



TECHNICAL REPORT	EIA-IS-115™	
	Issued 1995-12 Stabilized 2015-07	
CDIF Integrated Meta-model Data Flow Model Subject Area		

NOTICE

This document has been taken directly from the original TechAmerica document and contains only minor editorial and format changes required to bring it into conformance with the publishing requirements of SAE Technical Standards. The release of this document is intended to replace the original with the SAE International document. Any numbers established by the original document remain unchanged.

The original document was adopted as an SAE publication under the provisions of the SAE Technical Standards Board (TSB) Rules and Regulations (TSB 001) pertaining to accelerated adoption of specifications and standards. TSB rules provide for (a) the publication of portions of unrevised specifications and standards without consensus voting at the SAE committee level, and (b) the use of the existing specification or standard format.

SAE Technical Standards Board Rules provide that: "This report is published by SAE to advance the state of technical and engineering sciences. The use of this report is entirely voluntary, and its applicability and suitability for any particular use, including any patent infringement arising therefrom, is the sole responsibility of the user."

SAE reviews each technical report at least every five years at which time it may be revised, reaffirmed, stabilized, or cancelled. SAE invites your written comments and suggestions.

Copyright © 2016 SAE International

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted, in any form or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior written permission of SAE.

TO PLACE A DOCUMENT ORDER: Tel: 877-606-7323 (inside USA and Canada)
Tel: +1 724-776-4970 (outside USA)
Fax: 724-776-0790
Email: CustomerService@sae.org
<http://www.sae.org>

SAE WEB ADDRESS:

Full document details: [SAE EIAIS115](#)

SAE values your input. To provide feedback on this Technical Report, please visit <http://www.sae.org/technical/standards/EIAIS115>

Documents
Document Sharing Hub

Interim Standard

EIA/IS-115

CDIF

CASE Data Interchange Format
A Division of Electronic Industries Association

**CDIF Integrated Meta-model
Data Flow Model
Subject Area**

December 1995

**Electronic Industries Association
CDIF Technical Committee**



Documents

Document Sharing Hub

NOTICE

EIA Engineering standards and publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchaser in selecting and obtaining with minimum delay the proper product for his particular need. Existence of such standards and publications shall not in any respect preclude any member or nonmember of EIA from manufacturing or selling products not conforming to such standards and publications, nor shall the existence of such standards and publications preclude their voluntary use by those other than EIA members, whether the standard is to be used either domestically or internationally.

Recommended standards and publications are adopted by EIA without regard to whether or not their adoption may involve patents on articles, materials or processes. By such action, EIA does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the standard or publication.

EIA INTERIM STANDARDS

EIA Interim Standards contain information deemed to be of technical value to the industry, and are published at the request of the originating Committee without necessarily following the rigorous public review and resolution of comments which is a procedural part of the development of an EIA Recommended Standard.

EIA Interim Standards should be reviewed on an annual basis by the formulating Committee and a decision made on whether to proceed to develop an EIA Recommended Standard on this subject. EIA Interim Standards must be canceled by the Committee and removed from the EIA Standards Catalog before the end of their third year of existence.

This EIA standard is considered to have international standardization implication, but the International Standardization Organization activity has not progressed to the point where a valid comparison between the EIA standard and the ISO work can be made.

COPYRIGHT 1995

Published by
ELECTRONIC INDUSTRIES ASSOCIATION
Engineering Department
2500 Wilson Blvd.
Arlington, VA 22201

PRICE: Please refer to the current
Catalog of EIA & JEDEC STANDARDS & ENGINEERING PUBLICATIONS
or contact EIA Electronic Information Group
(Phone: (703) 907-2545, FAX: (703) 907-7501)

All rights reserved
Printed in U.S.A.

ISBN 0-7908-0021-7

PREFACE

This standard will assist the vendors and users of CASE tools in developing mechanisms for interchanging information between CASE tools. This standard specifies an element of a family of related standards. When used together, these standards specify a mechanism for transferring information between CASE tools.

EIA/IS-106 CDIF - CASE Data Interchange Format - Overview and EIA/IS-107 CDIF - Framework for Modeling and Extensibility should be read first when initially exploring CDIF. The first explains the overall CDIF Architecture and how the family of standards fits together. The second explains the scope, and modeling approach in CDIF. The CDIF Meta-meta-model and extensibility mechanism are also defined in that document.

This standard has been developed with the wide support and participation of vendors, users, academia and government involved in or familiar with the CASE industry, its products and the general requirements associated with interchanging information between these products.

The material contained in this publication was formulated under the cognizance of the CDIF Technical Committee.

CONTENTS

CONTENTS

1. SCOPE.....	1
2. INTRODUCTION.....	3
2.1 Audience	3
2.2 Structure of this Document	3
2.3 Conventions	5
3. STRUCTURE OF SUBJECT AREA DEFINITION.....	6
3.1 Introduction.....	6
3.2 Structure of Subject Area Overview.....	6
3.2.1 Summary	6
3.2.2 Description of Major Concepts	6
3.2.3 Diagram(s)	6
3.2.4 AttributableMetaObject Hierarchy	9
3.2.5 MetaEntity and MetaRelationship Summaries	9
3.3 Structure of Subject Area Detailed Definitions.....	10
3.3.1 Summary	10
3.3.2 Subject Area Definition	10
3.3.3 MetaEntity Definitions.....	10
3.3.4 MetaRelationship Definitions	12
3.4 Structure of Referenced Meta-object Definitions.....	13
4. SUBJECT AREA OVERVIEW	15
4.1 Description.....	15
4.1.1 Introduction	15
4.1.2 Data Flow Model	15
4.1.3 Processes.....	16
4.1.4 Flows	16
4.1.5 External Agents	17
4.1.6 Stores.....	17
4.1.7 The General Structuring Mechanism.....	17
4.1.7.1 Introduction	17
4.1.7.2 Meta-entities and Meta-relationships.....	18
4.1.7.3 DefinitionObject and ComponentObject.....	18
4.1.8 Process Decomposition and Definition.....	20
4.1.9 Flow Decomposition and Definition.....	21
4.1.10 Store Decomposition.....	22
4.1.11 Context Identifiers	23
4.1.12 Accessing a Specific Component in a Shared Definition	23
4.1.13 Identical Instances in Shared Definitions	25
4.1.14 Alternate Decompositions.....	27

4.1.15	Flow Splits and Merges	28
4.1.16	Interface of Processes, Stores and External Agents	32
4.1.17	Controls, Constraints and Mechanisms	36
4.1.18	Real-Time Extensions	36
4.1.19	Computable Languages	36
4.1.20	Formats for Computable Values	37
4.2	Diagrams	38
4.3	AttributableMetaObject Hierarchy	39
4.4	MetaEntity Summary	41
4.5	MetaRelationship Summary	50
5.	SUBJECT AREA DETAILED DEFINITIONS	52
5.1	Introduction	52
5.2	Subject Area Definition	53
5.3	MetaEntity Definitions	54
5.3.1	Attribute	54
5.3.2	ComponentObject	57
5.3.3	ConstraintPort	59
5.3.4	ControlPort	61
5.3.5	DataFlowModel	63
5.3.6	DefinitionObject	65
5.3.7	DFMProcess	68
5.3.8	DFMProcessDefinition	71
5.3.9	EquivalenceSet	76
5.3.10	ExternalAgent	78
5.3.11	ExternalAgentDefinition	81
5.3.12	Flow	83
5.3.13	FlowDefinition	86
5.3.14	FlowInputPort	89
5.3.15	FlowOutputPort	91
5.3.16	FlowPort	93
5.3.17	FlowProducerConsumer	95
5.3.18	Port	97
5.3.19	ReferencedElement	99
5.3.20	Store	101
5.3.21	StoreDefinition	104
5.3.22	SupportPort	109
5.4	Meta-relationship Definitions	111
5.4.1	ComponentObject.References.DefinitionObject	111
5.4.2	DataFlowModel.HasRoot.DFMProcessDefinition	112
5.4.3	DefinitionObject.Contains.ComponentObject	113
5.4.4	EquivalenceSet.HasMember.ComponentObject	115
5.4.5	FlowProducerConsumer.Consumes.Flow	116
5.4.6	FlowProducerConsumer.Produces.Flow	117
5.4.7	FlowProducerConsumer.ProducesOrConsumes.Flow	118
5.4.8	ReferencedElement.DefinesPath.ComponentObject	119

6. REFERENCED OBJECT DEFINITIONS	121
6.1 AttributableMetaObject Definitions.....	121
6.1.1 RootObject.....	121
6.2 MetaEntity Definitions.....	125
6.2.1 RootEntity.....	125
6.2.2 SemanticInformationObject	127
6.3 Meta-relationship Definitions	129
6.3.1 RootEntity.IsRelatedTo.RootEntity	129
APPENDIX A EXAMPLES	131
A1. Introduction.....	131
A2. Details of Example.....	132
A3. CDIF Objects for Example.....	136
A3.1 Data Flow Model	136
A3.2 Top Level Diagram.....	136
A3.3 Definition of Flow “Withdrawal Amount”	142
A3.4 Definition of Flow “Message”	143
A3.5 Decomposition of Flow “Message”	144
A3.6 The “User” Definition.....	149
A3.7 The “Withdrawal” Definition	156
A3.8 Definition of “Check and Update Balance”	166
A3.9 Definition of “Dispense Cash”	173
A3.10 Definition of “Account Balance”	178
A3.11 Binding of DFMPProcess “Withdrawal” to DFMPProcessDefinition “Withdrawal Definition”	180
APPENDIX B QUESTIONS AND ANSWERS.....	187
B.1 Introduction.....	187
B.2 General.....	187
GLOSSARY	195
INDEX.....	201
RELATED EIA STANDARDS	INSIDE BACK COVER

FIGURES

Figure 1	Position in the CDIF Family of Standards	1
Figure 2	Conventions for Representing Meta-entities	7
Figure 3	Conventions for Representing Meta-relationships	7
Figure 4	Conceptual Diagram and an Instance Diagram.....	8
Figure 5	Process Decomposition and hierarchy of data flow models	16
Figure 6	General Structuring Mechanism	18
Figure 7	Flows sharing the same structured definition	19
Figure 8	A process decomposition	21
Figure 9	Structured Flows.....	22
Figure 10	Shared definition for structured flows.....	24
Figure 11	Referring to a specific instance inside a shared definition	24
Figure 12	Denoting the equivalence of two Stores.....	26
Figure 13	Alternate Flow decompositions	28
Figure 14	A Flow Split into component Flows	29
Figure 15	A Flow Split.....	29
Figure 16	Instance diagram for splitting flows into component flows.....	30
Figure 17	Instance diagram for a flow split.....	31
Figure 18	Instance diagram for a bidirectional flow splitting into directed flows	31
Figure 19	Shared <i>DFMProcessDefinition</i> flow bindings	32
Figure 20	Interface of a process referencing a reusable definition	33
Figure 21	<i>FlowPorts</i> describing the formal interface of a <i>DFMProcessDefinition</i>	34
Figure 22	Interface of a Process using <i>FlowPorts</i>	35
Figure 23	Data Flow Model Subject Area - Diagram.....	38
Figure A1	Data Flow Model Example - Upper-level diagram	132
Figure A2	Data Flow Model Example - Lower-level diagram.....	132
Figure A3	Data Flow Model Example - Upper-level Instance Detail.....	133
Figure A4	Data Flow Model Example - Lower-level Instance Detail	134
Figure A5	Data Flow Model Example - Binding Instance Detail.....	135
Figure A6	Data Flow Model Example - Upper-level Instance diagram.....	148
Figure A7	Data Flow Model Example - User Instance diagram	155
Figure B1	Single Subtype.....	188
Figure B2	Instance diagram for an IDEF0 function with a Call arrow to several alternative functions.....	190
Figure B3	ExternalAgent equivalence example.....	192
Figure B4	Instance diagram showing ExternalAgent equivalence example ..	193

TABLES

Table 1	Contents of Subject Area Definition	10
Table 2	Contents of Meta-entity Definition	11
Table 3	Contents of Local Meta-attribute Definition.....	12
Table 4	Contents of Meta-relationship Definition	13
Table 5	Allowable DefinitionObject components	20

1. SCOPE

The CDIF Family of Standards is primarily designed to be used as a description of a mechanism for transferring information between CASE tools. It facilitates a successful transfer when the authors of the importing and exporting tools have nothing in common except an agreement to conform to CDIF. The language that is defined for the Transfer Format also has applicability as a general language for Import/Export from repositories. The CDIF Integrated Meta-model defined for CASE also has applicability as the basis of standard definitions for use in repositories.

The standards that form the complete family of CDIF Standards are documented in *EIA/IS-106 CDIF - CASE Data Interchange Format - Overview*. These standards cover the overall framework, the transfer format and the CDIF Integrated Meta-model.

