one FSM_model in the role of child. Each FSM_model acts as child for zero or one Hierarchical_decomposition_relationship.
The relationship is established via the Function_hierarchy_select select.

4.3.46 Hierarchical_decomposition_relationship to Function_instance
Each Hierarchical_decomposition_relationship refers to exactly one Function_instance in the role of child. Each Function_instance acts as child for zero or one Hierarchical_decomposition_relationship.
The relationship is established via the Function_hierarchy_select select.

4.3.47 Hierarchical_decomposition_relationship to Store
Each Hierarchical_decomposition_relationship refers to exactly one Store in the role of child. Each Store acts as child for zero or one Hierarchical_decomposition_relationship.
The relationship is established via the Function_hierarchy_select select.

4.3.48 Element_version to Element
Each Element_version refers to exactly one Element in the role of version_of. Each Element acts as version_of for zero or more Element_version .

4.3.49 Element_version_relationship to Element_version
Each Element_version_relationshiprefers to exactly one Element_version in the role of related_version. Each Element_version acts as related_version for zero or more Element_version_relationship.
Each Element_version_relationshiprefers to exactly one Element_version in the role of relating_version. Each Element_version acts as relating_version for zero or more Element_version_relationship.

4.3.50 Justification_relationship to Justification
Each Justification_relationship refers to exactly one Justification in the role of relating. Each Justification acts as relating for zero or more Justification_relationship.
Each Justification_relationship refers to exactly one Justification in the role of related. Each Justification acts as related for zero or more Justification_relationship.

4.3.51 Justification to Model_configuration_management_information
Each Justification refers to exactly one Model_configuration_management_informatiification acts as related for zero or more Justification_relationship.on in the role of applies_to. Each Model_configuration_management_item acts as applies_to for zero or more Justification.

4.3.52 Maturity to Model_configuration_management_information
Each Maturity refers to exactly one Model_configuration_management_item in the role of element_maturity. Each Model_configuration_management_item acts as element_maturity for zero or one Maturity.

4.3.53 Model_configuration_management_item to Element_version

Each Model_configuration_management_item refers to exactly one Element_version in the role of associated_version. Each Element_version acts as associated_version for exactly one Model_configuration_management_information.

4.3.54 Operational_scenario to Application

Each Operational_scenario refers to exactly one Application in the role of scenario_of. Each Application acts as scenario_of zero or more Operational_scenarios.

4.3.55 Organization_address to Organization
Each Organization_address refers to two or more Organization in the role of organizations. Each Organization acts as organisations for zero or one Organization_address.

4.3.56 Organization_relationship to Organization
Each Organization_relationship refers to exactly one Organization in the role of related_organization. Each Organization acts as related_organization for zero or more Organization_relationship.
Each Organization_relationship refers to exactly one Organization in the role of relating_organization. Each Organization acts as relating_organization for zero or more Organization_relationship.

4.3.57 Package_element_assignment to Model_configuration_management_information
Each Package_element_assignment refers to exactly one Model_configuration_management_item in the role of element. Each Model_configuration_management_item acts as element for zero or more Package.

4.3.58 Package_element_assignment to Package
Each Package_element_assignment refers to exactly one Package in the role of package. Each Package acts as package for two or more Package_element_assignment.

4.3.59 Person_and_organization to Organization
Each Person_and_organization refers to exactly one Organization in the role of the_organization. Each Organization acts as the_organization for zero or more Person_and_organization.

4.3.60 Person_and_organization to Person
Each Person_and_organization refers to exactly one Person in the role of the_person. Each Person acts as the_person for zero or more Person_and_organization.

4.3.61 Person_or_organization_assignment to Organization
Each Person_or_organization_assignment refers to zero or one Organization in the role of assigned_person_or_organization. Each Organization acts as assigned_person_or_organization for zero or more Person_or_organization_assignment.
This relationship is established via the Person_organization_select.

4.3.62 Person_or_organization_assignment to Person
Each Person_or_organization_assignment refers to zero or one Person in the role of assigned_person_or_organization. Each Person acts as assigned_person_or_organization for zero or more Person_or_organization_assignment.
This relationship is established via the Person_organization_select.

4.3.63 Person_or_organization_assignment to Person_and_organization
Each Person_or_organization_assignment refers to zero or one Person_and_organization in the role of assigned_person_or_organization. Each Person_and_organization acts as assigned_person_or_organization for zero or more Person_or_organization_assignment.
This relationship is established via the Person_organization_select.

4.3.64 Personal_address to Person
Each Personal_address refers to exactly one Person in the role of people. Each Person acts as people for zero or one Personal_address.

4.3.65 Project to Activity
Each Project refers to one or more Activity in the role of work_program. Each Activity acts as work_program for one or more Project.

4.3.66 Requirement_alternative_relationship to Requirement_instance
Each Requirement_alternative_relationship refers to exactly one Requirement_instance in the role of relating. Each Requirement_instance acts as relating for zero or more Requirement_alternative_relationship
Each Requirement_alternative_relationship refers to exactly one Requirement_instance in the role of related. Each Requirement_instance acts as related for zero or more Requirement_alternative_relationship.

4.3.67 Requirement_composition_relationship to Requirement_definition
Each Requirement_composition_relationship refers to exactly one Requirement_defintion in the role of parent_definition. Each Requirement_definition acts as child_instance for zero or more Requirement_composition_relationship.

4.3.68 Requirement_composition_relationship to Requirement_instance
Each Requirement_composition_relationship refers to exactly one Requirement_instance in the role of child_instance. Each Requirement_instance acts as child_instance for zero or more Requirement_composition_relationship.

4.3.69 Requirement_instance to Requirement_definition
Each Requirement_instance refers to exactly one Requirement_definition in the role of definition. Each Requirement_definition acts as definition for one or more Requirement_instance.

4.3.70 Requirement_system_view_relationship to System_view
Each Requirement_system_view_relationship refers to exactly one System_view in the role of system_view. Each System_view acts as system_view for zero or more Requirement_system_view_relationship.

4.3.71 Requirement_system_view_relationship to Requirement_instance
Each Requirement_system_view_relationship refers to exactly one Requirement_instance in the role of requirement. Each Requirement_instance acts as requirement for zero or more Requirement_system_view_relationship.

