



TECHNICAL REPORT

GEIA-HB-927

Issued 2007-07

Implementation Guide for Common Data Schema for Complex Systems

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 © 2013 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 GEIAHB927

SAE values your input. To provide feedback on this Technical Report, please visit <http://www.sae.org/technical/standards/GEIAHB927>

Documents

Document Sharing Hub

TechAmerica Engineering Bulletin

Implementation Guide for Common Data Schema for Complex Systems

GEIA-HB-927

July 2007



NOTICE

TechAmerica Engineering Standards and Publications are designed to serve the public interest by eliminating misunderstandings between manufacturers and purchasers, facilitating interchangeability and improvement of products, and assisting the purchasers in selecting and obtaining with minimum delay the proper product for their particular needs. Existence of such Standards and Publications shall not in any respect preclude any member or nonmember of TechAmerica 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 TechAmerica members, whether the standard is to be used either domestically or internationally.

Standards and Publications are adopted by TechAmerica in accordance with the American National Standards Institute (ANSI) patent policy. By such action, TechAmerica does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the Standard or Publication.

Technical Publications are distinguished from TechAmerica Standards in that they contain a compilation of engineering data or information useful to the technical community and represent approaches to good engineering practices that are suggested by the formulating committee.

This Bulletin is not intended to preclude or discourage other approaches that similarly represent good engineering practice, or that may be acceptable to, or have been accepted by, appropriate bodies. Parties who wish to bring other approaches to the attention of the formulating committee to be considered for inclusion in future revisions of this publication are encouraged to do so. It is the intention of the formulating committee to revise and update this publication from time to time as may be occasioned by changes in technology, industry practice, or government regulations, or for other appropriate reasons.

(Formulated under the cognizance of the TechAmerica Systems Engineering Committee, G-47.)

Published by

© 2011 TechAmerica
Standards & Technology Department
601 Pennsylvania Ave., NW
North Building, Suite 600
Washington, DC 20004-2650
All rights reserved
Printed in U.S.A.

PLEASE!
DON'T VIOLATE
THE
LAW!

This document is copyrighted by TechAmerica and may not be reproduced without permission.

Organizations may obtain permission to reproduce a limited number of copies by entering into a license agreement with our distributors.

For distributor information please see our web site www.techamerica.org/ or contact TechAmerica at 703-284-5355

TechAmerica

**Implementation Guide for Common Data Schema
for Complex Systems**

GEIA-HB-927

Revision	Description of change	Date
-	Initial Release	Jul 2007

TABLE OF CONTENTS

TABLE OF CONTENTS	I
TABLE OF FIGURES	III
1 PREFACE.....	1
1.1 The Problem	1
1.2 Approach to Problem Solution	2
2 INTRODUCTION	2
3 DATA MODELING GUIDELINES.....	3
3.1 Scope.....	4
3.2 General Requirements.....	4
3.3 The Six Principles	5
3.4 Naming.....	8
3.5 Specialization.....	8
3.6 Identity	8
3.7 Capitalization	9
3.8 Use of EXPRESS elements	9
3.9 Normalization.....	17
3.10 Documentation	17
3.11 EXPRESS-G Diagrams	19
3.12 Data Modeling Guidelines References	23
4 DATA MODEL USAGE GUIDE.....	24
4.1 Modeling Philosophy	24
4.2 GEIA-STD-927 DATA MODEL CONCEPTS.....	27
4.3 Model Usage Guide References	43
4.4 Acronyms.....	43
5 INTEGRATION PROCEDURE	44
5.1 Purpose.....	44
5.2 Procedure Model.....	44
5.3 Description.....	46
5.4 Roles	46
5.5 Inputs	46
5.6 Entry Criteria.....	46
5.7 Activities to Be Performed	46
5.8 Outputs.....	59

5.9	Exit Criteria.....	60
5.10	Policies.....	60
5.11	Standards	60
5.12	Measures	60
5.13	Training	60
5.14	Tools	60
5.15	Tailoring.....	61
5.16	Acronyms	61
5.17	Integration Procedure Terms and Definitions	61
5.18	Template Forms	62
5.19	Disposition Definitions.....	66
5.20	Integration Action Decision Matrix (Element Disposition)	66
5.21	Analysis – Instructions & Examples	67
5.22	Augmentation Material – Instructions & Examples.....	79
5.23	Mapping Table – Instructions & Examples.....	94
6	SCHEMA TAILORING GUIDELINES.....	111
6.1	Scope.....	111
6.2	Referenced Documents	111
6.3	Process Overview	111
6.4	Process Steps.....	114
6.5	Practical Application	120
6.6	Template Form.....	126

TABLE OF FIGURES

Figure 1 - Subtypes should be below the supertype and spaced evenly	21
Figure 2 - Subtypes should be balanced to the left and right of the supertype	22
Figure 3 - Separation of information representation from thing	25
Figure 4 - Use of ECM4 with Reference Data Library	26
Figure 5 - Representation of “pump” in Application Model	27
Figure 6 - Representation of “pump” in the GEIA-STD-927 Data Model	27
Figure 7 - Assigning a Description	29
Figure 8 - Assigning a Name	30
Figure 9 - Assigning an Identification	32
Figure 10 - Assigning a Version Identification	33
Figure 11 - First classification of a “process_activity” entity	36
Figure 12 - Second classification of a “process_activity” entity	37
Figure 13 - Composition relationship for a "realized_system" entity	39
Figure 14 - Composition and replacement relationships for a “system_instance” entity with relation to other entities and effectivity	42
Figure 15 - GEIA-STD-927 Integration Procedure Flow Diagram	45
Figure 16 – Data Source Specific Process Diagram	113
Figure 17 - Conceptual Data Model fragment	119
Figure 18 - Data Source Specific (Denormalized) Model fragment	120
Figure 19 - Logical Data Model for CDDB Tailoring	123
Figure 20 - Extended Abstract Model for CDDB Tailoring	124

1 Preface

1.1 The Problem

Enterprises that acquire and sustain complex modern systems face unprecedented challenges in containing costs while taking their systems through concept, design, development, deployment, and retirement. The schema and associated data set required for specifying, developing, operating, maintaining, and disposing of such systems is extremely large and involves myriad subtle relationships among seemingly disparate domains. Added to this complex reality, industry and government must develop, deliver, and manage systems meeting aggressive customer requirements and shifting mission objectives within stringent budget constraints. Affordability has become as important as mission performance when developing and sustaining such systems.

Similarly, product customization to satisfy specialized customer requirements, time-to-market, and affordability has become as important to the industrial world as product performance is for commercial complex systems. Commercial companies are increasingly operating in a virtual extended enterprise environment striving to share selective information with their disparate distributed stakeholders. Therefore the industrial approach to solving these problems through standards and partnering activities can be used as a model for the Department of Defense (DoD).

Military weapon system program offices, like industry, must be able to define technical or fiscal metrics to assess the Total Ownership Cost (TOC) of their systems. Traditional system acquisition and life cycle management practices include the use of automated tools for modeling and simulation, configuration management, and supply support systems to create and manage technical data for systems both in development and in the field. However, each tool uses its own data representation and storage mechanism causing major problems in communicating between systems.

With a few exceptions, mostly in the commercial industrial world, no real interoperability exists among tools, even for exchanging data concerning the same technical area of the system. Human operators most often re-enter data for each tool employed in the process. Interoperability problems grow with automated support. As program management offices use advanced process modeling and planning techniques, and work with complex sets of data across multiple databases, this “tower of Babel” increases. This presents a significant and growing challenge to programs to effectively integrate complex system data.

There is significant commonality of interest between DoD and U.S. industry in trying to achieve a high degree of interoperability among their Information Technology (IT) systems for complex engineer-to-order systems, products and processes over their life cycle. Success in so doing minimizes the number and cost of transactions and results in a lean more affordable operating mode for all involved. The path to this objective (and the specific objective) of the program involves the use of:

- Existing commercial enterprise software and standards
- A single schema for seamless integration of broad and varied data sets
- A framework or architecture for the communication and access to this data, over the life cycle of the system or product involved

1.2 Approach to Problem Solution

Meeting the challenge to effectively integrate complex system data is a key to ensuring that DoD complex systems (such as an Army Tank, a Navy Destroyer, and an Air Force Fighter) and commercial complex systems (such as wide-body airplanes, automobiles, trucks, ships, trains, offshore oil rig platforms, satellites, etc.) are both mission and performance effective and affordable. The response to the challenge involves three principal elements:

1. The first element is an organization of the system data through an integrated multi-domain data schema for representing system product and process data. This will be essential to developing and operating an advanced integrated environment.
2. The second element is an integrated environment that employs formal methods and automation to support the full range of data manipulation and communication required by complex system lifecycle activities. This environment will enable a broad spectrum of life cycle participants to evaluate alternatives in multiple domains simultaneously; provide a way for stakeholders to understand their needs in relation to the enterprise as a whole; and provide a continuous proactive means of identifying and successfully addressing key challenges for a complex system over time.
3. The third element is an evolved culture where enterprise-wide cooperation is the rule and individual contributions are encouraged and efficiently managed.

Working within an integrated environment based on these three elements will provide a common frame of reference in which sophisticated relationships across technical domains, and between these domains and a system's affordability, can be explicitly identified and analyzed. The key to realizing gains from the combined elements is the data schema, which is essential to integrating the disparate data sets in use by complex system program offices and other related enterprises.

2 Introduction

The purpose of this handbook is to provide the data modeler with the modeling conventions, philosophy and guidelines used during development of the GEIA-STD-927 schema; and the integration process used to integrate existing best-in-class standard data models into the GEIA-STD-927 schema. Most importantly, the handbook provides tailoring guidelines for the data modeler to use in applying the GEIA-STD-927 schema and mapping tables to a practical application for an existing program. The Handbook is organized into four sections:

- Data Modeling Guidelines,
- Data Model Usage Guide,
- Integration Procedure, and
- Schema Tailoring Guidelines.

The Data Modeling Guidelines establish modeling conventions and guidelines followed during the development of the GEIA-STD-927 schema. The section was expanded as a result of lessons learned during the project, liaisons with other groups developing schemas, and other input into the program. The Guidelines present general requirements for schema development, a set of principles to be followed, conventions to be used when naming data concepts, the use of supertype/subtype hierarchies, a recommended use of the Express

language constructs, and good graphical layout practice. It is intended to provide guidance for maintaining the Common Data Schema and for schema development efforts.

