# PLCS

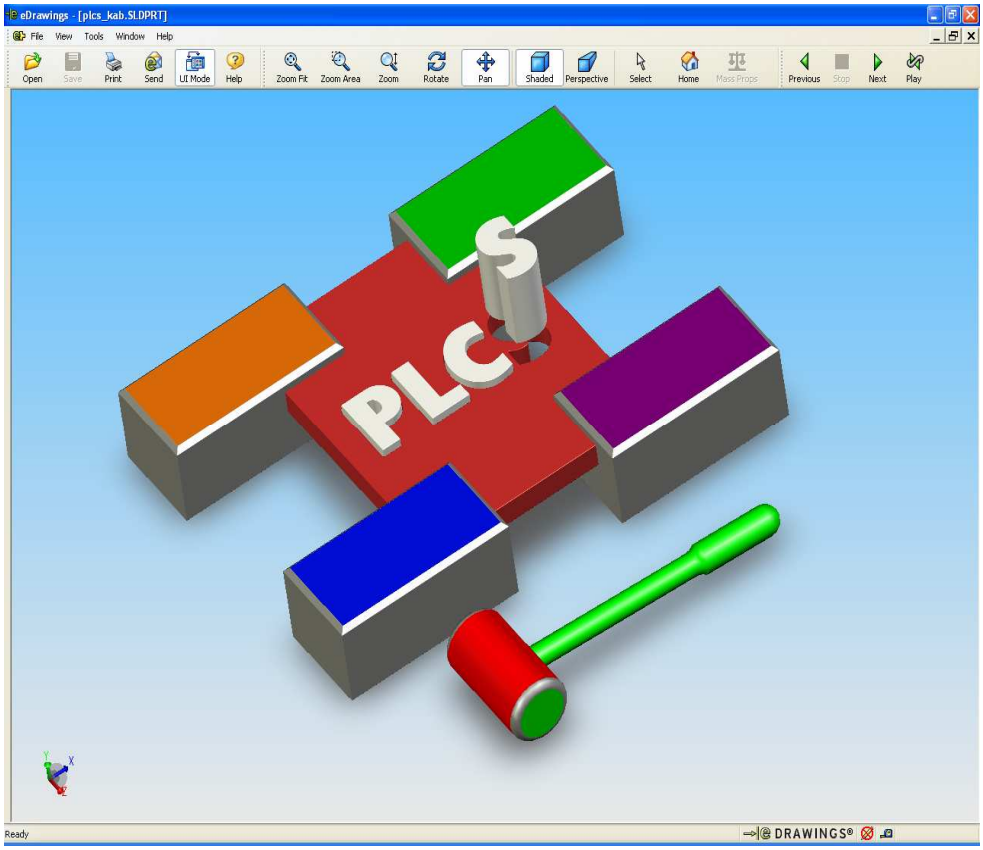
## Product Lifecycle Support

ISO 10303-239

# Handbook for Developing Data Exchange and Sharing Software, using ISO 10303-239

Last saved March 01, 2011 11:05

Version 1.3.1



# Table of Contents



## Table of Figures

# List of Tables



## Acknowledgements

The methods and procedures described in this handbook are developed by Jotne EPM Technology, based on and inspired by the work performed by the following organizations:

**International Organization for Standardization (ISO)**


**Aerospace Industries Association (AIA)**


**Organization for the Advancement of Structured Information Standards (OASIS)**


References to the actual documents that have been consulted are found in Appendix F – Reference Documents.

## Copyright Notice



## Disclaimer

Jotne EPM Technology AS assumes no responsibility for this Handbook's content or use, and disclaims any potential liability associated therewith.

Jotne JOTNE EPM TECHNOLOGY

# 1 Introduction

## 1.1 Executive Overview

The Aerospace Industries Association (AIA) states in the Guidelines for Implementing Interoperability Standards for Engineering Data [5] that:

"The variety of engineering tools used to support design, procurement, manufacturing, and support of aerospace products has never been greater." This is a statement that can certainly be endorsed by other industries as well.

The growing availability of software systems and tools has mainly been lucrative and profitable for the industries. There are, however, some significant challenges to overcome with respect to interoperability in such a heterogeneous environment. While numerous articles address these issues, it is not in the scope of this handbook to describe these challenges in detail. In this context we limit ourselves to references from the AIA Guidelines [5].

Along with an increasing number of organizations across multiple industries AIA has found standardization in general and ISO 10303-239(PLCS) in particular, to be the answer to many inherent interoperability challenges. Because the information model defined by ISO 10303-239 (PLCS) is a generic model supporting the whole life cycle of a product, it has a scope that is wider than most applications, business processes or single data exchanges. That is why *Data EXchange Specifications (DEXs)* have been developed by the OASIS [2] standardization body in order to support the usage of subsets of the model.

The standard by itself will not, however, provide its promised business benefits without being implemented in adapted business processes and in working software solutions.

Developing data exchange and sharing mechanisms can be cumbersome and costly, and there is a risk that the return of investment will be low if one does not approach the challenge in a systematic and methodical way.

Developing efficient data exchange mechanisms may require considerable effort even when best practices are applied. But once a standard based information backbone has been established investments will gradually start to pay off.

In this document, Jotne EPM Technology shares its experience from numerous projects developing PLCS based software solutions for data exchange.

This document describes the practices in use, and presents examples that aim to get the reader started on the road to a successful PLCS implementation.

## 1.2 Purpose of this handbook

The main purpose of the handbook is to provide practical guidelines for software developers setting out to implement data exchange mechanisms based on the PLCS standard. It is, however, believed that project managers, solution architects and data modellers can also gain insight from reading this handbook. Even though the handbook will best serve its purpose when treated as an extension of the AIA guidelines [5], care has been taken so that it can be used as a standalone item as well.

The handbook includes an introduction to the concepts and methodologies developed by OASIS [2]. Further in-depth reading might, however, be required for detailed explanations and clarifications of some of the concepts. Recommended readings are listed in Appendix F – Reference Documents.

Basically the handbook is neutral with respect to the use of tools and applications. However, specific tools or applications are mentioned when providing examples or to clarify a concept. Appendix C – PLCS in a Service Oriented Architecture, Appendix D – PLCS Tools (EDM™) and Appendix E – PLCS Data Exchange Framework provide further details of specific tools that will greatly aid in the implementation of PLCS. Those who consider making use of the EDMtemplateAPI™ or EDMplcsServicesAPI™ from Jotne EPM Technology should study the examples found in Appendix B – Translator Implementation Example.

# 2 Introduction to the main Standardization Concepts

In order to gain detailed in-sight in the main underlying concepts used to develop a standard based information backbone as lined out by AIA, in-depth reading of the documents listed in Appendix F – Reference Documents is recommended. It is the intention, however, that this introduction should give an overview, which is sufficient to make the handbook useful without any further study of the referenced material.

## *2.1 The Standards*

ISO 10303-239 (PLCS) specifies the information required to support a product throughout its life. It is one of the ISO 10303 families of standards, generally known as STEP (STandard for the Exchange of Product model data). This family of standards includes the Application Protocols (APs), which define particular business viewpoints of a common information model, and the PLCS standard belongs to this series. Application Protocols are numbered sequentially within STEP. ISO 10303-239 is often referred to as AP239 or PLCS (Product Life Cycle Support).

## *2.2 Scope of PLCS*

The scope of PLCS is as defined in the standard [6]:

- ✓ Information for defining a complex product and its support solution
- ✓ Information required to maintain a complex product
- ✓ Information required for through life configuration change management of a product and its support solution
- ✓ Representation of product assemblies
- ✓ Identification and representation of parts, their versions, definitions, and documentation and management information, such as dates and approvals assigned to parts
- ✓ Representation of a product through its entire lifecycle
- ✓ Specification and planning of activities for a product
- ✓ Representation of the activity history of a product
- ✓ Representation of the product history

To cover the entire information scope of product related data, PLCS must be extended with capabilities to represent design 3D data, structured documentation and training material. As it stands today, the PLCS standard does not represent these data types. This can be achieved with the use of complementary standards such as AP203, S1000D and SCORM.

## 2.3 Main Concepts

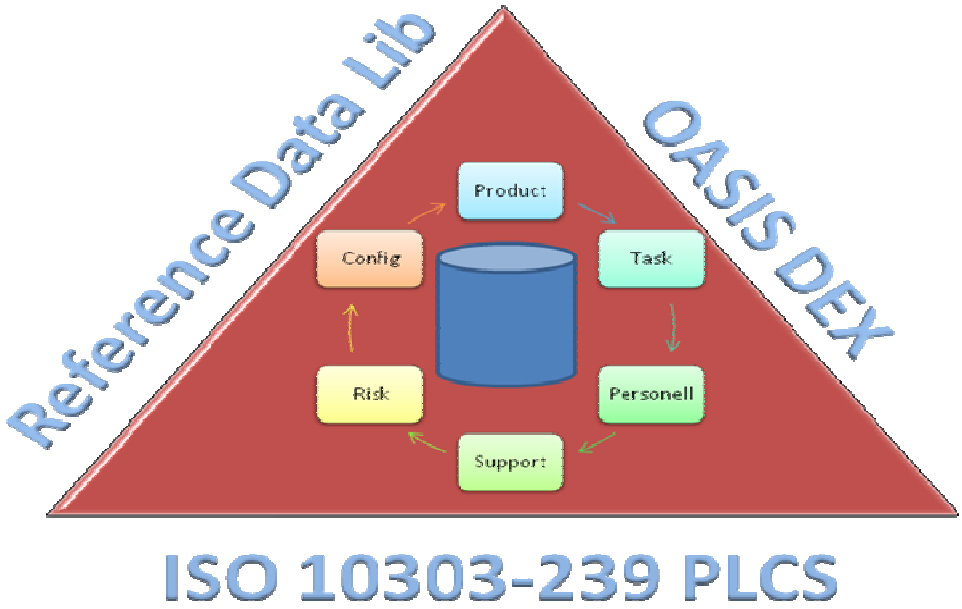



**Figure 1 - Information Model**

The information model defined by ***ISO 10303-239*** has a scope that is wider than most applications or any single data exchange. Therefore the **OASIS Technical Committee** has developed concepts and techniques in order to constrain the scope by means of so called **D**ata **EX**change Specifications. **DEX**es are specified using lower level components referred to as **templates** as reusable building blocks.

In order for the *ISO 10303-239* information model to be usable in many different *Business Context*s, it is deliberately generic. In other words, *EXPRESS* entities are provided for representing constructs such as parts and dates and their relationships. More specialized semantic definitions of constructs such as a "safety critical part", or a "creation date" are not represented directly in the PLCS standard. Instead provision is made to enable the same semantic precision by adding a classification scheme to the basic constructs using ***Reference Data***, thus refining or augmenting the meaning of the entity. It is out of scope for this document to explain the rationale behind these concepts. It is, however, recommended that the readers familiarize themselves with the specifications available on the OASIS web site.

To make oneself acquainted with PLCS, the OASIS' PLCS Technical Description web page [3] is considered to be an appropriate point to start, as is the case for the DEX and Reference Data concepts as well.

Another recommended introduction is the “Official OASIS PLCS presentation” [2].

In summary:

- A DEX defines a subset of the ISO 10303-239 (PLCS) information model, for which conformant software and data exchange contracts can be based on.
- The building blocks of DEXes are templates, which are reusable across different DEXes. Consequently, they provide a consistent representation of the same core concepts across the DEXes.
- The building blocks of a template are PLCS entities and other templates.
- Entities are ultimate "atoms" of the PLCS information model.
- Reference Data adds more specialized semantics to the information model entities.
- Business DEXes are specializations of the generic DEXes and are furnished with extensions to the core PLCS Reference Data Library (RDL).

### 2.3.1 Template usages

Even though it is expected that the reader of this document will develop software that is layered on top of a ready made Application Programming Interface, which encapsulates most of the complexity of the OASIS templates, it is considered a matter of necessity to be familiar with the essential properties of such templates. It is of great importance to understand the difference between template definitions and the rest of the template facets.

Please note that the order, in which the usages are listed, is the actual order in which templates are used.

1. Template definitions.
   Templates are defined in DEXlib [1]. Formal parameters are bound to the templates at this stage.
2. Template instances in DEXes.
   Templates are used or instantiated if you like, in various DEX specifications. Some actual parameter values, like class names (reference data) and relations to other templates and entities are bound to template instances at this stage.
3. Templates instances as mapping targets in an adapter.
   Source schema entity attribute names are mapped to template instance parameters at this stage. This is information related to a specific adapter and hence not found in DEXlib [1].
4. Population of template.
   At run time adapters will populate templates with real actual parameter values according to the specifications from the three earlier stages.

#### *2.3.1.1 Brief example*

The example below will take the reader through the four stages listed in the previous paragraph. Because of its simplicity the template "assigning_reference_data" is chosen. In this way principles can be highlighted while the example at the same time can be kept brief.

**1) Template definition**

The diagram below is a replica of the "assigning_reference_data" as defined on DEXlib [1].

The template has two formal parameters:

- assigning_reference_data.**class_name**
- assigning_reference_data**.ecl_id**

The strings prefixed with a circumflex are so called reference parameters. Such parameters provide hooks that one can refer to from the outside world.

A complete description of the syntax is found in "The PLCS technical descriptions" [3].

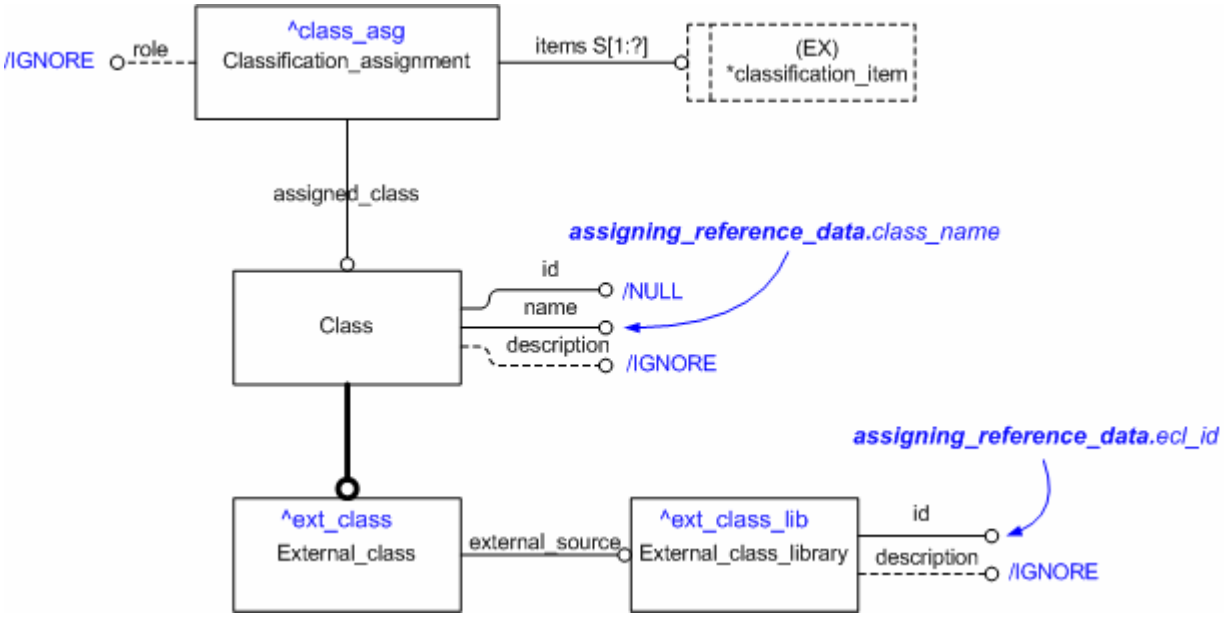



**Figure 2 Assigning_reference_data diagram**

### 2) Template instances

The sketch underneath is a fragment from one of the NOLITO business DEXes as found on DEXlib [1]. The template "assigning_reference_data" is instantiated twice, as one can see. Each of them is given a unique identification, namely DEX1.#4 and DEX1.#13.