4.3.72 Scenario_relationship to Operational_scenario

Each Scenario_relationship refers to exactly one Operational_scenario in the role of relating. Each Operational_scenario acts as relating for zero or more Scenario_relationship. Each Scenario_relationship refers to exactly one Operational_scenario in the role of related. Each Operational_scenario acts as related for zero or more Scenario_relationship.

The relating and related shall not refer to the same Operational_scenario.

4.3.73 State_instance to State_definition
Each State_instance refers to exactly one State_definition in the role of defintion. Each State_definition acts as definition for zero or more State_instance.

4.3.74 State_composition_relationship to State_instance
Each State_composition_relationship refers to exactly one State_instance in the role of child_state. Each State_instance acts as child_state for zero or more State_composition_relationship.

4.3.75 State_composition_relationship to State_definition
Each State_composition_relationship refers to exactly one State_definition in the role of parent_state. Each State_definition acts as parent_state for one or more State_composition_relationship.

4.3.76 Store to Data_type_definition
Each Store refers to exactly one Data_type_definition in the role of store_data_type. Each Data_type_definition acts as store_data_type for zero or more Store.

4.3.77 System_function_association to Function_context_definition
Each System_function_association refers to exactly one Function_context_definition in the role of associated_context. Each Function_context_definition acts as associated_context for exactly one System_function_association.

4.3.78 System_function_association to Function_instance
Each System_function_association refers to exactly one Function_instance in the role of system_function. Each Function_instance acts as system_function for zero or more System_function_association.

4.3.79 Transition to Transition_port
Each Transition refers to exactly one Transition_port in the role of source_port. Each Transition_port acts as source_port for zero or one Transition.

Each Transition refers to exactly one Transition_port in the role of destination_port. Each Transition_port acts as destination_port for zero or one Transition.

4.3.80 Transition_relationship to Transition
Each Transition_relationship refers to exactly one Transition in the role of relating. Each Transition acts as relating for zero or more Transition_relationship.
Each Transition_relationship refers to exactly one Transition in the role of related. Each Transition acts as related for zero or more Transition_relationship.

4.3.81 Transition_expression to Transition

Each Transition_expression refers to exactly one Transition in the role of transition. Each Transition acts as transition for zero or one Transition_expression.

4.3.82 Transition_port_binding to Actual_transition_port

Each Transition_port_binding refers to exactly one Actual_transition_port in the role of actual_port. Each Actual_transition_port acts as actual_port for zero or one Transition_port_binding.

4.3.83 Transition_port_binding to Formal_transition_port

Each Transition_port_binding refers to exactly one Formal_transition_port in the role of formal_port. Each Formal_transition_port acts as formal_port for zero or many Transition_port_binding.

4.3.84 Unit to Number_type_definition
Each Unit refers to zero or more Number_type_definition in the role of associated_with. Each Number_type_definition acts as associated_with for zero or one Unit.

4.3.85 Validation to Requirement_instance
Each Validation refers to exactly one Requirement_instance in the role of validated_requirement. Each Requirement_instance acts as validated_requirement for zero or more Validation.