The purpose of the Data Model Usage Guide is to enable the user of the GEIA-STD-927 Data Model to understand how their legacy data source information fits into the GEIA-STD-927 Data Model. After a brief overview of the modeling philosophy, this section presents various data model concepts and how they are approached. For each of these concepts, this section discusses typical representations in other models, the representation in the GEIA-STD-927 Schema, and related concepts.

The Integration Procedure section describes the activities used to integrate best-in-class standard data models into the GEIA-STD-927 Common Data Schema. An overview of the process is followed by detailed descriptions of each step of the work. A formal description approach is used, with a common template being used for all process steps. Key concepts in each Activity description include Roles to be filled, Inputs to the Activity, the Implementation steps and the Activity Outputs. The procedures defined in this section are used to perform the schema integrations and are referenced in the following section on schema tailoring.

[Section 6](#), Schema Tailoring Guidelines, provides a strategy whereby the GEIA-STD-927 Schema can be transitioned to practical application for an existing DoD program. The Schema Tailoring Guidelines address identifying data sources and the logical concepts contained within those sources; creating a form of Logical Data Model to support each data source; identifying areas where the GEIA-STD-927 Schema is relevant and performing the analysis necessary to map the data sources to the Schema; creating an Extended Abstract Model that takes into account all of the concepts represented by the individual data sources; and modifying the form of the GEIA-STD-927 Schema as necessary to facilitate the implementation of the Schema with a practical software application.

3 Data Modeling Guidelines

A good data model should have the following attributes:

- completeness;
- non-redundancy;
- data reusability;
- stability;
- flexibility;
- simplicity;
- elegance;
- extensibility;
- effectiveness as a communication tool.

This section provides some guidance in achieving the above attributes, but it ultimately is up to the modeler to ensure that they are met.

The examples that appear in this section were chosen to illustrate the points that precede them. Some of these examples may, for simplicity, violate other points in the document. Therefore, an understanding of this entire section, an understanding of decisions made during analysis (which is captured in the analysis documentation form) and familiarity with the existing schema are all necessary to achieve a proper modeling style.

Conventions Used in this Section

EXPRESS code is displayed in the Courier font. Ellipses (...) are used when portions of an EXPRESS definition are irrelevant to the example and therefore omitted.

Intended Audience

This section is intended primarily for use by data modelers who are involved in developing or tailoring the GEIA-STD-927 top level schema. It may also be of interest to system architects, software developers and project managers. Familiarity with the fundamentals of data modeling and an understanding of the EXPRESS modeling language are assumed.

3.1 Scope

This section establishes modeling conventions and guidelines to be followed during the development of the GEIA-STD-927 top level schema.

The following is within the scope of this document:

- Modeling rules, conventions and guidelines for the GEIA-STD-927 top level schema.

The following are outside the scope of this document:

- An introduction to information modeling;
- An introduction to the EXPRESS modeling language.

3.2 General Requirements

3.2.1 Perspective of the Schema

The perspective of the schema shall be the specification of the structure of data needed to represent useful and practical concepts within the GEIA-STD-927 domain, not a description of the structure of the real world.

EXAMPLE 1: Suppose we wish to model the concepts of "person," "male" and "female." We can write the following in EXPRESS:

```
ENTITY person
ABSTRACT SUPERTYPE OF (ONEOF(male, female));
...
END_ENTITY;

ENTITY male
SUBTYPE OF (person);
...
END_ENTITY;

ENTITY female
SUBTYPE OF (person);
...
END_ENTITY;
```

By declaring person to be an abstract supertype, we enforce the rule that every person is either male or female. This is true in the real world, but there may be cases where we do not know or care about a person's sex, but still want to store information about that person in the GEIA-STD-927 data store. The above declarations would make that impossible. So it is better to remove the "ABSTRACT" keyword from the person entity declaration.

EXAMPLE 2: The schema for a chemical database might consist of ATOM and MOLECULAR_BOND entity data types. We know from physics that atoms actually are made up of subatomic particles (electrons, protons, neutrons, etc.). However, for the purpose of a chemical database, it may be more useful to represent methane as four hydrogen atoms bonded to one carbon atom, rather than a configuration of m electrons, n neutrons and p protons.

3.2.2 Understandability to Domain Experts

The schema should be understandable to domain experts. Terms from information modeling, computer science, information technology, database technology, etc., should be avoided.

NOTE: This does not rule out the need to brief the domain experts on the EXPRESS modeling language or provide them with human or written guidance on how to read the schema.

3.2.3 Generality

The entities in the schema should be generic enough so the schema context is as broad as possible while still meeting requirements. However, they should not be so generic as to be meaningless.

EXAMPLE 1: A person may be a child of another person, a spouse of another person, an employee of another person, the teacher of another person, etc. These relationships could all be made subtypes of a single relationship called "person_person_relationship"; however, such a generalization would not be useful because person_person_relationship carries no meaning by itself.

A rule shall be embedded in the schema if and only if one of the following holds:

- The rule is necessary to ensure the integrity or meaningfulness of the data;
- There is some other clear benefit to having the rule and the rule is likely to remain in force throughout the life of the GEIA-STD-927 schema.

NOTE 1: Another way of thinking about this is: Can you imagine a possible world in which this rule does not hold? If you can, it may be better not to embed it in the schema.

EXAMPLE 2: The rule "An employee must have a home phone number" probably does not meet the first condition. One might think that it meets the second criterion, but some employees may not have a true home phone number only a cell phone number. So it may be best to leave this rule out of the schema and enforce it through some other means.

NOTE 2: There is no specified design life for the GEIA-STD-927 program and its schema.

NOTE 3: Business rules for a particular organization may be stored as data (in which case a meta-business rule schema is needed) or embedded in the software that makes updates to a GEIA-STD-927 data store.

3.3 The Six Principles

The following six principles shall be observed.

NOTE 1: These principles are adapted from "Developing high quality data models." The reader should consult that document for details, explanations, and additional examples.

Principle 1: Candidate attributes should be analyzed to determine whether they represent relationships to other entity data types.

EXAMPLE 1: Consider the following EXPRESS code:

```
ENTITY engine;
  vehicle_id: OPTIONAL STRING; (* the identifier of the vehicle
                                on which the engine is currently
                                mounted, or null if it is not
                                mounted to a vehicle *)
  ...
END_ENTITY;

ENTITY vehicle;
  vehicle_id: STRING; (* the unique identifier of the vehicle *)
  ...}
UNIQUE
  UR1: vehicle_id;
END_ENTITY;
```

There is a relationship between engine and vehicle implicit in the shared attribute vehicle_id. The vehicle_id attribute should be removed from engine and replaced with a link. The following EXPRESS code represents this concept:

```
ENTITY engine;
  mounted_on: OPTIONAL vehicle; (* the vehicle on which the engine
                                is currently mounted, or null if
                                it is not mounted to a vehicle *)
  ...
END_ENTITY;

ENTITY vehicle;
  vehicle_id: STRING; (* the unique identifier of the vehicle *)
  ...
UNIQUE
  UR1: vehicle_id;
END_ENTITY;
```

EXAMPLE 2: A product has a price attribute, which is represented in EXPRESS as follows:

```
ENTITY product;
  product_code: STRING;
  price: currency_amount;
  ...
END_ENTITY;
```

However, a product's price may change over time and there may be different prices for different markets, so the following EXPRESS code would be a better representation of this concept:

```
ENTITY product;
  product_code: STRING;
  ...
END_ENTITY;

ENTITY product_price;
  the_product: product;
  the_market_segment: market_segment;
  price: currency_amount;
END_ENTITY;
```

NOTE 2: This principle is worded as follows in Developing High Quality Data Models: "Candidate attributes should be treated as representing relationships to other entity types."

Principle 2: Entities and records should have an internal identifier within a database or exchange file. These should be artificial and managed to be unique. Relationships should not be used as part of the internal identifier.

NOTE 3: "Natural" identifiers represent data that have meaning to a domain expert, exist independently of the GEIA-STD-927 schema, or that must be maintained as part of an object's description.

Principle 3: Activities and associations between domain objects should be represented by EXPRESS entity data types (EDTs), not by EXPRESS relationships or attributes of EDTs.

Principle 4: EXPRESS relationships should only be used to express the involvement of entity data types with activities or associations.

Principle 5: Entity data types should represent, and be named after, the underlying nature of objects, not the roles they play in a particular context.

EXAMPLE: It is not good to create a subtype of the person entity data type called "mother," since the same person can be the mother to someone, the daughter of someone else, and the sister of a third person.

Principle 6: Entity data types within a model should be part of a single sub-type/super-type hierarchy of generic entity data types in order to define a universal context for the model, and to avoid duplication of concepts and data.

Notwithstanding the above, EDTs should not be so generic as to be meaningless or almost meaningless.

3.4 Naming

For the purpose of this clause, "EXPRESS identifier" or "identifier" refers to names given to items that are declared in a schema, as defined in Clause 7.4 of ISO 10303-11:1994.

Except as noted below, each EXPRESS identifier shall consist of one or more English words, separated by underscores.

EXAMPLE 1: Names of schemas, entities, attributes, user-defined types, constants, functions and rules are kinds of identifiers.

There shall be no leading or trailing underscores in an identifier, and there shall be exactly one underscore separating adjacent words. Each word of an identifier shall be in lower case. Schema, entity and attribute names shall be in the singular.

EXAMPLE 2: Person, organization, document are valid words as opposed to persons, organizations, documents.

Each identifier shall be no more than 80 characters long. Use of acronyms and abbreviations should be avoided. All acronyms and abbreviations should follow known conventions for the area being modeled.

3.5 Specialization

The proper use of specialization (subtype/supertype) hierarchies can greatly simplify the schema, and make it easier to understand. However, very deep hierarchies tend to be cumbersome and make the schema more difficult to understand.

Given two entity data types a, b, ..., the following factors support creating a common supertype from which they inherit:

- Domain experts believe that the entities are specialized from the same general concept;
- The entities a, b, ... share attributes, rules or other characteristics that are captured in the schema;
- There may be situations where it is known that an object is one of a or b, ... but it is not known which one it is.

EXAMPLE: It may be known that an object is a person, but not whether the person is a male or female.

Multiple inheritance should be used sparingly. Uses of multiple inheritance will be scrutinized carefully during validation and verification reviews of the schema.

3.6 Identity

EXPRESS considers each instance of an EDT to have an "identity" separate from all other instances. Two separately created instances retain separate identities even if all their attributes have identical values. This feature of the language, while quite powerful, can be confusing to people who are reading or implementing the schema. Therefore, it is best to avoid the situation where two instances of an EDT:

- have the same values for all attributes;
- are not interchangeable.