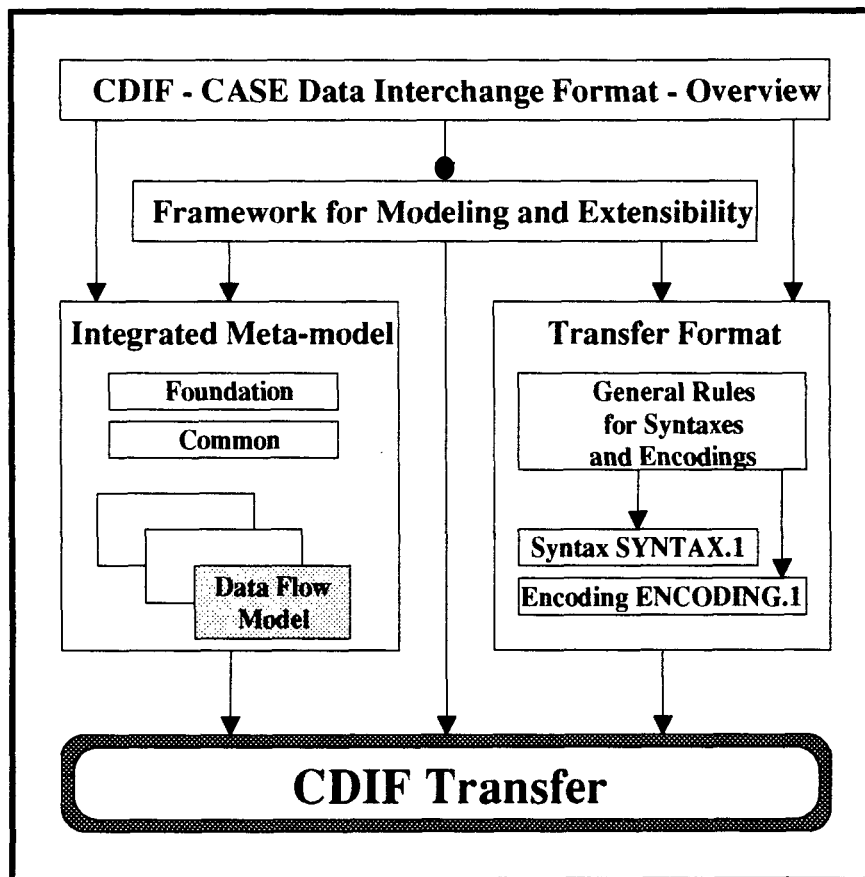


Figure 1
Position in the CDIF Family of Standards

The diagram in Figure 1 depicts the various standards that comprise the CDIF Family of Standards. The shaded box depicts this Standard and its position in the CDIF Family of Standards.

This standard describes the Data Flow Model subject area of the CDIF Integrated Meta-model. This provides support for the expressions of information about Data Flow Modeling techniques.

2. INTRODUCTION

2.1 Audience

This document is intended to be used by anyone wishing to understand and/or use CDIF. This document provides a definition of a single subject area of the CDIF Integrated Meta-model. It is suitable for:

- those evaluating CDIF
- those who wish to understand the principles and concepts of a CDIF transfer
- those developing importers and exporters.

The document *EIA/IS-106 CDIF - CASE Data Interchange Format - Overview* and the document *EIA/IS-107 CDIF - Framework for Modeling and Extensibility* should be read first when initially exploring CDIF and before attempting to read other documents in the CDIF Family of Standards.

While there are no specific prerequisites for reading this document, it will be helpful for the reader to have familiarity with the following:

- Entity-Relationship-Attribute modeling
- CASE tools
- Information repositories
- Data dictionaries
- Multiple meta-layer modeling.

2.2 Structure of this Document

This document is organized into the following sections:

Section 1 'Scope'

This describes the CDIF Family of Standards and positions the current document within the set of documents.

Section 2 'Introduction'

This describes the intended audience, structure, and conventions used in the document.

Section 3 'Structure of Subject Area Definition'

This describes in detail the structure of the subsequent sections of this standard including the notation and layout of the diagrams and the detailed definitions.

Section 4 'Subject Area Overview'

This gives an overview of the coverage of this subject area.

Section 5 'Subject Area Detailed Definitions'

This gives the detailed definitions of all the objects used in the subject area.

Section 6 'Referenced Meta-object Definitions'

This gives the detailed definitions of objects that are not used in the subject area defined in this Standard, but are referenced by objects that are. These definitions are included to avoid the need to reference multiple standards to obtain the complete definition of the current subject area.

Appendix A 'Examples'

This gives one or more examples of how information is represented using this CDIF Subject Area. Where there are different technique variants used within the coverage of a single Subject Area, the examples attempt to show how some of these different variants are represented in the CDIF Integrated Meta-model.

Appendix B 'Questions and Answers'

This includes detailed responses to questions that have been raised by those reviewing this document. It is intended that the inclusion of these will aid the reader in understanding this document, and related issues concerning CDIF.

Glossary

The Glossary contains definitions of terms used within this Standard.

Index

A comprehensive index is provided.

2.3 Conventions

The typographical and naming conventions defined below are used throughout the CDIF Family of Standards.

- Double quotes are used to introduce new terms (e.g., "model layer")
- Bold type is used for emphasis (e.g., this is **important**)
- All meta-objects and meta-meta-objects in CDIF (in meta-models and meta-meta-models) are named by concatenating all the words that name the meta-object or meta-meta-object; the first letter of each word is upper-case, the rest are lower-case. These names are shown in italic (e.g., *MetaAttribute*, *AttributeDerivation*, *IsDrawnUsing*, *IsOptional*)
- The names of objects and values for meta-attributes that appear in examples and instance diagrams are shown in bold italic (e.g., ***Customer***, ***True***, ***100***)

Full details of the CDIF Graphical Notation used in the Meta-model and Meta-meta-model can be found in *EIA/IS-107 CDIF - Framework for Modeling and Extensibility*. A summary of the major concepts is given in Section 3.2.3, Diagram(s).

3. STRUCTURE OF SUBJECT AREA DEFINITION

3.1 Introduction

The following subsections describe how the subject area is defined in the remainder of this document. The subject area is first introduced in the Subject Area Overview Section, and the detailed definitions of the objects contained in the subject area are given in the Subject Area Definitions Section.

3.2 Structure of Subject Area Overview

3.2.1 Summary

The Subject Area Overview Section consists of the following sub-sections:

- Description of Major Concepts
- Diagram(s)
- AttributableMetaObject Hierarchy
- MetaEntity Summary
- MetaRelationship Summary

3.2.2 Description of Major Concepts

This provides a textual description of the major concepts in the subject area. It explains the coverage of this subject area and outlines what it is capable of representing.

3.2.3 Diagram(s)

Included in the Subject Area Overview are several diagrams. One of these diagrams (found in section 4.2, Diagrams) presents graphically an overview of the meta-entities and meta-relationships used in this subject area. All CDIF subject area diagrams use the following diagramming notation conventions.

Meta-entities used in the subject area are shown as rectangular boxes, with the name of the meta-entity on a white background. An example is the meta-entity *A* in Figure 2. An additional special notation may be used to represent meta-entities on diagrams of meta-models. If the meta-entity is not used in the subject area, but is used in another subject area and is included for clarity, the box shall be shaded. An example is the meta-entity *B* in Figure 2.

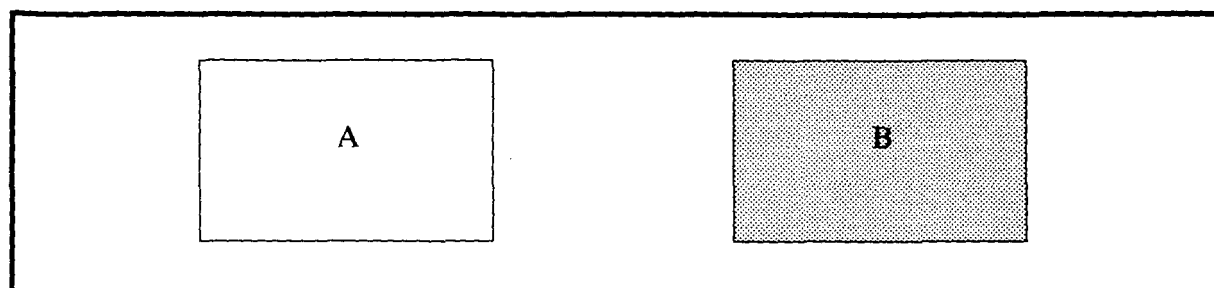


Figure 2
Conventions for Representing Meta-entities

Meta-relationships are normally represented by a line joining the source and destination entities, with the name of the meta-relationship given next to the line. An arrow on the line indicates the direction in which to read the name. The meta-relationship *C* in Figure 3 is an example of this. Additionally, two special notations may be used to represent meta-relationships on diagrams of meta-models.

If a meta-relationship is included in the diagram for clarification, but it is not used in the subject area, the text of the meta-relationship name shall be back-shaded. This is shown for the meta-relationship *D* in Figure 3.

If a supertype meta-entity that participates in a meta-relationship is not included in the diagram, the inherited meta-relationship shall be shown on the diagram as a broken line to/from the inheriting subtype meta-entity. This is shown for the meta-relationship *E* in Figure 3.

Subtype meta-relationships are shown with the meta-relationship name in italics. This is shown for the meta-relationship *F* in Figure 3.

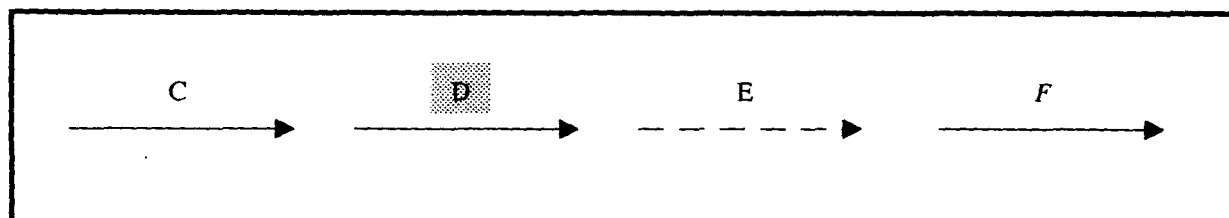


Figure 3
Conventions for Representing Meta-relationships

Full details of the CDIF Graphical Notation used in the Meta-model and the Meta-meta-model can be found in *EIA/ IS-107 CDIF - Framework for Modeling and Extensibility*.

To aid in the understanding of some of the concepts modeled in this subject area, instance diagrams have been used. These diagrams differ from those described above in that an instance

diagram depicts one or more occurrences or instances of meta-entities and meta-relationships depicted in the diagrams previously described. Where one of the diagrams previously described may show that the meta-model includes an *Automobile* meta-entity, an instance diagram might depict *Honda*, *Ford*, *Mercedes*, and *Fiat* as instances of *Automobile*.

In an instance diagram, instances of meta-entities are depicted as boxes with a horizontal divider. Above the divider is the name of the meta-entity that is instantiated. Below the divider may be a name or a value assigned to a significant meta-attribute of that meta-entity. Where the name is inherited, inferred, or otherwise not significant to the example (e.g., some associative meta-entity instances), no name or value may be present.

Meta-relationships are depicted as arrows, just as in the previous diagrams described. They do not, however, have cardinality or exclusivity indicators, since none are necessary.

Meta-attributes of both meta-entities and meta-relationships are represented as:

[meta-attribute name :] *value*

where the name of the meta-attribute may be omitted depending on the ease of understanding the example with or without the name (e.g., *Name* might be omitted, *MinLength* might not).

When an instance of a meta-entity or meta-relationship not in this Subject Area is included for clarity, the box/arrow shall be shaded just as in the E-R Diagrams.

Figure 4 shows both the conceptual diagram and an instance diagram.

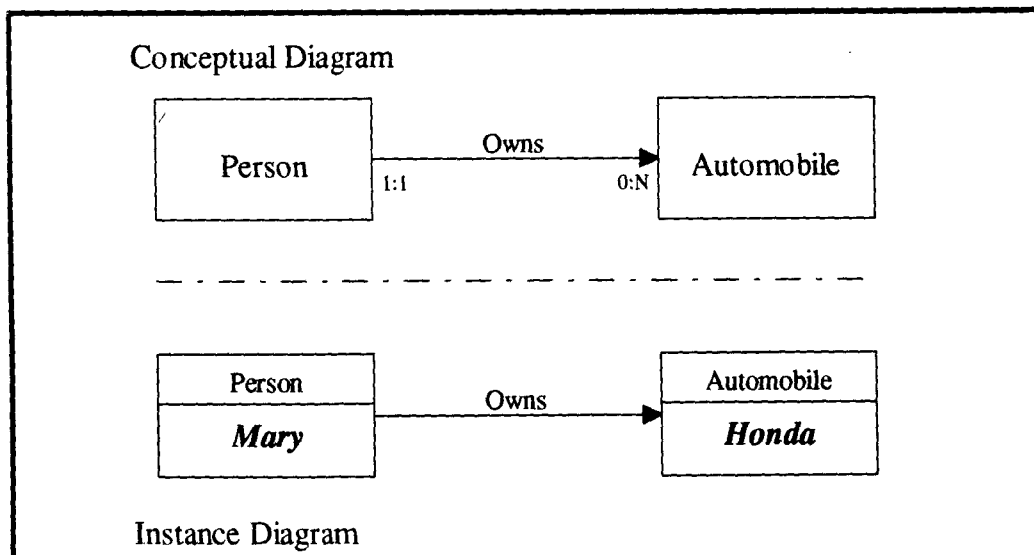


Figure 4
Conceptual Diagram and an Instance Diagram

3.2.4 AttributableMetaObject Hierarchy

The AttributableMetaObject Hierarchy in the Subject Area Overview section contains all the meta-entities and meta-relationships used in the subject area. It also identifies all supertypes required to relate the meta-objects used back to *RootObject*. The only subtypes of a meta-object appearing in the hierarchy that will also appear in the hierarchy are those that are used in the subject area or those required for clarity.