5 EXPRESS-G REPRESENTATION OF THE ARM

6 EXPRESS REPRESENTATION OF THE ARM

```
SCHEMA sedres_flat;

 TYPE status = text;
 END_TYPE;

 TYPE person_organization_select = SELECT (
   person,
   person_and_organization,
   organization
  );
 END_TYPE;

 TYPE text = STRING;
 END_TYPE;

 TYPE explicitly_structured_text = LIST [1:?] OF text_elements;
 END_TYPE;

 TYPE control_characters = ENUMERATION OF ( cr,tab  );
 END_TYPE;

 TYPE text_elements = SELECT (
   control_characters,
   text
  );
 END_TYPE;

 TYPE label = STRING;
 END_TYPE;

 TYPE text_select = SELECT (
   general_template,
   explicitly_structured_text,
   text
  );
 END_TYPE;

 TYPE data_direction = ENUMERATION OF ( to_system,from_system  );
 END_TYPE;

 TYPE port_data_relation = ENUMERATION OF ( consumer,producer  );
 END_TYPE;

 TYPE port_usage_select = SELECT (
   store,
   function_instance,
   fsm_model,
   external_entity
  );
 END_TYPE;

 TYPE relation_type = ENUMERATION OF ( sequence,concurrent  );
 END_TYPE;

 TYPE action_association_select = SELECT (
   general_transition,
   state_definition
  );
 END_TYPE;

 TYPE requirement_allocation_select = SELECT (
   function_instance,
   store,
   causal_chain,
   fsm_model,
   state_instance,
   data_instance
  );
 END_TYPE;

 TYPE function_hierarchy_select = SELECT (
   function_instance,
   store,
   fsm_model
  );
 END_TYPE;

 TYPE bound_type = ENUMERATION OF ( upper,lower  );
 END_TYPE;

 TYPE timing_type = ENUMERATION OF ( worst_case,best_case,nominal_case  );
 END_TYPE;

 TYPE port_direction = ENUMERATION OF ( input,output  );
 END_TYPE;

TYPE port_position_select = SELECT (
   transition_port,
   flow_port
  );
 END_TYPE;

 TYPE definition_select = SELECT (
   composite_state_definition,
   general_function_definition,
   function_context_definition
  );
 END_TYPE;

 TYPE identifier = STRING;
 END_TYPE;

 TYPE flow_element_select = SELECT (
   flow,
   flow_group
  );
 END_TYPE;

 TYPE flow_context_select = SELECT (
   composite_function_definition,
   function_context_definition
  );
 END_TYPE;

 TYPE node_select = SELECT (
   system_function_association,
   hierarchical_decomposition_relationship,
   external_entity_context_relationship,
   state_composition_relationship
  );
 END_TYPE;

 TYPE link_select = SELECT (
   flow_composition_relationship,
   transition_composition_relationship
  );
 END_TYPE;

 TYPE name_binding_select = SELECT (
   function_reference,
   actual_name_port
  );
 END_TYPE;

 TYPE action_function_operation_type = ENUMERATION OF ( start, stop, suspend, resume  );
 END_TYPE;


 ENTITY project
  SUBTYPE OF (
   Process_Configuration_Management_Item
  );
  work_program : SET [1:?] OF activity;
 END_ENTITY;

 ENTITY activity
  SUBTYPE OF (
   Process_Configuration_Management_Item
  );
   name : label;
   status : status;
 END_ENTITY;

 ENTITY activity_element_assignment;
  activity : activity;
  element : Model_Configuration_Management_Item;
  name : OPTIONAL label;
  description : OPTIONAL text;
 END_ENTITY;

 ENTITY activity_relationship;
  relating_activity : activity;
  related_activity : activity;
  name : label;
  description : OPTIONAL text_select;
 WHERE
 (* wr1: The relating and related activity must not be the same activity objects *)
wr1 : relating_activity :<>: related_activity;
 END_ENTITY;

 ENTITY approval;
  level : label;
  status : label;
 END_ENTITY;
```

```
ENTITY element;
 name : label;
 description : text_select;
 id : identifier;
UNIQUE
 id;
END_ENTITY;

ENTITY element_version;
 version_of : element;
 version_identifier : identifier;
 description : text_select;
UNIQUE
 single : version_identifier;
END_ENTITY;

ENTITY element_version_relationship
 ABSTRACT SUPERTYPE OF ( ONEOF(variant,revision,alternative,workspace_revision) );
 related_version : element_version;
 relating_version : element_version;
 change_description : OPTIONAL text_select;
WHERE
wr1 : related_version :<>: relating_version;
END_ENTITY;

ENTITY variant
 SUBTYPE OF (
  element_version_relationship
 );
END_ENTITY;

ENTITY revision
 SUBTYPE OF (
  element_version_relationship
 );
END_ENTITY;

ENTITY alternative
 SUBTYPE OF (
  element_version_relationship
 );
END_ENTITY;

ENTITY workspace_revision
 SUBTYPE OF (
  element_version_relationship
 );
END_ENTITY;

ENTITY date;
 year_component : INTEGER;
 month_component : INTEGER;
 day_component : INTEGER;
 hour_component : INTEGER;
 minute_component : INTEGER;
 second_component : INTEGER;
WHERE
 wr1 : {0<= hour_component < 24};
 wr2 : {0<= minute_component < 60};
 wr3 : {0<= second_component < 60};
```

```
  wr4 : {1<= month_component <= 12};
  wr5 : {1<= day_component <=31};
  wr6 : year_component >= 0;
 END_ENTITY;

 ENTITY organization;
  name : label;
  description : text_select;
  id : OPTIONAL identifier;
 END_ENTITY;

 ENTITY organization_relationship;
  related_organization : organization;
  relating_organization : organization;
  name : label;
  description : text_select;
 WHERE
 (* wr1: The relating and related organization must not be the same organization*)
wr1 : related_organization :<>: relating_organization;
 END_ENTITY;

 ENTITY address;
  town : OPTIONAL label;
  region : OPTIONAL label;
  postcode : OPTIONAL label;
  country : OPTIONAL label;
  facsimile_number : OPTIONAL label;
  electronic_mail_address : OPTIONAL label;
  telex_number : OPTIONAL label;
  postbox_number : OPTIONAL label;
  telephone_number : OPTIONAL label;
  street : OPTIONAL label;
  street_number : OPTIONAL label;
  internal_location : OPTIONAL label;
 END_ENTITY;

 ENTITY organizational_address
  SUBTYPE OF (
   address
  );
  organizations : SET [1:?] OF organization;
 END_ENTITY;

 ENTITY personal_address
  SUBTYPE OF (
   address
  );
  people : SET [1:?] OF person;
 END_ENTITY;

 ENTITY person;
  last_name : OPTIONAL label;
  first_name : OPTIONAL label;
  prefix_titles : OPTIONAL LIST [1:?] OF label;
  suffix_titles : OPTIONAL LIST [1:?] OF label;
  middle_names : OPTIONAL LIST [1:?] OF label;
  id : identifier;
UNIQUE
  no_duplicate_id: id;
 END_ENTITY;
```

```
ENTITY person_and_organization;
 the_person : person;
 the_organization : organization;
 name : label;
 description : text_select;
END_ENTITY;

ENTITY person_or_organization_assignment;
 item : Configuration_Management_Item;
 assigned_person_or_organization : person_organization_select;
 role : label;
END_ENTITY;

ENTITY approval_relationship;
 relating_approval : approval;
 related_approval : approval;
 name : label;
 description : text_select;
END_ENTITY;

ENTITY approval_person_organization;
 authorized_approval : approval;
 person_organization : person_organization_select;
 role : label;
END_ENTITY;

ENTITY approval_assignment;
 assigned_approval : approval;
 item : Configuration_Management_Item;
 status : label;
 description : text_select;
END_ENTITY;

ENTITY approval_date_time;
 dated_approval : approval_assignment;
 date : date;
 role : label;
END_ENTITY;

ENTITY package
 SUBTYPE OF (
  model_configuration_management_item
 );
 name : label;
 description : OPTIONAL text_select;
INVERSE
 elements : SET [2:?] OF package_element_assignment FOR package;
END_ENTITY;