Actually, the situation where two instances of an EDT are interchangeable in all circumstances should also be avoided, since it represents redundancy.

3.7 Capitalization

All EXPRESS identifiers shall be presented in lower case, with the exception that acronyms shall be in upper case.

NOTE 1: EXPRESS identifiers include entity names, attribute names, constant names, rule names, function names, and variable names.

EXAMPLE: The following are valid entity names: product, product_definition, product_process_association, CAD_system. The following are invalid entity names: Product, Product_Definition, ProductDefinition, cad_system.

The EXPRESS language is case-insensitive. Under no circumstances shall case alone be used to distinguish between two different identifiers.

3.8 Use of EXPRESS elements

This clause contains guidance on use of specific elements of the EXPRESS modeling language.

3.8.1 Schema

The name of the schema shall be “GEIA-STD-927 ver x.x,” where x.x represents the version identification of the schema at the time the name is assigned.

3.8.2 Constants

Constants shall be used for the following purposes:

- Defining a "configuration option" for the GEIA-STD-927 schema;

NOTE 1: It is not clear if it will be necessary to use constants for this purpose.

- Giving a name to an arbitrary-appearing value that has some conceptual significance and never changes;

EXAMPLE: Mathematical constants such as e and π would qualify under this case. Physical constants might qualify if it is strongly believed that they will not change during the life cycle of the GEIA-STD-927 schema.

NOTE 2: While physical constants conceptually never change, the values may be revised due to improvements in man's ability to measure them, or other scientific advances.

- Giving a name to a value that appears more than once in the schema.

Since constants are conceptually values that never change, they should not be used for values that change infrequently, but do change. They should also not be used for values that do not change for a given set of users but may be different for other sets of users, since many different groups of users may be operating off of the same version of the GEIA-STD-927 schema.

Changing the value of a constant in a constant declaration in the schema results in a new “unique” version of the schema, as does changing an entity declaration, a function definition, or any other element of the schema.

Constants do not appear in ISO 10303-21 [3] exchange files; therefore, use of a constant in an exchange context based on ISO 10303-21 requires that both sender and receiver have copies of the schema in which the constant is defined.

3.8.3 Named Types

Named types shall be used for the following purposes:

1. Defining SELECT and ENUMERATION types;

NOTE 1: The EXPRESS language requires that SELECT and ENUMERATION types be defined as named types.

2. Giving names to collection types where such collections have conceptual significance and appear in more than one place in the schema;

EXAMPLE 1: A real-valued three-dimensional vector may be defined in EXPRESS as follows:

```
TYPE real_vector_3d = ARRAY [1:3] OF REAL;
END_TYPE;
```

3. Giving names to simple types where such names have conceptual significance and appear in more than one place in the schema;

EXAMPLE 2: Assume that several attributes of the same or different entities are length measures and are defined to be real numbers, as shown below:

```
ENTITY pipe;
  diameter: REAL;
  length: REAL;
END_ENTITY;
```

```
ENTITY duct;
  width: REAL;
  height: REAL;
  length: REAL;
END_ENTITY;
```

Then those attributes should instead point to a defined type for length measure, as shown below.

```
TYPE length_measure = REAL;
END_TYPE;
```

```
ENTITY pipe;
  diameter: length_measure;
  length: length_measure;
END_ENTITY;
```

```
ENTITY duct;
  width: length_measure;
  height: length_measure;
  length: length_measure;
END_ENTITY;
```

NOTE 2: The definition "TYPE length_measure = REAL" ignores the fact that a length measure must have units associated. The above declarations would be valid only if somewhere else in the model, a global units context is defined.

4. Giving names to subsets of simple types where such subsets have conceptual significance and appear in more than one place in the schema;

EXAMPLE 3: Assume that several attributes of the same or different entities are counts and are defined to be positive integers, as shown below.

```
ENTITY physical_object;
  number_of_cycles: INTEGER;
WHERE
  WR1: number_of_cycles >= 0;
END_ENTITY;

ENTITY vehicle;
  number_of_seats: INTEGER;
WHERE
  WR1: number_of_seats >= 0;
END_ENTITY;
```

Then those attributes should instead point to a defined type for positive integer, as shown below.

```
TYPE positive_integer = INTEGER;
WHERE
  WR1: SELF >= 0;
END_TYPE;

ENTITY physical_object;
  number_of_cycles: positive_integer;
END_ENTITY;

ENTITY vehicle;
  number_of_seats: positive_integer;
END_ENTITY;
```

5. Giving names to subsets of other named types where such subsets have conceptual significance and appear in more than one place in the schema.

EXAMPLE 4: Suppose that the type positive_real is already defined as follows:

```
TYPE positive_real = REAL;
WHERE
  WR1: SELF >= 0;
END_TYPE;
```

Assume that several attributes of the same or different entities are distance values and point to type positive_real. Then those attributes should point to the following defined type for distance instead:

```
TYPE distance = positive_real;
END_TYPE;
```

NOTE 3: Distance may be more appropriately modeled as a length_measure entity.

3.8.4 SELECT Types

SELECT types are similar to supertypes.

EXAMPLE: Consider the following declarations:

```
ENTITY a;  
...  
END_ENTITY;  
  
ENTITY b;  
...  
END_ENTITY;  
  
TYPE sel = SELECT(a,b);
```

The declaration for sel is roughly equivalent to:

```
ENTITY sel;  
ABSTRACT SUPERTYPE OF (ONEOF(a,b));  
END_ENTITY;
```

One major difference between SELECT types and supertypes is that one cannot declare subtypes of SELECT types.

SELECT types should be used sparingly.

A SELECT type may be appropriate if:

1. The members of the SELECT type already appear in separate subtype/supertype hierarchies, and changing the SELECT type to an abstract supertype would result in multiple inheritance;
2. The members of the select type do not share any conceptual similarities worthy of putting them into a subtype/supertype hierarchy;

EXAMPLE: Consider the following declaration:

```

TYPE documentable_item = SELECT (system, design, shape_aspect,
tolerance, process);
END_TYPE;

ENTITY documentation;
  item_documented: documentable_item;
  documentation:   STRING;
END_ENTITY;

```

The entities system, design, shape_aspect, tolerance and process probably do not have enough in common to warrant making them subtypes of a common supertype.

3. Some members of the select type are not EDTs.

NOTE: EXPRESS requires that all members of a subtype/supertype hierarchy be EDTs.

3.8.5 Subtypes and Supertypes

A supertype X shall not be declared as abstract unless every instance at X falls into at least one known subtype of X. Abstract supertypes shall not be used to enforce business rules.

EXAMPLE: See the example in Clause 3.2.1.

Care should be taken to properly specify the constraints on the subtypes.

NOTE: The ONEOF operator is often overlooked.

3.8.6 Simple Datatypes

Simple datatypes shall not contain structured information, except as specifically called for in the data architecture.

EXAMPLE 1: The following is wrong:

```

person;
  name: STRING; (* should be in the format last name, comma,
                 space, first name, space, middle name *)
END_ENTITY;

```

Instead, the attribute name should be broken down into its constituents.

```

ENTITY person;
  last_name:   STRING;
  first_name:  STRING;
  middle_name: OPTIONAL STRING;
END_ENTITY;

```

EXAMPLE 2: The data architecture may specify that dates are to be represented as strings in accordance with ISO 8601 [\[1\]](#).

3.8.7 Collection Datatypes

Arrays are often used to represent mathematical concepts such as vectors and matrices. If there is a need to associate additional information with a concept beyond the array contents, then an EDT should be used, with the array being one of the EDT's attributes.

3.8.8 Entities

The entity declaration is the workhorse of EXPRESS. Most concepts in the schema should be captured as EDTs. An EDT shall not represent an individual. An EDT should represent a class with a reasonable number of members.

EXAMPLE 1: `steve_olson` would not be a good EDT, but `person` would be.

Attributes of entities shall be listed in alphabetical order. Only one attribute shall be defined in an attribute definition statement.

EXAMPLE 2: The following is wrong:

```
ENTITY ent;
  c, p: REAL;
END_ENTITY;
```

Instead, write:

```
ENTITY ent;
  c: REAL;
  p: REAL;
END_ENTITY;
```

3.8.9 Functions and Procedures

A function or procedure should be defined whenever the same code, or code that is conceptually similar appears in more than one place in the schema.

Function calls should also be substituted for subexpressions in complex nested expressions.

NOTE: In traditional EXPRESS usage (e.g., in the Application Interpreted Model of a typical STEP Application Protocol), very complex nested expressions occur abundantly in WHERE clauses.

3.8.10 Rules

EXPRESS has two kinds of rules: local rules and global rules. Local rules are defined in the WHERE clause of an ENTITY declaration. Global rules are defined with the RULE statement.

Local and global rules shall only be used to express constraints that cannot be easily expressed using cardinalities in forward or inverse attribute declarations. Local rules shall be used instead of global rules wherever possible.

NOTE: If a constraint applies only to a subset of the members of a class defined by an EDT, one can declare a subtype to represent the members of the subset, and then write the constraint as a local rule on that subtype.

Rule names shall summarize the meanings of the rules and shall conform to Clause 5.0.

EXAMPLE: `"rule27"` and `"axiom_A"` are not valid rule names.

3.8.11 Attributes

Attribute names should be chosen to be as meaningful as possible without being excessively long. In most cases, attribute names should be kept to three or four words at most.

If there are several attributes that play a similar role, but need to be distinguished, their names may end with a number.

EXAMPLE 1:

```
ENTITY line;
  endpoint1: point;
  endpoint2: point;
END_ENTITY;
```

Attribute names should be singular, even if the attributes are aggregations.

EXAMPLE 2: The following is wrong:

```
ENTITY person;
  addresses: SET [0:?] OF address;
END_ENTITY;
```

Instead, write:

```
ENTITY person;
  the_address: SET [0:?] OF address;
END_ENTITY;
```

An attribute name may start with "the_" if it would otherwise be a reserved word or an entity name. If the attribute is boolean or logical, then it may instead start with "is_."