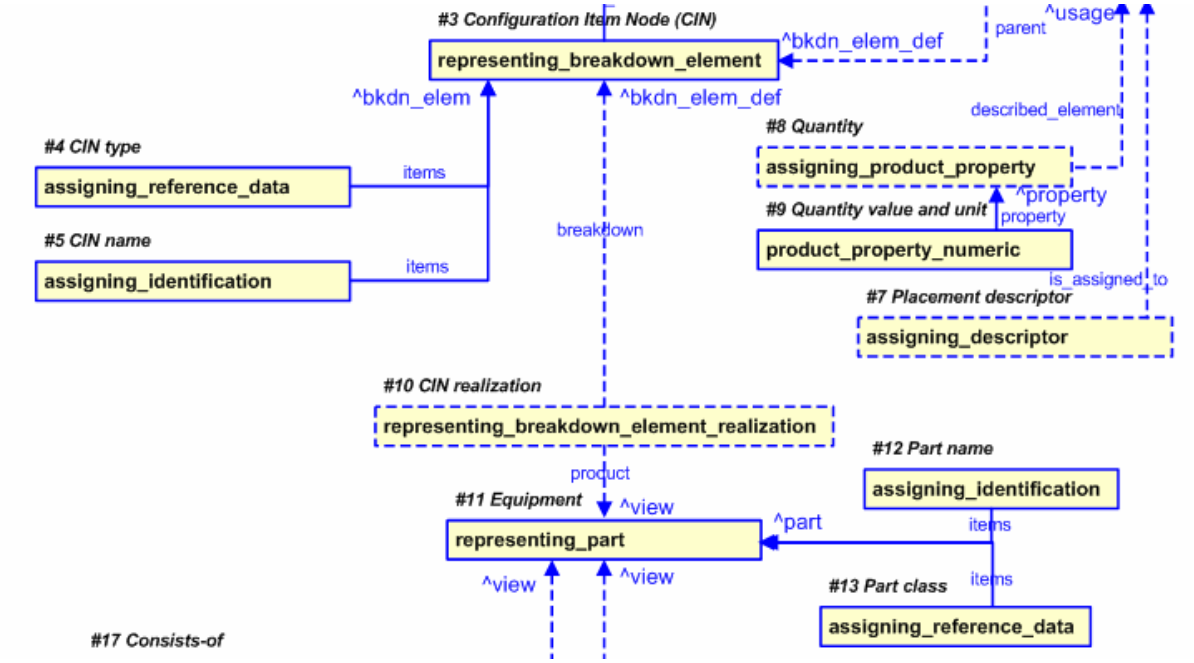



**Figure 3 DEX1 fragment**

A table is associated with each template instance. There is a row for each template parameter. Some parameters, like 'ecl_id', are bound to their final values at this stage. Others are partly resolved by narrowing their scope, but their final binding cannot take place until mapping or run time.

| **Description** | **Configuration Item Node (CIN) Type.** | |
|---|---|---|
| **Parameter name** | **Parameter value** | **Parameter description** |
| *class_name* | urn:epm:rdl:std:Product_variant<br>urn:epm:rdl:std:Subsystem<br>urn:epm:rdl:std:Hardware_item<br>urn:epm:rdl:std:Software_item | The configuration item type. A choice of four classes. One of them is selected and bound at adapter runtime. |
| ecl_id | 'urn:plcs:rdl:std' | The id of the External_class_library where the class definitions are found |
| *items* | Breakdown_element | To be bound at adapter runtime |

**Table 1 - Template instance: DEX1 #4 assigning_reference_data**

| **Description** | **Part class** | |
|---|---|---|
| **Parameter name** | **Parameter value** | **Parameter description** |
| *class_name* | urn:epm:rdl:std:Assembly<br>urn:epm:rdl:std:Component<br>urn:epm:rdl:std:Standard_part | The part class. A choice of three classes. One of them is selected and bound at adapter runtime. |
| ecl_id | 'urn:plcs:rdl:std' | The id of the External_class_library where the class definitions are found |
| *items* | Part | To be bound at adapter runtime |

**Table 2 - Template instance DEX1 #13 assigning_reference_data**

### 3) Templates instances as mapping targets.

Suppose we are going to develop an adapter and we have a legacy source model that among other tables has the ones named CONFIG_ITEM and PART. CONFIG_ITEM has among other fields the attribute named TYPE and PART has among other fields the attribute named CLASS. The mapping table underneath is a simplified one. Most of the template instances in the DEX fragment Figure 3 are left out and the parameters of template instance #3 are deliberately left out. The intention is just to exemplify template instances #4 and #13 used as mapping targets.

| TI_ID | Template type and params | Comment | Source |
|---|---|---|---|
| *Dex1.#3* | *representing_breakdown_element* | One-to-one | CONFIG_ITEM |
| *Dex1.#4* | *assigning_reference_data* | | |
| | *class_name* | Mapping of types to class names:<br>1 -> urn:epm:rdl:std:Product_variant<br>2 -> urn:epm:rdl:std:Subsystem<br>3 -> urn:epm:rdl:std:Hardware_item<br>4 -> urn:epm:rdl:std:Software_item | type |
| | *Items = #3* | | |
| *Dex1.#11* | *representing_part* | One-to-one | PART |
| *Dex1.#13* | *assigning_reference_data* | | |
| | *class_name* | Mapping of part types to class names:<br>A -> urn:epm:rdl:std:Assembly<br>B -> urn:epm:rdl:std:Component<br>C -> urn:epm:rdl:std:Standard_part | class |
| | *Items = #11* | | |

**Table 3 - Mapping table**

### 4) As populated

An adapter implemented in accordance with the above three steps and the mapping shown in Table 3, should produce a resulting PLCS population as the one shown below (in P21 format), provided that the source population has one CONFIG_ITEM whose type attribute equals mapping option 3 (i.e. Hardware_item) and one PART whose class attribute equals mapping option 'C' (i.e. Standard_part).

Please note that templates are development artefacts, and that they merely are abstractions in the final PLCS population.

```
#1= EXTERNAL_CLASS_LIBRARY('urn:epm:rdl:std,'/IGNORE');
#2= EXTERNAL_CLASS('/NULL','Standard_part','/IGNORE',#1);
#3= CLASSIFICATION_ASSIGNMENT(#2,(#4),'/IGNORE');
#4= PART_VIEW_DEFINITION('/IGNORE','/IGNORE',$,$,(),#5);
#5= PART_VERSION('/IGNORE','/IGNORE',#6);
#6= PART('/IGNORE','/IGNORE','/IGNORE');
#7= EXTERNAL_CLASS('/NULL','Hardware_item','/IGNORE',#1);
#8= CLASSIFICATION_ASSIGNMENT(#7,(#9),'/IGNORE');
#9= BREAKDOWN_ELEMENT('/IGNORE','/IGNORE','/IGNORE');
```

# 3 Overview of the Development Process

The table below is meant to be a compressed overview entire development process as lined out in the AIA EDIG Guidebook [5]. For each phase there is a set of activities with their corresponding deliverables. In software development projects of such a scale other deliverables might occur, but the table is limited to the essential key deliverables.

| | **Analysis** | **Specification** | **Implementation** | **Testing** |
|---|---|---|---|---|
| **Activities** | Describe Business Process<br>Identify Data Exchange Requirements<br>Describe IT Systems (Data Sources / Targets)<br>Define Data Flow | Specify Communication Interfaces<br>Specify DEX(s)<br>Specify Adapters<br>Specify Data Exchange Architecture | Implement Adapters<br>Implement Data Exchange Architecture | Verify Adapters<br>Verify Data Exchange Processes<br>Verify Complete Business Process |
| **Related Deliverables** | Business Process Model | | | Test reports for business requirements |
| | Data Model of Data Exchange Requirements | Business DEX specifications<br>Mapping specifications<br>Interface definitions and specifications | Adapter code | Test reports for adapters |
| | Data Flow Model between IT Systems | Detailed Data Flow Diagram / Sequence Diagram | Data Exchange and Integration System | Test reports for data exchange and integration |
| **Types of Tools** | See examples in Appendix D – PLCS Tools (EDM™) and Appendix E – PLCS Data Exchange Framework | See examples in Appendix D – PLCS Tools (EDM™) and Appendix E – PLCS Data Exchange Framework | See examples in Appendix D – PLCS Tools (EDM™) and Appendix E – PLCS Data Exchange Framework | See examples in Appendix D – PLCS Tools (EDM™) and Appendix E – PLCS Data Exchange Framework |

**Table 4 - Development process overview**

# 4 The Business Case

The motivation behind the introduction or enhancement of standard based data exchange solution(s) in an organization might be one or more of the ones listed below. (The list is not considered to be an exhaustive one)

The introduction or enhancement might be triggered by:

- Governmental regulations
- Long-term data retention and archival requirements motivated by possible liability issues
- Interoperability requirements in the customer-supplier chain
- The necessity to eliminate the costs of maintaining several point-to-point legacy links
- The strive to improve process and product quality by eliminating manual and hybrid links, which are potentially error-prone

The business case is more of a prerequisite than a result of the development process. The requested solution is limited by the scope defined by the business case, and by the budget and resources granted. Company policies and strategies might moreover put constraints on the way software is developed, acquired and modified. These issues are elaborated in finer detail in chapter 5 the AIA EDIG Guidebook [5].

# 5 Analysis Phase

The activities in the analysis phase are guided by the scope and limitations defined by the business case. In this phase one should team up with domain experts, system architects and stakeholders from the user communities for the purpose of describing the present enterprise solutions. Business processes along with the IT-systems that support them should be modelled. The flow of data between processes and systems should be described regardless of whether the methods are manual or automated, standardized or native. Such an analysis is explained in further detail in chapter 4 in [5], the AIA EDIG Guidebook - Aerospace Industry Guideline for Implementing Interoperability Standards for Engineering Data. Figure 4 below, depicts an overview of the activities in the analysis phase.

The report from the analysis phase might conclude that the present architecture serves its purpose well, and that the requirement is to automate manual exchanges and standardize native ones.

In other cases gaps between the present solution and a required improved solution is reported. In such case one might conclude that it is beneficial to alter parts of the overall architecture at the same time as the standard based information backbone is established.

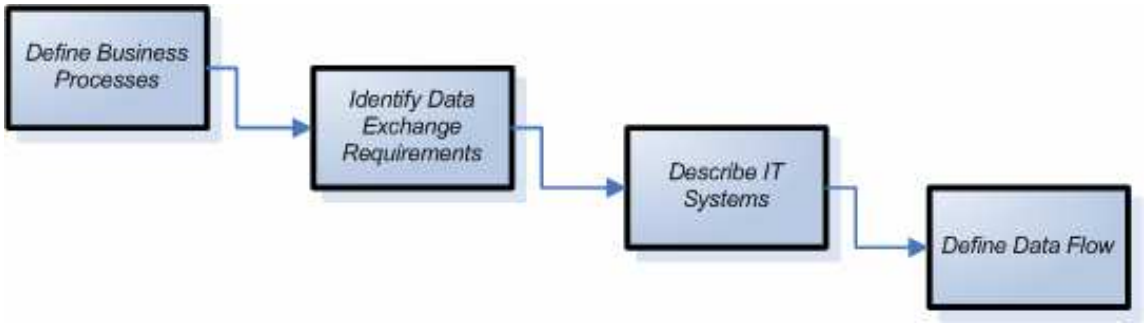



**Figure 4: Activities in the analysis phase**

## 5.1 Define Business Processes

An IT solution supports one or several business processes. In large enterprises such a solution normally consists of a plethora of systems and their corresponding applications. Some particular task belonging to a business process can be supported by several applications. The need for data-exchange arises when one task belonging to a business process produces data, which are consumed by another task, and those tasks are supported by different systems. In this context a business process model aims to identify data exchange requirements. The business process model should be kept at a level of breakdown granularity that enables all stakeholders to comprehend and evaluate the requirements.

Figure 5 illustrates a process model, which spans across four different systems.

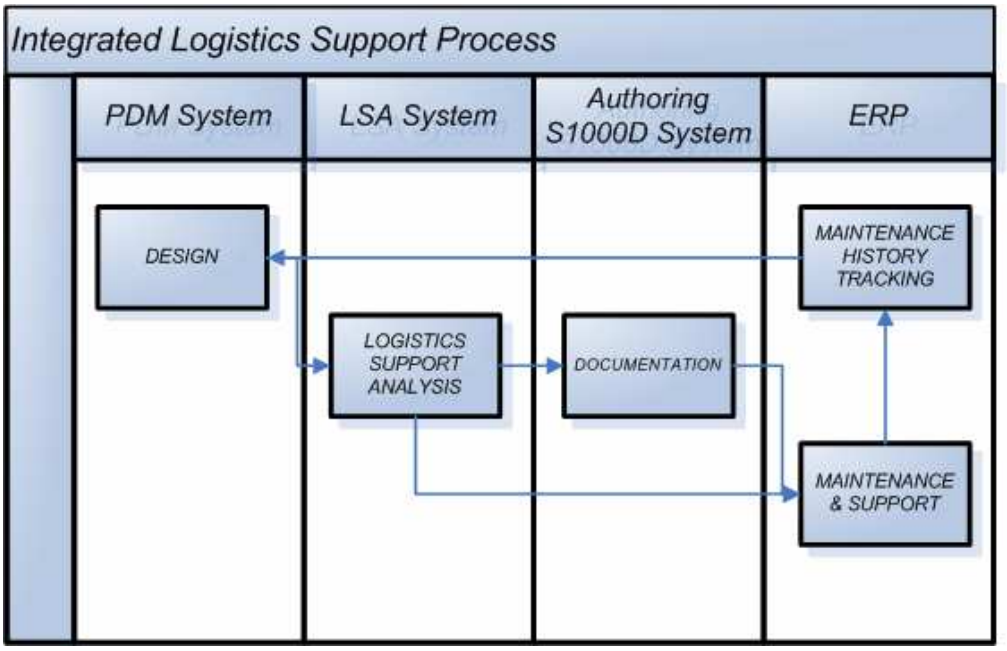



**Figure 5: Example of a business process model**

A survey can span the entire product life cycle from requirement to product retirement and result in a main overview chart as the one above. But implementing a standard based information backbone, which encompasses all systems, is for most organizations probably a resource-demanding endeavour. The scope should therefore be narrowed with cost benefit optimization in mind. Most organization would probably make use of an iterative strategy by initially trying to eliminate the worst process bottleneck with respect to missing or poor data exchange capabilities. Based on the outcome of the evaluation, performed at iteration end, one can plan for a next iteration, or abandon the development due to limited success.

The process model illustrated in Figure 5 is too coarse grained for most practical purposes and must be broken down into further details. Figure 6 shows an example of how the integrated logistics support processes can be broken down in order to identify the data exchange requirements.

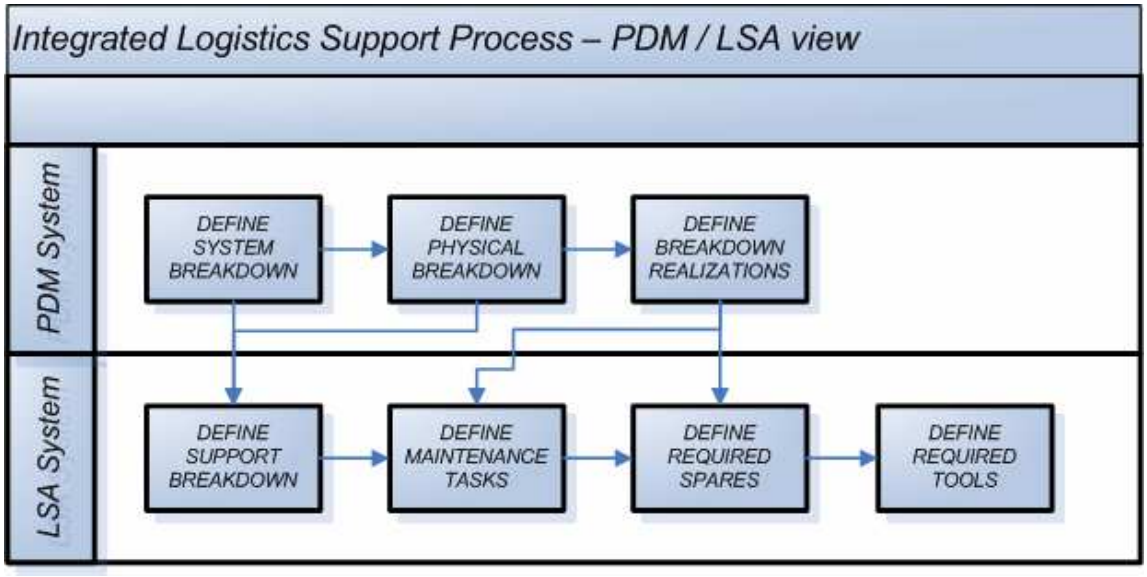



**Figure 6: Example of a detailed process**

## 5.2 Identify Data Exchange Requirements

Once an overview of the business process is in place, a model of the data to be exchanged has to be defined. It is not required for such a model to be defined in formal syntax using a data-modelling tool. The purpose is just to provide all stakeholders with the means to determine that all required information is included.

Figure 7 illustrates in an informal way the data exchange requirements between the design and integrated logistics support analysis modules. The requirement is to exchange product data that consists of physical breakdown, system breakdown and part assembly breakdowns. The best approach for defining the requirements is to work backwards from the receiving end - identifying its input requirements rather than identifying the full capacity of the sender as this may result in more data being exported than is necessary.

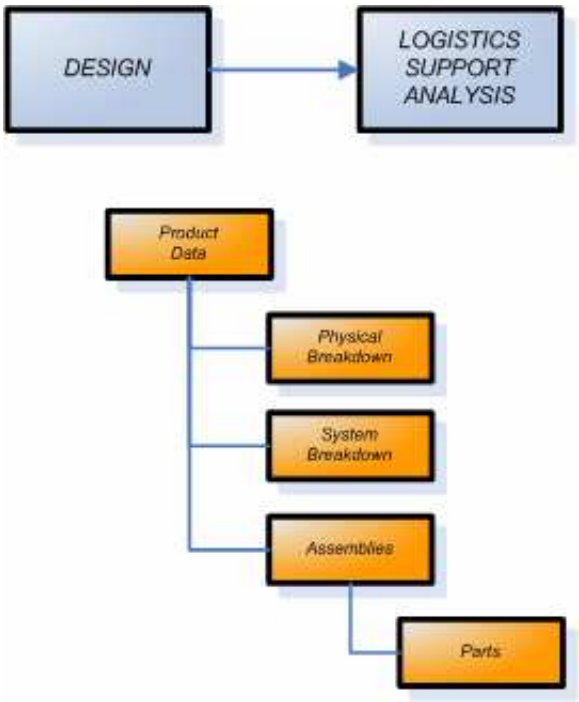



**Figure 7: Example of high-level exchange requirements**

Data models with such a coarse granularity are useful to some extent, but for the purpose of implementation it has to be broken down into further detail. Classes (or entities in the Express language), along with their associated attributes and relationships should be modeled. The purpose is to establish a common reference model as a vehicle for communication between the various stakeholders and across organizations and disciplines.

Figure 8 shows a simple common reference model including one single entity named PART. The PART has three attributes by the name of PART_NUMBER, PART_NAME and PART_TYPE. The PART_TYPE can either be EQUIPMENT or STANDARD_PART.

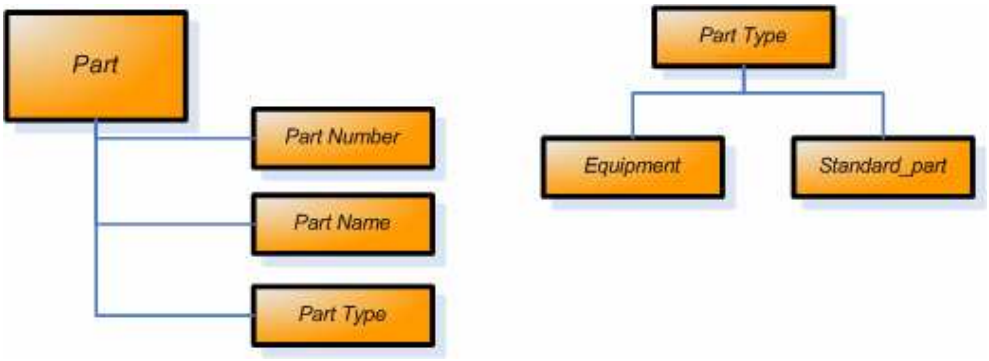



**Figure 8: Example of a common reference model**

The common reference model can be kept in an informal style, like the one in Figure 8, using any general-purpose drawing tool. However, JOTNE advocates that to comply with standards one should use the EXPRESS-G notation to illustrate the reference model. Figure 9 introduces a more detailed example using an EXPRESS-G model produced with EDMVisualExpress™ from Jotne EPM Technology.