The AttributableMetaObject Hierarchy is presented as an indented list of meta-object names. The indentation indicates the subtyping; all names indented below a given name are subtypes of that meta-object. For example, if *Car* and *Truck* are both subtypes of *Vehicle*, this would be shown as follows:

```
Vehicle
  Car
  Truck
```

A meta-entity or meta-relationship and all its subtypes can be duplicated in the hierarchy due to multiple inheritance. Where this occurs, the duplicate entries are shown in *italics*.

The overall AttributableMetaObject supertype, *RootObject*, defined in *EIA/IS-111 CDIF - Integrated Meta-model - Foundation Subject Area*, is shown at the top of the hierarchy.

Where a page break occurs in the list, all the supertypes of the first meta-entity or meta-relationship on the new page are repeated for clarity with a shaded background.

3.2.5 MetaEntity and MetaRelationship Summaries

Summaries are given of all the meta-entities and meta-relationships used in the Subject Area. The meta-entity definitions are given first, in alphabetical order, followed by the meta-relationship definitions, also in alphabetical order.

For each meta-entity used in the subject area, all the meta-attributes, both inherited and local, are listed. The inherited meta-attributes are shown first, in *italics*, sequenced alphabetically within parent meta-entity. The local meta-attributes are then listed alphabetically in normal font. The optionality, whether mandatory or optional, of each meta-attribute is given.

For each meta-relationship all the meta-attributes are listed sequenced alphabetically. Inherited and local meta-attributes are shown in the same way as for meta-entities. The optionality of each meta-attribute is given.

3.3 Structure of Subject Area Detailed Definitions

3.3.1 Summary

The Subject Area Detailed Definitions section consists of the following sub-sections:

- Subject Area Definition
- MetaEntity Definitions
- MetaRelationship Definitions

3.3.2 Subject Area Definition

This is the formal definition of the instance of the meta-entity called *SubjectArea* (SA). It contains the information listed in Table 1.

Table 1
Contents of Subject Area Definition

5.2 Subject Area Definition

NAME.....	subject area name
CDIFMETAIDENTIFIER	
DESCRIPTION	
USAGE.....	
ALIASES	
CONSTRAINTS.....	
VERSIONNUMBER	

3.3.3 MetaEntity Definitions

The definition of each meta-entity used in the subject area is structured according to the classes of information listed in Table 2.

For each meta-entity:

- Inherited meta-attributes are listed by supertype meta-entity in a top-down hierarchical order. Within each sub-list, meta-attributes are listed alphabetically.
- Inherited meta-relationships are grouped by source meta-entity (listed in alphabetical order). Within each group, the inherited meta-relationships are sorted alphabetically by "verb" name and then alphabetically by destination meta-entity name. Inherited abstract meta-relationships are not listed.
- Local meta-attributes are listed alphabetically.

- Detailed local meta-attribute descriptions are listed alphabetically. These definitions are structured according to the classes of information listed in Table 3.

Table 2
Contents of Meta-entity Definition

5.N.N Meta-entity Name
NAME meta-entity name
CDIFMETAIDENTIFIER
SUBTYPEOF List of direct supertypes of the meta-entity...
meta-entity name [not] in this SA
SUPERTYPEOF List of direct subtypes of the meta-entity within this
subject area...
meta-entity name
DESCRIPTION
USAGE
ALIASES
CONSTRAINTS
TYPE

INHERITED META-ATTRIBUTES

List of meta-attributes inherited from all supertypes of the meta-entity, with the name of the meta-object for which the meta-attribute is defined...

meta-attribute name from meta-entity name [not] in this SA

LOCAL META-ATTRIBUTES

List of meta-attributes defined for the meta-entity...

meta-attribute name

INHERITED META-RELATIONSHIPS

List of instantiable meta-relationships inherited from all supertypes of the meta-entity...

meta-relationship name [not] in this SA

LOCAL META-RELATIONSHIPS

List of meta-relationships defined for the meta-entity...

meta-relationship name

Full definitions of all local meta-attributes appear at this point. See Table 3 for format of local meta-attribute definitions.

Table 3
Contents of Local Meta-attribute Definition

META-ATTRIBUTE NAME.....	meta-attribute name
CDIFMETAIDENTIFIER	
DESCRIPTION	
USAGE.....	
ALIASES	
CONSTRAINTS	
DATA TYPE	
DOMAIN	
LENGTH.....	
ISOPTIONAL.....	

3.3.4 MetaRelationship Definitions

The definition of each meta-relationship used in the subject area is structured according to the classes of information listed in Table 4.

For each meta-relationship:

- Inherited meta-attributes are listed by supertype meta-relationship in a top-down hierarchical order. Within each sub-list, meta-attributes are listed alphabetically.
- Local meta-attributes are listed alphabetically.
- Detailed local meta-attribute descriptions are listed alphabetically.

Table 4
Contents of Meta-relationship Definition

5.N.N Full Meta-relationship Name

NAME	meta-relationship name
CDIFMETAIDENTIFIER	
SUBTYPEOF	List of direct supertypes of the meta-relationship... meta-relationship name [not] in this SA
SUPERTYPEOF	List of direct subtypes of the meta-relationship within this subject area... meta-relationship name
MINSOURCECARD	Minimum source meta-entity cardinality
MAXSOURCECARD	Maximum source meta-entity cardinality
MINDESTCARD	Minimum destination meta-entity cardinality
MAXDESTCARD	Maximum destination meta-entity cardinality
DESCRIPTION	
USAGE	
ALIASES	
CONSTRAINTS	

INHERITED META-ATTRIBUTES

List of meta-attributes inherited from all supertypes of the meta-relationship, with the name of the meta-object for which the meta-attribute is defined...

meta-attribute name from meta-relationship name [not] in this SA

LOCAL META-ATTRIBUTES

List of meta-attributes defined for the meta-relationship...

meta-attribute name

Full definitions of all local meta-attributes appear at this point. See Table 3 for format of local meta-attribute definitions.

3.4 Structure of Referenced Meta-object Definitions

This section contains the formal definitions of meta-objects referenced from the definitions of meta-objects used in the subject area.

These will include the *AttributableMetaObject RootObject*, since this is the root of the *AttributableMetaObject* Hierarchy, and it may also include other meta-entities and meta-relationships.

The definitions are given in the following order: *AttributableMetaObject*, meta-entities, then meta-relationships. Within each section, they are listed in alphabetical order. The definition layout for the *AttributableMetaObject* is the same as that of the meta-entity; both meta-entity and meta-relationship definitions follow the same layout as that defined for the Subject Area Detailed Definitions section.

4. SUBJECT AREA OVERVIEW

4.1 Description

4.1.1 Introduction

The Data Flow Model Subject Area provides support for interchanging the semantic information found in existing data flow and some function modeling techniques. It covers the semantics associated with processes or functions, flows, stores, and external agents. This subject area does not cover physical process modeling found in structured design techniques.

Although some of the meta-entities, meta-relationships and meta-attributes in this subject area can be used for specifying models using the concepts of control flows and material flows, complete modeling of these techniques (e.g., real-time extensions) is outside the scope of this subject area.

Rules and constraints enforced by specific data flow or function modeling techniques are not enforced by this subject area, since the subject area must be capable of supporting all the different variants, and the interchange of data between all the variants.

4.1.2 Data Flow Model

A data flow model is the root container context for a set of processes, stores, external agents, and flows. The concept of a data flow model is used in this subject area to identify a unit of reusability that can represent subsystems or modules.

The definitions of processes, flows, stores, and external agents may be shared. Therefore different data flow models may have the same decomposition, or a data flow model may use another data flow model as a submodel.

Figure 5 shows an example of the process decomposition hierarchy in connection with data flow models in a CDIF transfer. Processes are defined by process definitions, which may be structured. Process definitions may be identified as data flow models.

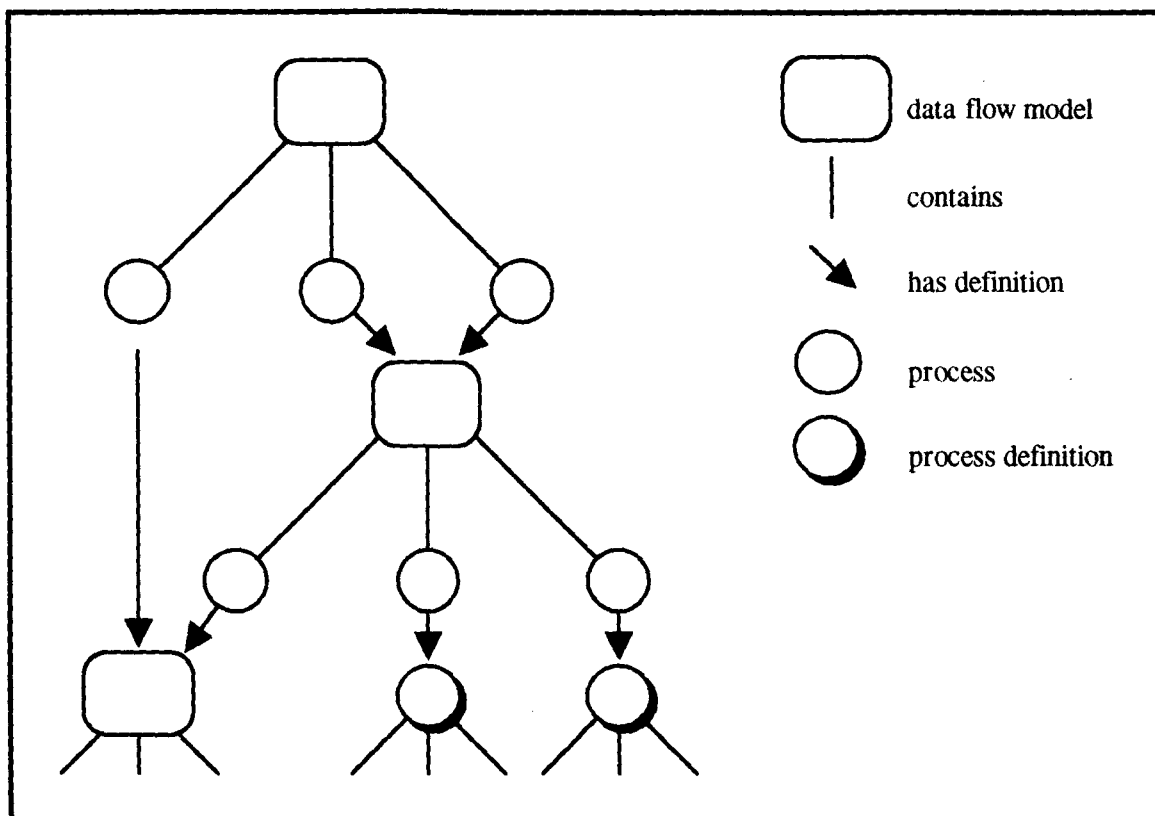


Figure 5
Process decomposition and hierarchy of data flow models

4.1.3 Processes

The Data Flow Model Subject Area supports the concept of a process or function. A process consumes data carried in data flows, transforms the data, and sends the results on the data flows which lead out of that process.

Processes may be structured; a process at a certain level of abstraction may be refined into a number of lower-level processes, stores and external agents, connected by data flows. The higher-level data flows are bound to the interface the lower level model exposes.

Processes at any level of abstraction may be specified textually.

4.1.4 Flows

Flows represent channels for potential information exchange between processes, stores and external agents. At any instant of time, there may or may not be data carried over a flow.

A flow may be elemental or structured. A structured flow is a composition of other flows. A flow that is not decomposed into other flows may contain attributes representing the structure of the flow. A flow may also directly reference a data type as defined in the *EIA/IS-113 CDIF - Integrated Meta-model - Data Definition Subject Area*.

A flow may be unidirectional or bidirectional. A unidirectional, structured flow may only be decomposed into flows that are unidirectional in the same direction. A bidirectional structured flow may be decomposed into both unidirectional and bidirectional flows.

4.1.5 External Agents

An external agent is an object whose detailed definition is outside the scope of a system or subsystem under consideration, but is referenced in order to show the interaction of the system or subsystem with its environment.

All the input and output flows of an external agent are part of the interface of the system or subsystem to its environment.

The concept of representing stores and processes as an external interface in another model is accomplished using external agents (e.g., an input data store produced in a different system should appear as an external agent).

4.1.6 Stores

A store produces and consumes flows. A store may consume the incoming data or controls, store them, and, at a later time, send the stored data or controls over the outgoing flows. The store does not perform any transformations on the data or controls; it can be considered a set of flows at rest.

Stores may be data stores, control stores, or both.

4.1.7 The General Structuring Mechanism

4.1.7.1 Introduction

Support for the decomposition and structuring of objects into other objects, and the reuse of definitions, is provided by a general structuring mechanism. This mechanism is also used in other CDIF subject areas.

In this subject area the following concepts may be structured: processes, stores, external agents, and flows. Structuring of those concepts is performed using the general structuring mechanism because their definitions may be used for several objects in different contexts.

4.1.7.2 Meta-entities and Meta-relationships

The general structuring mechanism makes use of the following meta-entities:

- *ComponentObject*
- *DefinitionObject*
- *ReferencedElement*
- *EquivalenceSet*

and the following meta-relationships:

- *DefinitionObject.Contains.ComponentObject*
- *ComponentObject.References.DefinitionObject*
- *ReferencedElement.DefinesPath.ComponentObject*
- *EquivalenceSet.HasMember.ComponentObject*

Figure 6 shows the meta-model for the general structuring mechanism.