EXAMPLE 3: The following is wrong:

```
ENTITY address;
  ...
END_ENTITY;

ENTITY person;
  address: address; (* conflicts with entity name *)
  local: boolean;   (* reserved word in EXPRESS *)
END_ENTITY;
```

Instead, use the following:

```
ENTITY address;
...
END_ENTITY;

ENTITY person;
  the_address: address;
  is_local: boolean;
END_ENTITY;
```

3.8.12 Derived Attributes

A derived attribute shall only be used if a calculable value is referenced in more than one place within the schema;

EXAMPLE: A square is defined as follows:

```
ENTITY square;
  length: REAL;
  width: REAL;
WHERE
  WR1: ... (* some rule that calculates the area *)
  WR2: ... (* another rule that calculates the area *)
END_ENTITY;
```

Then area can be defined as a derived attribute as follows:

```
ENTITY square;
  length: REAL;
  width: REAL;
DERIVE
  area: REAL := length * width;
WHERE
  WR1: ... (* some rule that references the derived
            attribute area *)
  WR2: ... (* another rule that references the derived
            attribute area *)
END_ENTITY;
```

However, if area were a concept that the domain experts associate with a square but the area is not referenced anywhere in the schema, it would not be captured as a derived attribute.

3.8.13 Inverse Attributes

An INVERSE attribute shall only be used to constrain the cardinality of a relationship. An INVERSE attribute shall not be used just to give a name to a backward role. An INVERSE attribute shall not be referenced in a local rule.

3.8.14 UNIQUE

A UNIQUE statement in EXPRESS constrains a data population. Uniqueness rules shall not be used by themselves to make assertions that are true of all data in the world.

3.9 Normalization

The schema should, in general, conform to the following normal forms:

- first normal form;
- second normal form;
- third normal form;
- Boyce-Codd normal form.

Definitions of these normal forms can be found in most database textbooks.

If the principles and rules stated earlier in this document have been followed, then the model will likely already be in normal form. Therefore, instead of normalization being applied early on as a part of the modeling process, models shall first be developed according to the principles of this document and the referenced documents. Then they shall be checked to ensure that they are properly normalized. Any deviations shall be corrected or explained.

3.10 Documentation

3.10.1 Entities

Entities should be defined following the rules of good English definition writing. All terms that are not defined either in entity definitions or in a separate definition section attached to the schema are to be interpreted as defined in the Concise Oxford Dictionary of Current English [4].

The following format shall be used to define an entity data type (EDT):

A(n) <entity_name> is <description of the entity within the context of the application.>

The definition of an EDT should start with the general category of the thing represented by the EDT, followed by one or more adjective clauses that give specific characteristics that differentiate things that are members of the EDT from things that are not members of the EDT.

If an EDT is a subtype of another EDT, then the following format shall be used:

A(n) <entity_name> is a type of <supertype-entity-name> <description of the entity within the context of the application.>

The definition of the concept and any supporting text necessary to define the intent of the concept shall be contained in the entity definition text. Examples may be provided to clarify the intent of the concept or to illustrate the population of the attributes of the entity. Each example shall be a separate paragraph and shall start with the text "EXAMPLE:". Extra explanation and references to other sources for explanations should also be given as notes. Each note shall be a separate paragraph and shall begin with the text "NOTE:."

EXAMPLE 1: The product entity could be defined as follows:

A product is a thing or substance that is produced by a natural or artificial process.

NOTE: This includes the end item acquired in an acquisition as well as end item components, intermediate work products and possibly input materials.

3.10.2 Attributes

The following format shall be used to define an attribute of an EDT:

The <attribute_name> specifies the <role and nature of the attribute with respect to the object.>

The following format shall be used to define Boolean or logical attributes:

The <attribute_1> specifies whether or not <condition the attribute creates with respect to the object.>

When referring to the EDT to which the attribute belongs, use the phrase "the <entity_name>." Following the rule given in [Section 3.10.1](#), the entity name shall be given in angle brackets.

EXAMPLE 1: Consider the following EXPRESS definition:

```
ENTITY product;
  name: STRING;
END_ENTITY;
```

The name attribute could be defined as: "The name specifies the designation for the product."

When referring to other attributes of the EDT to which the attribute belongs, use the phrase "the <attribute_name>." Do not enclose the names of attributes in angle brackets.

EXAMPLE 2: Consider the following EXPRESS definition:

```
ENTITY thing;
  name: STRING;
  definition_of_thing: STRING;
  definition_normative: BOOLEAN;
END_ENTITY;
```

The normative attribute could be defined as: "The definition_normative specifies whether or not the definition_of_thing is mandatory."

Each attribute definition shall be written in the singular, even if the attribute is an aggregation. The definition of an aggregation attribute shall be written as if it were followed by the following sentence:

There may be more than one <attribute-name> for a(n) <entity-name>.

However, the above sentence shall not actually be included in the documentation.

NOTE: The above sentence may be automatically added to the documentation at a later date through the use of a script.

EXAMPLE 3: Consider the following EXPRESS definition:

```
ENTITY person;
  the_address: SET [0:?] OF address;
END_ENTITY;
```

The following attribute definition is incorrect:

"The the_address specifies the addresses associated with the person."

The following is correct:

"The the_address specifies the address associated with the person. *There may be more than one <attribute-name> for a(n) <entity-name>.*"

The second sentence, shown in dark yellow, is imaginary.

3.11 EXPRESS-G Diagrams

3.11.1 General

Each schema shall be presented in both EXPRESS and EXPRESS-G form. The EXPRESS and EXPRESS-G shall be kept synchronized through use of a toolkit. The EXPRESS-G diagrams shall conform to Annex D of ISO 10303-11.

EXPRESS-G drawings should be arranged so as to minimize visual complexity [4], while also emphasizing important relationships and grouping elements logically.

Except as specified below, all text in the EXPRESS-G drawings shall be in the Times New Roman font with a regular font style (not bold and not italic). All text on the diagram shall use the same font size.

The drawing shall include a document change table. If the drawing tool does not support tables, this element may be in the Courier font to allow for alignment of columns.

The following rules apply to layout of EXPRESS-G drawings. For further information, see the literature on "graph layout" and "graph visualization." Reference [1] is a good starting point.

NOTE: The literature on graph layout is not specific to EXPRESS, so one must determine which guidelines, examples, and rules apply to EXPRESS drawings and which do not.

3.11.2 Overall Layout

Graphical elements shall not overlap, except that relationship lines may cross each other, or may cross subtype lines, where necessary.

Boxes and text shall not overlap page boundaries. Only lines may cross page boundaries.

All line crossings shall be at right angles.

The number of line crossings shall be minimized.

The number of bends in a line shall be minimized.

A Manhattan layout shall be used, i.e., all lines shall be vertical or horizontal.

Related items shall be grouped together where possible.

A label for a relationship line shall be close to the associated line.

A label for a relationship line shall be positioned so that it is clear which relationship line it applies to.

The number of long relationship lines should be minimized.

3.11.3 Subtypes

Subtypes should be shown below their supertype where possible (see [Figure 1](#)).

Subtypes should be placed evenly with respect to each other where possible.

Subtypes should be balanced to the left and right of their supertype, where possible (see [Figure 2](#)).

Long subtype lines, especially those that cross page boundaries, are to be avoided, since they are very distracting. If it is not possible to place all subtypes near the supertype, then break up the subtype lines using off-page references.

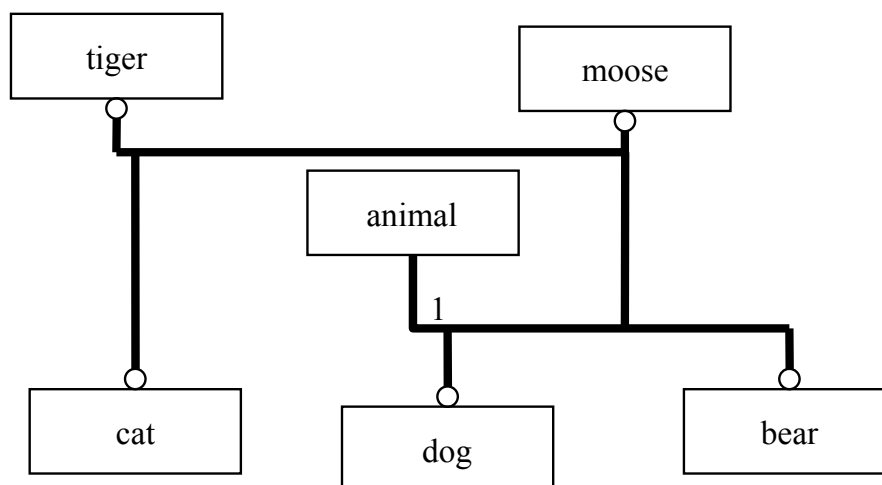
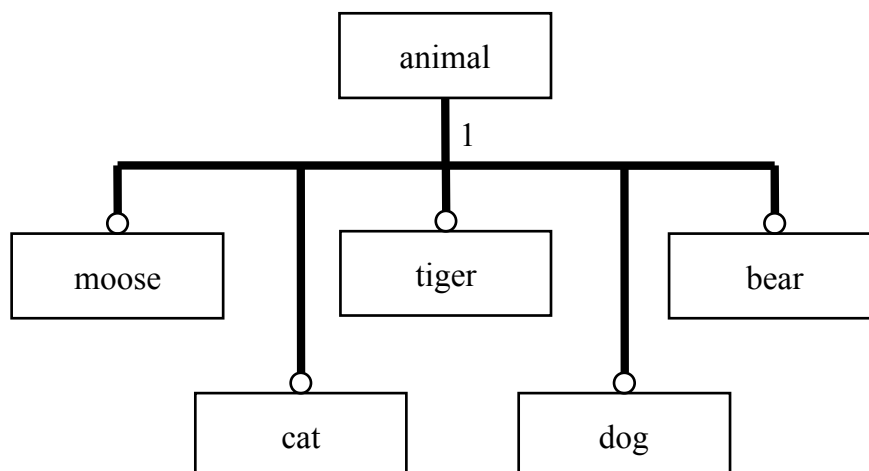
Bad**Good**