ENTITY package_element_assignment;
 package : package;
 element : Model_Configuration_Management_Item;
 name : OPTIONAL label;
 description : OPTIONAL text_select;
END_ENTITY;

ENTITY documentation
 SUBTYPE OF (
  documentation_reference
  );
  text : text_select;
 END_ENTITY;

 ENTITY documentation_relationship;
  relating_documentation : documentation_reference;
  related_documentation : documentation_reference;
  name : OPTIONAL label;
  description : OPTIONAL text_select;
 WHERE
(* correct_relation: The related and relating documentation must not be the same object *)
correct_relation : relating_documentation :<>: related_documentation;
 END_ENTITY;

 ENTITY Process_Configuration_Management_Item
  ABSTRACT SUPERTYPE OF ( ONEOF(project,activity,maturity,validation,justification) )
  SUBTYPE OF (
   configuration_management_item
  );
 END_ENTITY;

 ENTITY Configuration_Management_Item
  ABSTRACT SUPERTYPE OF (ONEOF (process_configuration_management_item,
model_configuration_management_item));
 END_ENTITY;

 ENTITY Model_Configuration_Management_Item
  ABSTRACT SUPERTYPE OF (ONEOF (package,vmcmi,nvmcmi))
  SUBTYPE OF (
   Configuration_Management_Item
  );
  id : identifier;
 INVERSE
  maturity : SET [0:1] OF maturity FOR element_maturity;
 UNIQUE
  single : id;
 END_ENTITY;

 ENTITY documentation_reference
  SUPERTYPE OF ( ONEOF(documentation,external_document) )
  SUBTYPE OF (
   VMCMI
  );
  name : label;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY template_element;
  name : label;
  element_value : explicitly_structured_text;
 END_ENTITY;

 ENTITY general_template;
  elements : LIST [1:?] OF template_element;
 END_ENTITY;

 ENTITY requirement_definition
  ABSTRACT SUPERTYPE OF (
ONEOF(constraint_definition,functional_requirement_definition,operational_requirement_definition,phy
sical_requirement_definition) )
```

```
 SUBTYPE OF (
  VMCMI
 );
 name : OPTIONAL label;
 description : text_select;
INVERSE
 composed_of : SET [0:?] OF requirement_composition_relationship FOR parent_definition;
END_ENTITY;

ENTITY constraint_definition
 SUBTYPE OF (
  requirement_definition
 );
END_ENTITY;

ENTITY functional_requirement_definition
 SUBTYPE OF (
  requirement_definition
 );
END_ENTITY;

ENTITY operational_requirement_definition
 SUBTYPE OF (
  requirement_definition
 );
END_ENTITY;

ENTITY physical_requirement_definition
 SUBTYPE OF (
  requirement_definition
 );
END_ENTITY;

ENTITY maturity
 SUBTYPE OF (
  Process_Configuration_Management_Item
 );
 element_maturity : Model_Configuration_Management_Item;
 maturity_description : label;
END_ENTITY;

ENTITY external_entity
 SUBTYPE OF (
  NVMCMI
 );
 name : label;
 description : OPTIONAL text_select;
INVERSE
 referenced_in_requirement : SET [0:?] OF external_to_requirement_association FOR
implied_external;
 in_context : SET [0:?] OF external_entity_context_relationship FOR external;
END_ENTITY;

ENTITY external_to_requirement_association;
 associated_requirement : requirement_instance;
 implied_external : external_entity;
INVERSE
 associated_data : SET [0:1] OF data_transfer FOR transfer;
END_ENTITY;
```

```
 ENTITY data_transfer;
  transfer : external_to_requirement_association;
  data : data_instance;
  direction : data_direction;
 END_ENTITY;

 ENTITY requirement_instance
  SUBTYPE OF (
   NVMCMI
  );
  name : label;
  definition : requirement_definition;
 INVERSE
  implied_externals : SET [0:?] OF external_to_requirement_association FOR associated_requirement;
  child_of : SET [0:?] OF requirement_composition_relationship FOR child_instance;
  validated : SET [0:?] OF validation FOR validated_requirement;
 END_ENTITY;

 ENTITY requirement_composition_relationship;
  parent_definition : requirement_definition;
  child_instance : requirement_instance;
  description : text_select;
 END_ENTITY;

  ENTITY operational_scenario
  SUBTYPE OF (
   system_view
  );
  scenario_of : application;
 END_ENTITY;

 ENTITY requirement_system_view_relationship;
  system_view : system_view;
  requirement : requirement_instance;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY scenario_relationship;
  related : operational_scenario;
  relating : operational_scenario;
  transition_condition : text_select;
  description : OPTIONAL text_select;
 WHERE
(* correct_relationship : the relating and related scenario must not be the same scenario *)
correct_relationship : related :<>: relating;
 END_ENTITY;

 ENTITY derived_requirement_relationship;
  original_requirement : SET [1:?] OF requirement_instance;
  derived_requirement : requirement_instance;
  description : text_select;
 WHERE
(* correct_derivation : the derived_requirement must not be in the set of original requirements *)
correct_derivation : NOT (derived_requirement IN original_requirement);
 END_ENTITY;

 ENTITY requirement_alternative_relationship;
  relating : requirement_instance;
  related : requirement_instance;
  description : OPTIONAL text_select;
 WHERE
(* no_self : the relating and related requirement_instance must not be the same instance *)
no_self : relating :<>: related;
 END_ENTITY;

 ENTITY validation
  SUBTYPE OF (
   Process_Configuration_Management_Item
  );
  validated_requirement : requirement_instance;
  validation_comment : OPTIONAL text_select;
 END_ENTITY;

 ENTITY function_context_definition
  SUBTYPE OF (
   VMCMI
  );
  name : OPTIONAL label;
  description : OPTIONAL text_select;
 INVERSE
  has_externals : SET [1:?] OF external_entity_context_relationship FOR associated_context;
  system : SET [1:1] OF system_function_association FOR associated_context;
  in_view : SET [0:1] OF context_system_view_relationship FOR function_context;
 END_ENTITY;

  ENTITY context_system_view_relationship;
  function_context : function_context_definition;
  system_view : system_view;
 END_ENTITY;


 ENTITY external_entity_context_relationship;
  associated_context : function_context_definition;
  external : external_entity;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY system_function_association;
  associated_context : function_context_definition;
  system_function : function_instance;
 END_ENTITY;

 ENTITY composite_function_definition
  SUBTYPE OF (
   general_function_definition
  );
 INVERSE
  parent_of : SET [1:?] OF hierarchical_decomposition_relationship FOR parent;
 END_ENTITY;

 ENTITY function_instance
  SUBTYPE OF (
   NVMCMI
  );
  definition : general_function_definition;
  name : label;
 INVERSE
  system_function_for : SET [0:?] OF system_function_association FOR system_function;
  control_ports : SET [0:?] OF control_port FOR port_of;
 END_ENTITY;
 ENTITY hierarchical_decomposition_relationship;
  child : function_hierarchy_select;
  parent : composite_function_definition;
  description : OPTIONAL text_select;
 END_ENTITY;


 ENTITY store
  SUBTYPE OF (
   NVMCMI
  );
  store_data_type : data_type_definition;
  store_size : INTEGER;
  name : label;
  permanent : LOGICAL;
  read_only : LOGICAL;
  global_scope : BOOLEAN;
 END_ENTITY;

 ENTITY data_instance;
  definition : data_type_definition;
  is_constant : boolean;
  name : label;
 INVERSE