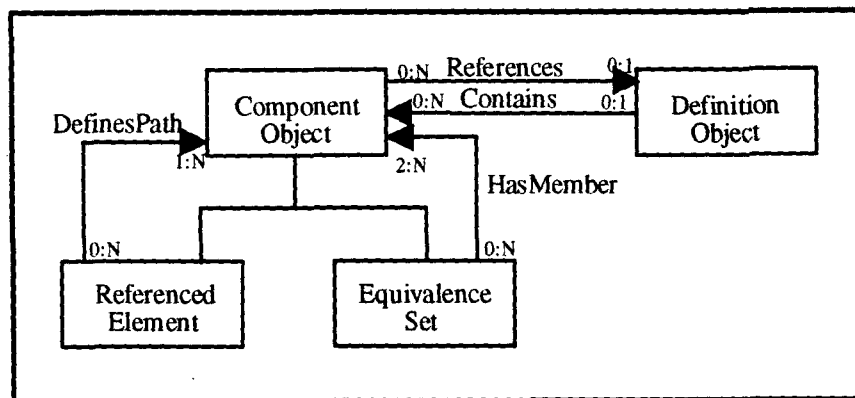


Figure 6
General Structuring Mechanism

4.1.7.3 DefinitionObject and ComponentObject

DefinitionObject serves as an abstract supertype for all decomposition and structure definitions that may be reused. A *DefinitionObject* represents the definition of a *ComponentObject*, using meta-relationship *ComponentObject.References.DefinitionObject*. A *DefinitionObject* may contain *ComponentObjects*, using the *DefinitionObject.Contains.ComponentObject* meta-relationship. The *ComponentObjects* contained in a *DefinitionObject* represent the components of the definition.

Both *ComponentObject* and *DefinitionObject* represent abstract concepts and thus shall not be instantiated. Instead, appropriate subtypes, provided by this or other subject areas, are used.

A subtype of *DefinitionObject* may be referenced by any number of *ComponentObjects* indicating that all those *ComponentObjects* share the same definition.

Instances of the subtypes of *ComponentObject* are used to describe the structure of a *DefinitionObject*. There may be any number of *ComponentObjects* per *DefinitionObject*.

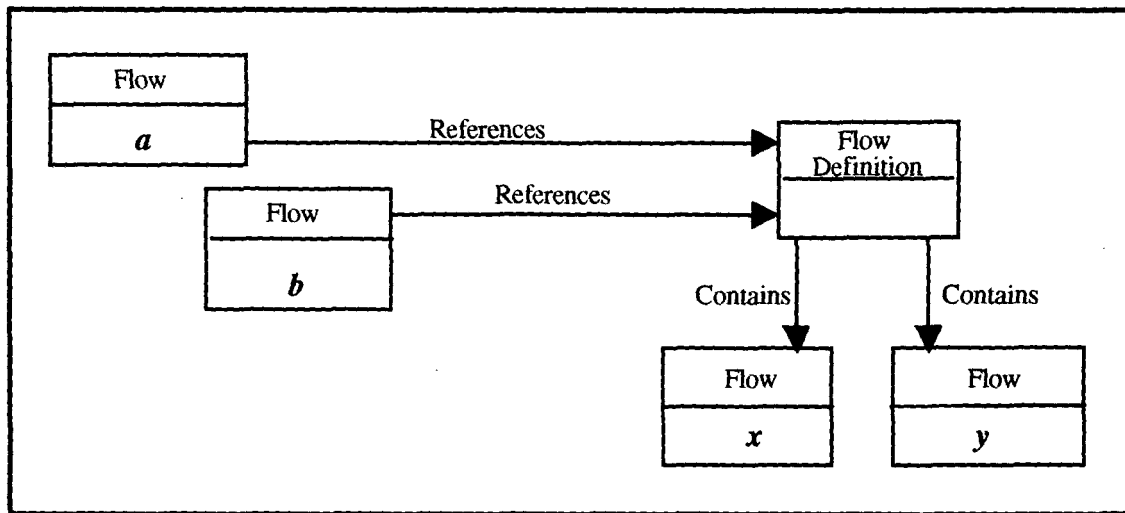


Figure 7
Flows sharing the same structured definition

For example, if a structure is built containing two components that share the same definition, the structured *DefinitionObject* will contain two distinct *ComponentObjects* which refer to the same shared *DefinitionObject*.

In this subject area, *DefinitionObjects* may contain one or more instances of one or more *ComponentObjects*. The allowed components for a given *DefinitionObject* are shown in Table 5. For example, a *Flow* may reference a *FlowDefinition* that contains *Flows* and *Attributes*. This represents the fact that the *Flow* is refined into lower-level flows and attributes. The *FlowDefinition* may be shared by another *Flow* by using *ComponentObject.References.DefinitionObject* to refer to this *FlowDefinition*.

Table 5
Allowable *DefinitionObject* components

DefinitionObject	ComponentObject
DFMProcessDefinition	DFMProcess Store ExternalAgent FlowPort Flow Attribute
ExternalAgentDefinition	DFMProcess Store ExternalAgent FlowPort Flow Attribute
StoreDefinition	Store FlowPort Flow Attribute
FlowDefinition	Flow Attribute

The general structuring mechanism ensures that, for example, the instances of the components of one *DFMProcess* are different from those of another *DFMProcess* even if both *DFMProcesses* share the same *DFMProcessDefinition*. This applies equally for all *DefinitionObjects*. It can also be used to specify, using the *EquivalenceSet* meta-entity, that two components in different reuse situations are actually the same instance (see 4.1.13 Identical Instances in Shared Definitions).

4.1.8 Process Decomposition and Definition

Processes are represented using the meta-entity *DFMProcess*, a subtype of *ComponentObject*. Process definitions are represented using meta-entity *DFMProcessDefinition*, a subtype of *DefinitionObject*.

Figure 8 shows an example of a process decomposition. *DFMProcess P* decomposes into *Q*, *R* and *S*. *R* and *S* share the same decomposition into *T* and *U*. The general structuring mechanism requires that each process refers to a process definition. *DFMProcesses* cannot contain *DFMProcesses* or any other *ComponentObject* directly; the *DFMProcessDefinition* contains the *ComponentObjects*. Sharing definitions does not imply anything about the instances. For example, the *DFMProcess T* in the decomposition of *R* shares only its structure with *S*, not its data or state.

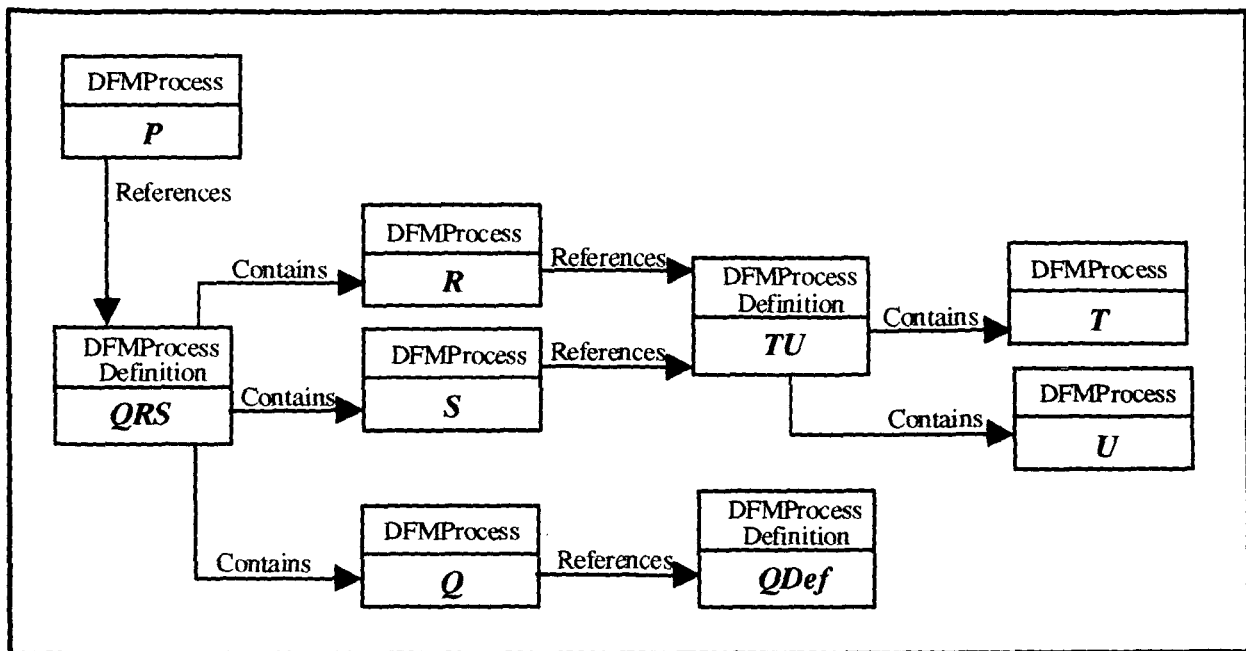


Figure 8
A process decomposition

For the definition of leaf processes (i.e., processes which are not decomposed into other processes), an instance of the meta-entity *DFMPProcessDefinition* is used which does not contain any *DFMPProcesses*, *Stores*, *ExternalAgents* or *Flows*.

4.1.9 Flow Decomposition and Definition

Analogous to the structuring of processes, *Flows* are also structured using the general structuring mechanism. A *FlowDefinition* represents the potential structure of a *Flow*. A *Flow* may be a structured *Flow* containing other *Flows* as members or containing *Attributes*, or completely unstructured.

Figure 9 shows an example for structured *Flows* where *Flows* *f1* and *f2* share the same structured *FlowDefinition*. This *FlowDefinition* indicates that both *f1* and *f2* each contain a *Flow* *x*, a *Flow* *y* and a *Flow* *z*. Each *Flow* has a *FlowDefinition*. *Flows* *x* and *y* share a *FlowDefinition* called *flow-def2*. Those *FlowDefinitions* contain *Attributes* that are the lowest-level components of the flows.

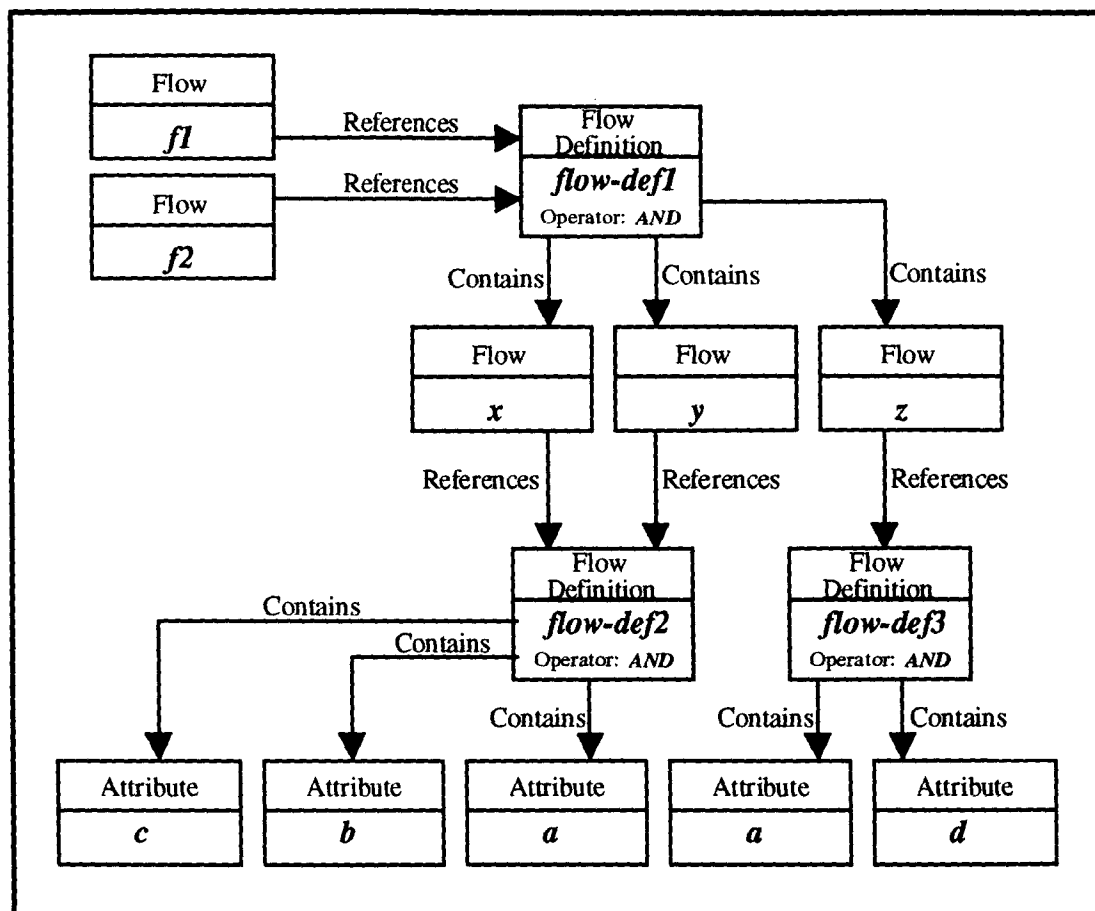


Figure 9
Structured Flows

The sharing of *FlowDefinitions* does not imply anything about the data on the *Flows* which share the same definitions. In this case, for example, the data on *Attributes* *f1.x.a* (*Attribute a* on member flow *x* of flow *f1*), *f1.y.a* and *f2.z.a* is completely unrelated. If the values of the *Attributes* *f1.x.a* and *f1.y.a* are the same, for example, an *EquivalenceSet* with two *ReferencedElements* would be used. This *EquivalenceSet* would be contained in *FlowDefinition* *flow-def1*. (see section 4.1.13 Identical Instances in Shared Definitions)

4.1.10 Store Decomposition

Stores may be decomposed into other stores. This is done by having a *StoreDefinition* contain instances of meta-entity *Store*. A *StoreDefinition* can also contain *Flows* and *Attributes*. The *Attributes* represent the lowest level data elements of the *Store*.