Figure 1 - Subtypes should be below the supertype and spaced evenly

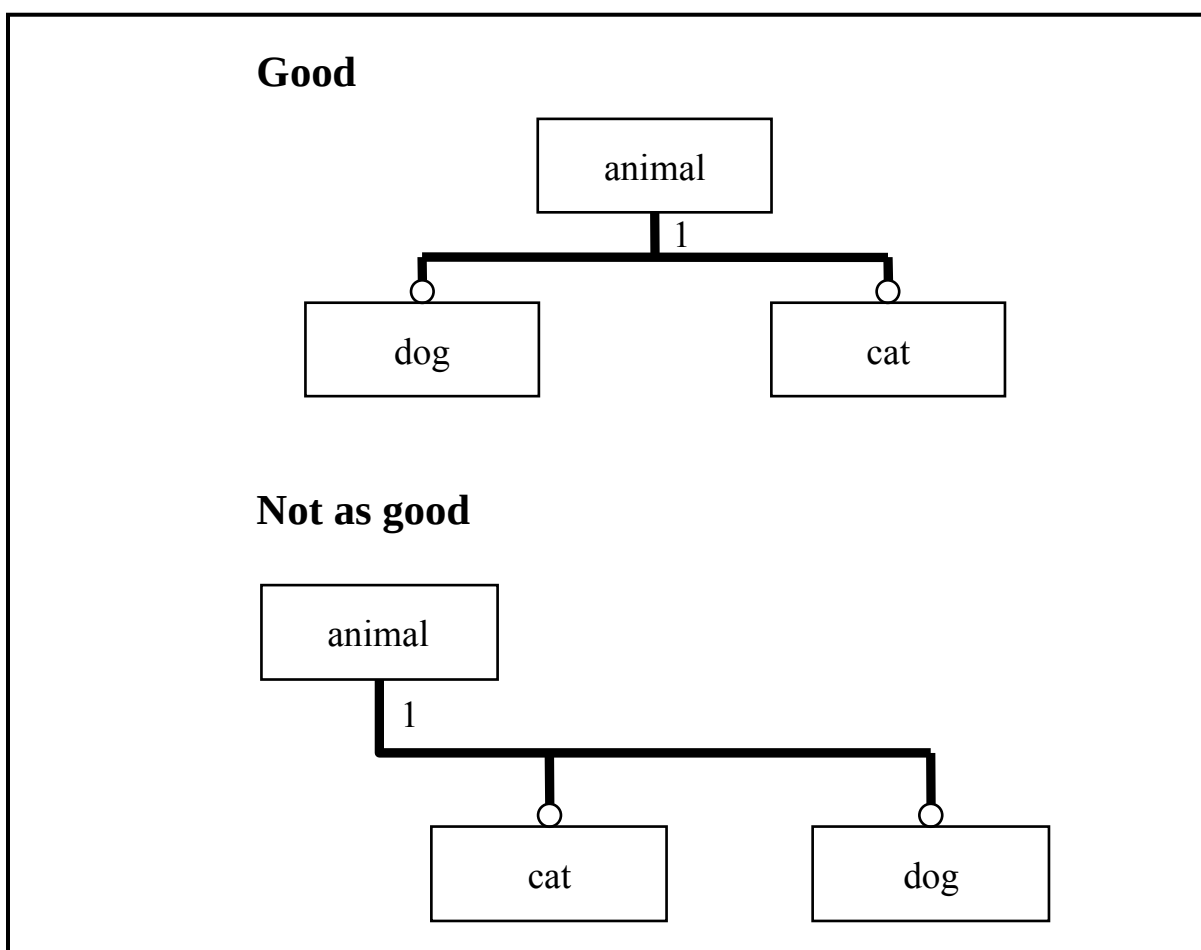


Figure 2 - Subtypes should be balanced to the left and right of the supertype

3.12 Data Modeling Guidelines References

- [1] Herman, I., G. Melançon, and M. S. Marshall, "Graph Visualization and Navigation in Information Visualization,"
<http://www.cwi.nl/InfoVisu/Survey/StarGraphVisuInInfoVis.html>.
- [2] ISO 8601:1988, *Data elements and interchange formats — Information interchange — representation of dates and time*.
- [3] ISO 10303-21:1994, *Industrial automation systems and integration — Product data representation and exchange — Part 21: Implementation methods: Clear text encoding of the exchange structure*.
- [4] Kimelman, Doug, Bruce Leban, Tova Roth, and Dror Zernik: Reduction of visual complexity in dynamic graphs, *Proceedings of Graph Drawing 1994*, pp. 218-225,
<http://www.research.ibm.com/pvres/gd94abs.html>.
- [5] Pearsall, Judy, ed., *The Concise Oxford Dictionary of Current English*, 10th edition (also known as *The Concise Oxford English Dictionary*, 10th edition), Oxford University Press, 1999.
- [6] "Pitfalls in Writing Definitions,
<http://www.inform.umd.edu/EdRes/Colleges/ARHU/Depts/MEI/.faculty/ACavanaugh/pitfall2.html>.
- [7] Simsion, Graeme, *Data Modeling Essentials: Analysis, Design, and Innovation*, International Thomson Computer Press, 1994.
- [8] Schenck, Douglas, and Peter Wilson, *Information Modeling the EXPRESS Way*, Oxford University Press, New York, 1994.
- [9] Warren, Thomas, QC Training Session Notebook, http://www.okstate.edu/ind-engr/step/WEBFILES/QC%20Support%20Material/QC_Notebook_index.html.

4 Data Model Usage Guide

4.1 Modeling Philosophy

The GEIA-STD-927 Data Model is in the EXPRESS modeling language [1]. It is an extension of the EXPRESS schema documented in ISO/DIS 15926-2 [2], which is also known as the EPISTLE Core Model version 4.0 (ECM4). ECM4 is modeled in accordance with the principles given in the "high quality data modeling" (HQDM) paper [4].

ECM4 is intended to have a universal context; therefore, it contains no constraints that are specific to particular applications or views.

In accordance with HQDM, ECM4 has a root entity (named "thing") from which all other entities inherit, in order to provide a "universal context" for the model. Each "thing" is an individual, a class, or a relation. It cannot be more than one of these.

In ECM4, most attributes relate entities to other entities. The exception is a set of entity data types defined as EXPRESS simple types, which are used for standardized computer representations of numbers, text characters, and binary patterns, but not their possible meanings. Associations and activities are represented by entity data types. Entity data types represent the underlying nature of objects, not the roles they play in specific contexts.

An individual is defined as "a possible extension of space and time." Thus, an individual is a set of points in a four-dimensional Cartesian space, whose axes are the three spatial axes plus time. Different types of individuals are characterized mainly by the geometry of their spatio-temporal extent.

ECM4 distinguishes between a real-world thing and the representation of information about some aspect of that thing. The link between these two is the entity data type: "class_of_representation."

EXAMPLE 1: The fact that a pump is described by the string "main bilge pump" can be represented as shown in Figure 3. In the figure, if "class_of_description" = x and "class_of_representation" = y, "x <= y" means that x is a subtype of y.

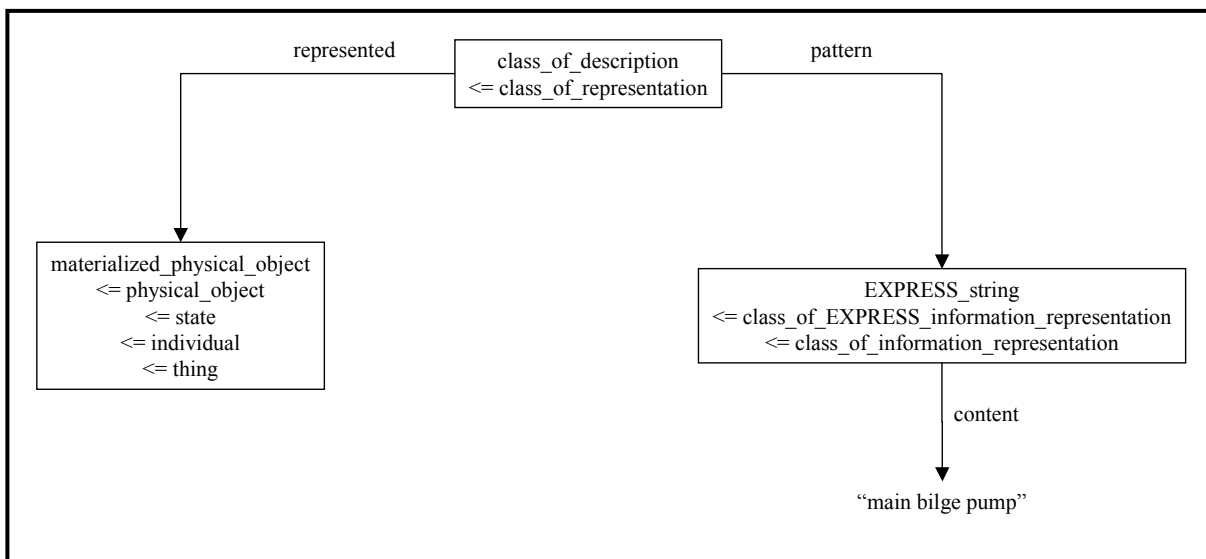


Figure 3 - Separation of information representation from thing

Traditionally, designers of entity-relationship or object-oriented data models make a clear distinction between data and metadata. Different representation schemes are often used for the two different types of data. The authors of ECM4 do not feel such a distinction is warranted. Data and metadata can be freely mixed in an ECM4 data set.

For more details on ECM4, see Clause 4 (Fundamental concepts and assumptions) of [\[2\]](#).

The ECM4 Data Model is designed to be used in conjunction with reference data. ISO 15926-2 [\[2\]](#) defines reference data as “process plant life-cycle data that represents information about classes or individuals which are common to many process plants or of interest to many users.” One type of reference data are reference classes, which are standard instances of subtypes of the “class” entity data type that, in effect, extend the ECM4 Data Model.

The EPISTLE modeling team intended ISO 15926-2 [\[2\]](#) to be a small stable “core” model that would be extended by a much larger and constantly-growing Reference Data Library (RDL), not by changing the EXPRESS model. [Figure 4](#) shows a small core model extended by a much larger RDL.