  in_type : SET [0:?] OF data_composition_relationship FOR child;
  aggregation : SET [0:1] OF aggregation FOR aggregate_for;
 WHERE
  (* if a constant then the type of the constant may not be a composite data type *)
  (is_constant = FALSE) OR (NOT
('sedres_flat.configuration_management_information.model_configuration_management_information.
    vmcmi.composite_data_type_definition' IN TYPEOF(SELF.definition)));
 END_ENTITY;

 ENTITY data_type_definition
  ABSTRACT SUPERTYPE
  SUBTYPE OF (
   VMCMI
  );
  name : OPTIONAL label;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY composite_data_type_definition
  SUBTYPE OF (
   data_type_definition
  );
  composition_type : text;
 INVERSE
  composed_of : SET [1:?] OF data_composition_relationship FOR parent;
 END_ENTITY;

(* Check_integer_bounds:Every other element in the bounds list shall be an upper_bound and every
other shall
   be a lower_bound, and the bound values shall be monotonically increasing *)

 Function Check_integer_bounds (i : integer_type_definition) : BOOLEAN;
 LOCAL
    return_value : BOOLEAN := TRUE;
 END_LOCAL;
```

```
    IF HiIndex (i.bounds) < 2 THEN
        RETURN (TRUE);
    ELSE
        REPEAT index:=0 TO (HiIndex (i.bounds) -1) UNTIL return_value = false;
            IF i.bounds[index].bound_type = i.bounds[index+1].bound_type THEN
                return_value := false;
            END_IF;
            IF i.bounds[index].bound > i.bounds[index+1].bound THEN
                return_value := false;
            END_IF;
        END_REPEAT;
        RETURN (return_value);
    END_IF;
END_FUNCTION;

ENTITY integer_type_definition
 SUBTYPE OF (
  number_type_definition
 );
 SELF\number_type_definition.default_value : OPTIONAL INTEGER;
 SELF\number_type_definition.initial_value : OPTIONAL INTEGER;
 SELF\number_type_definition.resolution : OPTIONAL INTEGER;
 SELF\number_type_definition.accuracy : OPTIONAL INTEGER;
 bounds : LIST [0:?] OF integer_bound;
WHERE
 correct_integer_bounds : Check_integer_bounds (SELF);
END_ENTITY;

(* Check_real_bounds:Every other element in the bounds list shall be an upper_bound and every other
shall
   be a lower_bound, and the bound values shall be monotonically increasing *)

Function Check_real_bounds (i : real_type_definition) : BOOLEAN;
LOCAL
    return_value : BOOLEAN := TRUE;
END_LOCAL;
    IF HiIndex (i.bounds) < 2 THEN
        RETURN (TRUE);
    ELSE
        REPEAT index:=0 TO (HiIndex (i.bounds) -1) UNTIL return_value = false;
            IF i.bounds[index].bound_type = i.bounds[index+1].bound_type THEN
                return_value := false;
            END_IF;
            IF i.bounds[index].bound > i.bounds[index+1].bound THEN
                return_value := false;
            END_IF;
        END_REPEAT;
        RETURN (return_value);
    END_IF;
END_FUNCTION;

ENTITY real_type_definition
 SUBTYPE OF (
  number_type_definition
 );
 SELF\number_type_definition.default_value : OPTIONAL REAL;
 SELF\number_type_definition.initial_value : OPTIONAL REAL;
 SELF\number_type_definition.resolution : OPTIONAL REAL;
 SELF\number_type_definition.accuracy : OPTIONAL REAL;
 bounds : LIST [0:?] OF real_bound;
WHERE
 correct_real_bounds : Check_real_bounds (SELF);
END_ENTITY;

ENTITY string_type_definition
 SUBTYPE OF (
  data_type_definition
 );
 maximum_size : OPTIONAL INTEGER;
 initial_value : OPTIONAL text;
 default_value : OPTIONAL text;
WHERE
no_overflow : stringoverflowcheck(SELF);
END_ENTITY;

ENTITY data_composition_relationship;
 child : data_instance;
 parent : composite_data_type_definition;
 role : OPTIONAL text;
END_ENTITY;

ENTITY flow;
 destination_port : flow_port;
 source_port : flow_port;
 name : label;
 control_flow : BOOLEAN;
DERIVE
    data_on_flow : SET [1:?] OF data_instance := Derive_Flow_Data(SELF);
WHERE
 correct_ports : ((destination_port.role = consumer) AND (source_port.role = producer));
  (* The source_port and destination_port may not both be buffered *)
  (* no_double_buffer : (not (('SEDRES_flat.flow_port.actual_flow_port' IN TYPEOF (destination_port)
AND
                              (destination_port\actual_flow_port.buffered)) AND
                         (('SEDRES_flat.flow_port.actual_flow_port' IN TYPEOF (source_port)) AND
                              (source_port\actual_flow_port.buffered)))); *) --error on this line. type conflict in
where clause - I can't see why???
END_ENTITY;

FUNCTION Derive_Flow_Data (f : flow ) : SET [1:?] OF data_instance;
LOCAL
    dummy : SET [1:?] OF data_instance;
END_LOCAL;
 (* This function is not yet implemented *)
 (* Apply recursive traversal of data on source_port and destination_port respectively until all common
elements are derived *)
 (* Flows between flow_ports whose data have no common elements are illegal! *)
    RETURN (dummy); -- dummy return statement
END_FUNCTION;

ENTITY flow_group;
 name : label;
 elements : SET [2:?] OF flow;
END_ENTITY;

ENTITY event_definition
 SUBTYPE OF (
  data_type_definition
 );
END_ENTITY;
```

```
ENTITY state_instance
 SUBTYPE OF (
  NVMCMI
 );
  definition : state_definition;
  name : label;
INVERSE
 in_context : state_composition_relationship FOR child_state;
 name_ports : SET [0:?] OF actual_name_port FOR port_of;

END_ENTITY;

ENTITY state_definition
 ABSTRACT SUPERTYPE OF ( ONEOF(leaf_state_definition,composite_state_definition) )
 SUBTYPE OF (
  VMCMI
 );
 name : OPTIONAL label;
 description : OPTIONAL text_select;
INVERSE
 name_ports : SET [0:?] OF formal_name_port FOR port_of;
END_ENTITY;

ENTITY leaf_state_definition
 SUBTYPE OF (
  state_definition
 );
END_ENTITY;


ENTITY composite_state_definition
 ABSTRACT SUPERTYPE OF ( ONEOF(and_state_definition,or_state_definition) )
 SUBTYPE OF (
  state_definition
 );
INVERSE
 root_for : SET [0:1] OF fsm_model FOR definition;
 child_instances : SET [1:?] OF state_composition_relationship FOR parent_state;
 associated_transition_ports : SET [0:?] OF formal_transition_port FOR port_of;
 transitions : SET [1:?] OF transition_composition_relationship FOR state_definition;
END_ENTITY;

ENTITY and_state_definition
 SUBTYPE OF (
  composite_state_definition
 );
END_ENTITY;

ENTITY or_state_definition
 SUBTYPE OF (
  composite_state_definition
 );
END_ENTITY;

ENTITY fsm_model
 SUBTYPE OF (
  NVMCMI
 );
 name : label;
```

```
  definition : composite_state_definition;
 INVERSE
  function_references : SET [0:?] OF function_reference FOR port_of;
 WHERE
  no_interface : (HiIndex (SELF.definition.associated_transition_ports) = 0);
 END_ENTITY;

 ENTITY transition_port
  ABSTRACT SUPERTYPE;
  direction : port_direction;
 END_ENTITY;

(*  involving_state_instance : a transition must be connected to at least one actual_transition_port
    transitions between formal port does not make sense *)
(* CorrectTransition : A transition shall be produced by a producer port and consumed
by a consumer port *)

 ENTITY transition
  SUBTYPE OF (
   general_transition
  );
  source_port : transition_port;
 INVERSE
  transition_expression : SET [0:1] OF transition_expression FOR transition;
 WHERE
  involving_state_instance : (TYPEOF (SELF.source_port) <> TYPEOF (SELF.destination_port))