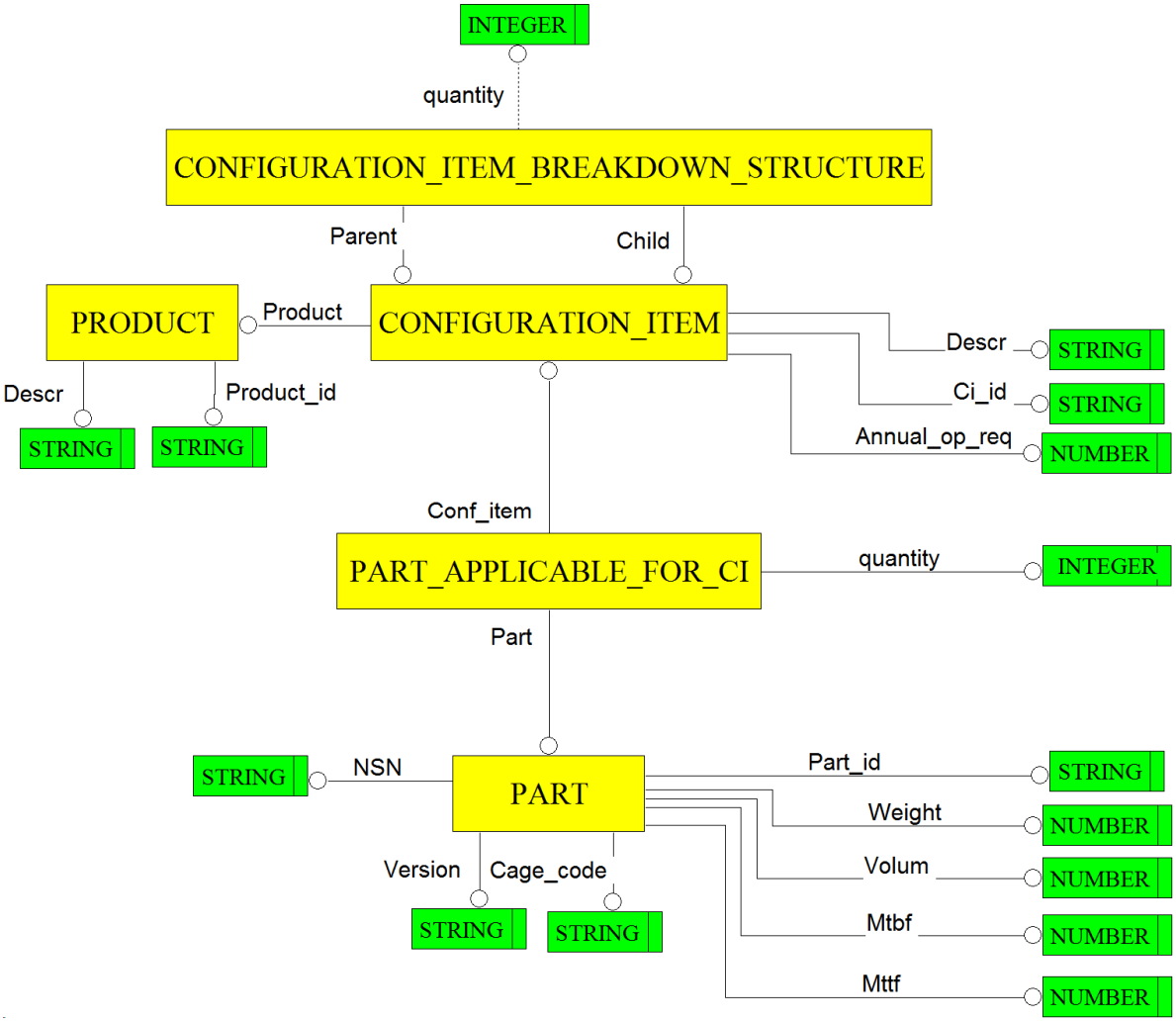



**Figure 9 A common reference model showing a product breakdown structure**

## 5.3 Describe IT Systems

Once the data exchange requirements have been identified, a survey of the existing IT-systems and applications should be performed, and a sketch of the architecture should be established, provided that such documentation does not already exist. The purpose is to probe the systems already in use for exchange capabilities that are in compliance with the common reference model. Based on the gap analysis report the decision will be one of the following:

- Purchase a new module that possesses the required capabilities.
- Develop export/import functionality layered on top of an embedded data access API.
- Develop export/import functionality on top of the bare bones of the database system.

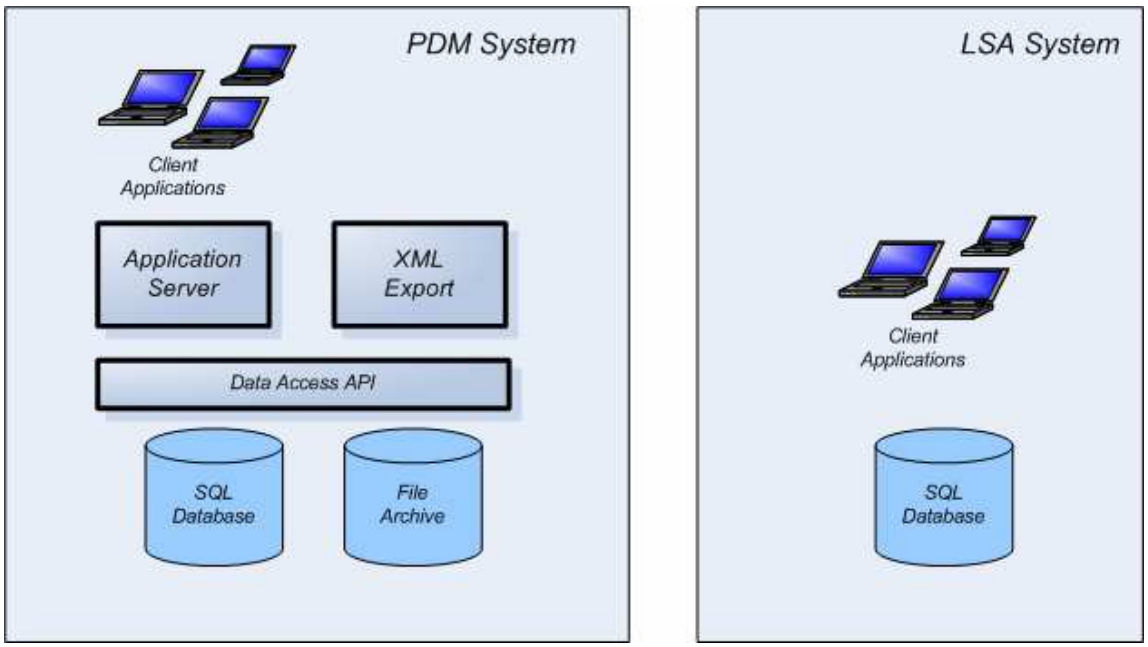



**Figure 10: System architecture**

It serves the purpose to keep the system architecture overview as simple as possible by only including those components and modules that are relevant for the data exchange described by the common reference model.

## 5.4 Define Data Flow

The business process model defines how tasks are related to each other and how IT systems support the various tasks [see Figure 5]. It is not obvious, however, how data actually flows between the various systems. The data flow overview diagram is a crucial input for the development of the data exchange architecture specification.

Each data exchange scenario needs to be orchestrated. In the most advanced case a workflow engine orchestrates such scenarios, thus ensuring that the process tasks are synchronized as intended. In the simplest scenarios messages are manually relayed (mail/phone). The control offered by an advanced system will result in fewer delays and less faulty situations. The upfront investment is higher though. This cost benefit trade-off has to be performed from case to case.

Figure 11 depicts an example of such a data flow scenario. There is one simple data exchange between the PDM system and the LSA system, and another one between the ERP and PDM system. The Support Data flow is, however, a little bit trickier. The ERP system requires Support Data supplied by the LSA system as well as by the Authoring S1000D system. The Authoring S1000D system requires Support Data from the LSA System as well, before it can deliver its part of the Support Data to the ERP system. Subsequently a point-to-point exchange is not possible, since the Authoring S1000D system does not handle the full scope of the Support Data. It is therefore necessary to integrate the data from the LSA System with the data from the Authoring S1000D before it can be delivered to the ERP system.

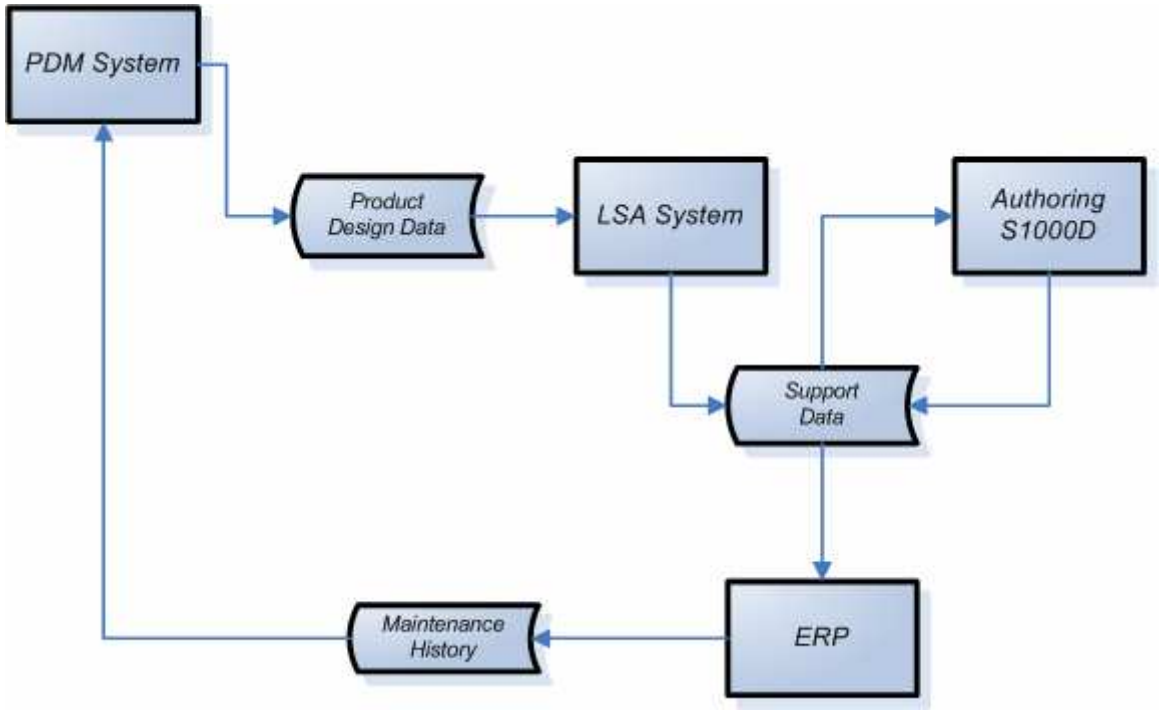



**Figure 11: Example of data flow**

Sequence diagrams, as the one in Figure 12, can be applied as a basis for orchestration regardless of how sophisticated the implemented is.

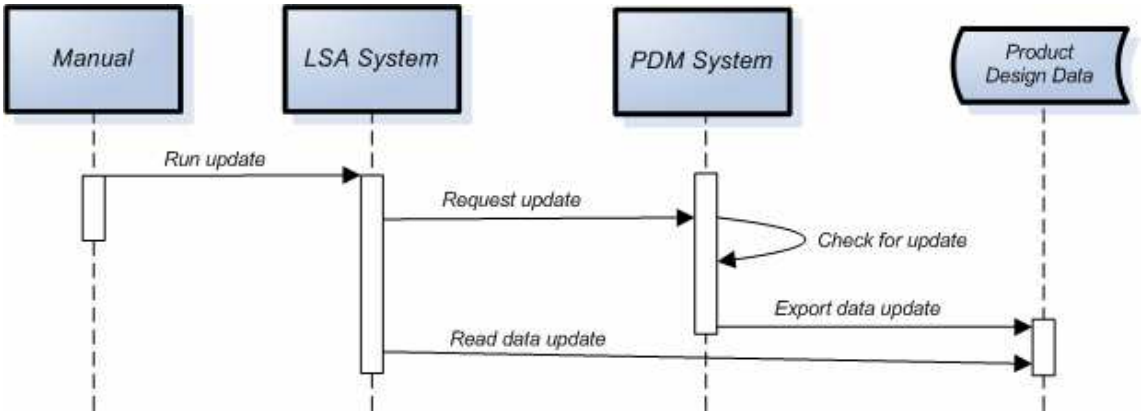



**Figure 12: Example of data flow sequence**

# 6 Specification Phase

There are, as depicted by Figure 13, four main sections of the specification phase.

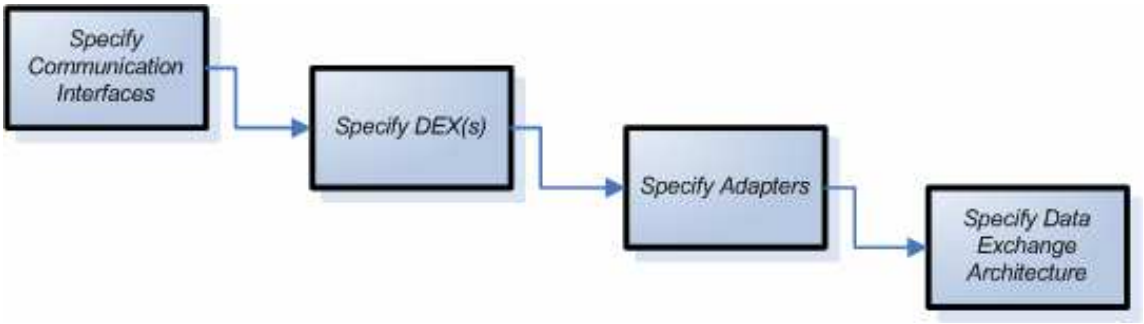



**Figure 13: Activities in specification phase**

## *6.1 Specify Communication Interfaces*

Based on the data exchange requirements from the analysis phase, export interfaces for the involved IT-systems have to be specified.

Communication interfaces can be characterized as:

1. File export -

   where an already existing capability of one of the software modules of the donor IT-system might suffice and hence utilized. Otherwise, there might be a possibility to develop an export in case one of the modules is equipped with an API.

2. Relational Database dump -

   where a pure replica of the relevant database tables dumped onto file(s).

3. Web Service XML streaming -

   where one of the modules has a capability to envelope XML formatted export into a stream, which is consumed by an XML parser embedded in the adapter. The same considerations for reuse versus development as for file export are relevant.

The example common reference model might be streamed according to the schema shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://www.example.org/NewXMLSchema"
xmlns:tns="http://www.example.org/NewXMLSchema"
elementFormDefault="qualified">

    <complexType name="Assembly">
      <sequence>
            <element name="Part_number" type="string"></element>
            <element name="Part_name" type="string"></element>
            <element name="Consist_of" type="tns:Component"></element>
      </sequence>
    </complexType>