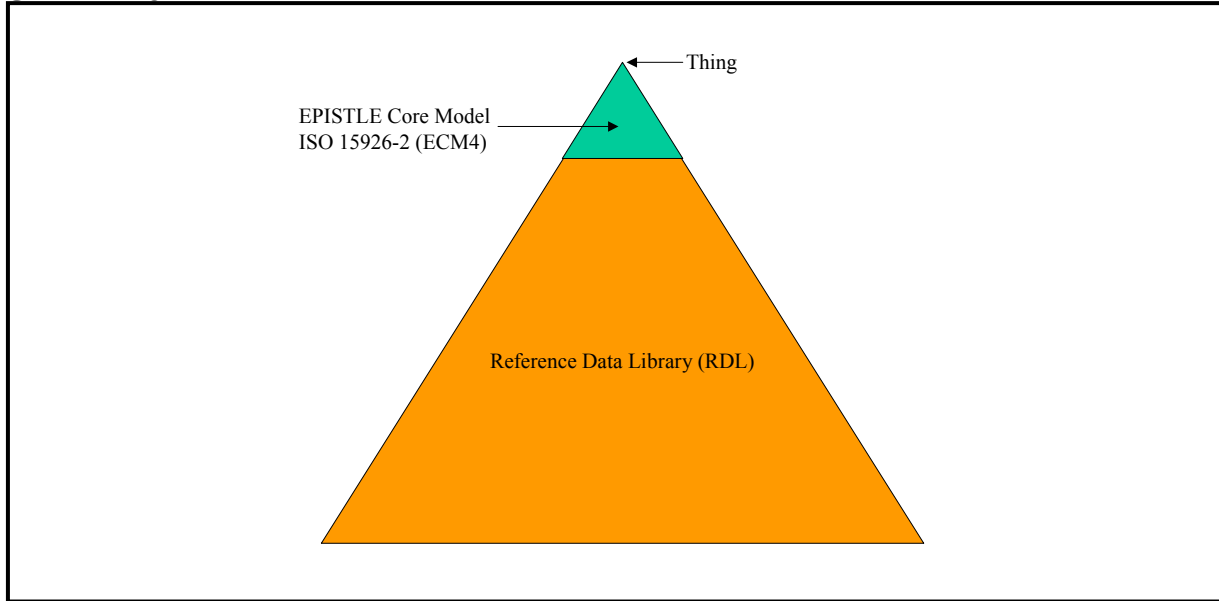


Figure 4 - Use of ECM4 with Reference Data Library

However, the GEIA-STD-927 modeling team found it more expedient to extend ECM4 by creating a new EXPRESS schema with additional subtypes of ECM4 entity data types, rather than creating a RDL.

This decision was based on the following considerations:

- A RDL was being developed to accompany ECM4, but it was not published at the time work started on the GEIA-STD-927 Data Model.
- No guidelines were available from EPISTLE on how to create a RDL or how it would be used for data exchange.
- Tools for modeling RDLs that conform to ECM4 were not available.
- The RDL paradigm does not provide a mechanism for representing models graphically, as does EXPRESS with the EXPRESS-G modeling tool.

In extending ECM4, the modeling team followed the GEIA-STD-927 Data Modeling Guidelines [3]. The GEIA-STD-927 Data Modeling Guidelines [3] incorporated principles from the HQDM paper [4] by reference but also added other GEIA-STD-927-specific requirements.

EXAMPLE 2: Suppose that a particular Application Model defines the concept of “pump” and gives it attributes of “part_number” and “flow_rate.” A particular pump with a part number of 1234 and a flow rate of 1200 might be represented in that Application Model as shown in Figure 5. In the GEIA-STD-927 Integration Model, there could be a subtype of “physical_object” called “pump,” a subtype of “class_of_description” called “flow_rate,” and a subtype of “class_of_identification” called “part_number.” The instance of “pump” in Figure 5 could then be represented as shown in Figure 6. Within this figure subtypes that would then be contained within the GEIA-STD-927 Integration Model representing extensions to ECM4 are shown in **bold face** type.

NOTE: This representation is somewhat simplified for illustration purposes.

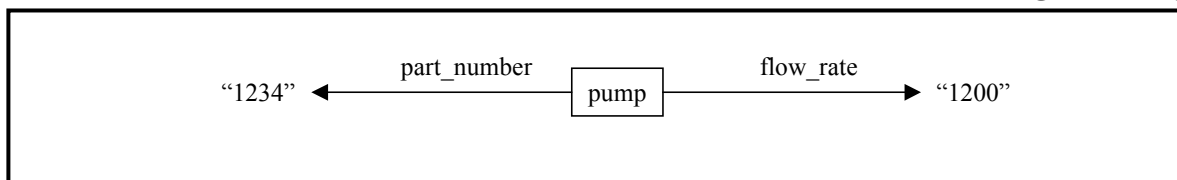


Figure 5 - Representation of “pump” in Application Model

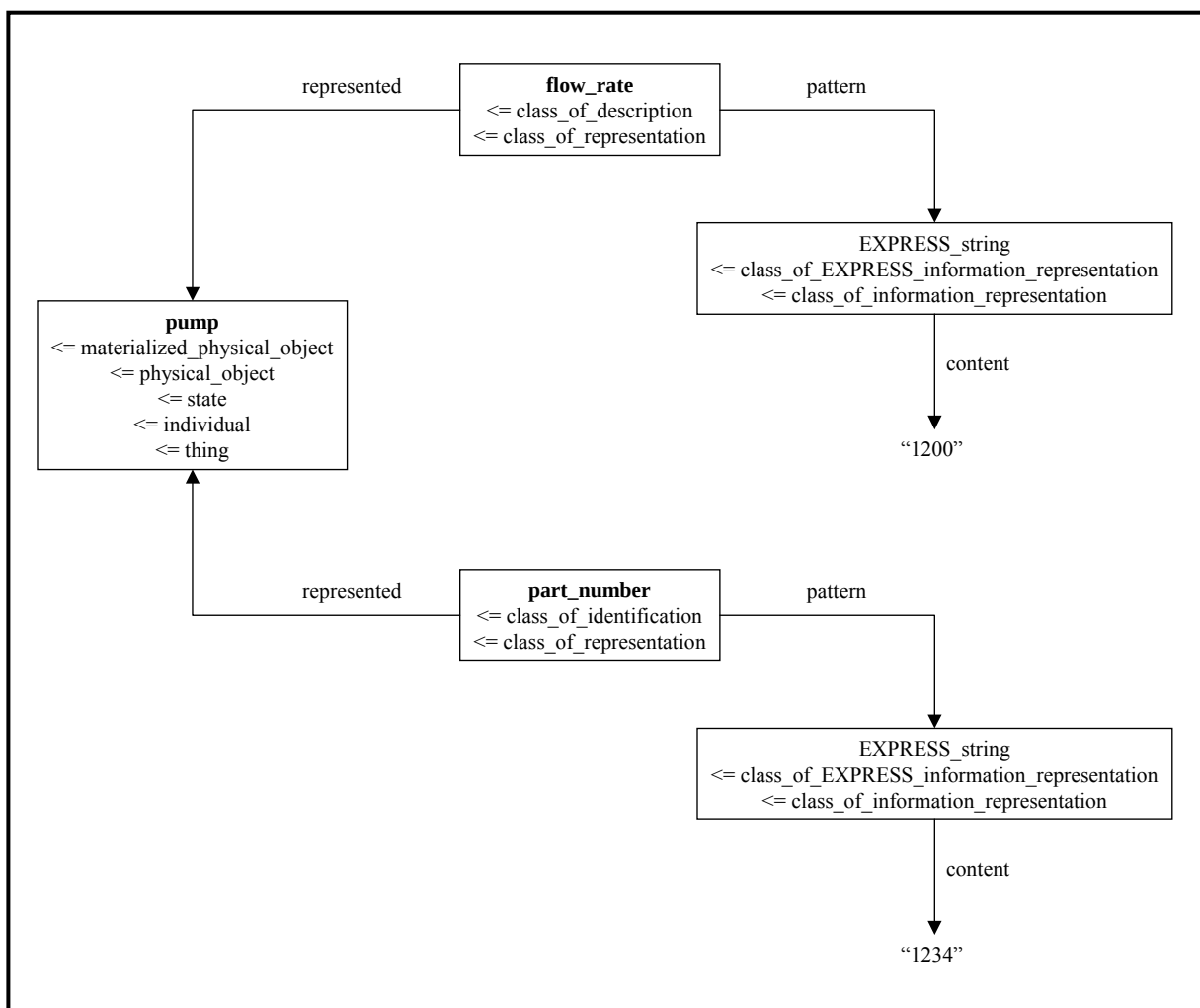


Figure 6 - Representation of “pump” in the GEIA-STD-927 Data Model

4.2 GEIA-STD-927 DATA MODEL CONCEPTS

This section describes the various types of data model concepts that recur in the GEIA-STD-927 Data Model. An overview of each concept is first provided followed by three other subsections which more fully detail and define the individual concept under study. The first subsection explains how each particular concept is used in other application models pertinent to the GEIA-STD-927 domain. The second subsection explains how each concept is depicted, accessed and made use of in the GEIA-STD-927 Data Model. This second subsection may also provide a representative example and an instance diagram to graphically show the entities

involved in each of the concepts. The third subsection references the other concepts needed to complete the current concept being represented.

The figures in this section show how entities relating to each of the concepts are connected in the actual GEIA-STD-927 Schema. The figures are presented using a graphical notation intended to illustrate the instance model only. The diagrammatic notation is not EXPRESS-G and does not illustrate the true GEIA-STD-927 EXPRESS Schema, but it is a graphical illustration of a specific population of a particular concept instance model involved in the Schema. The large boxes shown within the figures represent the full hierarchy of the instance being illustrated for each of the concepts. The significant inherited attributes are also shown within these figures.

4.2.1 Representation (Identifying and Describing Things)

The representation concept is the link between a thing and a pattern that is used to define, describe, name, or identify that same thing. The pattern can be anything from one character, to one word such as TRUE or GEORGE, to a number such as 12345 or 5.0, to even that of a full document.

Naming something is a type of representation that is a special method for identification. People use names most often to identify things. For instance, you would not call a friend by their social security number; you would use their name instead. A name is a simple string of text. Names are not pictures. Although it is said that a picture is worth a thousand words, no one would want to refer to something with a name that is a thousand words long.

Identification is a form of representation often used within databases and models as a way to mark a record or an entity so as to be able to distinguish it from other records or entities. Identifications are most often designed or intended to be unique within a certain context.

Version Identification is a type of representation which provides the user with a way to keep track of the changes that may occur to things. A new version of something often means that some change has been made and in some cases that change may turn out not to be desired; therefore, it should be possible to return to the previous version of that particular item.

4.2.1.1 Concept in Other Models

Most models represent names, identifications, definitions, and descriptions of entities as attributes attached to those entities that in turn point to basic string types.

4.2.1.2 Concept in the GEIA-STD-927 Data Model

4.2.1.2.1 How Descriptions are shown in the GEIA-STD-927 Data Model

The “class_of_description” entity is used to indicate that the pattern is a description of the represented “thing.” A “thing” can have many descriptions attached to it. For example a “class_of_description” object can link a car object to the pattern “old” and another “class_of_description” object can link the same car object to the pattern “rusty.” There is no limit to the number of patterns that can be linked to a particular object to provide a description.