4.1.11 Context Identifiers

A *DFMProcessDefinition*, *StoreDefinition*, or *ExternalAgentDefinition* may be reused. Storing identifiers to describe the location of the *DFMProcesses*, *Stores* or *ExternalAgents* contained in the decomposition hierarchy is not possible. This is because the containing *DefinitionObject* may be referenced by many *ComponentObjects* in different locations in the hierarchy.

The meta-attribute *ContextIdentifier* of meta-entities *DFMProcess*, *Store* and *ExternalAgent* provides the mechanism to store the part of the identifier of the process, store or external agent which is local to the *DFMProcessDefinition*, *StoreDefinition*, or *ExternalAgentDefinition*. The full identifier used by some techniques then needs to be computed by the tool.

For example, if, in a specific technique, a process has the full identifier *1.2.3*, only the value *3* is stored. The values *1* and *2* can be derived from the context of the process.

4.1.12 Accessing a Specific Component in a Shared Definition

If a *ComponentObject* in a shared *DefinitionObject* needs to be referenced, a simple reference is not sufficient because definitions can be shared and it would be not clear whose structured object's component is meant if its definition is reused. Instead, the full path to the *ComponentObject*, traversing the loop *ComponentObject*, *ComponentObject.References.DefinitionObject*, *DefinitionObject*, *DefinitionObject.Contains.ComponentObject*, is needed to uniquely identify a specific instance.

This path is formed by the associative meta-entity *ReferencedElement* and the meta-relationship *ReferencedElement.DefinesPath.ComponentObject*. The *ReferencedElement* is the *ComponentObject* which refers to the specific *ComponentObject* contained in a shared *DefinitionObject*. The meta-relationship *ReferencedElement.DefinesPath.ComponentObject* carries a meta-attribute *SequenceNumber* which describes the sequence in which the path is traversed.

An example of this is shown in figure 10 for the case of structured data flows. Figure 11 shows the corresponding instance diagram. In this example, there are two instances of the *x* component flow, one in *Flow a* (called *a.x*), and one in *Flow b* (called *b.x*). In order to refer to the *x* component of *a* (as opposed to the *x* component of *b*) a *ReferencedElement* is created which identifies the complete path consisting of *a* and *x*, using instances of the meta-relationship *ReferencedElement.DefinesPath.ComponentObject*.

Paths define a top-down hierarchical navigation of a structure. This is done using the meta-attribute *SequenceNumber* of the meta-relationship *ReferencedElement.DefinesPath.ComponentObject*. Sequence numbers must be consecutive to properly specify the navigation.

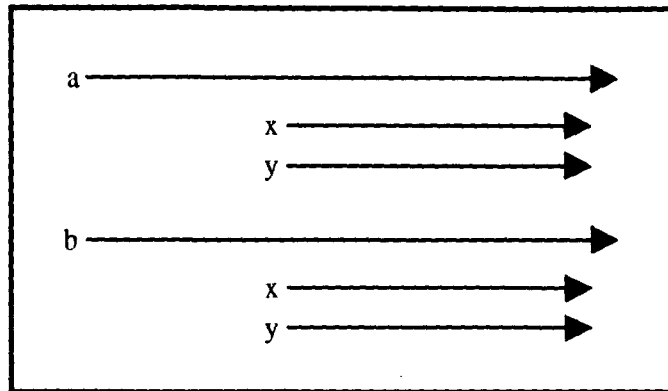


Figure 10
 Shared definition for structured flows

The values of meta-attribute *SequenceNumber* are also shown in figure 11. The *ComponentObject* *Flow a* is the first element of the path and hence the instance of meta-relationship *ReferencedElement.DefinesPath.ComponentObject* referring to it carries the value *1* in its *SequenceNumber* meta-attribute. *Flow x* is the second element and thus *SequenceNumber* has a value of *2*.

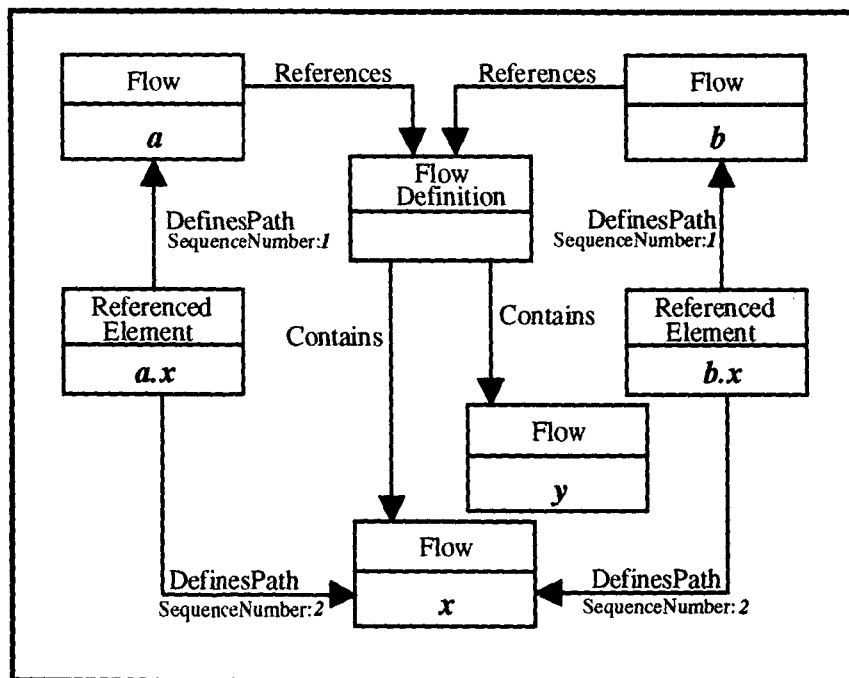


Figure 11
 Referring to a specific instance inside a shared definition

To form a path, every *ComponentObject* on the path shall be contained in the path formed by *ReferencedElement* and *ReferencedElement.DefinesPath.ComponentObject*, even if a specific *DefinitionObject* on the path is not reused.

Note that, except for the case of an incomplete model, any *ReferencedElement* must also be contained within a *DefinitionObject* (not shown in figure 11). Thus a reference to a specific lower-level component is also located in a definition that could be reused.

In this subject area the *ReferencedElement* meta-entity is used for referencing *DFMProcesses*, *Stores*, *ExternalAgents*, *Flows*, *Attributes*, and *FlowPorts* contained in a *DFMProcessDefinition*; for referencing *DFMProcesses*, *Stores*, *ExternalAgents*, *Flows*, *Attributes*, and *FlowPorts* contained in an *ExternalAgentDefinition*; for referencing *Stores*, *FlowPorts*, *Flows* and *Attributes* in a *StoreDefinition*; and for referencing *Flows* and *Attributes* in a *FlowDefinition*.

4.1.13 Identical Instances in Shared Definitions

When two or more *ComponentObjects* are identical, they are specified as such using the mechanism of the meta-entity *EquivalenceSet* and the meta-relationship *EquivalenceSet.HasMember.ComponentObject*.

By using the meta-relationship *EquivalenceSet.HasMember.ComponentObject*, an *EquivalenceSet* groups all *ComponentObjects* which represent the same object instance although they may be represented by different instances of potentially different subtypes of *ComponentObject*. An *EquivalenceSet* may also contain *ReferencedElements*. This allows specific components in specific reuse situations of shared definitions to be equivalent.

An *EquivalenceSet* must be contained in a *DefinitionObject*, except for the case of an incomplete model. It may only have members which are contained by the same *DefinitionObject* or are not contained by any *DefinitionObject* (in case of an incomplete model). If a *ComponentObject* is member of an *EquivalenceSet*, and this *ComponentObject* is contained in another *DefinitionObject*, the *ReferencedElement* mechanism must be used for referencing components in shared definitions.

For example, to express that two *Stores* in two branches of a process decomposition hierarchy are identical, an *EquivalenceSet* is used to equate two *ReferencedElements* which refer to the *Stores*. This is shown in figure 12.

It is not possible to have both *Stores* as members in the *EquivalenceSet* directly without referencing them using *ReferencedElements* because the intermediate *DFMProcessDefinitions* might be reused and the equivalence is only given for a specific situation. This is also shown in figure 12. *DFMProcess other* references the same definition as *DFMProcess B*, but because it is not included in the *EquivalenceSet*, the *Store other.Z* is a different instance than *Stores B.Z* and *A.Y*.

All other *DFMProcesses* which might share the *DFMProcessDefinition* are not affected by the *EquivalenceSet* containing *A.Y* and *B.Z* and so do not influence other uses of the same definition. When the *EquivalenceSet* is contained within another *DFMProcessDefinition*, and thus is shared, the property that *A.Y* and *B.Z* are identical is preserved for all uses of the "upper-level" *DFMProcessDefinition*.

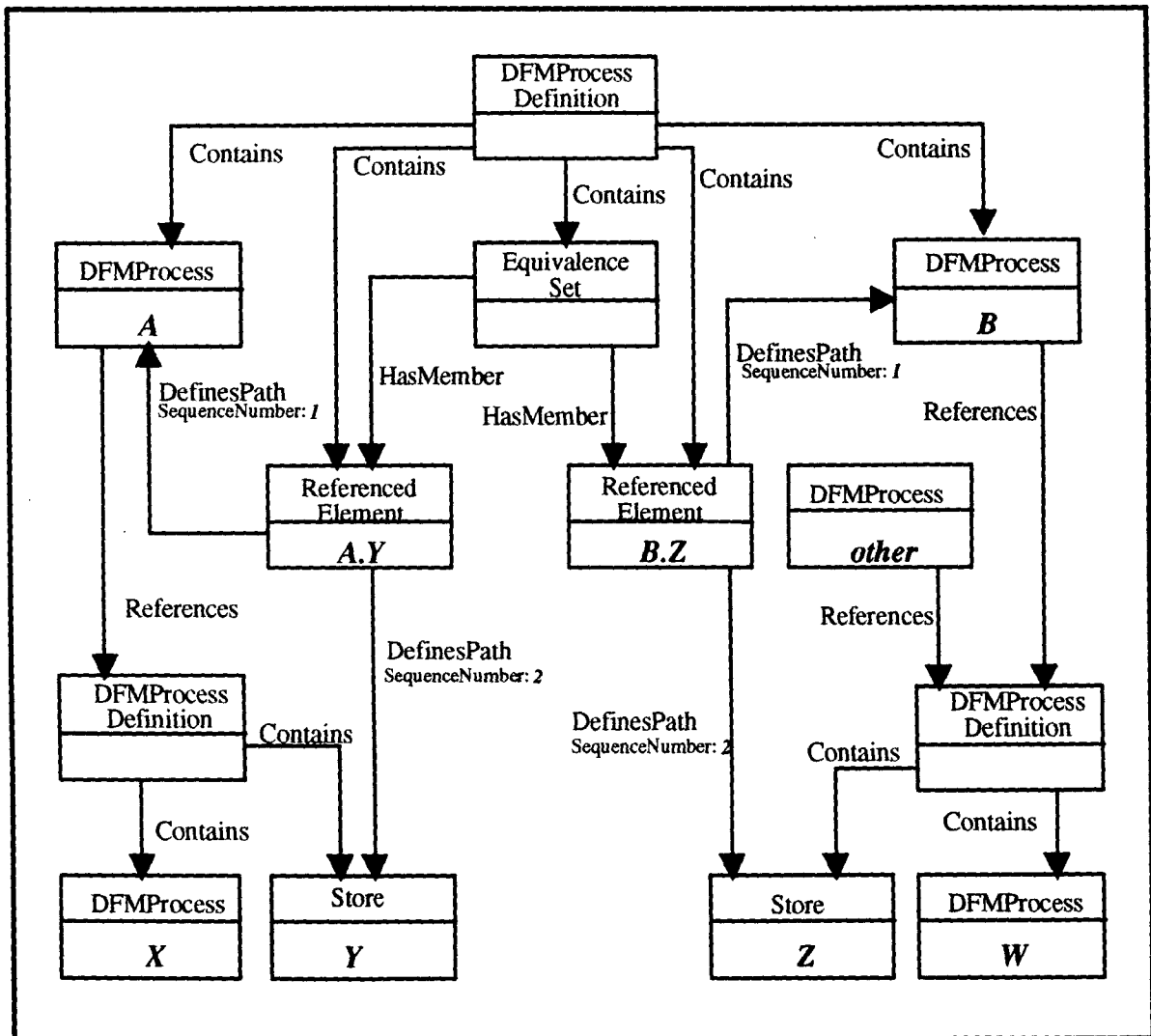


Figure 12
Denoting the equivalence of two Stores

4.1.14 Alternate Decompositions

This subject area supports alternate decompositions of *Flows* and *DFMProcesses*, using the general structuring mechanism.

Decompositions may have one of two characteristics, represented by the *Operator* meta-attribute of the *DefinitionObject* meta-entity. Both *FlowDefinition* and *DFMProcessDefinition* are subtypes of *DefinitionObject* and thus inherit the meta-attribute *Operator*.