    <complexType name="Component">
      <sequence>
            <element name="Part_number" type="string"></element>
```

4. Proprietary API -

   where a proprietary API belonging to one of the modules is embedded in the adapter.

5. Open Database Connectivity software API -

   where an ODBC connection is embedded in the adapter.

It does not make sense to recommend one single practice. It all depends on the possibilities for reuse of existing export capabilities.

A thorough documentation explaining how the export data-formats relate to the common reference model will ease the specification and implementation of adapters.

Figure 14 shows an example with the relationship between the XML Schema for the PDM system and the common reference model.

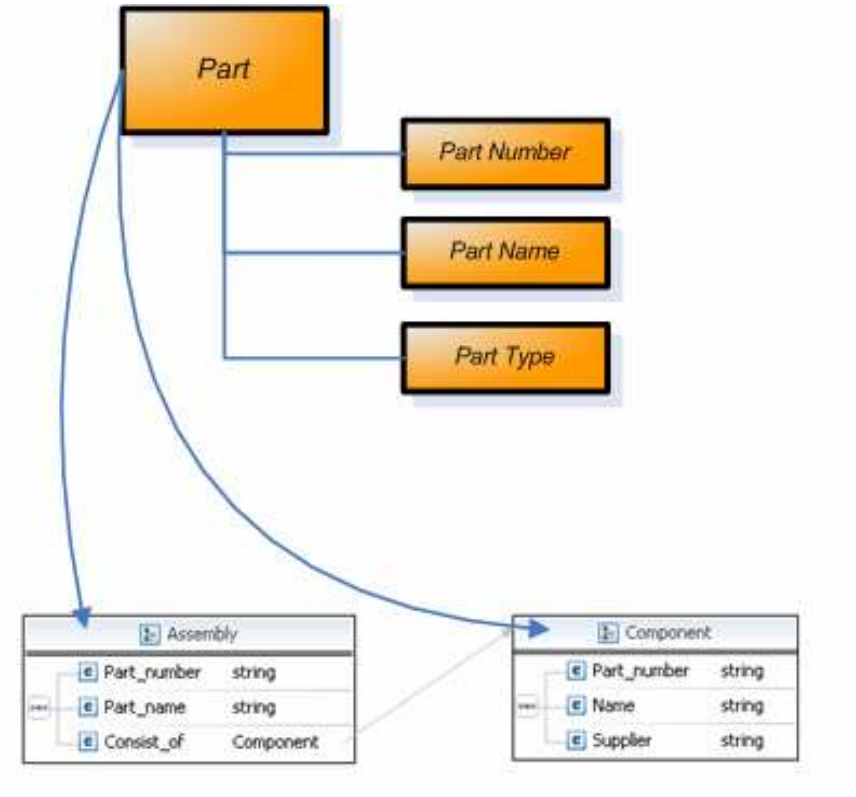



**Figure 14: Relationship between common reference-model and interface data-model**

## 6.2 Specify DEX(s)

The purpose of this activity is to identify or specify the DEX(s) needed to support the required data-exchange. Such specifications are the basis for implementation of adapters.

Given the high-level data exchange requirements and the common reference model from previous chapters [Figure 9], the DEX(s) listed in Table 5 - Available DEXes must be inspected for coverage of these requirements.

| DEX # | Name | Description |
|---|---|---|
| D001 | Product Breakdown for Support | |
| D002 | Faults related to product structures | |
| D003 | Task Set | |
| D004 | Work Package Definition | |
| D005 | Maintenance plan | |
| D007 | Operational Feedback | |
| D008 | Product as individual | |
| D009 | Work Package Report | |
| D010 | System Requirements | |
| D011 | Aviation maintenance | |

**Table 5 - Available DEXes**

Hopefully DEX(s) that meet the requirements are found. However, if that is not the case, there are two possibilities. If there is a minor gap between the requirements and one of the available standard DEXes, a business DEX should be derived thereof, otherwise a brand new DEX must be specified. The procedure is elaborated in somewhat more detail in chapter 6 in [5], AIA EDIG Guidebook - Aerospace Industry Guideline for Implementing Interoperability Standards for Engineering Data.

### 6.2.1 Specify a new DEX

However, if no standard DEX matches the requirements with sufficient coverage, then this may indicate that this is an uncovered domain area or discipline so far. It is very likely that many of the requirements will be met, but not precisely or fully. Therefore to produce a satisfactory result it is advisable for the project team to engage someone with sufficient PLCS expertise and someone with a good knowledge of the domain. One should strive to reuse existing templates as far as possible. But the introduction of new ones is not strictly prohibited – specifically when creating a business DEX. The working drafts of the project, like for example instance diagrams and template instance tables, might be kept in any format and style using the tools of ones choice. However, whenever publishing a DEX on DEXlib, however, the conventions and rules prescribed by OASIS must be obeyed.

### 6.2.2 Specify a business DEX

There are three areas of concern when a business DEX is derived from an existing standard DEX:

1. The extension of reference data
2. The extraction of a suitable subset of the underlying standard DEX
3. The definition of specific business rules

By the extension of reference data new properties or classification can be added whereas existing ones can be sub classed.

It is not mandatory for the business DEX to include all sections of the underlying DEX to its full extent. In many cases a subset of the DEX is sufficient to cover the data exchange requirements. The DEX can then be further constrained with business rules to validate dependent data.

As an example Figure 16 shows a fragment of a business DEX derived from DEX003 (Figure 15). The common reference model as shown in Figure 8, specifies that a part should have a part number, a part name and a part type. Since the template representing_part already has an identifier embedded in it, we do not need the additional identifier. The part support classification is not required, but a classification for the part type should be added instead. Note that in Figure 15 the three 'assignments (#2, #3 and #4) are all optional (indicated by the dashed lines), but those in Figure 16 for #2 and #3 are not (i.e. they become mandatory in the business DEX).

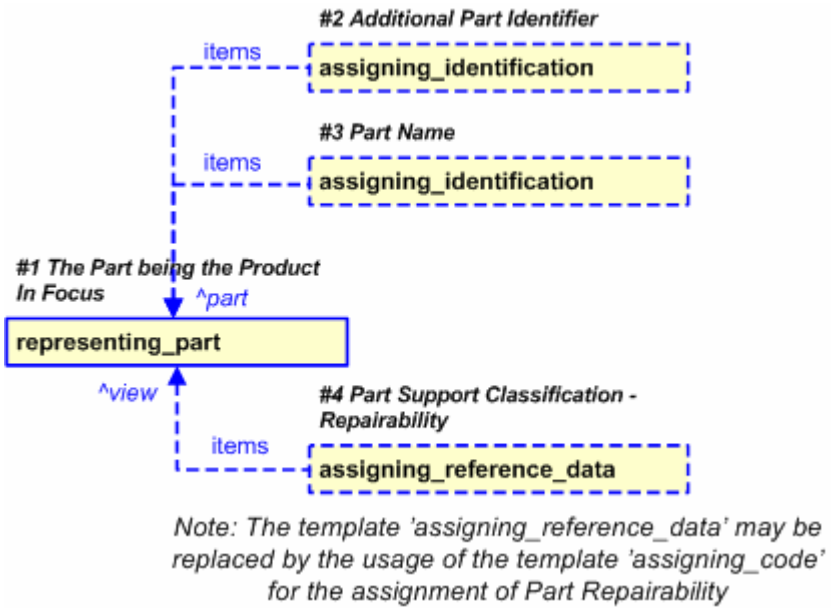



**Figure 15: representing_part in DEX003**

The fragment of the business DEX below is derived from the DEX003 standard DEX shown above.

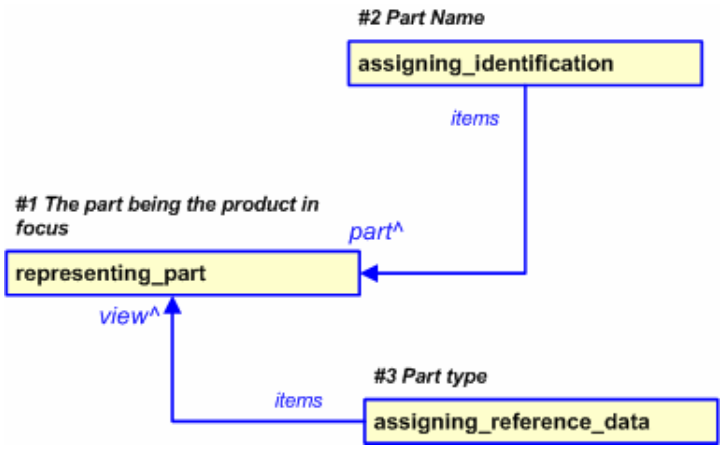



**Figure 16: Example of representing_part in a business DEX**

## 6.3 Specify Adapters

Once the requirements in the form of a common reference data model, (the source – see section 5.2), and a DEX, (the target – see section 6.2), has been selected or developed, the mapping between the two of them has to be performed. This mapping is basically a table, which specifies the correspondence between entity attributes of the source model with template instance parameters of the target DEX.

In case one of the existing export capabilities, as described in chapter 6.1, is utilized there might be a considerable gap between the common reference model and the way data is structured on the file or stream. In such a case one should consider to implement the adapter in two steps making use of an intermediate mapping.

Unlike the examples shown in Table 3 and Table 6 mapping tables in most real projects are extremely detailed, voluminous and not so easy to oversee. That is why it is useful to make use of a graphical overview in order to visualize how the main concepts (entities) of the source model map to template instances of the target DEX.

Table 6 shows exemplifies attribute to template parameter mapping.

| DEX-Template | Maps from |
|---|---|
| #1 representing_part.id | Assembly.Part_number<br>Component.Part_number |
| #2 assigning_identification.id | Assembly.Part_name<br>Component.Name |
| #3 assigning_reference_data.class_name | if(Assembly) -> 'Equipment'<br>if(Component) -> 'Standard_part' |

**Table 6 - Mapping table example**

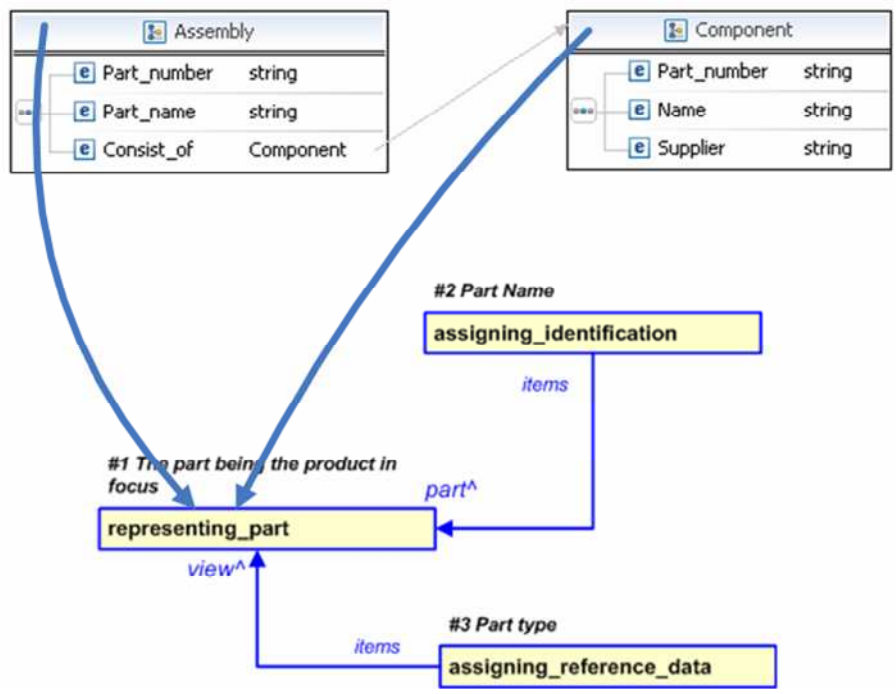



**Figure 17: Example of high level mapping specification**

## 6.4 Specify Exchange Architecture

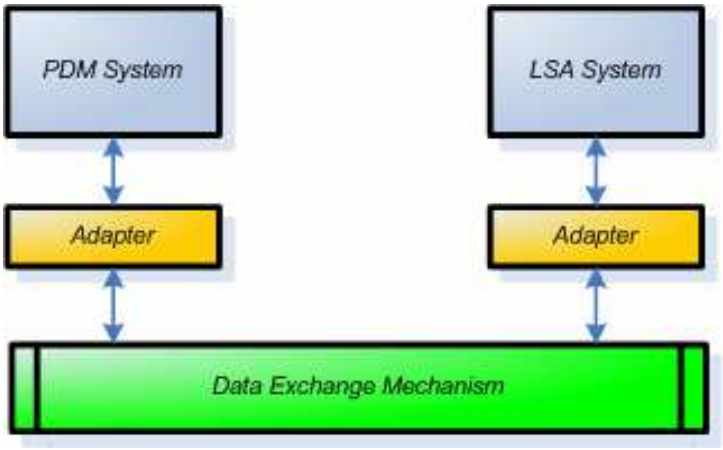



**Figure 18: Data Exchange Architecture**

Data exchange mechanisms can range from very simple point-to-point exchanges to advanced data sharing solutions.

An exchange file can in the simplest case be handed over manually by means of memory stick or an email. In the more advanced solutions the exchange scenario might be orchestrated within a Service Oriented Architecture (SOA) by means of a workflow engine where the master data is kept in an integrated repository and web services provide part of the exchange mechanism.

It is considered out of the scope of this handbook to recommend a data exchange practice. It all depends on the scale of the exchange scenario, as well as the ambitions of the organization and the resources made available for the project. However, we do provide an overview of one of Jotne's tools that provides an example of an advanced solution in Appendix C – PLCS in a Service Oriented Architecture.

# 7 Implementation Phase

## *7.1 Implementation of Adapters*

Initially one should recognize the fact that the OASIS specifications are organized in layers. Based on the experience gained from PLCS projects over the past years, the authors of the handbook have reached the conclusion that the best practice is to organize the software as a mirror of these layers.

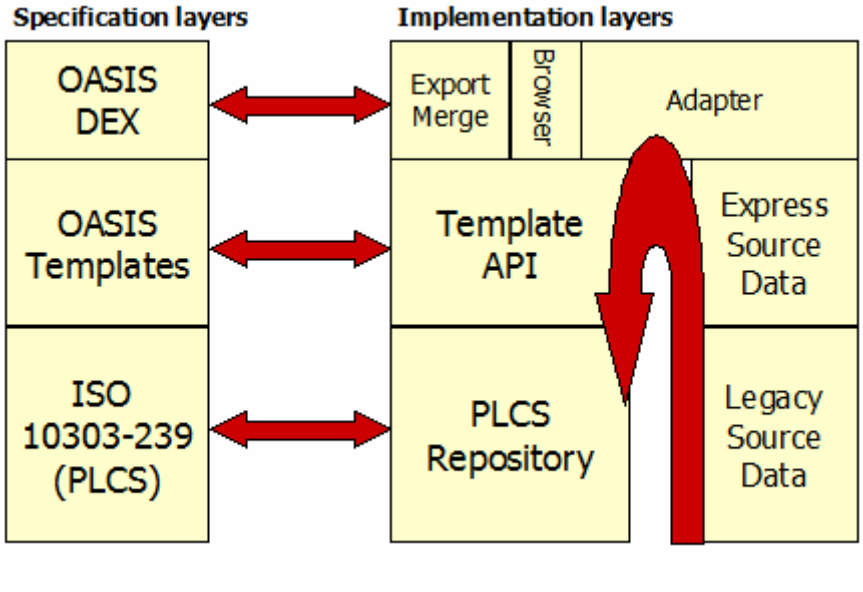



**Figure 19 Specification and implementation layers**

The core of the specification is the AP239 schema. A pure PLCS data repository is, as illustrated by the sketch above, arranged exactly in compliance with the AP239 schema. Other solutions using the AP239 schema as a logical schema mapping to another physical storage structure might also be possible.

There are, however, some major advantages with a PLCS repository that is in complete compliance with AP239. Data models defined in EXPRESS are human readable. Equally important is the fact that the language obeys a strict grammar so that it is also "computer readable". The dictionary model for a data model defined in EXPRESS can be established directly by a compiler. There is no such thing as a physical data base schema, and consequently the "impedance" one face when interfacing from a logical to a physical model, is simply avoided. Hence the risk that data is lost or distorted in such an interface is eliminated.

In order to fully exploit the potential of EXPRESS, one needs a database management system with built-in support for this language. Such a database is said to be EXPRESS-driven. All methods like translations, validations and queries can access the data repository and the corresponding meta-data directly by means of the EXPRESS-X language (ISO-10303-14).

The OASIS template definitions are layered on top of the AP239 schema. Each and every template has a number of formal parameters as part of its definition. That is why it is feasible to implement the OASIS templates as an API, like Jotne EPM Technology has done with its EDMtemplateAPI™. In this API there is a "create" and a "get" function for each template, where the formal parameters of the templates are mapped to the signature of each function. Such an API encapsulates and hides the relatively high complexity of the internals of the templates. Clients of the API, like developers of adapter code, can concentrate on the logic of the adapter without having to spend time studying the details of PLCS and the OASIS template definitions.

As seen from the example below, the EDMtemplateAPI™ implements signatures of the "create" and "get" functions for the "assigning_reference_data" template corresponds to the structure and parameters of the template (see Table 1 and/or Table 2 in section 2.3.1.1).

Whenever invoking a "create" function, values for all the parameters must be given. The "create" functions are implemented in such a way that the uniqueness rules as described in the template definitions are obeyed. This means that a handle to an already existing object is returned in case an object with the very same parameter set has already been created.

```
Function  create_assigning_reference_data
             (class_name : string;
             item : plcs::classification_item
             ) : plcs::assigning_reference_data;
```

The "get" functions are implemented in such a way that any of the parameters may be omitted upon invocation.  Consequently one does not always match a single object, and a list of objects is returned. For instance if only the 'item' parameter is given, and the value of 'class_name' is not specified (indeterminate in the EXPRESS-X language) all classifications for the item will be returned.

```
Function get_assigning_reference_data
            (class_name : string;
             item : plcs:: classification_item
            ) : LIST OF plcs:: assigning_reference_data;
```

The reader might have noticed that the 'ecl_id' parameter is left out. This is done in order to shorten the parameter lists, which are rather long for many of the templates. This is done under the assumption that the reference data library is the same throughout a session. The 'ecl_id' is handed over to the API by means of a separate function.

Typical clients of the API are adapters, browsers and export/import programs. These applications correspond to the DEX layer of the specification stack. Adapters might be seen as consisting of a front end as well as a back end. The back end will be the same for all of the adapters, namely the template API, whereas the front end that is tied to a specific, legacy model will be a different.

### 7.1.1 Template instance wrappers

The "create" functions of the template API may of course be called directly from the translator code. It is, however, a recommended practice to implement a wrapper function for each template instance. The names of such wrapper functions should act as references from the code to the template instance specifications and vice versa. It is believed that such an approach helps improving the readability of the code in the sense that it is absolutely clear at which point in the translator code a specific template instance is populated. The function, which corresponds to the template instance "DEX1#4 assigning_reference_data, could therefore have the name assigning_reference_data_1_4.

The arguments that are statically bound at translator development time will be bound in the template instance wrapper layer. As a consequence there is a strict rule for how to update the code if/when specifications are modified. In the brief EXPRESS-X example below a configuration item is classified as a hardware item.

```
asg_ref := assigning_reference_data_1(config_item);
.
.

FUNCTION assigning_reference_data_1_4 (classified_item : tar::classification_item
                                      ) :   tar::classification_assignment;
    LOCAL
      asg_ref : tar:: classification_assignment;
    END_LOCAL;