     OR
     ('sedres_flat.transition_port.actual_transition_port' IN TYPEOF (SELF.destination_port));
  (*correcttransition : ((source_port.role = producer) AND (destination_port.role =
                  consumer));*)
  (* The source and destination port shall belong to the same state_definition *)
  (*same_context : source_port.port_of.in_context.parent_state :=:
destination_port.port_of.in_context.parent_state;*)
 END_ENTITY;

 ENTITY action_instance;
  action_for : action_association_select;
  language : label;
  definition : text_select;
 END_ENTITY;

 ENTITY state_composition_relationship;
  child_state : state_instance;
  parent_state : composite_state_definition;
 END_ENTITY;

 ENTITY flow_port
  ABSTRACT SUPERTYPE OF ( ONEOF(formal_flow_port,actual_flow_port,control_port) );
  direction : port_direction;
  data : data_instance;
 DERIVE (* Dummy derive to avoid strange model for control_ports *)
  role : port_data_relation := producer;
 END_ENTITY;

 ENTITY control_port
  SUBTYPE OF (
   flow_port
  );
  port_of : function_instance;
  offset : REAL;
 DERIVE
  SELF\flow_port.role : port_data_relation := consumer;
 INVERSE
  triggered_by_clock : SET [0:1] OF clock FOR trigger_for;
 WHERE
  (* The data types of a control port may only be numeric, boolean or event *)
  good_data_types:
('SEDRES_flat.configuration_management_information.model_configuration_management_informatio
n.
                         vmcmi.data_type_definition.event_definition' IN TYPEOF (data.definition)) OR
     ('SEDRES_flat.configuration_management_information.model_configuration_management_inform
ation.vmcmi.data_type_definition.boolean_type_definition' IN TYPEOF (data.definition)) OR
     ('SEDRES_flat.configuration_management_information.model_configuration_management_inform
ation.vmcmi.data_type_definition.number_type_definition' IN TYPEOF (data.definition));

  (* Control ports are always input ports *)
  port_data_direction : (SELF\flow_port.direction = input);
  good_offset: offset >= 0.0;
 END_ENTITY;

(* Correct_reference : The actual_port must be connected to an instance which has as its
definition the definition object the formal_port is a port_of.
   Correct_direction : If the formal port has as its role the produce data then the actual
   port must consume data and vice versa. Since there are only two possible values for role
   it is sufficient to test for inequality *)


 ENTITY flow_port_binding;
  formal_port : formal_flow_port;
  actual_port : actual_flow_port;
 WHERE
correct_reference : formal_port.port_of :=: actual_port.port_of.definition;
correct_port_type : (SELF.formal_port.data :<>: SELF.actual_port.data) AND
                         (SELF.formal_port.data.definition :=: SELF.actual_port.data.definition);
correct_direction : formal_port.role <> actual_port.role;
 END_ENTITY;


ENTITY action_precedence_relationship;
  relating : action_instance;
  related : action_instance;
  relationship_type : relation_type;
 WHERE
(* NotCircular : the relating and related function_instance must not be the same instance *)
notcircular : relating :<>: related;
 END_ENTITY;

 ENTITY causal_relationship;
  causing : function_instance;
  caused : function_instance;
  chain : causal_chain;
  relation_type : label;
  description : OPTIONAL text_select;
 WHERE
 (* NotCircular : the causing and caused function_instance must not be the same instance *)
  notcircular : causing :<>: caused;
 END_ENTITY;

 ENTITY external_document
  SUBTYPE OF (
   documentation_reference
  );
  location : text;
 END_ENTITY;

 ENTITY causal_chain
  SUBTYPE OF (
   VMCMI
  );
  name : label;
  description : text_select;
 INVERSE
  chain_elements : SET [1:?] OF causal_relationship FOR chain;
 END_ENTITY;

 ENTITY formal_flow_port
  SUBTYPE OF (
   flow_port
  );
  port_of : general_function_definition;
 DERIVE
  SELF\flow_port.role : port_data_relation := determineformalportrole(SELF);
 END_ENTITY;

 ENTITY actual_flow_port
  SUBTYPE OF (
   flow_port
  );
(* Derived role attribute added to formal and actual transition port *)

  buffered : BOOLEAN; (* A buffered input port logs all incoming values, a buffered output
                               port make sure output values are only read once *)
  port_of : port_usage_select;
 DERIVE
  SELF\flow_port.role : port_data_relation := determineactualportrole(SELF);
 END_ENTITY;

 ENTITY formal_transition_port
  SUBTYPE OF (
   transition_port
  );
  port_of : composite_state_definition;
 DERIVE
(* DetermineFormalPortRole defines whether the formal_transition_port is a consumer
or producer of transitions *)

  role : port_data_relation := determineformaltransitionportrole(SELF);
 END_ENTITY;

 ENTITY actual_transition_port
  SUBTYPE OF (
   transition_port
  );
  port_of : state_instance;
 DERIVE
(* DetermineActualPortRole defines whether the actual_transition_port is a consumer
or producer of transitions *)
  role : port_data_relation := determineactualtransitionportrole(SELF);
 END_ENTITY;
```

```
 ENTITY transition_relationship;
  relating : transition;
  related : transition;
  role : label;
 WHERE
(* The relating and related transition must not be the same transition object *)
notcircular : relating :<>: related;
 END_ENTITY;

 ENTITY justification
  SUBTYPE OF (
   Process_Configuration_Management_Item
  );
  applies_to : Model_Configuration_Management_Item;
  role : text_select;
  justification_text : text_select;
 END_ENTITY;

 ENTITY justification_relationship;
  relating : justification;
  related : justification;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY transition_expression;
  transition : transition;
  language : label;
  definition : text;
 WHERE
  (* A transition expression may only be associated with the first segment of a state
     transition spanning multiple levels in a state hierarchy *)
  correct_expression : 'sedres_flat.transition_port.actual_transition_port' IN TYPEOF
(transition.source_port);
 END_ENTITY;

 ENTITY general_function_definition
  ABSTRACT SUPERTYPE OF ( ONEOF(composite_function_definition,leaf_function_definition) )
  SUBTYPE OF (
   VMCMI
  );
  name : OPTIONAL label;
  description : OPTIONAL text_select;
 INVERSE
  formal_ports : SET [0:?] OF formal_flow_port FOR port_of;
  function_timing : SET [0:3] OF execution_time FOR timing;
 WHERE
  no_duplicates : checkduplicate(SELF);
 END_ENTITY;

 ENTITY leaf_function_definition
  SUBTYPE OF (
   general_function_definition
  );
  specification_language : label;
  specification : text_select;
  function_type : label;
  predefined : BOOLEAN;
 END_ENTITY;

 ENTITY enumeration_type_definition
  SUBTYPE OF (
   data_type_definition
  );
  enumeration_elements : SET [2:?] OF label;
  initial_value : OPTIONAL label;
  default_value : OPTIONAL label;
 END_ENTITY;

 ENTITY unit;
  associated_with : number_type_definition;
  unit_name : label;
 END_ENTITY;

 ENTITY number_type_definition
  SUBTYPE OF (
   data_type_definition
  );
  default_value : OPTIONAL NUMBER;
  initial_value : OPTIONAL NUMBER;
  resolution : OPTIONAL NUMBER;
  accuracy : OPTIONAL NUMBER;
 INVERSE