If the value of this meta-attribute is *AND*, this indicates that all the *ComponentObjects* related to the *DefinitionObject* using the meta-relationship *DefinitionObject.Contains.ComponentObject* are contained at the same time (i.e., inclusive).

For example, in figure 13, the *FlowDefinition Y1Def* representing a structured flow definition would set the value of *Operator* to *AND* because it contains all the member flows concurrently.

Some data flow modeling techniques support alternate flow decompositions. In this case the *FlowDefinition*'s meta-attribute *Operator* would have the value *XOR*. The value *XOR* of the *Operator* meta-attribute specifies that the *ComponentObjects* contained in the *DefinitionObject* are exclusive alternates. This means that the object is decomposed into exactly one of the *ComponentObjects* but which one is not specified. An alternate decomposition leaves it open as to which of the decompositions is to be used (see Section 4.1.15 Flow Splits and Merges).

For example, in figure 13, if a *Flow X* has two decompositions into *Flows A, B and C*, or into *Flows D and E*, a *FlowDefinition XDef* is created which is the definition of *Flow X*, using meta-relationship *ComponentObject.References.DefinitionObject*. *FlowDefinition XDef* has its meta-attribute *Operator* set to *XOR*. There are two *Flows Y1* and *Y2* which are contained in *XDef* using meta-relationship *DefinitionObject.Contains.ComponentObject*, indicating that the *FlowDefinition* has two possible structures. The *Flows Y1* and *Y2* have *FlowDefinitions Y1Def* and *Y2Def*, using *ComponentObject.References.DefinitionObject*. Those *FlowDefinitions* have set their meta-attribute *Operator* to *AND*, indicating that all their components are contained and they do not represent alternates. Note that if the *Operator* meta-attribute has a value of *XOR*, then the contained *Flows* do not actually exist as a part of the containing *Flow's* structure, but are simply playing the role of a grouping mechanism. Those *FlowDefinitions Y1Def* and *Y2Def* contain *Flows A, B and C*, or *D and E*, respectively.

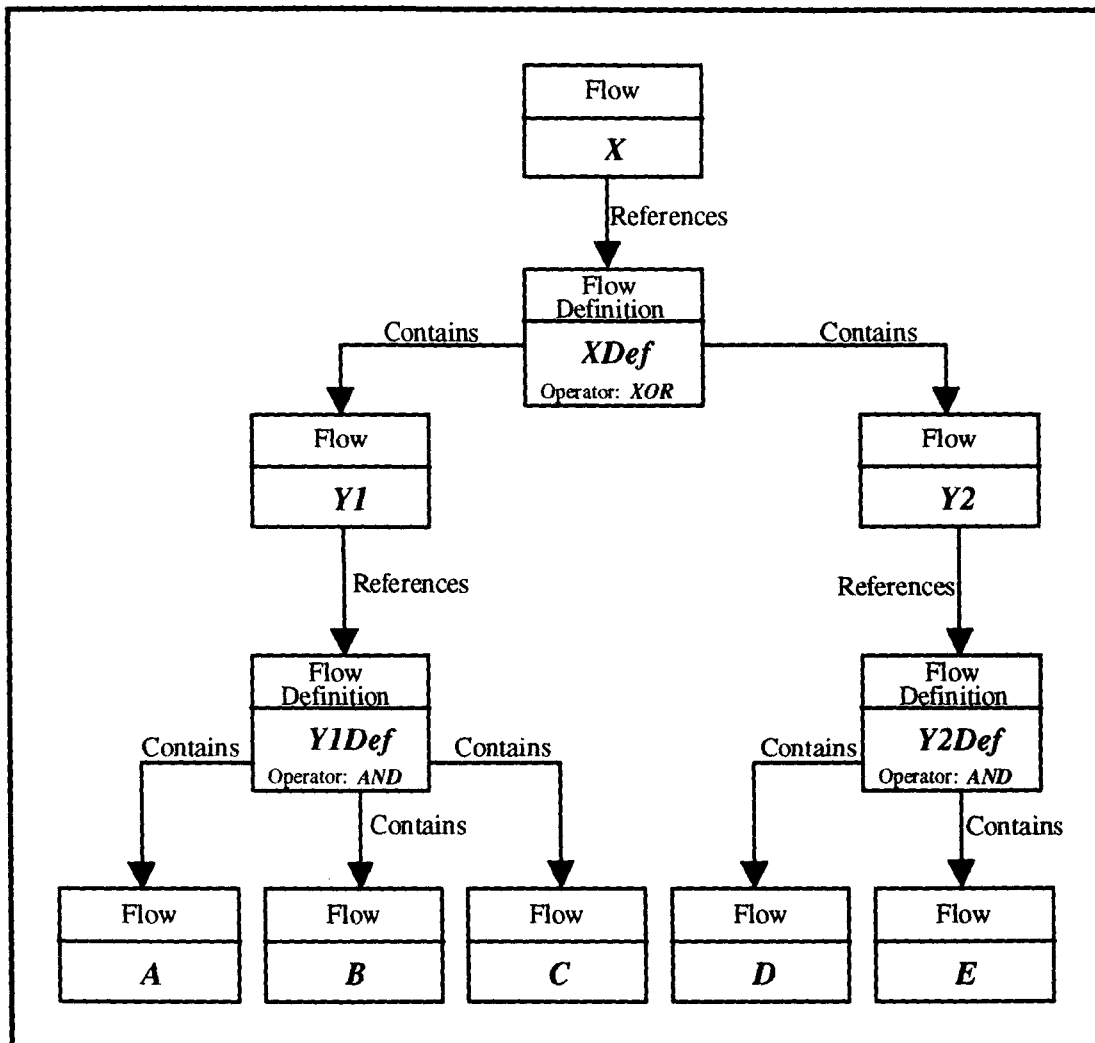


Figure 13
 Alternate Flow decompositions

4.1.15 Flow Splits and Merges

Flows may split and merge. Figure 14 and figure 15 are examples for *Flow* splits.

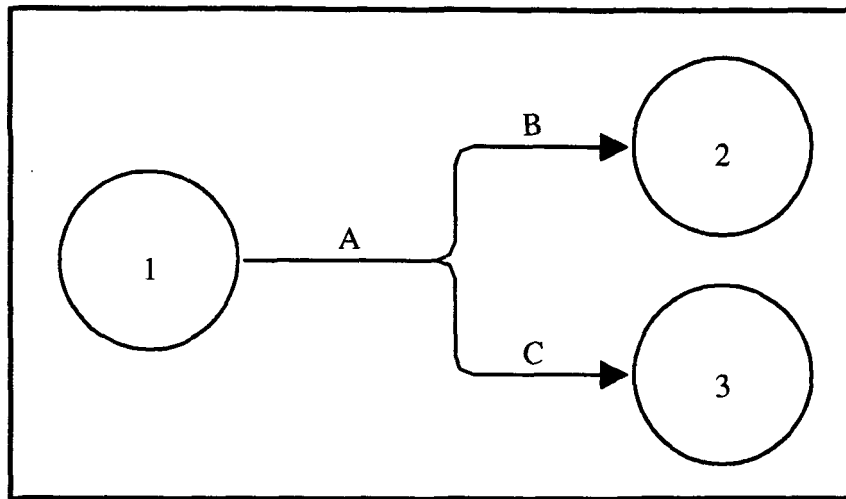


Figure 14
A Flow Split into component Flows

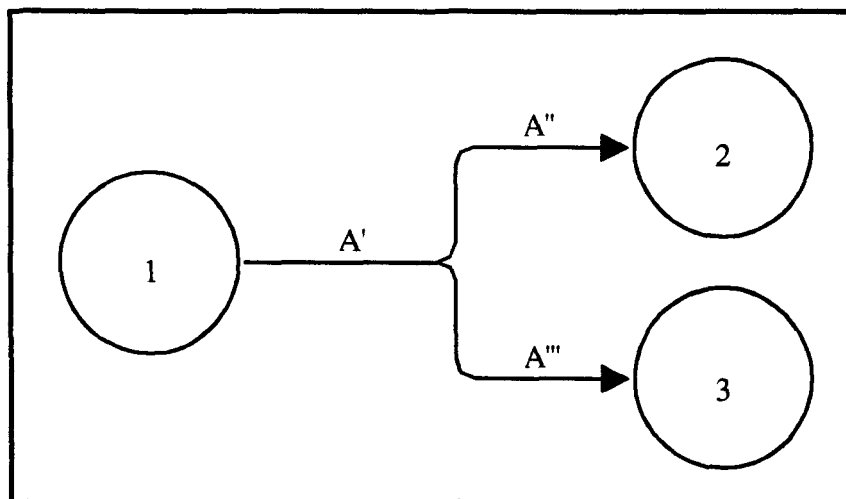


Figure 15
A Flow Split

Figure 14 depicts that *Flows B* and *C* carry the data from component *Flows* of *Flow A*. Figure 15 shows a split where *Flow A''* and *A'''* both carry all the data from *Flow A'* (i.e., they are instances of *A'*). The primes are added for explanatory purposes only.

For the first case, where *Flows B* and *C* carry the data of component flows of *Flow A*, it has to be defined which component *Flow* of *A* is consumed by *B* and which by *C*. Note that this subject area does not rely on any scheme which matches the names, so it is not possible to rely on naming in figure 14 to achieve this identification.

Figure 16 shows the instance diagram for this situation.

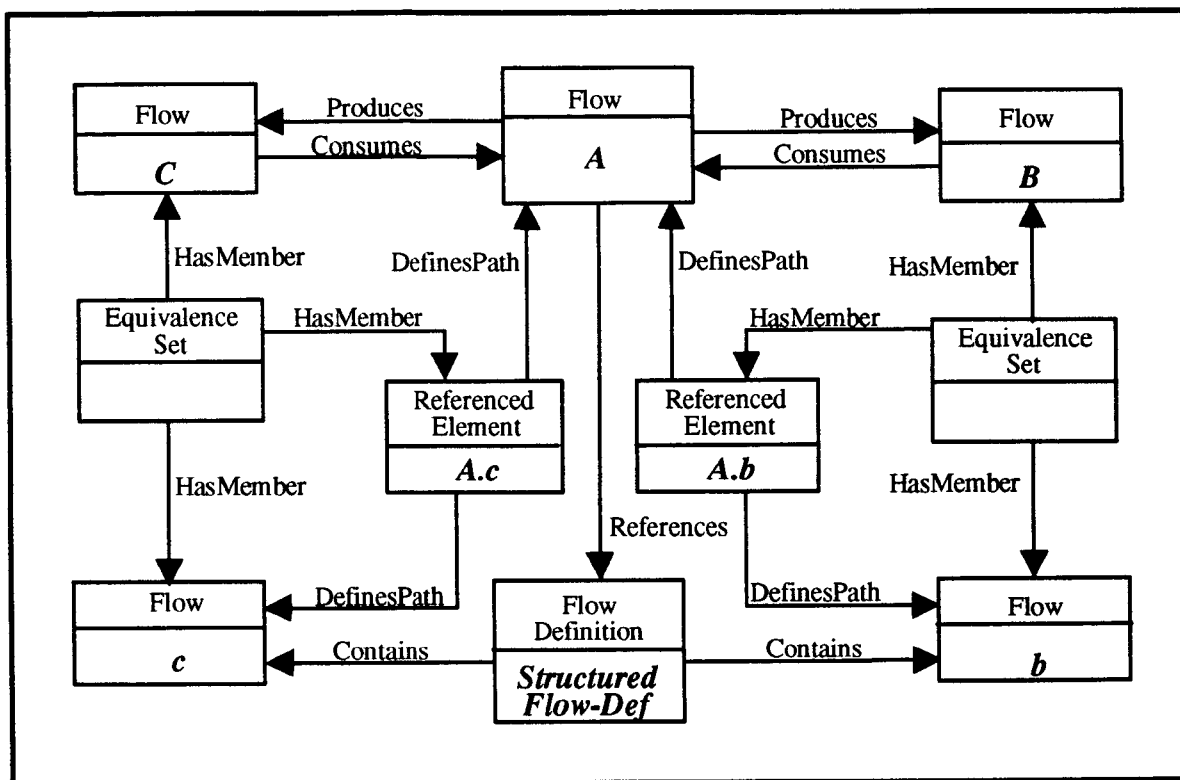


Figure 16
Instance diagram for splitting flows into component flows

For the second case, *Flows A''* and *A'''* carrying all the data from *Flow A'*, the instance diagram is shown in figure 17.

The flow split is represented using the meta-relationships *FlowProducerConsumer.ProducesOrConsumes.Flow*, *FlowProducerConsumer.Produces.Flow* and *FlowProducerConsumer.Consumes.Flow*. Note that for both split and merge the outgoing flows are produced by the incoming flows, and the incoming flows are consumed by the outgoing flows.

Merging flows is represented by reversing the source and destination of meta-relationships *FlowProducerConsumer.Produces.Flow* and *FlowProducerConsumer.Consumes.Flow*.