    asg_ref := create_assigning_reference_data(HARDWARE_ITEM, classified_item);
    RETURN(asg_ref);
  END_FUNCTION;
```

### 7.1.2 Reference data

In OASIS DEXlib specifications [1] reference data is defined by means of the **OWL, Ontology Web Language,** language and located on a collection of files. Such a representation is not suitable for an operational Reference Data Library. There might be several approaches for establishing an operational and an online RDL. In this handbook we will present Jotne EPM Technology's concept, which is based on **the ISO-12006-3** (IFD) standard. The **EDMrds™** [13] is a suite of modules tailored for the purpose of handling reference data.

#### *7.1.2.1 RDL Initialization*

For the purpose of initializing an RDL, based on the DEXIlib [1] OWL definitions the **owl2ifd** application is available. It takes a collection of OWL files as input and creates an RDL population with ISO-12006-3 as the underlying schema located on an EDMdatabase™. Please consult chapter 6 in the [13] for detailed user guidance.

#### *7.1.2.2 Development time*

At translator specification time an operational RDL is not actually needed. The mapping table simply refers to the class names as they are defined in the template instances of the actual target DEX. That is also the case for the source code. The present technique is early binding. e.g. class names are represented as string constants in the source code and the EDMexpressXCompiler™ has of course no means to check for proper class names against an RDL. The EDMtemplateAPI™, however, uses an operational RDL. Consequently, an RDL must have been initialized before the test debugging cycles starts (in most projects this should be at an early stage).

It is feasible, however, to make translators more flexible with respect to reference data. The approach is to make use of symbolic class names at translator development time, and switch those symbolic names into actual RDL class names by means of a mapping table at run time. In this method translators can be configured without altering the source code, but such an approach might require an RDL enhancement some time after the translator deployment. For example, a *"part version"*, which has been mapped to 'Version_identification_code' at development time should be specialised into 'Part_version_identification_code' afterwards. Two actions have to be taken: The table keeping the mapping between the symbolic name and the actual class name must be modified and 'Part_version_identification_code' has to be introduced as a sub class of 'Version_identification_code' in the RDL. See paragraph 7.1.2.3.

#### *7.1.2.3 RDL updates*

There are two possible ways to modify an RDL: One is to make use of the **EDMReferenceDataManager™**, please consult [13] for further information. The other way is to develop an application layer on top of the **IFD API** to read reference data definitions from a flat file or an Excel sheet into the RDL. Please see [14] for more information.

#### *7.1.2.4 Run time performance considerations*

Real-time access to an external reference data library at adapter's run time is not considered feasible performance wise. Generally, while large volumes of data are being processed, numerous lengthy or even nested iterations may be needed. Under such conditions accessing an RDL server by means of web services is not advised.

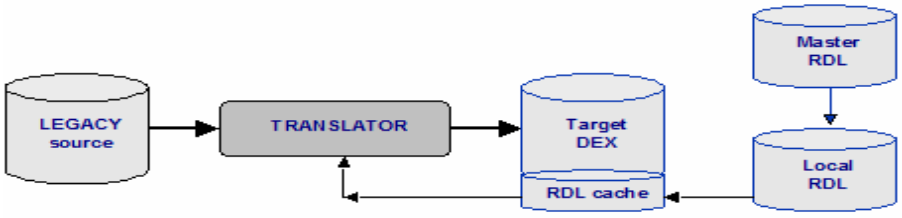



**Figure 20 RDL cache**

Jotne EPM Technology's solution is to make use of two levels of caching. A copy of the master RDL is copied into the operational database upon database initialization. Afterwards, the local RDL is cached into PLCS populations upon their initialization. For this reason the EDMtemplateAPI™ has refresh cache functionality. It works in such way that the PLCS populations are synchronized with the local RDL upon start of each session (web service, query). The "inner" cache is stripped off whenever the PLCS population are exported to the P21 or P28 files.

## 7.2 Implement Data Exchange Architecture

The purpose of this activity is to implement the specified data exchange architecture. An implementation can involve developing new software, customising existing systems or using commercial software packages. The work includes connecting adapters to the actual IT systems and to the data exchange mechanism.

Figure 21 shows an example of an implementation using custom developed software, open source software and commercial software.

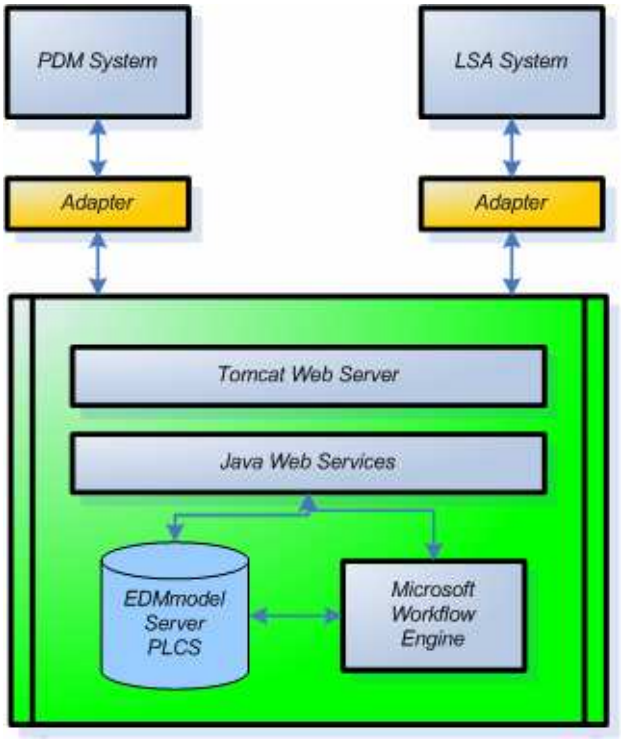



**Figure 21: Example of implementation of a data exchange architecture**

Section 6.1 identified a number of types of communication interfaces, including web services. In a traditional implementation using relational database technology, the standard (e.g. PLCS) schema has to be mapped to the proprietary implementation schema. Then in order to define a set of web services a set of (application dependent) stored sql procedures are defined to provide the functionality for the services being made available. Hence the code is written against the implementation schema, not the standard itself.

Using an EXPRESS-driven database, such as Jotne's EDM, means that there is no 'implementation schema' – thus, the functionality is coded against the standard itself. Thus implementers not need to learn, understand and maintain an implementation schema, nor will the implementation schema ever limit PLCS usage.

# 8 Verification & Validation

## *8.1 Verify Translators*

Four methods are recommended for the purpose of verifying translators:

- Verification of the legacy source data
- Generic AP239 verification
- DEX level verification
- Verification by extraction

### 8.1.1 Verification of source data

The legacy data model schemas will specify the subset of information that will be exchanged. Such schemas will in most cases reflect relational tables, from which the information will be extracted. In practice this means that the schemas will define entities that mimic relational records where the primary and foreign keys are modelled as strings. Apart from these keys there are no other entity relationships in the legacy data models. The rest of the entity attributes will be primitive attributes. This means that the validation of native populations will be quite simple. Checks for referential integrity are the most important rules. No intrinsic rules apart from the resident simple generic ones (like mandatory/optional attributes, max string lengths etc.) are considered necessary.

### 8.1.2 Generic AP239 verification

This level of verification is applied on the target population.

Such verification methods are ready made as an integral part of the AP239 schema, implemented as so called rules. In EXPRESS-driven systems, like the EXPRESSDataManager™, PLCS populations can be checked against the rules directly. In solutions other than EXPRESS-driven ones, the rule checker software has to be implemented using a conventional programming language, like Java. This level of verification is necessary but not sufficient in order to verify that a translator works as specified.

### 8.1.3 DEX level verification

This level of verification is applied on the target PLCS population. Such verification checks that the target population obeys the constraints of the actual DEX by executing the rules derived from DEXlib. In EXPRESS-driven systems, like the EXPRESSDataManager™, PLCS populations can be directly checked against the rules of the DEX schema. In solutions other than EXPRESS-driven ones, the rule checker software has to be implemented using a conventional programming language, like Java.

### 8.1.4 Verification by extraction

The purpose of this level of verification is to make sure that data is not lost, added or distorted by a translator. This is done by developing a suite of queries that extract reports from the source as well as from the target population. A typical report would be the flattening of a product breakdown structure on to a file. Another such report would the listing of all resources connected to all maintenance tasks. One should develop as many such extractions as needed to cover all structures, instances and attributes in a population. The principle is a simple one. The complete population must be flattened onto files for the purpose of comparison. Obviously this might be a time consuming operation.

Extraction queries should be developed as pairs. Whereas one query of such a pair extracts a report from the source population, the other one should extract a report from the target population. A translation is verified when all pairs of extraction queries have been executed, and all pairs of report files have been compared and no discrepancies have been detected.

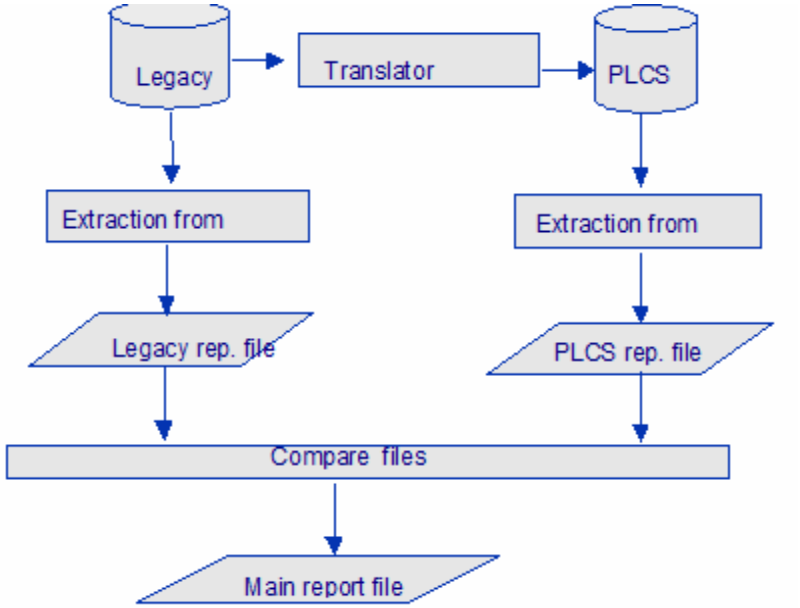



**Figure 22 Comparison by extraction**

## *8.2 Verify Entire Data Exchange Processes*

Whereas the method described in chapter 8.1 is applicable for the purpose of verifying one single translator step, the task to prove that the data has reached its final destination is pending.

End to end tests shall verify that data exchanged between two applications by means of the Standards-based-information-backbone is accurate; meaning that novel data elements and relations between them should be added, whereas already existing data elements and relations should not be replicated. The individual steps in the exchange shall be passive in the sense that data element values shall neither be modified nor distorted. Moreover, such tests should verify that no redundant or faulty data is conveyed to receiving end.

## 8.3 Verify Complete Business Process

Whereas the methods of automated verification described above are meant to assure that data is translated and exchanged without any loss or distortion, the verification of the complete business process can hardly be automated. Consequently, this level of verification must include the involvement of end user domain experts. By means of a detailed analysis, such experts should verify that the new PLCS based solution represents an integration vehicle that:

- Allows applications (in-house as well as external ones) to exchange information more efficiently than before, using the Standards Based Information Backbone.
- Eliminates/obsoletes manual procedures that so far has been time consuming and prone to errors.
- Sharpens the competitive edge through the capability to deliver standardized information to customers, as well as the capability to exchange such information with subcontractors.
- Improves the quality of the information being exchanged/shared (since not any data is lost or distorted)
- Ensures that the return of investment is as expected, due to the points listed above.

# 9 Appendix A – Abbreviations and Definitions

| | |
|---|---|
| **2D** | **Two (2) dimensional** |
| **3D** | **Three (3) dimensional** |
| **Adapter** | **Software that imports, transforms, loads, merges and exports data from one system to another. (An adapter is a device whose purpose is to convert attributes of one device or system to those of an otherwise incompatible device or system.** |
| **AIA** | **Aerospace Industries Association** |
| **AIC** | **Application interpreted construct** |
| **AIM** | **Application Interpreted Model (ISO 10303)** |
| **AP** | **Application Protocol (ISO 10303)** |
| **AP239** | **ISO 10303-239, "Product life cycle support" a.k.a. PLCS** |
| **AP203** | **ISO 10303-203, "Configuration controlled 3D design"** |
| **API** | **Application Programming Interface** |
| **ARM** | **Application Reference Model (ISO 10303)** |
| **ASD** | **AeroSpace and Defence Industries Association of Europe** |
| **CAD** | **Computer Aided Design** |
| **Characteristic** | **Abstraction of a property of an object or of a set of objects (ISO 1087-1 [8])** |
| **Class** | **Category or division of things based on one or more criteria for inclusion and exclusion (ISO 15926-1 [10])** |
| **Concept** | **A human understanding of an object unit of knowledge created by a unique combination of characteristics (ISO 1087-1 [8])** |
| **COTS** | **Commercial off-the-shelf** |
| **Data** | **Representation of information in a formal manner suitable for communication, interpretation, or processing by human beings or computers (ISO 10303-1 [6])** |
| **DBMS** | **Database management system** |
| **DEX** | **Data Exchange Specification (OASIS PLCS TC)** |
| **DQA** | **Data Quality Assistant** |
| **EDIG** | **Engineering Data Interoperability Working Group (AIA)** |
| **EDOM3** | **EDM Object Model, version 3** |
| **EDM** | **EXPRESS Data Manager™** |
| **EDMS** | **EDMSupervisor™** |
| **EPMT** | **Jotne EPM Technology AS** |

| | |
|---|---|
| **ERP** | **Enterprise Resource Planning** |
| **EXPRESS** | **Data modelling language, defined in ISO 10303-11** |
| **EXPRESS-X** | **Data manipulation language, defined in ISO 10303-14** |
| **GUI** | **Graphical user interface** |
| **ICT** | **Information & Computer Technology** |
| **ILS** | **Integrated Logistics Support** |
| **LSA** | **Logistics Support Analysis** |
| **Information** | **Knowledge concerning objects, such as facts, events, things, processes, or ideas, including concepts, that within a certain context has a particular meaning (ISO/IEC 2382-1 [7])** |
| **Information** | **Facts, concepts, or instructions (ISO 10303-1 [6])** |
| **ISO** | **International Organization for Standardization** |
| **ISO 10303** | **Industrial automation systems and integration - Product data representation and exchange** |
| **ISO 10303-11** | **Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual** |
| **ISO 10303-21** | **Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure** |
| **ISO/DIS 10303-28e2** | **Industrial automation systems and integration - Product data representation and exchange - Part 28: Implementation methods: XML representations of EXPRESS schemas and data** |
| **ISO 10303-239** | **Industrial automation systems and integration - Product data representation and exchange - Part 239: Application protocol: Product life cycle support** |
| **OASIS** | **Organization for the Advancement of Structured Information Standards** |
| **Object** | **Anything perceivable or conceivable (ISO 1087-1 [8])** |
| **P21** | **ISO 10303-21,** |
| **P28** | **ISO/DIS 10303-28e2** |
| **PDM** | **Product Data Management** |
| **PDT** | **Product Data Technology** |
| **PLCS** | **Product Life Cycle Support (ISO 10303 239)** |
| **RDL** | **Reference Data Library** |
| **Reference data** | **Data that represents information about classes or individuals which are common to many application areas or of interest to many users (15926-1 [10])** |

| | |
|---|---|
| **Reference data library** | **Managed collection of reference data (15926-1 [10])** |
| **R&D** | **Research & development** |
| **S1000D** | **International Specification for Technical Publications Utilizing a Common Source Database** |
| **SCORM** | **Sharable Content Object Reference Model** |
| **SDAI** | **Standard Data Access Interface (ISO 10303-22)** |
| **SRD** | **System Requirement Document** |
| **STEP** | **Standard for the Exchange of Product Model Data** |
| **URD** | **User Requirement Document** |
| **Validation** | **The process of evaluating software during or at the end of the development process to determine whether it satisfies specified requirements. [12] Confirms that the system meets the requirements defined in the user requirement document** |
| **Verification** | **The process of evaluating software to determine whether the products of a given development phase satisfy the conditions imposed at the start of that phase [12]. Checks that each component meets its specific requirement, usually as defined in the design document** |
| **XML** | **Extensible Mark-up Language** |

# 10 Appendix B – Translator Implementation Example

A notional example has been developed using the tools from Jotne EPM Technology that uses EXPRESS, EXPRESS-G and the EXPRESS-X ISO standards. The example assumes that a legacy relational database needs to be mapped to PLCS. It is based on the data model from chapter 5.2 and the diagram below is replica of *[Figure 9 A common reference model showing a product breakdown structure*].

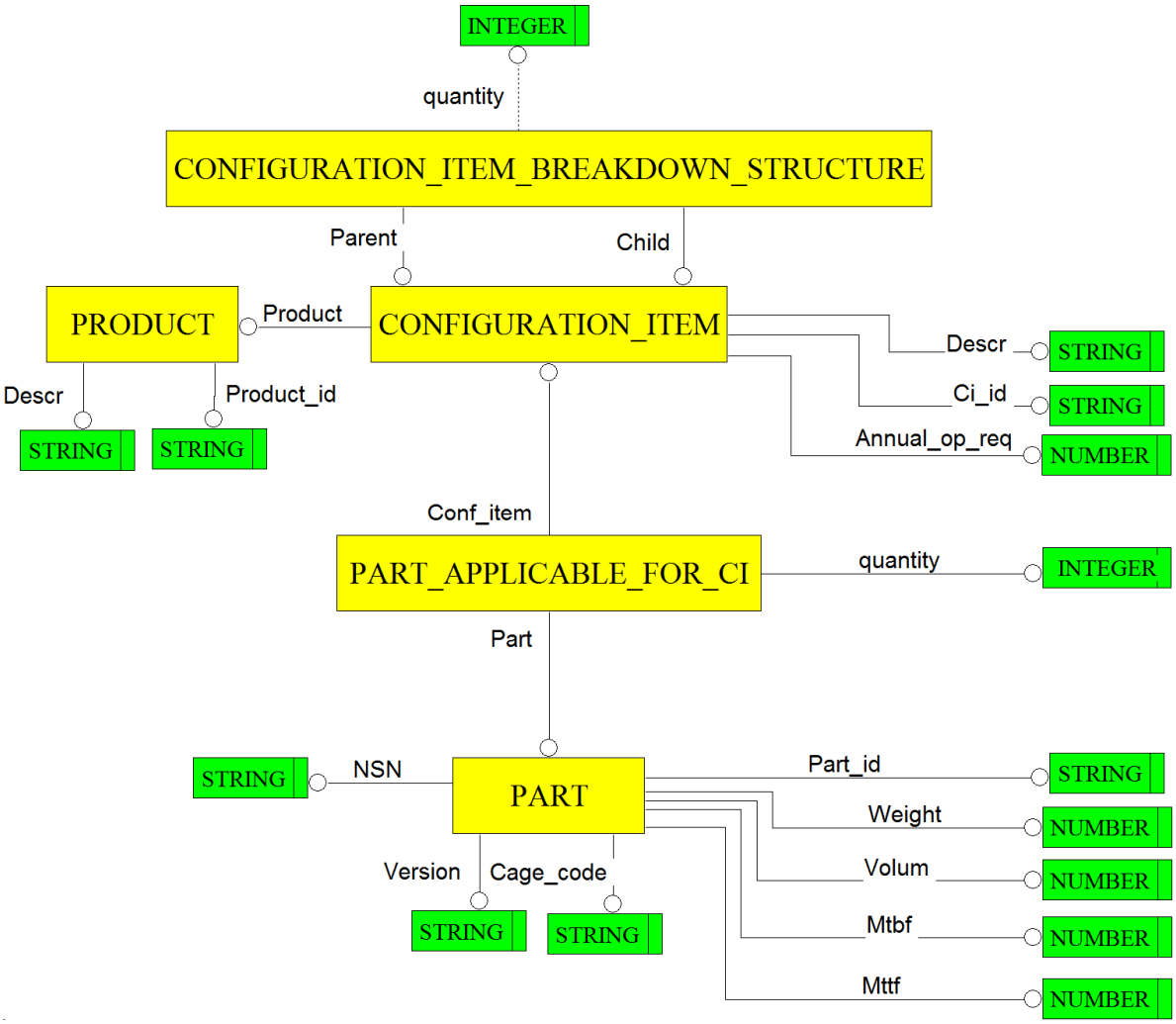



**Figure 23 Object model (in EXPRESS-G)**

In most cases, the actual legacy data source originates from a relational database. For example, the source (object) data model above, was created from an analysis of the (notional) relational data model depicted in Figure 24 below.

The EXPRESS-G standard is a graphical representation of the corresponding EXPRESS schema (a clear text representation that can be compiled). Using JOTNE's VisualEXPRESS™ tool, both the object schema e.g. the PDM object model (referred to as PDM_SCHEMA_O below), and the relational source schema e.g. a PDM relational model (referred to as PDM_SCHEMA_R below), can be created.

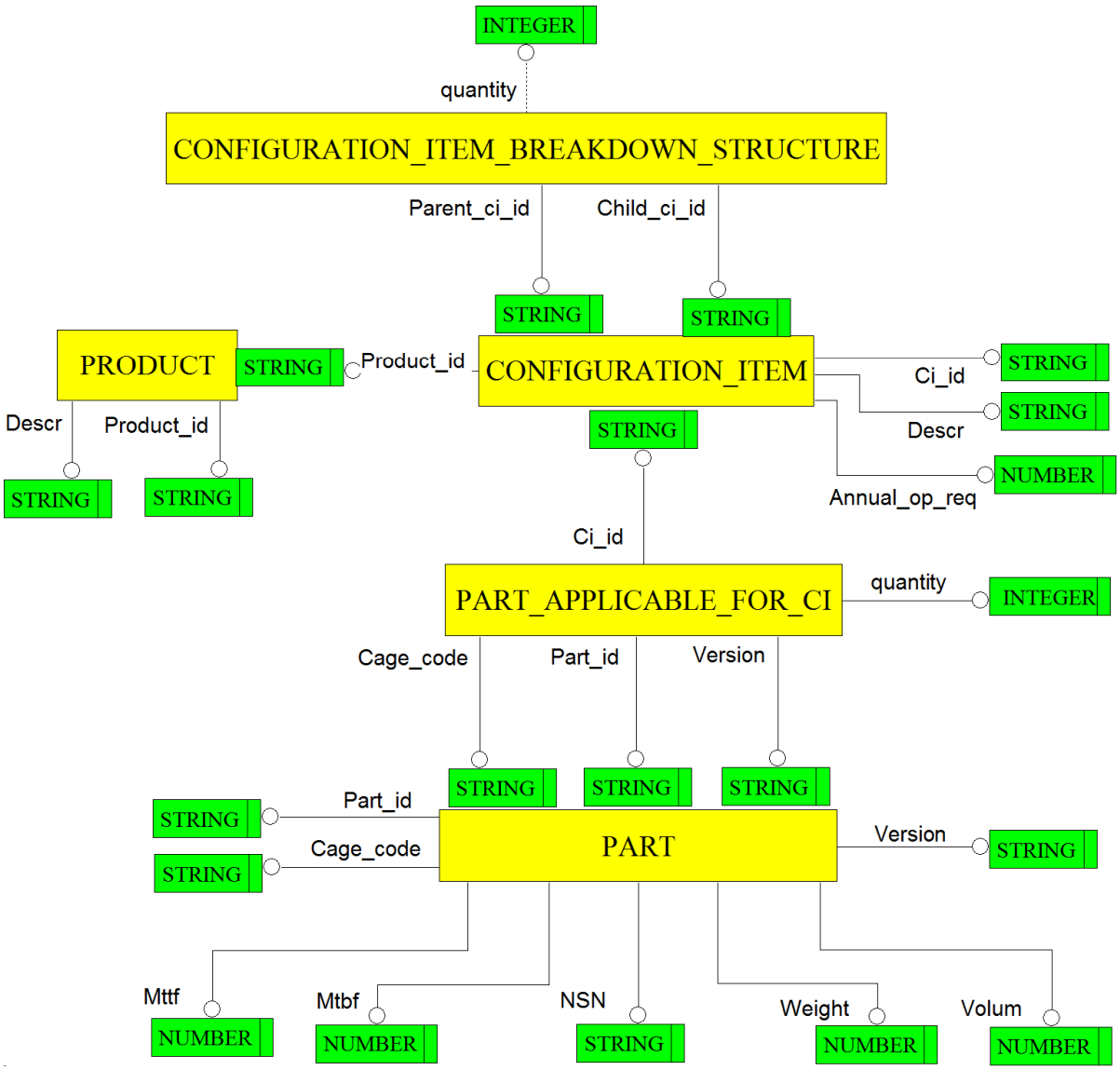



**Figure 24 Relational model equivalent (in EXPRESS-G)**

The translation to the ultimate target DEX is performed in two steps. First a translation from the legacy relational model to the object model, thereafter, a translation from the object model to the final target, the DEX needs to be performed. Each step is achieved in two parts, (a) a mapping table is used to document the required translation, and (b) the creation of the EXPRESS-X code to implement the mapping.

## STEP-1: Relational instances to Object Instances

The first translation is a simple one. Foreign keys are "switched" to object/instance references whereas other attribute values are simply copied.

This is illustrated in the two figures above. They look identical – and semantically they are, but there are significant differences. In this EXPRESS-G notation, one can see the redundancy of the foreign keys in the relational model (Figure 24) when compared with the object model (Figure 23). This is charted in table (STEP-1A: Mapping Table

| PRIMARY KEY | | FOREIGN KEY | | OBJECT REFERENCE | |
|---|---|---|---|---|---|
| ENTITY | ATTRIBUTE | ENTITY | ATTRIBUTE | ENTITY | ATTRIBUTE |
| CONFIGURATION_ITEM | CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | PARENT_CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | PARENT |
| CONFIGURATION_ITEM | CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | CHILD_CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | CHILD |
| PRODUCT | PRODUCT_ID | CONFIGURATION_ITEM | PRODUCT_ID | CONFIGURATION_ITEM | PRODUCT |
| PART | CAGE_CODE<br>PART_ID<br>VERSION | PART_APPLICABLE_FOR_CI | CAGE_CODE<br>PART_ID<br>VERSION | PART_APPLICABLE_FOR_CI | PART |

**Table 7 - Mapping between relational keys and object references**

) below, which shows the general mapping between the relational keys and object references used in the example above.

## STEP-1A: Mapping Table

| PRIMARY KEY | | FOREIGN KEY | | OBJECT REFERENCE | |
|---|---|---|---|---|---|
| ENTITY | ATTRIBUTE | ENTITY | ATTRIBUTE | ENTITY | ATTRIBUTE |
| CONFIGURATION_ITEM | CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | PARENT_CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | PARENT |
| CONFIGURATION_ITEM | CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | CHILD_CI_ID | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE | CHILD |
| PRODUCT | PRODUCT_ID | CONFIGURATION_ITEM | PRODUCT_ID | CONFIGURATION_ITEM | PRODUCT |
| PART | CAGE_CODE<br>PART_ID<br>VERSION | PART_APPLICABLE_FOR_CI | CAGE_CODE<br>PART_ID<br>VERSION | PART_APPLICABLE_FOR_CI | PART |

**Table 7 - Mapping between relational keys and object references**

Having both the source & target models for the legacy data in EXPRESS allows us to use EXPRESS-X to translate (map) instances of one schema to the other.

The listing below provides an example of the EXPRESS-X code required to translate the source PDM_SCHEMA_R to the target PDM_SCHEMA_O representation.

## STEP-1B: Mapping Code