  unit : SET [0:1] OF unit FOR associated_with;
 END_ENTITY;

 ENTITY date_assignment;
  date : date;
  item : Configuration_Management_Item;
  role : label;
 END_ENTITY;

 ENTITY integer_bound;
  bound_type : bound_type;
  bound : INTEGER;
 END_ENTITY;

 ENTITY real_bound;
  bound_type : bound_type;
  bound : REAL;
 END_ENTITY;

 ENTITY boolean_type_definition
  SUBTYPE OF (
   data_type_definition
  );
  default_value : OPTIONAL BOOLEAN;
  initial_value : OPTIONAL BOOLEAN;
 END_ENTITY;

(* In the model there are three possible timing attributes {worst_case, best_case,
nominal_case}. Only one of each type can be assigned to a general_function_definition.
The relation between the timing types are the following {worst_case >= nominal_case >=
best_case} This is very difficult to test in the current model. *)


 ENTITY execution_time;
  timing : general_function_definition;
  time : REAL;
  unit : label;
  role : timing_type;
 END_ENTITY;

ENTITY clock;
  control_signal : data_instance;
  trigger_for : control_port;
  name : OPTIONAL label;
  description : OPTIONAL text;
  period : REAL;
 WHERE
  (* a clock can only generate events *)
  correct_data_definition :
'SEDRES_flat.configuration_management_information.model_configuration_management_information
.vmcmi.data_type_definition.event_definition' IN TYPEOF(control_signal.definition);
 END_ENTITY;

 ENTITY null_type_definition
  SUBTYPE OF (
   data_type_definition
  );
 END_ENTITY;

 ENTITY requirement_allocation_relationship;
  requirement : requirement_instance;
  relation_to : requirement_allocation_select;
  role : label;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY transition_port_binding;
  actual_port : actual_transition_port;
  formal_port : formal_transition_port;
  WHERE
correct_reference : formal_port.port_of :=: actual_port.port_of.definition;
correct_direction : formal_port.role <> actual_port.role;

 END_ENTITY;

 ENTITY VMCMI
  ABSTRACT SUPERTYPE OF ( ONEOF(system_view,
documentation_reference,requirement_definition,function_context_definition,general_function_definitio
n,data_type_definition,state_definition, causal_chain) )
  SUBTYPE OF (
   model_configuration_management_item
  );
  associated_version : OPTIONAL element_version;
 END_ENTITY;

 ENTITY NVMCMI
  ABSTRACT SUPERTYPE OF (
ONEOF(requirement_instance,function_instance,store,state_instance,fsm_model, external_entity) )
  SUBTYPE OF (
   model_configuration_management_item
  );
 END_ENTITY;

ENTITY application
  SUBTYPE OF (
   system_view
  );
 INVERSE
  operational_scenarios : SET [0:?] OF operational_scenario FOR scenario_of;
 END_ENTITY;

 ENTITY system_view
  ABSTRACT SUPERTYPE OF ( ONEOF(operational_scenario,application) )
 SUBTYPE OF (
   VMCMI
  );
  name : OPTIONAL label;
 INVERSE
  context_definition : SET [0:1] OF context_system_view_relationship FOR system_view;
 END_ENTITY;

  ENTITY general_transition
  ABSTRACT SUPERTYPE OF ( ONEOF(default_transition,transition) );
  destination_port : transition_port;
 INVERSE
  in_context : SET [1:1] OF transition_composition_relationship FOR transition;
 WHERE
  same_context : Check_Transition_destination (SELF);
 END_ENTITY;

(* Check_Transition_destination : check that the transition and destination state
   is part of the same Composite_state_definition object, either as child objects
   or the destination state is the Composite_state_definition object itself.  *)

 FUNCTION Check_Transition_destination (tr : general_transition ) : BOOLEAN;
  (* This function is not yet implemented *)
     RETURN (TRUE);
 END_FUNCTION;


 ENTITY default_transition
  SUBTYPE OF (
   general_transition
  );
 END_ENTITY;

 ENTITY transition_composition_relationship;
  state_definition : composite_state_definition;
  transition : general_transition;
 END_ENTITY;

  ENTITY graphics_node
  SUBTYPE OF (
   visual_element
  );
  associated_with : node_select;
  top_left : graphics_point;
  bottom_right : graphics_point;
 END_ENTITY;

 ENTITY graphics_point;
  x_coordinate : REAL;
  y_coordinate : REAL;
 END_ENTITY;

 ENTITY graphics_link
  SUBTYPE OF (
   visual_element
  );
  associated_with : link_select;
  has_points : LIST [2:?] OF graphics_point;
 END_ENTITY;

 ENTITY actual_port_position
  SUBTYPE OF (
   visual_element
  );
  positioned_port : port_position_select;
  position : graphics_point;
  position_in_decomposition : node_select;
 END_ENTITY;

 ENTITY formal_port_position
  SUBTYPE OF (
   visual_element
  );
  position : graphics_point;
  positioned_port : port_position_select;
 END_ENTITY;

 ENTITY view_relationship;
  parent : graphics_view;
  child : graphics_view;
  valid_in : multi_level_view;
 END_ENTITY;

 ENTITY graphics_view;
  definition_for : definition_select;
 INVERSE
  view_elements : SET [1:?] OF visual_element FOR view;
  root_view : SET [0:1] OF multi_level_view FOR top_view;
 END_ENTITY;

 ENTITY visual_element
  ABSTRACT SUPERTYPE OF (
ONEOF(graphics_node,actual_port_position,formal_port_position,graphics_link) );
  view : graphics_view;
 END_ENTITY;

 ENTITY multi_level_view;
  top_view : graphics_view;
  name : OPTIONAL label;
  description : OPTIONAL text_select;
 END_ENTITY;

 ENTITY flow_composition_relationship;
  flow_component : flow_element_select;
  flow_context : flow_context_select;
 END_ENTITY;

 ENTITY actual_name_port;
  port_of : state_instance;
  reference_port : formal_name_port;
 END_ENTITY;

 ENTITY formal_name_port;
  port_of : state_definition;
 INVERSE
```

```
  referenced_by : SET [0:?] OF actual_name_port FOR reference_port;
 END_ENTITY;

 ENTITY name_binding;
  formal_port : formal_name_port;
  actual_port : name_binding_select;
 END_ENTITY;

 ENTITY function_reference;
  function_link : function_instance;
  port_of : fsm_model;
 END_ENTITY;

 ENTITY action_link;
  element : formal_name_port;
  command : action_function_operation_type;
  associated_action : action_instance;
 END_ENTITY;

 ENTITY aggregation;
  aggregate_for : data_instance;
  aggregation_list : LIST [1:?] OF INTEGER;
 WHERE
  correct_aggregate : correct_aggregate_bound (SELF);
 END_ENTITY;

 FUNCTION correct_aggregate_bound (element : aggregation) : BOOLEAN;
  LOCAL
     return_value : BOOLEAN := TRUE;
 END_LOCAL;
     REPEAT index:=1 TO (HiIndex (element.aggregation_list) ) UNTIL return_value = false;
          return_value := (element.aggregation_list[index] >= 0);
     END_REPEAT;
     RETURN (return_value);
 END_FUNCTION;


(* FUNCTION StringOverflowCheck makes sure the initial and default attributes of a
string_type_definition is of equal length or shorter than the maximum_size attribute
(if supplied)*)

 FUNCTION stringoverflowcheck(s : string_type_definition) : BOOLEAN;
  IF EXISTS(s.maximum_size)
   THEN
    IF EXISTS(s.initial_value)
     THEN
      RETURN (LENGTH(s.initial_value) <= s.maximum_size);
    END_IF;
    IF EXISTS(s.default_value)
     THEN
      RETURN (LENGTH(s.default_value) <= s.maximum_size);
    END_IF;
   ELSE
    RETURN (TRUE);
  END_IF;
 END_FUNCTION;
(* no_overflow : if a maximum_size has been defined for the string the initial and
default value must not exceed the maximum_size *)
```