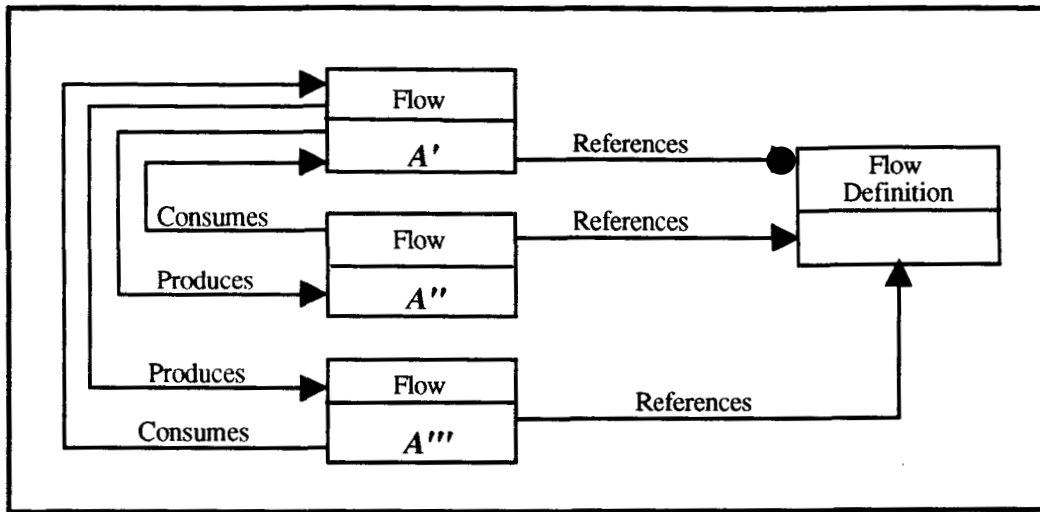


Figure 17
Instance diagram for a Flow Split

For example, when reversing the directions of the flows in figure 17, all instances of meta-relationships *FlowProducerConsumer.Produces.Flow* become instances of *FlowProducerConsumer.Consumes.Flow*, and vice versa.

This subject area also supports the concept of bidirectional *Flows*. If, for example, in figure 17 *Flows A' and A''* were bidirectional instead, the instance diagram would then change to figure 18.

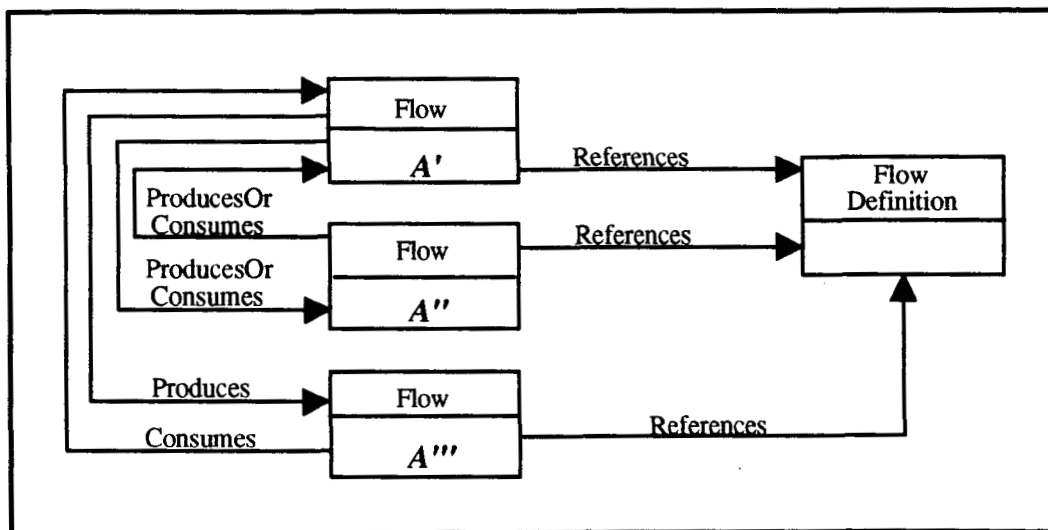


Figure 18
Instance diagram for a bidirectional Flow splitting into directed Flows

4.1.16 Interface of Processes, Stores and External Agents

The definition of processes, stores, external agents and flows may be shared, i.e., used by one or more processes, stores, external agents and flows, respectively, or by none. A process, store or external agent that is shared is required to explicitly specify its interface in terms of its input flows and output flows. This is similar to the binding mechanism of actual and formal parameters in programming languages.

Figure 19 depicts the need for this. Processes *F* and *G* both share the same definition, *PDef*, which contains process *P1*, input-flow *f1* and output-flow *f2*. There must be a correspondence between the definition flows and the actual flows. In the context of process *F*, flow *x* is the actual flow for definition flow *f1* and flow *y* is the actual flow for *f2*. Flow *y* is the actual flow for *f1* and flow *z* is the actual flow for *f2* in the context of process *G*.

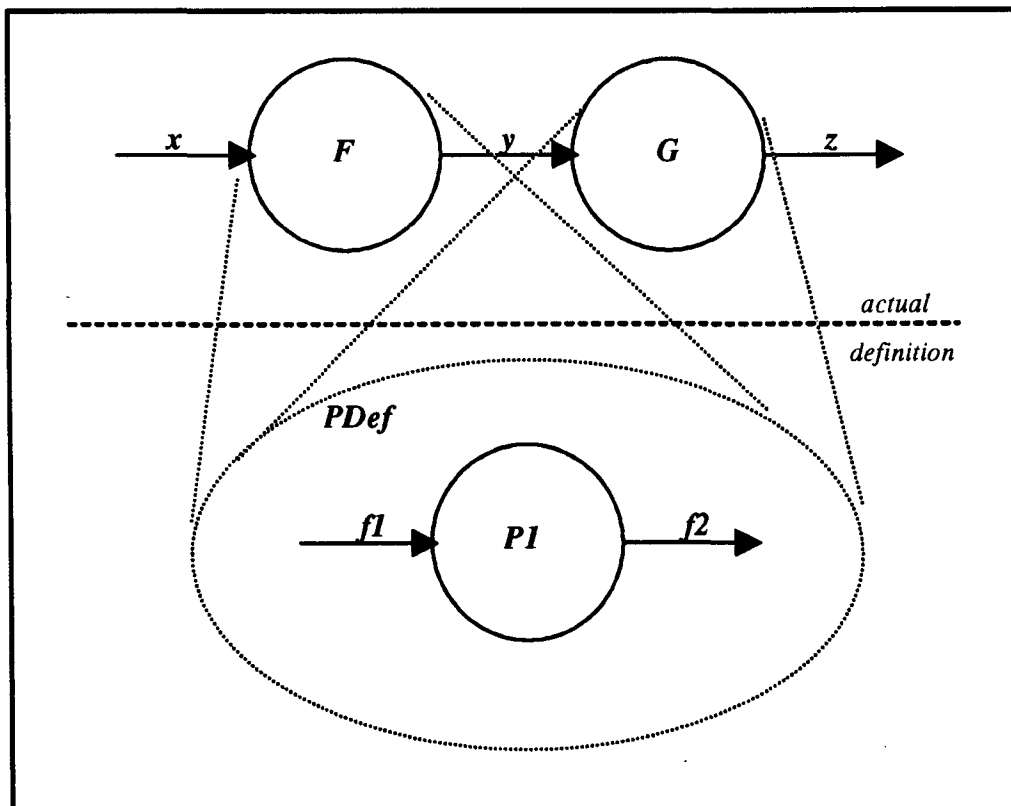


Figure 19
Shared *DFMPProcessDefinition* flow bindings

The meta-entity *Port* is a generic, abstract concept for *DefinitionObjects* to declare their interfaces to other objects. Because, in this subject area, the interface is accessed only by *Flows*, a concrete meta-entity, *FlowPort*, is provided instead of the abstract meta-entity *Port*. The formal

interface to the outside of the *DefinitionObject* is formed by all instances of *FlowPort* which have the value of the *IsFormal* meta-attribute set to **True**. To specify the actual *Ports* that reference the formal *Ports*, instances of *FlowPort* are used which have the value of *IsFormal* set to **False**. *Flows* cannot cross the boundary of a *DFMProcess*, *Store* or *ExternalAgent* without using the concept of a *FlowPort*.

Figure 20 shows a process, *P*, which references a process definition, *Pdef*. It also depicts the binding of *P*'s interface of actual flow ports (*A*, *B*, and *C*) with *Pdef*'s interface of formal flow ports (*A'*, *B'*, and *C'*).

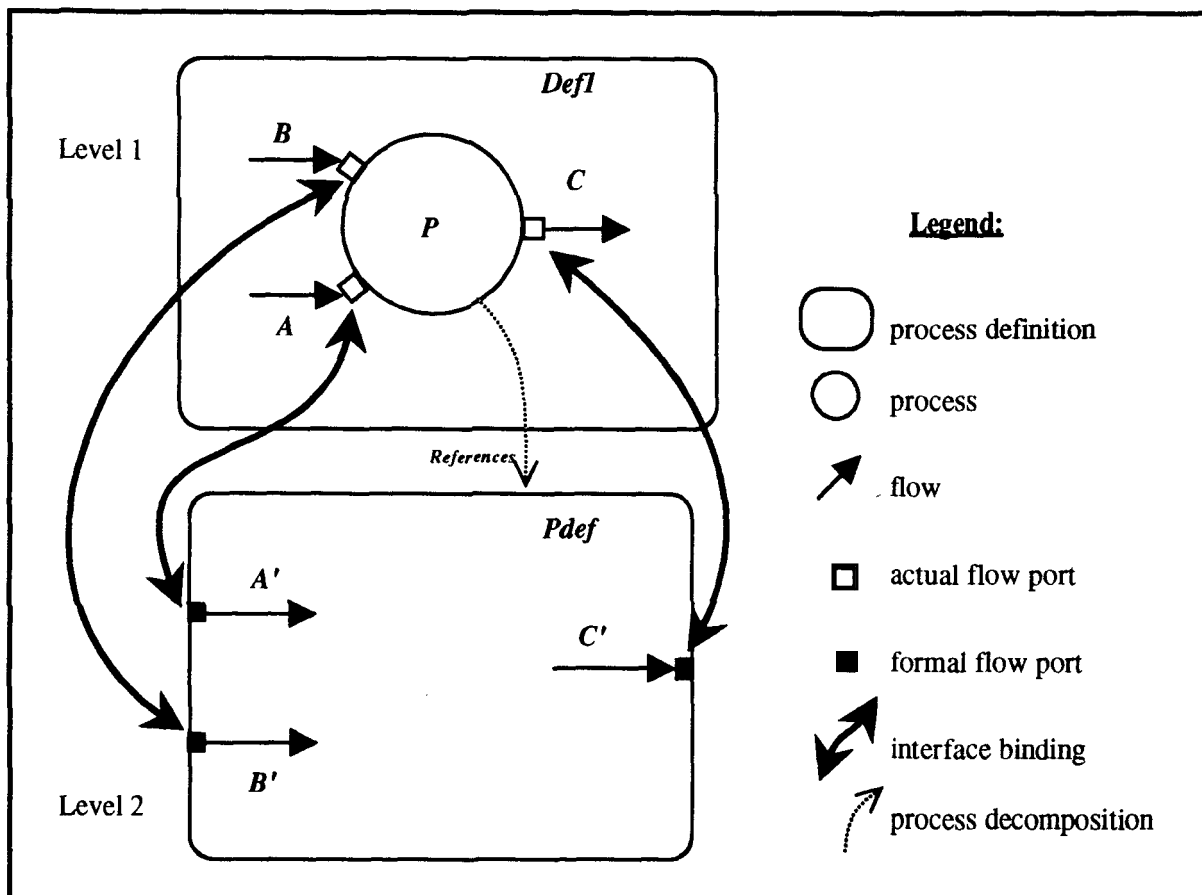


Figure 20
Interface of a process referencing a reusable definition

Figure 21 is an instance diagram of level 2 of Figure 20.

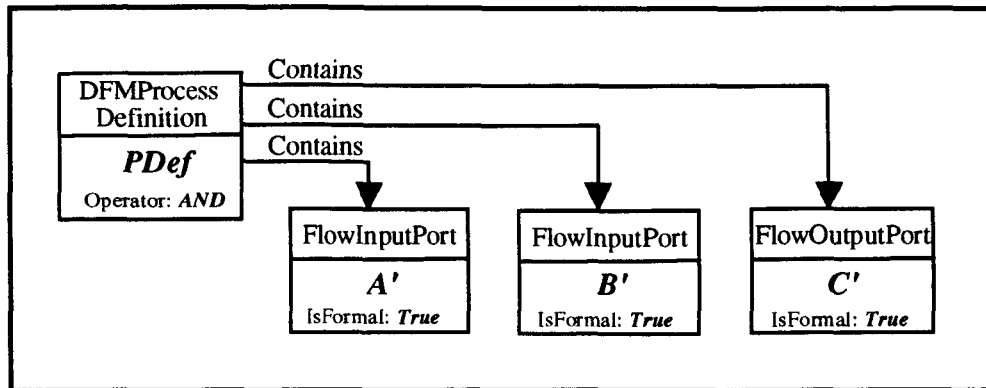


Figure 21
FlowPorts describing the formal interface of a *DFMPProcessDefinition*

Those *FlowPorts* contained in a *DFMPProcessDefinition* which have set their meta-attribute *IsFormal* to **False** form the actual interface of a process.

If a *FlowPort* describes the actual input of a process, or a store, or an external agent, the *FlowInputPort* meta-entity shall be used. In case of an output, *FlowOutputPort* shall be used. A flow port which is not directed (i.e., a bidirectional interface) is indicated by instantiating *FlowPort* directly.

For every *Process*, *Store* and *ExternalAgent* referencing a *DFMPProcessDefinition*, *StoreDefinition* or *ExternalAgentDefinition*, the formal and actual *FlowPorts* have to be bound to each other.