```
SCHEMA_MAP PDM_SCHEMA_R_to_PDM_SCHEMA_O;

GLOBAL
  DECLARE src INSTANCE OF SOURCE_SCHEMA PDM_SCHEMA_R;
  DECLARE tar INSTANCE OF TARGET_SCHEMA PDM_SCHEMA_O;

  product_hash_index : INTEGER;
  ci_item_hash_index : INTEGER;
  part_hash_index    : INTEGER;

END_GLOBAL;

COPY_MODEL
  INCLUDE(PRODUCT,PART);
END_COPY_MODEL;

STATEMENTS;
  LOCAL
    entityId : GENERIC;
  END_LOCAL;

  ON_ERROR_DO
    handleException;
```

```
  END_ON_ERROR_DO;

-----------------------------------------
-- Initialize hash tables
-----------------------------------------
    entityId := xpfGetEntity(XPXSOURCEMODELID, 'CONFIGURATION_ITEM');
    ci_item_hash_index := xpfMakeAttrsHashTable(XPXSOURCEMODELID, entityId,
FALSE, 'CI_ID');

    entityId := xpfGetEntity(XPXSOURCEMODELID, 'PRODUCT');
    product_hash_index := xpfMakeAttrsHashTable(XPXTARGETMODELID, entityId,
FALSE, 'PRODUCT_ID');

    entityId := xpfGetEntity(XPXTARGETMODELID, 'PART');
    part_hash_index := xpfMakeAttrsHashTable(XPXTARGETMODELID, entityId,
FALSE,'CAGE_CODE' ,'PART_ID','VERSION');
END_STATEMENTS;

MAP Map_Ci_Breakdown_strucure FOR
t_ci_item_bs:tar::CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE;
  FROM(s_ci_item_bs:src::CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      s_parent    : src::CONFIGURATION_ITEM;
      s_child     : src::CONFIGURATION_ITEM;

      t_parent    : tar::CONFIGURATION_ITEM;
      t_child     : tar::CONFIGURATION_ITEM;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    t_ci_item_bs.quantity := s_ci_item_bs.quantity;

    s_parent :=
xpfGetAttrsHashedValue(ci_item_hash_index,1,s_ci_item_bs.parent_ci_id);
    s_child  :=
xpfGetAttrsHashedValue(ci_item_hash_index,1,s_ci_item_bs.child_ci_id);

    IF((s_parent :<>: XPXNULLID) AND (s_child :<>: XPXNULLID)) THEN
      t_parent := Map_Ci(s_parent);
      t_child  := Map_Ci(s_child);
      t_ci_item_bs.parent := t_parent;
      t_ci_item_bs.child  := t_child;
    END_IF;
  END_MAP;

MAP Map_Ci FOR t_ci_item:tar::CONFIGURATION_ITEM;
  FROM(s_ci_item:src::CONFIGURATION_ITEM)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      t_product : tar::PRODUCT;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    t_ci_item.ci_id := s_ci_item.ci_id;
    t_ci_item.descr := s_ci_item.descr;
```

```
    t_ci_item.annual_op_req := s_ci_item.annual_op_req;

    t_product :=
xpfGetAttrsHashedValue(product_hash_index,1,s_ci_item.product_id);
    t_ci_item.product := t_product;
  END_MAP;

MAP Map_Part_App_For_Ci FOR t_pafc:tar::PART_APPLICABLE_FOR_CI;
  FROM(s_pafc:src::PART_APPLICABLE_FOR_CI)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      s_citem : src::CONFIGURATION_ITEM;
      t_citem : tar::CONFIGURATION_ITEM;

      t_part  : tar::PART;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    t_pafc.quantity := s_pafc.quantity;

    s_citem := xpfGetAttrsHashedValue(ci_item_hash_index,1,s_pafc.ci_id);
    t_citem := Map_Ci(s_citem);
    t_pafc.conf_item := t_citem;

    t_part :=
xpfGetAttrsHashedValue(part_hash_index,1,s_pafc.cage_code,s_pafc.part_id,s_pafc.
version);
    t_pafc.part := t_part;
  END_MAP;

-------------------------------------------
-- handleException
-------------------------------------------
PROCEDURE handleException;
  IF(NOT(EXISTS(xpxExceptionid.errmessage))) THEN
    xpxStringPrintf(xpxExceptionid.errmessage, 'ERROR: %s \n   error code: %u\n
error in line: %u',
                    xpfGetErrorText(xpxExceptionid.errcode),
xpxExceptionid.errcode, xpxExceptionid.lineNumber);
  END_IF;
  xpxTerminate(xpxExceptionid.errcode, xpxExceptionid.errmessage);
  xpxTerminate(100002, 'Unhandled error');
END_PROCEDURE;

END_SCHEMA_MAP;
```

Once this mapping is specified and compiled, JOTNE's tools allow the process to be automated, creating the first translator. This can then be used to create a population from the legacy source format to one that conforms to typical object modeling principles and removes redundant relational references.

## STEP-2: Business DEX Mapping

Once the legacy data has been translated into the object model (PDM_SCHEMA_O), the EXPRESS instances need to be mapped to the new (PLCS – AP239) representation. This requires a mapping to the business DEX that has been chosen or developed (in accordance with this handbook).

The business DEX for this example implementation has been depicted graphically in Figure 25 below. This provides the basic arrangement of PLCS templates required to represent the source requirements in PLCS and is used as an input to the mapping table. The explicit mapping between the schema of the legacy object model (PDM_SCHEMA_O) and the DEX schema is provided in the tables following (see

## STEP-2A: Mapping Table

The table below is the detailed mapping of the PDM_Schema_O entity attributes to PLCS template instance parameters. The template instance identifiers refer to the Business Dex1 diagram above. The template API is basically a carbon copy of the template definitions as found in DEXlib. However, there is one basic discrepancy. Since the name of the RDL in most cases is the same for the total scope of a translator (or query or any other application), the name is set in global space upfront. The same is true for the name of the organization that issues the identifiers (like part identifiers). This is why parameters with names like xxx_id_ecl_id , xxx_org_id , xxx_org_id_class_name, xxx_org_id_ecl_id are omitted from the template API interface, and hence the mapping. In this way programming against the API becomes more efficient.

Table 8).

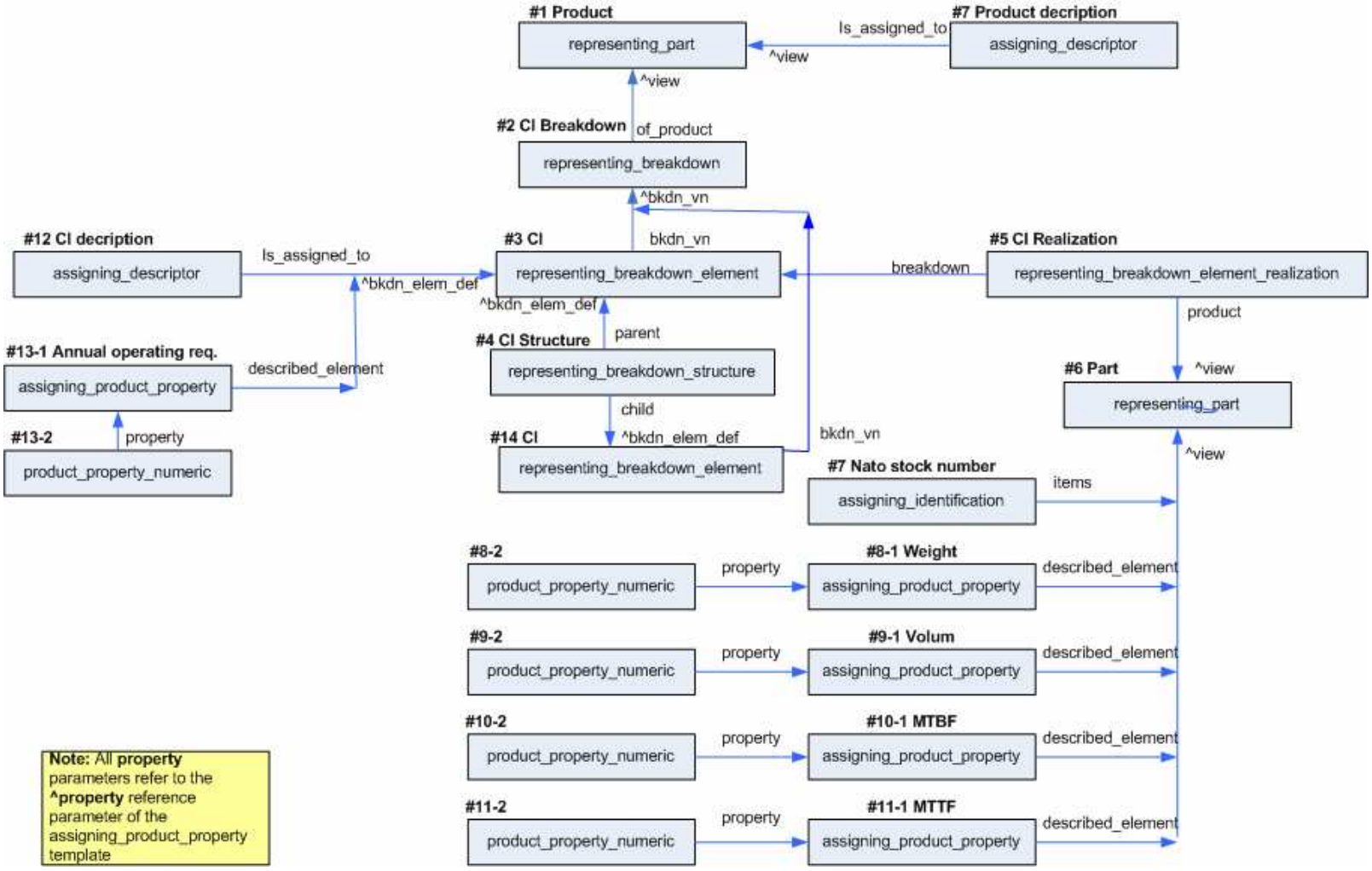



**Figure 25 Business DEX (Derived from DEX1)**

## STEP-2A: Mapping Table

The table below is the detailed mapping of the PDM_Schema_O entity attributes to PLCS template instance parameters. The template instance identifiers refer to the Business Dex1 diagram above. The template API is basically a carbon copy of the template definitions as found in DEXlib. However, there is one basic discrepancy. Since the name of the RDL in most cases is the same for the total scope of a translator (or query or any other application), the name is set in global space upfront. The same is true for the name of the organization that issues the identifiers (like part identifiers). This is why parameters with names like xxx_id_ecl_id , xxx_org_id , xxx_org_id_class_name, xxx_org_id_ecl_id are omitted from the template API interface, and hence the mapping. In this way programming against the API becomes more efficient.

**Table 8 - Mapping Table from the PDM_SCHEMA_O and PLCS TEMPLATES**

| *Templ inst_id* | *Templ type and parameters* | *Cardinality* | *Static binding* | *PDM entity (source)* |
|---|---|---|---|---|
| #1 | representing_part | one-to-one | | PRODUCT |
| | id | | | product_id |
| | id_class_name | | Product_variant_identification_code | |
| | vn_id | | /NULL | |
| | vn_id_class_name | | Version_identification_code | |
| | life_cycle_stage | | Support_stage | |
| | domain | | Product_life_cycle_support | |
| | | | | |
| #7 | assigning_descriptor | | | |
| | descr | | | descr |
| | id_class_name | | Product_description | |
| | items = #1 (^view) | | | |
| #2 | representing_breakdown | one-to-one | | PRODUCT |
| | id | | | product_id |
| | id_class_name | | Breakdown_identification_code | |
| | bkdn_type | | Configuration_item_breakdown | |
| | vn_id | | /NULL | |
| | vn_id_class_name | | Version_identification_code | |
| | of_product = #1 (^view) | | | |
| #3 | representing_breakdown_element | one-to-one | | CONFIGURATION_ITEM |

| | | | | |
|---|---|---|---|---|
| | id | | | ci_id |
| | id_class_name | | Breakdown_element_identification_code | |
| | vn_id | | /NULL | |
| | vn_id_class_name | | Version_identification_code | |
| | life_cycle_stage | | Support_stage | |
| | domain | | Product_life_cycle_support | |
| | bkdn_vn = #2 (^bkdn_vn) | | | |
| | | | | |
| #12 | assigning_descriptor | | | |
| | descr | | | descr |
| | id_class_name | | Breakdown_element_description | |
| | items = #3 (^bkdn_elem_def) | | | |
| | | | | |
| #13-1 | assigning_product_property | | | |
| | property_class_name | | Annual_operating_requirement | |
| | described_element = #3 (^bkdn_elem_def) | | | |
| #13-2 | product_property_numeric | | | |
| | value | | | annual_op_req |
| | unit | | Hours | |
| | si_unit | | TRUE | |
| | context | | Numerical_representation_context | |
| | property = #13-1 (^property) | | | |
| #14 | representing_breakdown_element | one-to-one | | CONFIGURATION_ITEM |
| | The parameter and template instance mappings are similar to those that are specified for #3, and are hence not repeated here. | | | |
| #4 | representing_breakdown_structure | one-to-one | | CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE |
| | rel_type_name | | Breakdown_element_usage | |
| | parent = #3 (^bkdn_elem_def) | | | carent_ci_id |
| | child = #14 (^bkdn_elem_def) | | | child_ci_id |

| | | | | |
|---|---|---|---|---|
| | | | | quantity |
| #6 | representing_part | one-to-one | | PART |
| | id | | | part_id |
| | id_class_name | | Part_identification_code | |
| | vn_id | | | version |
| | vn_id_class_name | | Version_identification_code | |
| | life_cycle_stage | | Support_stage | |
| | domain | | Product_life_cycle_support | |
| | | | | |
| #7 | assigning_identification | | | |
| | id | | | NSN |
| | id_class_name | | Nato_stock_number | |
| | items = #6 (^view) | | | |
| | | | | |
| #8-1 | assigning_product_property | | | |
| | property_class_name | | Weight | |
| | described_element = #6 (^view) | | | |
| #8-2 | product_property_numeric | | | |
| | value | | | weight |
| | unit | | Kilo_gram | |
| | si_unit | | TRUE | |
| | context | | Numerical_representation_context | |
| | property = #8-1 (^property) | | | |
| | | | | |
| #9-1 | assigning_product_property | | | |
| | property_class_name | | Volum | |
| | described_element = #6 (^view) | | | |
| #9-2 | product_property_numeric | | | |
| | value | | | volum |
| | unit | | Cubic_meter | |
| | si_unit | | TRUE | |
| | context | | Numerical_representation_context | |
| | property = #9-1 (^property) | | | |
| | | | | |

| | | | | |
|---|---|---|---|---|
| #10-1 | assigning_product_property | | | |
| | property_class_name | | Mean_time_between_failure | |
| | described_element = #6 (^view) | | | |
| #10-2 | product_property_numeric | | | |
| | value | | | mtbf |
| | unit | | Hours | |
| | si_unit | | TRUE | |
| | context | | Numerical_representation_context | |
| | property = #10-1 (^property) | | | |
| | | | | |
| #11-1 | assigning_product_property | | | |
| | property_class_name | | Mean_time_to_failure | |
| | described_element = #6 (^view) | | | |
| #11-2 | product_property_numeric | | | |
| | value | | | mttf |
| | unit | | Hours | |
| | si_unit | | TRUE | |
| | context | | Numerical_representation_context | |
| | property = #11-1 (^property) | | | |
| #5 | representing_breakdown_element_realization | one-to-one | | PART_APPLICABLE_FOR_CI |
| | breakdown = #3 (^bkdn_elem_def) | | | ci_id |
| | product = #6 (^view) | | | cage_code,part_id,version |
| | | | | |
| #16 | representing_part | one-to-one | | PART |
| | The parameter and template instance mappings are similar to those that are specified for #6, and are hence not repeated here. | | | |

## STEP-2B Mapping Code

The mapping table above provides the context for the next stage of the process – generating the EXPRESS-X mapping. This is based upon two EXPRESS schemas – the schema of the legacy object model (PDM_SCHEMA_O) and the schema of the DEX, which could also be the PLCS schema (since the DEXs are subsets of this).

These become the source and target schemas respectively, and are the first items to be declared in any EXPRESS-X mapping. Having both the source and target models in EXPRESS means that tools that can process EXPRESS-X commands can be automated.

As you will see, the mappings are identifiable in the code by the corresponding # numbers in the mapping table above.