(* CheckFlowPortType : If we have a binding involving flow_ports the type of the instance associated with the formal port shall be the same as the type of the instance associated with the actual port, and the same instance shall not appear on both ports *)

```
(*
 FUNCTION checkflowporttype(b : port_binding) : BOOLEAN;
  IF 'sedres_flat.flow_port' IN TYPEOF(b.formal_port)
   THEN
    IF (b.formal_port.data :<>: b.actual_port.data) AND (b.formal_port.data.
       definition :=: b.actual_port.data.definition)
     THEN
      RETURN (TRUE);
     ELSE
      RETURN (FALSE);
    END_IF;
-- if else branch is taken it was not a flow port.
   ELSE
    RETURN (TRUE);
  END_IF;
 END_FUNCTION;
*)

 FUNCTION determineformalportrole(p : formal_flow_port) : port_data_relation;
  IF p.direction = input
   THEN
    RETURN (producer);
   ELSE
    RETURN (consumer);
  END_IF;
 END_FUNCTION;


 FUNCTION determineactualportrole(p : actual_flow_port) : port_data_relation;
  IF p.direction = input
   THEN
    RETURN (consumer);
   ELSE
    RETURN (producer);
  END_IF;
 END_FUNCTION;


 FUNCTION determineformaltransitionportrole(p : formal_transition_port) :
                                          port_data_relation;
  IF p.direction = input
   THEN
    RETURN (producer);
   ELSE
    RETURN (consumer);
  END_IF;
 END_FUNCTION;


 FUNCTION determineactualtransitionportrole(p : actual_transition_port) :
                                          port_data_relation;
  IF p.direction = input
   THEN
    RETURN (consumer);
   ELSE
    RETURN (producer);
  END_IF;
 END_FUNCTION;


 FUNCTION checkduplicate(f : general_function_definition) : BOOLEAN;
  LOCAL
   check1,check2,check3 : BOOLEAN;
  END_LOCAL;
  IF SIZEOF(f.function_timing) < 2
   THEN
    RETURN (TRUE);
--  2 or more timings
   ELSE
    IF SIZEOF(f.function_timing) = 2
     THEN
      RETURN (f.function_timing[1].role = f.function_timing[2].role);
-- 3 timings, checkX will be true if the role of two is equal
     ELSE
      check1 := (f.function_timing[1].role = f.function_timing[2].role);
      check2 := (f.function_timing[1].role = f.function_timing[3].role);
      check3 := (f.function_timing[2].role = f.function_timing[3].role);
      RETURN (check1 OR check2 OR check3);
    END_IF;
  END_IF;
 END_FUNCTION;

END_SCHEMA;
```