In the GEIA-STD-927 Data Model any entity can have a description attached. The “class_of_description” entity is used to connect a description to the “thing” that it describes.

The actual description for an entity in the GEIA-STD-927 Data Model must be an instance of the “class_of_information_representation” entity or an instance of one of the subtypes of the “class_of_information_representation” entity.

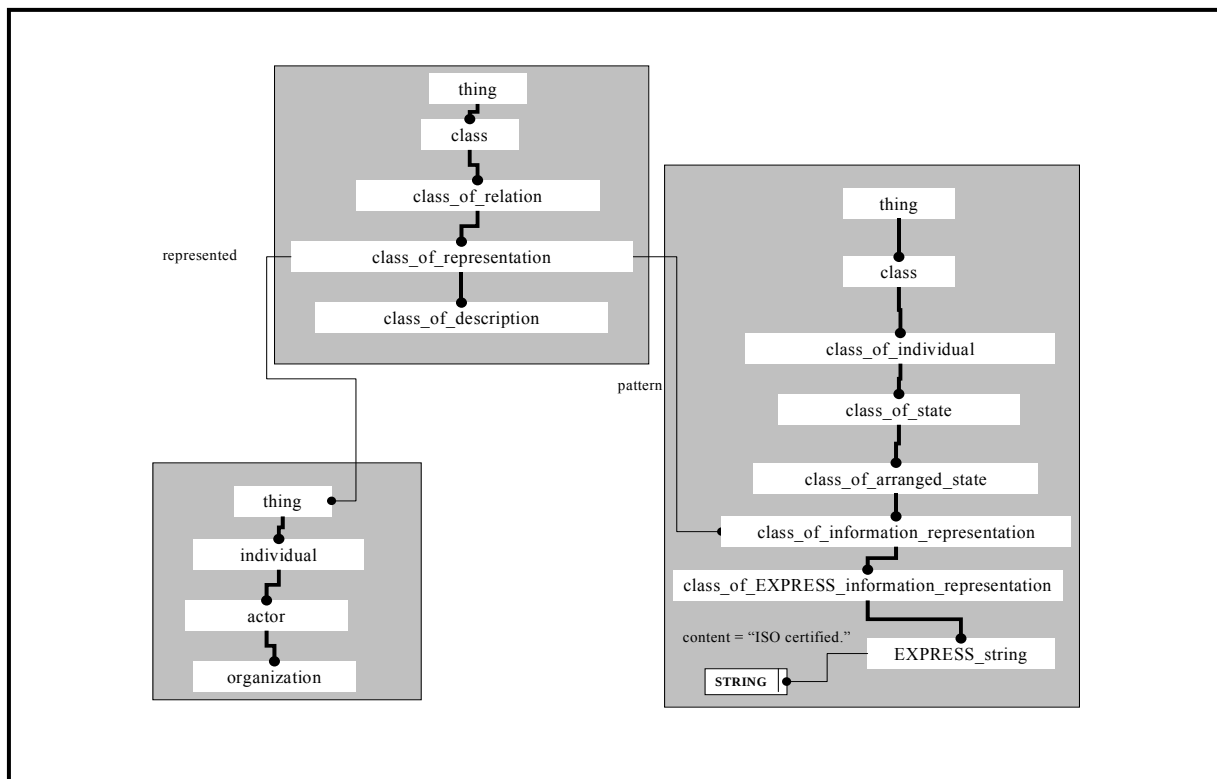


Figure 7 - Assigning a Description

Figure Entity List

This section lists the entities needed for the example shown in [Figure 7](#). The figure also depicts the supertypes of the required entities.

class_of_description - The “class_of_description” entity's “represented” attribute points to the entity that is being described. The “class_of_description” entity's “pattern” attribute points to the instance of the “EXPRESS_string” entity that contains the description. In [Figure 7](#) above an “organization” is being described as "ISO certified," by way of the “pattern” attribute, which is the description contained in the string of text pointed to by the “content” attribute of the “EXPRESS_string” entity.

EXPRESS_string - This particular entity is a sub type of the “class_of_information_representation” entity that contains the pattern: “ISO certified.” via the “content” attribute pointing to “STRING.”

organization - In the example in [Figure 7](#), an instance of an “organization” is being described as a result of being pointed to by the “represented” attribute. In the GEIA-STD-927 Data Model, an instance of any entity can be described with a simple string of text by using instances of the “class_of_description” and “EXPRESS_string” entities.

4.2.1.2.2 How Names are shown in the GEIA-STD-927 Data Model

In the GEIA-STD-927 Data Model, the “class_of_name” entity is used to attach a name to a “thing.” The “class_of_name” entity is a subtype of the “class_of_identification” entity because a name is a more specific type of a “class_of_identification.” The name of an instance can be used to separate it from other instances. For example the “status_class” entity with the name “resolved” can be used to classify member instances that have been resolved, whereas an instance of the “status_class” entity with the name “in_work” can be used to classify members that are still in the process of being resolved.

[Figure 8](#) depicts the instances needed to assign a name to an instance of the “person” entity.

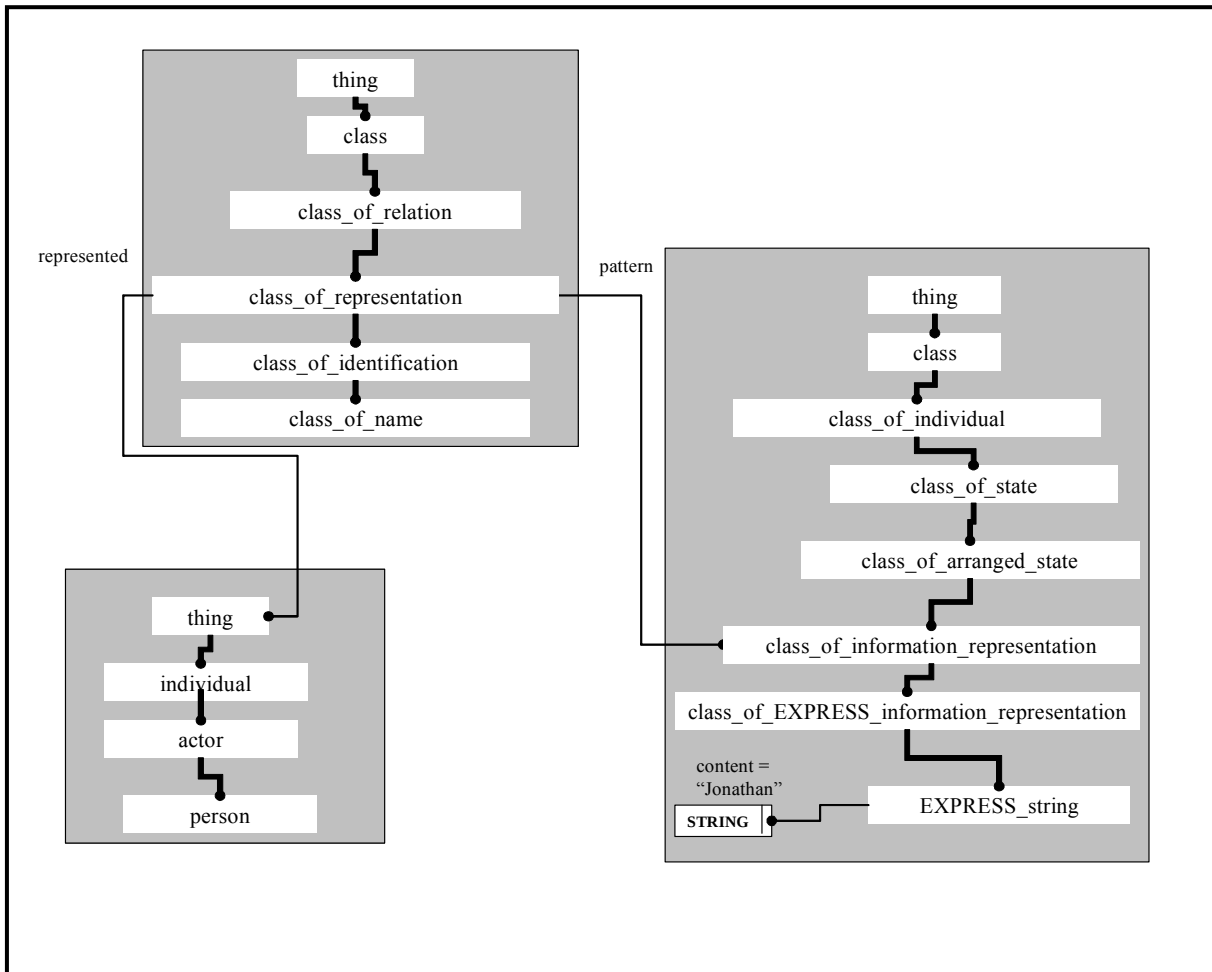


Figure 8 - Assigning a Name

Figure Entity List

This section lists the entities needed for the example shown in [Figure 8](#). The diagram also depicts the supertypes of the required entities.

person - The example in [Figure 8](#) shows an instance of the “person” entity being represented by the “pattern” name pointed to by the “content” attribute of the “EXPRESS_string” entity.

In the GEIA-STD-927 Data Model, an instance of any entity can be named with a simple string of text by using instances of the “class_of_name” and “EXPRESS_string” entities.

class_of_name - The “class_of_name” entity's “represented” attribute points to the entity that is being named. The “class_of_name” entity's “pattern” attribute points to an instance of a “EXPRESS_string” entity that contains the name. In [Figure 8](#), a “person” is being named “Jonathan,” by way of the “pattern” attribute, which is the name contained in the string of text pointed to by the “content” attribute of the “EXPRESS_string” entity.

EXPRESS_string - This particular entity is a subtype of the “class_of_information_representation” entity that holds the pattern: "Jonathan" via the “content” attribute pointing to STRING. Any appropriate subtype of the “class_of_information_representation” entity can be used to hold a pattern.

4.2.1.2.3 How Identification with a context is shown in the GEIA-STD-927 Data Model.

In the GEIA-STD-927 Data Model, the “class_of_identification” entity is used to link an instance of the “thing” entity to a “pattern” attribute that is that instance’s identification. The identification of an instance can therefore be used to separate it from other instances. The “pattern” attribute for the instance of the “thing” entity would ultimately point to and be represented by the “element_identifier” entity, which is a direct subtype of the “class_of_information_representation” entity, that contains an identification value and the actor (a person or organization) that determines the identification value. The “element_identifier” entity has an “identifier_value” attribute attached to it that is a pointer to a string of text, which is where the actual “id” value is held. The list of entities to follow depicts those that are needed to assign an identification to an instance of the “person” entity as shown in [Figure 9](#).