```
SCHEMA_MAP PDM_to_DEX1;

GLOBAL
  DECLARE src  INSTANCE OF SOURCE_SCHEMA PDM_SCHEMA_O;
  DECLARE plcs INSTANCE OF TARGET_SCHEMA
AP239_PRODUCT_LIFE_CYCLE_SUPPORT_ARM_LF_TIS_EXTENSION;

  NULL_ID : STRING := '/NULL';
  ignore  : STRING := '/IGNORE';

--#include "templateAPI/template_global.inn"

--#include "string_list_of_class_names.inn"

  my_default_org_id : STRING;

END_GLOBAL;

STATEMENTS;
  LOCAL
    entityId : GENERIC;
  END_LOCAL;

  ON_ERROR_DO
    handleException;
  END_ON_ERROR_DO;

-------------------------------------------
-- Initialize cache
-------------------------------------------
    init;

    DEFAULT_ORG_ID := 'EPM';
END_STATEMENTS;


(*================================================*)
-- #1 Product and #2 CI Breakdown
(*================================================*)
MAP Map_Breakdown FOR;
  FROM(prod:src::PRODUCT)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      rep_prod      : plcs::REPRESENTING_PART;
```

```
      rep_breakdown : plcs::REPRESENTING_BREAKDOWN;
      ass_descr     : plcs::assigning_descriptor;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

-------------------------------------------
-- #1 Product
-------------------------------------------
    rep_prod := create_representing_part(prod.product_id,
                                         PRODUCT_VARIANT_IDENTIFICATION_CODE,
                                         NULL_ID,
                                         VERSION_IDENTIFICATION_CODE,
                                         PRODUCT_LIFE_CYCLE_SUPPORT,
                                         SUPPORT_STAGE
                                        );

-------------------------------------------
-- #7 Product description
-------------------------------------------
    ass_descr := ASS_DESCR_7(prod.descr,rep_prod.version);

-------------------------------------------
-- #2 CI Breakdown
-------------------------------------------
    rep_breakdown := create_representing_breakdown(prod.product_id,

BREAKDOWN_IDENTIFICATION_CODE,
                                                   DEFAULT_BREAKDOWN_TYPE,
                                                   NULL_ID,
                                                   VERSION_IDENTIFICATION_CODE,
                                                   rep_prod.view
                                                  );
    xpxDefineMapInstances(rep_breakdown);
  END_MAP;
(*== END Map_Breakdown ==============================*)



(*================================================*)
-- #3 CI
(*================================================*)
MAP Map_Ci_Item FOR;
  FROM(ci_item:src::CONFIGURATION_ITEM)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      prod          : src::PRODUCT;

      rep_breakdown : plcs::REPRESENTING_BREAKDOWN;
      rep_break_el  : plcs::REPRESENTING_BREAKDOWN_ELEMENT;
      ass_descr     : plcs::ASSIGNING_DESCRIPTOR;
      ass_prod_prop : plcs::ASSIGNING_PRODUCT_PROPERTY;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    rep_breakdown := Map_Breakdown(ci_item.product);
```

```
-----------------------------------------
-- #3 CI
-----------------------------------------
    rep_break_el := create_representing_breakdown_element(ci_item.ci_id,
                                          BREAKDOWN_ELEMENT_IDENTIFICATION_CODE,
                                          NULL_ID,
                                          VERSION_IDENTIFICATION_CODE,
                                          SUPPORT_STAGE,
                                          PRODUCT_LIFE_CYCLE_SUPPORT,
                                          rep_breakdown.bkdn_vn
                                          );
-----------------------------------------
-- #12 CI Description
-----------------------------------------
    ass_descr := ASS_DESCR_12(ci_item.descr,rep_break_el.bkdn_elem_vn);

-----------------------------------------
-- #13 Annual operating req.
-----------------------------------------
    ass_prod_prop :=
ASS_PROD_PROP_13(rep_break_el.bkdn_elem_def,ci_item.annual_op_req);
    xpxDefineMapInstances(rep_break_el);
  END_MAP;
(*==END Map_Ci_Item ================================*)



(*=================================================*)
-- #4 CI Structure
(*=================================================*)
MAP Map_Ci_Breakdown_strucure FOR;
  FROM(ci_item_bs:src::CONFIGURATION_ITEM_BREAKDOWN_STRUCTURE)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      s_parent    : src::CONFIGURATION_ITEM;
      s_child     : src::CONFIGURATION_ITEM;

      t_parent    : plcs::REPRESENTING_BREAKDOWN_ELEMENT;
      t_child     : plcs::REPRESENTING_BREAKDOWN_ELEMENT;
      bkdn_struct : plcs::REPRESENTING_BREAKDOWN_STRUCTURE;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    t_parent := Map_Ci_Item(ci_item_bs.parent);
    t_child  := Map_Ci_Item(ci_item_bs.child);

-----------------------------------------
-- #4 CI Structure
-----------------------------------------
    bkdn_struct :=
create_representing_breakdown_structure(DEFAULT_BREAKDOWN_TYPE,t_parent.bkdn_ele
m_def,t_child.bkdn_elem_def);

  END_MAP;
(*== END Map_Ci_Breakdown_strucure ==================*)



(*=================================================*)
```

```
-- #6 Part
(*================================================*)
MAP Map_Part FOR;
  FROM(mypart:src::PART)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      rep_part      : plcs::REPRESENTING_PART;
      ass_id        : plcs::ASSIGNING_IDENTIFICATION;
      ass_prod_prop : plcs::ASSIGNING_PRODUCT_PROPERTY;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    my_default_org_id := DEFAULT_ORG_ID;
    DEFAULT_ORG_ID    := mypart.cage_code;

-----------------------------------------
-- #6 Part
-----------------------------------------
    rep_part := create_representing_part(mypart.part_id,
                                         PART_IDENTIFICATION_CODE,
                                         mypart.version,
                                         VERSION_IDENTIFICATION_CODE,
                                         PRODUCT_LIFE_CYCLE_SUPPORT,
                                         SUPPORT_STAGE
                                        );

    DEFAULT_ORG_ID := my_default_org_id;

-----------------------------------------
-- #7 Nato stock number
-----------------------------------------
    ass_id := ASS_ID_7(mypart.NSN,mypart.cage_code,rep_part.view);

-----------------------------------------
-- #8 Weight
-----------------------------------------
    ass_prod_prop := ASS_PROD_PROP_8 (rep_part.view,mypart.weight);

-----------------------------------------
-- #9 Volum
-----------------------------------------
    ass_prod_prop := ASS_PROD_PROP_9 (rep_part.view,mypart.volum);

-----------------------------------------
-- #10 MTBF
-----------------------------------------
    ass_prod_prop := ASS_PROD_PROP_10(rep_part.view,mypart.mtbf);

-----------------------------------------
-- #11 MTTF
-----------------------------------------
    ass_prod_prop := ASS_PROD_PROP_11(rep_part.view,mypart.mttf);

    xpxDefineMapInstances(rep_part);
  END_MAP;
(*==END Map_Part ====================================*)

(*================================================*)
-- #5 CI Realization
```

```
(*================================================*)
MAP Map_Bel_Realization FOR;
  FROM(papp:src::PART_APPLICABLE_FOR_CI)
  WHEN(TRUE);
  BEGIN_MAP
    LOCAL
      s_part   : src::PART;
      s_bel    : src::CONFIGURATION_ITEM;

      rep_part : plcs::REPRESENTING_PART;
      rep_bel  : plcs::REPRESENTING_BREAKDOWN_ELEMENT;
      rep_ber  : plcs::REPRESENTING_BREAKDOWN_ELEMENT_REALIZATION;
    END_LOCAL;

    ON_ERROR_DO
      handleException;
    END_ON_ERROR_DO;

    rep_part := Map_Part(papp.part);
    rep_bel  := Map_Ci_Item(papp.conf_item);

------------------------------------------
-- #5 CI Realization
------------------------------------------
    rep_ber :=
create_representing_breakdown_element_realization(rep_bel.bkdn_elem_def,rep_part
.view,BREAKDOWN_ELEMENT_REALIZATION);
  END_MAP;
(*== END Map_Bel_Realization ========================*)


------------------------------------------
-- handleException
------------------------------------------
PROCEDURE handleException;
  IF(NOT(EXISTS(xpxExceptionid.errmessage))) THEN
    xpxStringPrintf(xpxExceptionid.errmessage, 'ERROR: %s \n   error code: %u\n
error in line: %u',
                    xpfGetErrorText(xpxExceptionid.errcode),
xpxExceptionid.errcode, xpxExceptionid.lineNumber);
  END_IF;
  xpxTerminate(xpxExceptionid.errcode, xpxExceptionid.errmessage);
  xpxTerminate(100002, 'Unhandled error');
END_PROCEDURE;


END_SCHEMA_MAP;
```

The following section is an alternate .NET coded example implementation of the mapping from PDM (`PDM_SCHEMA_0`) to business DEX1 (via the `AP239_PRODUCT_LIFE_CYCLE_SUPPORT_ARM_ LF_TIS_EXTENSION`)using the PLCS services described earlier (and in [15]).

This example should result in a PLCS population identical to the one produced by the EXPRESS-X mapping example above[1].

[1] At present this is not exactly the case. In the .net web service example, text properties are used instead of numerical properties.

```
//Program.cs



using System;

using System.Collections.Generic;

using System.Text;

using System.Xml;



namespace PDM_to_DEX1_client

{

    class Program

    {

        static void Main(string[] args)

        {

            Access.EDMAccessControlService access = new
Access.EDMAccessControlService();

            // access.Url = ... - it can be changed

            const String user = "superuser";

            const String password = "db";

            String sessionId = access.login(user, "", password);



            List<pdm.Product> products = new List<pdm.Product>();

            List<pdm.Configuration_Item> ci_items = new
List<pdm.Configuration_Item>();

            List<pdm.Configuration_Item_Breakdown_Structure> ci_breakdown_items =
new List<pdm.Configuration_Item_Breakdown_Structure>();

            List<pdm.Part> parts = new List<pdm.Part>();

            List<pdm.Part_applicable_for_CI> parts_for_ci = new
List<pdm.Part_applicable_for_CI>();
```

```
            // all aggregates will be populated below

            ReadXMLData("pdm_r.xml", ref products, ref ci_items, ref
ci_breakdown_items, ref parts, ref parts_for_ci);



            plcs.EDMplcsServices_ContentManagementService service = new
plcs.EDMplcsServices_ContentManagementService();

            // service.Url = ... - it can be changed



            List<plcs.srvBreakdownElement> all_breakdown_elments = new
List<plcs.srvBreakdownElement>();



            for (int i = 0; i < products.Count; i++)

            {

                plcs.srvProduct product = new plcs.srvProduct();

                plcs.srvAttribute prod_id_attr = new plcs.srvAttribute();

                prod_id_attr.attrVal = products[i].product_id;

                prod_id_attr.uriName = "urn:rdl:epm-
std:Product_variant_identification_code";

                prod_id_attr.state = "CREATE";

                product.productId = prod_id_attr;



                plcs.srvAttribute prod_descr_attr = new plcs.srvAttribute();

                prod_descr_attr.attrVal = products[i].descr;

                prod_descr_attr.uriName = "urn:rdl:epm-std:Product_description";

                product.descriptions = new plcs.srvAttribute[] { prod_descr_attr };



                // create product

                product = service.storeProduct(sessionId, product);
```

```
                plcs.srvBreakdown brkdown = new plcs.srvBreakdown();

                brkdown.breakdownFor = product;



                plcs.srvAttribute brkdown_id_attr = new plcs.srvAttribute();

                brkdown.bkId = brkdown_id_attr;

                brkdown_id_attr.attrVal = products[i].product_id;

                brkdown_id_attr.uriName =
"urn:plcs:rdl:std:Breakdown_identification_code";



                plcs.srvClass brkdown_class_attr = new plcs.srvClass();

                brkdown_class_attr.uriName = "urn:plcs:rdl:std:Breakdown";

                brkdown_class_attr.state = "CREATE";

                brkdown.bkType = brkdown_class_attr;



                // create breakdown

                brkdown = service.storeBreakdown(sessionId, brkdown);



                List<plcs.srvBreakdownElement> elments_list = new
List<plcs.srvBreakdownElement>();

                for (int j = 0; j < ci_items.Count; j++)

                {

                    if ( !ci_items[j].product_id.Equals(product.productId.attrVal)
)

                        continue;

                    plcs.srvBreakdownElement brkdwn_elment = new
plcs.srvBreakdownElement();



                    // set breakdown identifiers

                    plcs.srvAttribute element_id = new plcs.srvAttribute();
```

```
                    element_id.attrVal = ci_items[j].ci_id;

                    element_id.uriName =
"urn:plcs:rdl:std:Breakdown_element_identification_code";

                    element_id.state = "CREATE";

                    brkdwn_elment.elementId = element_id;



                    // set breakdown descriptions

                    plcs.srvAttribute element_description = new
plcs.srvAttribute();

                    element_description.attrVal = ci_items[j].desc;

                    element_description.uriName =
"urn:plcs:rdl:std:Descriptor_assignment";

                    element_description.state = "CREATE";

                    brkdwn_elment.descriptions = new plcs.srvAttribute[] {
element_description };



                    // set breakdown properties

                    plcs.srvAttribute element_prop = new plcs.srvAttribute();

                    element_prop.attrVal = ci_items[j].annual_op_req.ToString();

                    element_prop.uriName =
"urn:epm:rdl:std:Annual_operating_requirement";

                    element_prop.state = "CREATE";

                    brkdwn_elment.properties = new plcs.srvAttribute[] {
element_prop };

                    elments_list.Add(brkdwn_elment);

                }

                // create breakdown elements

                plcs.srvBreakdownElement[] elments =
service.storeBreakdownElements(sessionId, brkdown, elments_list.ToArray());

                all_breakdown_elments.AddRange(elments);
```

```
                List<plcs.srvBreakdownStructure> structures = new
List<plcs.srvBreakdownStructure>();

                for (int j = 0; j < ci_breakdown_items.Count; j++)

                {

                    plcs.srvBreakdownStructure brkdown_structure = new
plcs.srvBreakdownStructure();

                    List<plcs.srvBreakdownStructure> children_list = new
List<plcs.srvBreakdownStructure>();

                    for (int k = 0; k < elments.Length; k++)

                    {

                        String id = elments[k].elementId.attrVal;

                        if ( id.Equals( ci_breakdown_items[j].parent_ci_id ) )