For binding the *FlowPorts*, meta-entity *EquivalenceSet* is used. The *EquivalenceSet* is contained in the *DefinitionObject* which contains the *Process*, *Store* or *ExternalAgent* whose interface is described. The *EquivalenceSet* contains the actual *FlowPort*, and a reference to the formal *FlowPort*.

The general structuring mechanism requires that a *ReferencedElement* is used every time a decomposition level is crossed, so instead of directly using the formal *FlowPort* as member of the *EquivalenceSet*, a *ReferencedElement* references the formal *FlowPort*. This *ReferencedElement* becomes a member of the *EquivalenceSet*.

Figure 22 is an instance diagram of both levels of figure 20. It describes the context for a process along with its process definition, which consumes two flows and produces one flow.

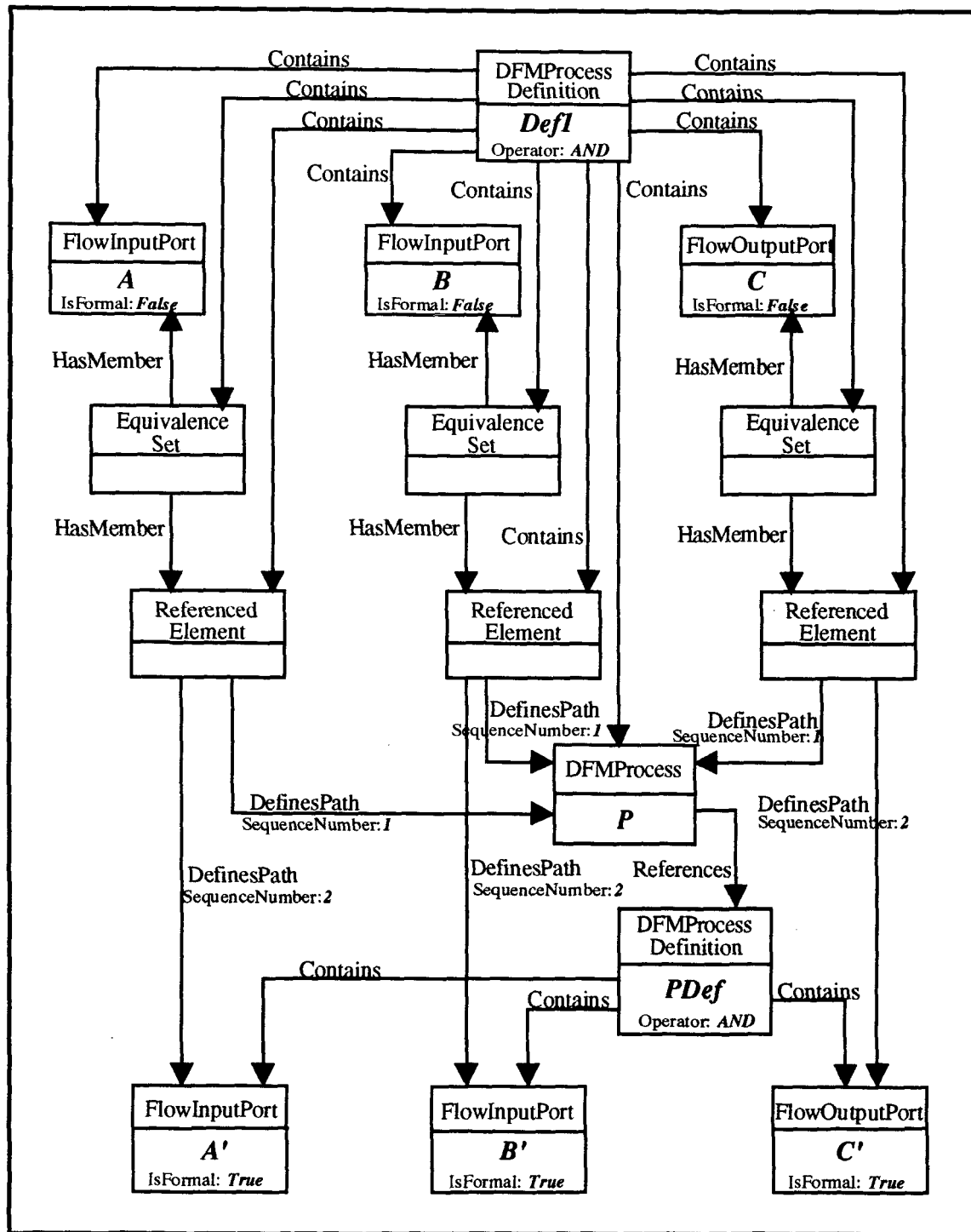


Figure 22
Interface of a Process using FlowPorts

A complete *DFMProcessDefinition* requires that *FlowPorts* are used to define the formal interface of the definition to the outside. So the *DFMProcessDefinition* contains formal *FlowPorts* which correspond to the external interface of the definition (i.e., there is one formal *FlowPort* for every external input, output, or bidirectional channel). Formal *FlowPorts* may reference *FlowDefinitions* or *DataTypes* to indicate the structure of information they can exchange, and they may be bound to multiple actual *FlowPorts* by use of separate *ReferencedElements* for each binding.

4.1.17 Controls, Constraints and Mechanisms

In some function modeling techniques, flows entering a function represent the constraints or conditions required to produce the correct output. When a *Flow* is a constraint for a function this is established by using the meta-entity *ConstraintPort* as the input port rather than the *FlowInputPort* meta-entity.

Input flows can also represent the mechanism or means used to perform a function. This is accomplished by using the meta-entity *SupportPort* as the input port rather than the *FlowInputPort* meta-entity.

A *Flow* entering a *DFMProcess* or *Store* through a *ControlPort* must be a control flow, i.e., a *Flow* whose *IsControl* meta-attribute is set to **True**. This subject area does not support the detailed description of the kind of operations applied to the *DFMProcesses* or *Stores* that are found in some real-time oriented data flow modeling techniques.

4.1.18 Real-Time Extensions

DFMProcessDefinitions, *StoreDefinitions* and *FlowDefinitions* carry a meta-attribute *IsControl* which indicates whether the *DFMProcesses*, *Stores* or *Flows* referencing the definition process, store or carry control information. This is needed by some techniques which extend data flow modeling to cover real time information.

Meta-entity *DFMProcessDefinition* provides meta-attribute *HasRealTimeSemantics* which identifies those *DFMProcessDefinitions* that use concepts usually found in real-time extensions to data flow modeling.

For a process which references a *DFMProcessDefinition* which has real time semantics, the executing state can be influenced by control signals that are sent to the process using control flows.

4.1.19 Computable Languages

The set of computable languages that can be supported is defined in Section 4 of *EIA/IS-112 CDIF - Integrated Meta-model - Common Subject Area*.

4.1.20 Formats for Computable Values

The supported formats for computable values are defined in Section 4 of
EIA/IS-113 CDIF - Integrated Meta-model - Data Definition Subject Area.

4.2 Diagrams

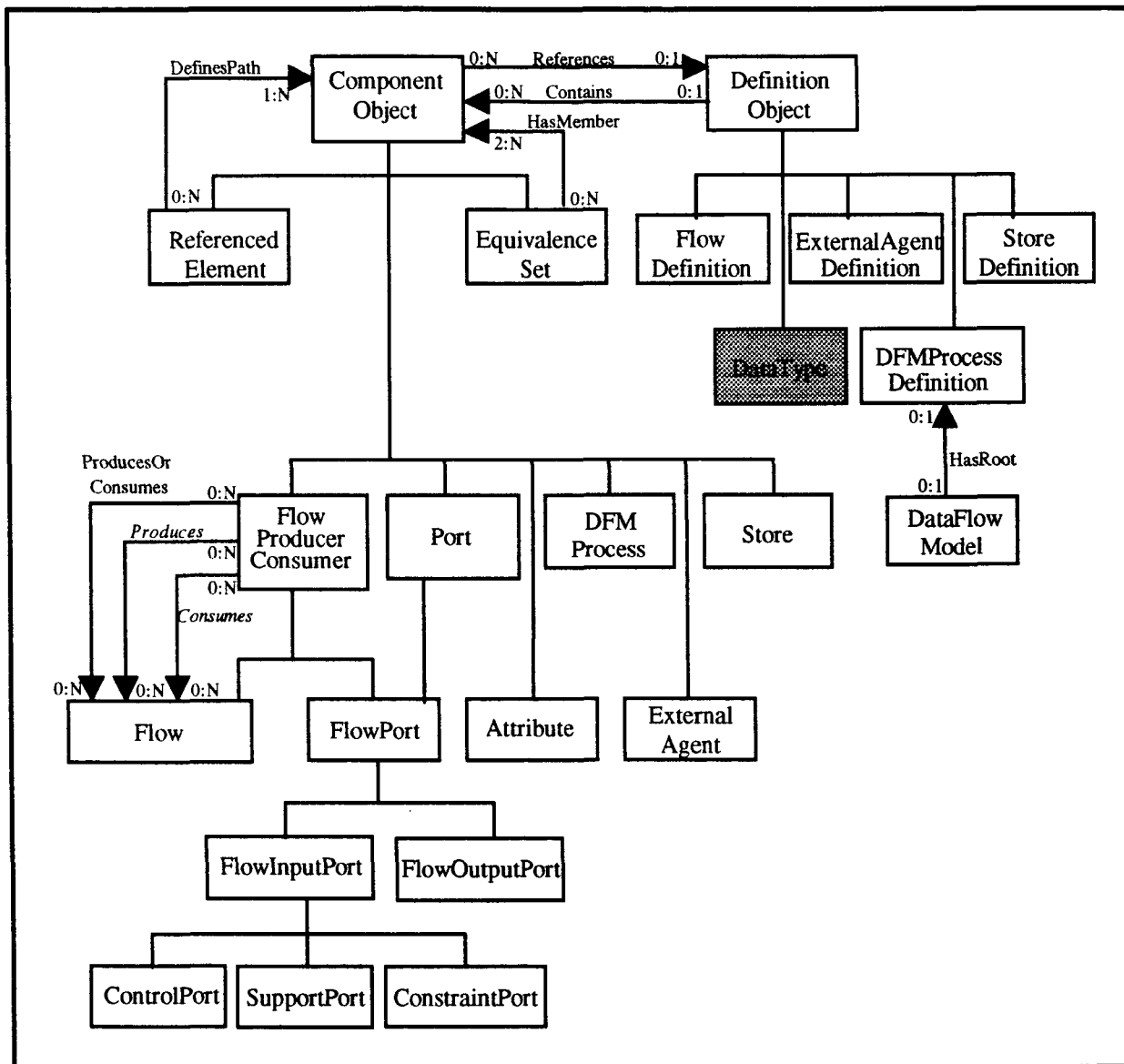


Figure 23
 Data Flow Model Subject Area – Diagram

4.3 AttributableMetaObject Hierarchy

MetaObjectName	Subject Area
RootObject	not in this SA
RootEntity	not in this SA
SemanticInformationObject	not in this SA
ComponentObject	in this SA
Attribute	in this SA
DFMProcess	in this SA
EquivalenceSet	in this SA
ExternalAgent	in this SA
FlowProducerConsumer	in this SA
Flow	in this SA
FlowPort	in this SA
FlowInputPort	in this SA
ConstraintPort	in this SA
ControlPort	in this SA
SupportPort	in this SA
FlowOutputPort	in this SA
Port	in this SA
FlowPort	in this SA
FlowInputPort	in this SA
ConstraintPort	in this SA
ControlPort	in this SA
SupportPort	in this SA
FlowOutputPort	in this SA
ReferencedElement	in this SA
Store	in this SA
DataFlowModel	in this SA
DefinitionObject	in this SA
DFMProcessDefinition	in this SA
ExternalAgentDefinition	in this SA
FlowDefinition	in this SA
StoreDefinition	in this SA
RootEntity.IsRelatedTo.RootEntity	not in this SA
ComponentObject.References.DefinitionObject	in this SA
DataFlowModel.HasRoot.DFMProcessDefinition	in this SA
DefinitionObject.Contains.ComponentObject	in this SA
EquivalenceSet.HasMember.ComponentObject	in this SA

MetaObjectName	Subject Area
RootObject	not in this SA
RootEntity.IsRelatedTo.RootEntity	not in this SA
FlowProducerConsumer.ProducesOrConsumes.Flow	in this SA
FlowProducerConsumer.Consumes.Flow	in this SA
FlowProducerConsumer.Produces.Flow	in this SA
ReferencedElement.DefinesPath.ComponentObject	in this SA

4.4 MetaEntity Summary

MetaEntity Name

Inherited MetaAttribute Name

Local MetaAttribute Name

Mandatory/Optional

Attribute

CDIFIdentifier

Mandatory

DateCreated

Optional

DateUpdated

Optional

TimeCreated

Optional

TimeUpdated

Optional

BriefDescription

Optional

FullDescription

Optional

DefaultValue

Optional

IsOptional

Optional

Name

Optional

ComponentObject

CDIFIdentifier

Mandatory

DateCreated

Optional

DateUpdated

Optional

TimeCreated

Optional

TimeUpdated

Optional

BriefDescription

Optional

FullDescription

Optional

ConstraintPort

CDIFIdentifier

Mandatory

DateCreated

Optional

DateUpdated

Optional

TimeCreated

Optional

TimeUpdated

Optional

BriefDescription

Optional

FullDescription

Optional

IsFormal

Mandatory

Name

Optional