Figure Entity List

This section lists the entities needed for the example shown in [Figure 9](#). The diagram also depicts the supertypes of the required entities (unless stated otherwise in the diagram).

person - The example in [Figure 9](#) shows an instance of the “person” entity being represented by the “pattern” identification pointed to by the “identifier_value” attribute of the “element_identifier” entity. In the GEIA-STD-927 Data Model, an instance of any entity can be identified by using instances of the “class_of_identification” and “element_identifier” entities.

class_of_identification - The “class_of_identification” entity's “represented” attribute points to the entity that is being identified. The “class_of_identification” entity's “pattern” attribute points to an instance of the “element_identifier” entity that contains the identification. In [Figure 9](#), a “person” is being identified by the number “123-45-6789” in the context of the “Social Security Administration” organization by way of the “pattern” attribute. This “id” value is the identification number contained in the string of text pointed to by the “identifier_value” attribute of the “element_identifier” entity.

element_identifier - This particular entity is a subtype of the “class_of_information_representation” entity that holds the pattern: "123-45-6789" via the “identifier_value” attribute pointing to STRING. For the concept of identification, the “element_identifier” entity is the only appropriate subtype of the “class_of_information_representation” entity that can be used to hold the pattern.

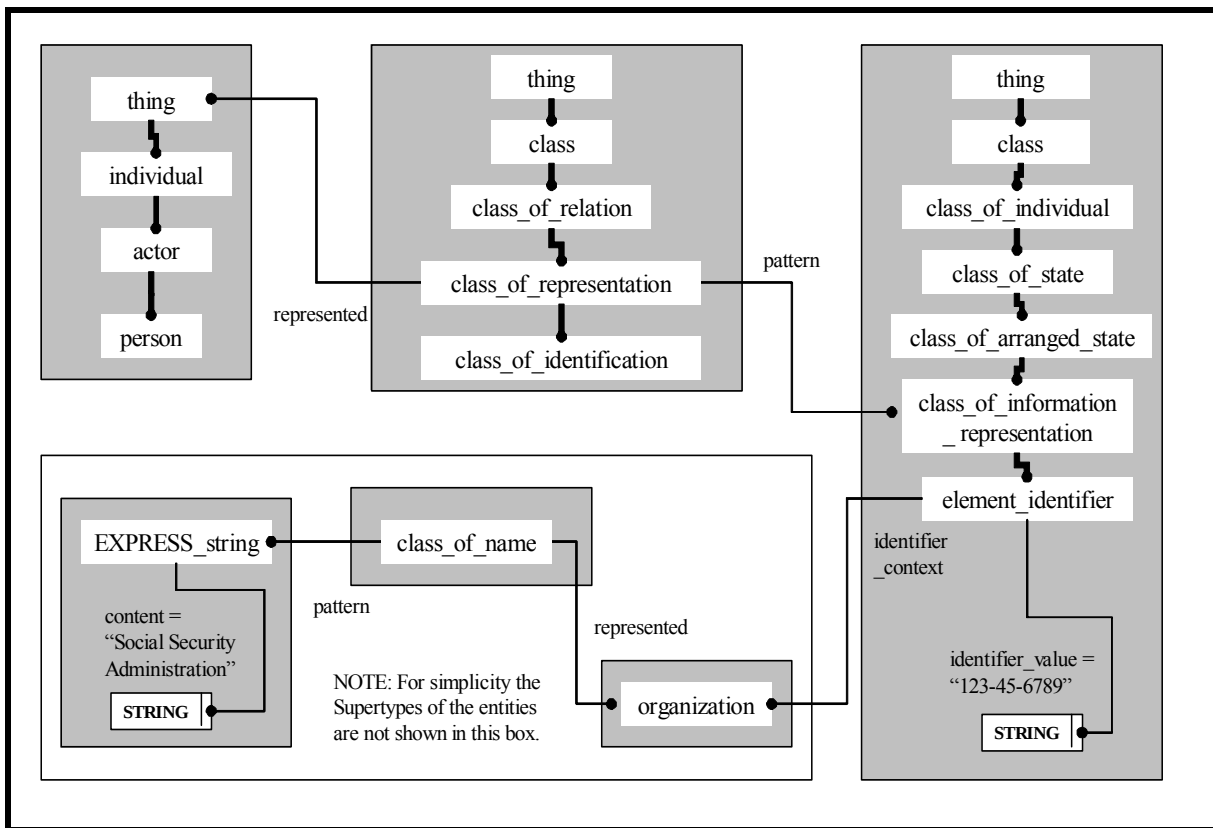


Figure 9 - Assigning an Identification

4.2.1.2.4 How Version Identification with a context is shown in the GEIA-STD-927 Data Model.

In the GEIA-STD-927 Data Model, the “class_of_version_identification” entity is used to link an instance of the “thing” entity to a “pattern” attribute that is that instance’s version identification. The “class_of_version_identification” entity is a subtype of the “class_of_identification” entity because a version “id” is a more specific type of a “class_of_identification” entity. The version identification of an instance can therefore be used to separate it from other instances. The “pattern” attribute for the instance of the “thing” entity would therefore point to and be represented by the “element_identifier” entity, which again is the direct subtype of the “class_of_information_representation” entity that will contain a version identification value and the actor (a person or organization) that determines the value of the version identification. The “element_identifier” entity has an “identifier_value” attribute attached to it that is a pointer to the string of text, which is where the actual version “id” value is held. The list of entities to follow depicts those that are needed to assign a version identification to an instance of the “event” entity as shown in [Figure 10](#).

Figure Entity List

This section lists the entities needed for the example shown in [Figure 10](#). The diagram also depicts the supertypes of the required entities (unless stated otherwise in the diagram).

event - The example in [Figure 10](#) shows an instance of the “event” entity being represented by the “pattern” version identification pointed to by the “identifier_value” attribute of the “element_identifier” entity. In the GEIA-STD-927 Data Model, an instance of any entity can be version identified by using instances of the “class_of_version_identification” and “element_identifier” entities.

class_of_version_identification - The “class_of_version_identification” entity's “represented” attribute points to the entity that is being identified by version. The “class_of_version_identification” entity's “pattern” attribute points to an instance of the “element_identifier” entity that contains the version identification. In [Figure 10](#), an “event” is being version identified by the text which reads “Games of the XXVII Olympiad” in the context of the “International Olympic Committee” organization by way of the “pattern” attribute. This version “id” value is the version identification information which is contained in the string of text pointed to by the “identifier_value” attribute of the “element_identifier” entity.

element_identifier - This particular entity is a subtype of the “class_of_information_representation” entity that holds the pattern: “Games of the XXVII Olympiad” via the “identifier_value” attribute pointing to STRING. For the concept of version identification the “element_identifier” entity is the only appropriate subtype of the “class_of_information_representation” entity that can be used to hold the pattern.

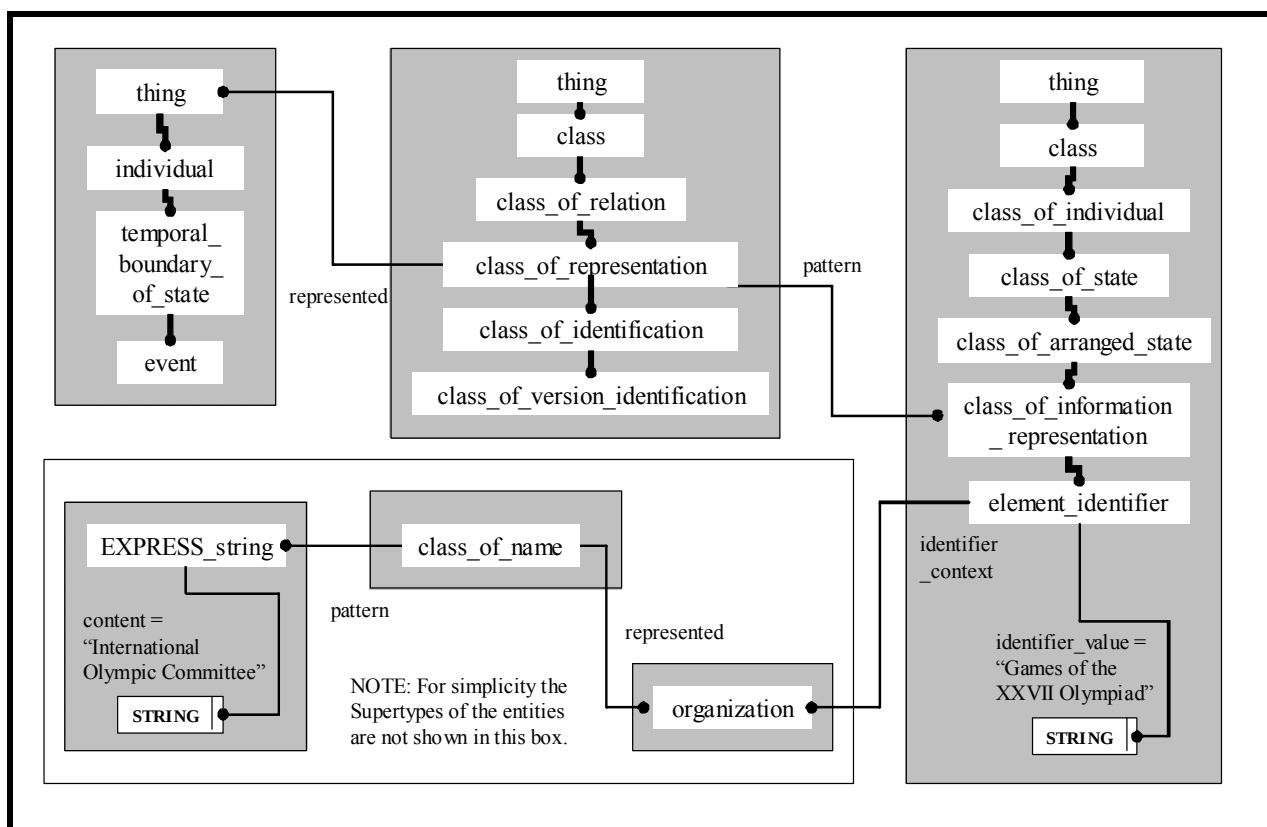


Figure 10 - Assigning a Version Identification