                        {

                            brkdown_structure.element = elments[k];

                        }

                        else if ( id.Equals( ci_breakdown_items[j].child_ci_id ) )

                        {

                            plcs.srvBreakdownStructure child = new
plcs.srvBreakdownStructure();

                            child.element = elments[k];

                            child.parent = brkdown_structure;

                            children_list.Add(child);

                        }

                    }

                    brkdown_structure.children = children_list.ToArray();

                    structures.Add(brkdown_structure);

                }

                // create breakdown structure

                // breakdown structure tree will appear in the database then
```

```
                plcs.srvBreakdownStructure[] stored_structures =
service.storeBreakdownStructure(sessionId, brkdown, structures.ToArray());

            }

            for (int i = 0; i < parts.Count; i++)

            {

                plcs.srvProduct product = new plcs.srvProduct();

                plcs.srvAttribute prod_id_attr = new plcs.srvAttribute();

                prod_id_attr.attrVal = parts[i].cage_code + "|" + parts[i].part_id;

                prod_id_attr.uriName = "urn:rdl:epm-
std:Product_variant_identification_code";

                prod_id_attr.state = "CREATE";

                product.productId = prod_id_attr;



                prod_id_attr = new plcs.srvAttribute();

                prod_id_attr.attrVal = parts[i].cage_code + "|" + parts[i].nsn;

                prod_id_attr.uriName = "urn:rdl:epm-
std:Product_variant_identification_code";

                prod_id_attr.state = "CREATE";

                product.identifiers = new plcs.srvAttribute[] { prod_id_attr };



                // There is no possibility to assign following properties to
Product using PLCS services

                {

                    plcs.srvAttribute prop_weight = new plcs.srvAttribute();

                    prop_weight.attrVal = parts[i].weight;

                    prop_weight.uriName = "urn:epm:rdl:std:Weight";

                    prop_weight.state = "CREATE";



                    plcs.srvAttribute prop_volum = new plcs.srvAttribute();
```

```
                    prop_volum.attrVal = parts[i].volum;

                    prop_volum.uriName = "urn:epm:rdl:std:Volum";

                    prop_volum.state = "CREATE";



                    plcs.srvAttribute prop_mtbf = new plcs.srvAttribute();

                    prop_mtbf.attrVal = parts[i].mtbf;

                    prop_mtbf.uriName =
"urn:epm:rdl:std:Mean_time_between_failure";

                    prop_mtbf.state = "CREATE";



                    plcs.srvAttribute prop_mttf = new plcs.srvAttribute();

                    prop_mttf.attrVal = parts[i].mttf;

                    prop_mttf.uriName = "urn:epm:rdl:std:Mean_time_to_failure";

                    prop_mttf.state = "CREATE";

                }

                product = service.storeProduct(sessionId, product);

            }



            plcs.srvProduct[] stored_parts = service.getProduct(sessionId);

            for (int i = 0; i < parts_for_ci.Count; i++)

            {

                plcs.srvBreakdownElement element = null;

                for (int j = 0; j < all_breakdown_elments.Count; j++)

                {

                    String id = all_breakdown_elments[j].elementId.attrVal;

                    if (id.Equals(parts_for_ci[i].ci_id))

                    {
```

```
                        element = all_breakdown_elments[j];

                        break;

                    }

                }

                plcs.srvProduct part = null;

                for (int j = 0; j < stored_parts.Length; j++)

                {

                    String id = stored_parts[j].productId.attrVal;

                    if (id.Equals(parts_for_ci[i].part_id)) {

                        part = stored_parts[j];

                        break;

                    }

                }

                service.addProductToBreakdownElement(sessionId, element, part);

            }

        }

        static void ReadXMLData(

            String xml_file_name,

            ref List<pdm.Product> products,

            ref List<pdm.Configuration_Item> ci_items,

            ref List<pdm.Configuration_Item_Breakdown_Structure>
ci_breakdown_items,

            ref List<pdm.Part> parts,

            ref List<pdm.Part_applicable_for_CI> parts_for_ci)

        {

            XmlDocument xDoc = new XmlDocument();

            xDoc.Load(xml_file_name);
```

```
            //get products
            XmlNodeList product_list = xDoc.GetElementsByTagName("Product", "*");
            for (int i = 0; i < product_list.Count; i++)
            {
                pdm.Product product = new pdm.Product();
                XmlNodeList children = product_list[i].ChildNodes;
                for (int j = 0; j < children.Count; j++)
                {
                    switch (children[j].LocalName)
                    {
                        case "Product_id":
                            product.product_id = children[j].InnerText;
                            break;
                        case "Descr":
                            product.descr = children[j].InnerText;
                            break;
                    }
                }
                products.Add(product);
            }


            //get configuration items
            XmlNodeList ci_list = xDoc.GetElementsByTagName("Configuration_item",
"*");
            for (int i = 0; i < ci_list.Count; i++)
            {
                pdm.Configuration_Item ci_item = new pdm.Configuration_Item();
```

```
                XmlNodeList children = ci_list[i].ChildNodes;

                for (int j = 0; j <children.Count; j++)

                {

                    switch (children[j].LocalName)

                    {

                        case "Ci_id":

                            ci_item.ci_id = children[j].InnerText;

                            break;

                        case "Product_id":

                            ci_item.product_id = children[j].InnerText;

                            break;

                        case "Descr":

                            ci_item.desc = children[j].InnerText;

                            break;

                        case "Annual_op_req":

                            ci_item.annual_op_req = children[j].InnerText;

                            break;

                    }

                }

                ci_items.Add(ci_item);

            }



            //get configuration item breakdown structures

            XmlNodeList ci_breakdown_list =
xDoc.GetElementsByTagName("Configuration_item_breakdown_structure", "*");

            for (int i = 0; i < ci_breakdown_list.Count; i++)

            {
```

```
                pdm.Configuration_Item_Breakdown_Structure ci_breakdown_item = new
pdm.Configuration_Item_Breakdown_Structure();

                XmlNodeList children = ci_breakdown_list[i].ChildNodes;

                for (int j = 0; j < children.Count; j++)

                {

                    switch (children[j].LocalName)

                    {

                        case "Parent_ci_id":

                            ci_breakdown_item.parent_ci_id = children[j].InnerText;

                            break;

                        case "Child_ci_id":

                            ci_breakdown_item.child_ci_id = children[j].InnerText;

                            break;

                        case "quantity":

                            ci_breakdown_item.quantity = children[j].InnerText;

                            break;

                    }

                }

                ci_breakdown_items.Add(ci_breakdown_item);

            }



            //get parts

            XmlNodeList part_list = xDoc.GetElementsByTagName("Part", "*");

            for (int i = 0; i < part_list.Count; i++)

            {

                pdm.Part part_item = new pdm.Part();

                XmlNodeList children = part_list[i].ChildNodes;
```

```
                for (int j = 0; j < children.Count; j++)
                {
                    switch (children[j].LocalName)
                    {
                        case "Part_id":
                            part_item.part_id = children[j].InnerText;
                            break;
                        case "Version":
                            part_item.version = children[j].InnerText;
                            break;
                        case "Cage_code":
                            part_item.cage_code = children[j].InnerText;
                            break;
                        case "NSN":
                            part_item.nsn = children[j].InnerText;
                            break;
                        case "Weight":
                            part_item.weight = children[j].InnerText;
                            break;
                        case "Volum":
                            part_item.volum = children[j].InnerText;
                            break;
                        case "Mtbf":
                            part_item.mtbf = children[j].InnerText;
                            break;
                        case "Mttf":
                            part_item.mttf = children[j].InnerText;
```

```
                            break;

                    }

                }

                parts.Add(part_item);

            }



            //get part applicable for ci

            XmlNodeList part_for_ci_list =
xDoc.GetElementsByTagName("Part_applicable_for_ci", "*");

            for (int i = 0; i < part_for_ci_list.Count; i++)

            {

                pdm.Part_applicable_for_CI item = new pdm.Part_applicable_for_CI();

                XmlNodeList children = part_for_ci_list[i].ChildNodes;

                for (int j = 0; j < children.Count; j++)

                {

                    switch (children[j].LocalName)

                    {

                        case "Ci_id":

                            item.ci_id = children[j].InnerText;

                            break;

                        case "Cage_code":

                            item.cage_code = children[j].InnerText;

                            break;

                        case "Part_id":

                            item.part_id = children[j].InnerText;

                            break;

                        case "Version":
```

```
                            item.version = children[j].InnerText;

                            break;

                        }

                    }

                    parts_for_ci.Add(item);

                }

            }

        }
}
```

# 11 Appendix C – PLCS in a Service Oriented Architecture

The figure below, Figure 26, depicts the arrangement of a PLCS repository and adaptors within a Service Oriented Architecture (SOA) that provides web services to other applications or clients. This sections uses the implementation by Jotne as an example.

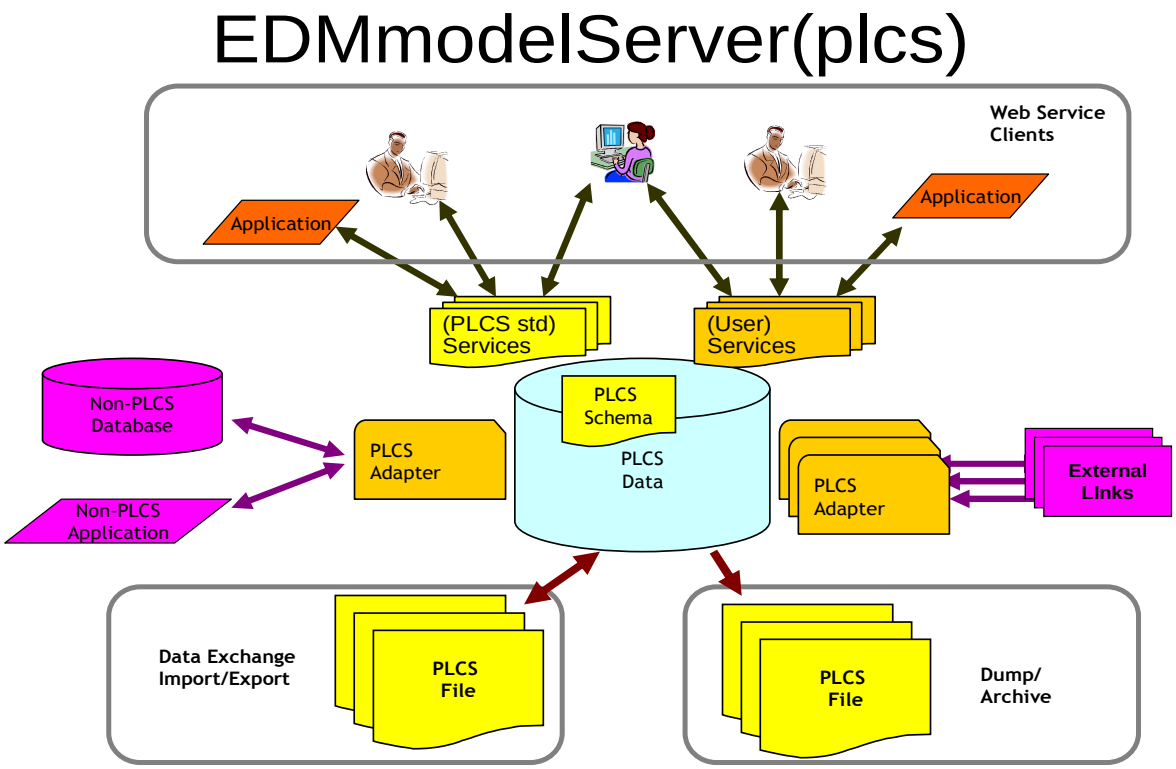



**Figure 26 Overview of EDMmodelServer™(plcs)**

The *EDMmodelServer*™(plcs) offers both server and – optionally - client functionality. This is supported by a database management system (DBMS) and includes the service-oriented architecture (SOA) by offering web-services that are easily generated from database queries. It provides a set of PLCS services that are based upon the *EDMtemplateAPI*™ and the underlying web services. It also has the usual (Java, C, C++, and EXPRESS-X) access to the PLCS repository and other repositories housed in the database. These APIs facilitate the interfacing of specific domain applications — PDM, ILS, etc., via PLCS DEX adaptors, from import of PLCS DEX data exported by PLCS capable systems or from those applications that can communicate via the PLCS services.

These services ensure that the applications connected to such a repository are replaceable without loss of information (both as it is created, and then into the future) as it is used throughout the supported product's life. Information owners are thus better able to change or update their applications and still maintain control of the information stored in the repository.

The *EDMmodelServer*™(plcs) supports functions such as common check-in/check-out, validation, business and engineering rule checking, etc. The benefits of using *EDMmodelServer*™(plcs) and PLCS include:

- Maintaining product information quality and integrity even in highly-distributed and heterogeneous environments
- Merging capability helps to obtain information from different sources into a common PLCS repository
- Keeping product information synchronized throughout the product's lifecycle with baseline capability
- Retaining enough relevant information to serve products for long-term archiving
- Reducing the cost of developing and maintaining interfaces across the supply network
- Enabling customers, partners, and suppliers to work together while using the different development applications that each has chosen for the individual business
- Establishing a common terminology used throughout the product lifecycle.

# 12 Appendix D – PLCS Tools (EDM™)

Below is a list of products from Jotne EPM Technology for development and implementation of PLCS solutions.

| **Task or function** | **Product** |
|---|---|
| Systems analysis | EDMmodelMigrator™ |
| Data modelling and specification | EDMvisualExpress™, EDMmodelMigrator™ |
| Translator development | EDMdeveloper™ (incl PLCS Template API) |
| Data quality, verification and validation | EDMmodelChecker™ |
| Data sharing, interoperability | EDMmodelServer(plcs)™ |
| Reference data and terminology | EDMreferenceDataSystems™ |
| Service oriented architecture and web services | EDMserver™, EDMmodelServer(plcs)™ |

EXPRESS Data Manager

http://www.epmtech.jotne.com/

The appendix below provides a framework within which these tools might be used and the associated contexts (taken from the handbook processes above) that may help the reader identify when such tools might be applicable.

# 13 Appendix E – PLCS Data Exchange Framework

| Activity | Data (What) | Function (How) | People (Who) | Time (When) | Motivation (Why) |
|---|---|---|---|---|---|
| **1. Define Business Process** | Business process modelling (processes, connections, systems) | IDEF0, BPMN | Stakeholder, Domain expert, Business consultant | Analysis phase | To define the scope and boundaries of the exchange and define why the exchange is taking place from a business perspective.. |
| **2. Identify Data Exchange Requirements** | Model domain concepts and their relations that are requried for the exchange. | EDMvisualExpress or UML tool | Domain expert, Business consultant | Analysis phase | To define the data exchange requirements |
| **3. Describe IT Systems** | Model IT systems involved in the exchange. | | Software engineer | Analysis phase | To describe the components and modules of IT system, to use for deciding how to implement data-exchange. |
| **4. Define Data Flow** | Describe where and when data should be exchanged from an IT system perspective | Data flow diagrams, sequence diagrams | Software engineer | Analysis phase | To define the high level requirements for the behavior of a data-exchange mechanism |
| **5. Specify Communication Interfaces** | Describe interfaces and data formats | | Software engineer | Specification phase | Specify how to get data in or out of the IT systems |
| **6. Specify Business DEX(s)** | External class library, PLCS DEX, mapping between domain model and templates | EDMmodelMigrator | Business consultant | Specification phase | |
| **6.1. Select a DEX** | | EDMmodelMigrator | Business consultant | Specification phase | |
| **6.2. Create concept library** | Domain concepts added to reference data library. | EDMmodelMigrator | Domain expert, Business consultant | Specification phase | To classify domain concepts using an existing reference data library (PLCS OWL, ISO 12006, ISO 15926) |
| **6.3. Define business DEX specialization** | | EDMmodelMigrator | Domain expert, Business consultant | Specification phase | To specialize (or qualify) a PLCS DEX. This will constrain the population to the requirements specified in the common reference model.. |
| **7. Specify Adapters** | | | Domain expert, Business consultant | Specification phase | |
| **7.1. Mapping specification** | The mapping relation between the source model and business DEX. Mappings are done on the template variants. | EDMmodelMigrator | Source system vendor | Specification phase | To document how the source model maps to the business DEX. |
| **8. Specify Data Exchange** | | | Software engineer | Specification phase | |
| **9. Implement Adapters** | | EDMmodelServer for PLCS | Software engineer | Implementation phase | |
| **10. Implement Data Exchange Architecture** | | EDMmodelServer for PLCS | Software engineer | Implementation phase | |
| **11. Conformance testing and validation** | Reports created as a result of running conformance testing on import and export. | EDMmodelServer for PLCS | Software engineer | Testing phase | To test the quality of the implementation. |
| **11.1. Verify Adapters** | | | Software engineer | Testing phase | |
| **11.2. Verify Data Exchange** | | | Software engineer | Testing phase | |
| **11.3. Verify Complete Business** | | | Domain expert, Business consultant | Testing phase | |

# 14 Appendix F – Reference Documents

[1] DEXlib
http://www.plcs-resources.org/plcs/dexlib.

[2] Official OASIS PLCS presentation
http://oasis-open.org/committees/document.php?document_id=29411&wg_abbrev=plcs

[3] PLCS technical description
http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_intro.htm

[4] AIA Position Paper: Engineering Data Interoperability- A Standards Based Approach
http://www.aia-aerospace.org/pdf/wp_engineering-data-interoperability.pdf

[5] AIA EDIG Guidebook - Aerospace Industry Guideline for Implementing Interoperability Standards for Engineering Data.

[6] ISO 10303 - Industrial automation systems and integration - Product data representation and exchange. AP239 Product life cycle support (A.k.a. PLCS)

[7] ISO/IEC 2382-1:1993, Information technology - Vocabulary - Part 1: Fundamental terms. 3rd ed.

[8] ISO 1087-1:2000, Terminology work - Vocabulary - Part 1: Theory and application

[9] ISO 8000 - Information quality

[10] ISO 15926 - Industrial automation systems and integration - Integration of life-cycle data for process plants including oil and gas production facilities

[11] Technical description of PLCS Reference Data
http://www.plcs-resources.org/plcs/dexlib/help/dex/techdes_refdata.htm

[12] IEEE Std 610.12-1990 - IEEE Standard Glossary of Software Engineering Terminology

[13] EDMrds User Guide – see the EDMReferenceDataManager™ Help Manual.

[14] http://dev.ifd-library.org/index.php/Ifd:IFD_API

[15] EDMplcsServices Interface Documentation v1.1829