A Concept for a Generic Connector as Middleware Between Data Sources and the Engineering Network Framework

Volume 4

Author:

Dipl.-Wirtsch.-Ing. Alexander Keßler

Prof., Prof. e.h. (mult), Dr.-Ing. Martin Eigner
Kaiserslautern | 12 12th, 2011
Impressum

Herausgeber: Prof. Dr.-Ing. Martin Eigner
Redaktion: Prof. Dr.-Ing. Martin Eigner
Anschrift: Lehrstuhl für Virtuelle Produktentwicklung (VPE)
Technische Universität Kaiserslautern
Postfach 3049
67653 Kaiserslautern

Gebäude: 44
Raum: 314.2
Gottlieb-Daimler-Straße
67663 Kaiserslautern

Internet: http://vpe.mv.uni-kl.de
E-mail: vpeinfo@mv.uni-kl.de
Telefon: + 49 631 - 205 3871
Fax: + 49 631 - 205 3872
ISSN: 2190-2135
A Concept for a Generic Connector as Middleware Between Data Sources and the Engineering Network Framework

Prof. Dr.-Ing. M. Eigner
Dipl.-Wirtsch.-Ing. A. Keßler
The University of Kaiserslautern, Germany

November 02, 2011

Abstract

In order to remain successful and competitive on the market, today’s companies and their IT systems must be capable of managing the product and process relevant information of their products from the early stages of the product development right through to recycling. They have to be able to provide this information at the right time at the right place for the right user. Nowadays there exists mainly multidisciplinary product information that is spread across various IT systems which have historically grown apart and have to be federated and integrated for a successful development. To facilitate this, the Engineering Network Concept[1] has been developed. In this paper, the extension of the Engineering Network (EN) Concept by a concept for a generic connector that allows building data source specific connectors with little effort will be described.

1 Introduction

As part of the development of the Engineering Network Framework we had to cope with different data sources. As the EN aims to integrate and federate data from different data sources, the framework needs some data source dependent connectors. In the beginning, an own connector for every data source that should be accessible has been developed. That was a very elaborative task and did cost a lot of time. The same functionalities had to be rewritten over and over again. Some parts of the connector code could be copied, but most parts had to be customized so widely, that it was easier to completely rewrite it, than to copy and change it.

Since this makes no sense in the long run, a search for commonalities in the data sources was started, with the aim to develop a common code base for all data source-specific connectors. However, the structure of the data sources is very different from each other and a common code base can not be derived in
a trivial way. Therefore, a thorough investigation with the aim of developing a concept for a generic connector was initiated.

First, in Chapter 2 an overview of the systems and data sources in the field of Product Data Management (PDM) and Product Lifecycle Management (PLM) is given. Then, in Chapter 3 the filing types and formats used by these systems are examined and classified. Following the structures in these formats are examined and a generic structure is deduced from them. The results of this investigation are shown in Chapter 4. Finally, a summary is given in Chapter 5.

2 Data Sources

As the Engineering Network is a framework for federating and integrating multidisciplinary product information and processes, it is focused on data sources that are used in the field of Product Data Management (PDM) and Product Lifecycle Management (PLM). Figure 1 depicts typical IT-System topics in the field of PDM and PLM.

![Figure 1: Example of IT-Systems in a PLM solution](image)

In the market that underlies this PLM working area there are many different vendors for PDM Software. Since it was not possible to investigate all vendors and their software solutions, the decision was to take a closer look at the five major vendors. For the investigation we assumed that the technology that is used by the major vendors to store their data and their interfaces to access it is the industry standard and that the smaller players in the market have similar solutions. Of course there might be other solutions, but because of the limited resources for that study we can not investigate them all. The five largest vendors...
which were identified are:

**SAP:** With the PLM module of the SAP Business Suite. The SAP system can use different relational databases such as SAP’s own MaxDB, DB2, Oracle, MS SQL Server, Sybase ASE and further. As additional access options, SAP offers connectors for different platforms (.NET, Java, etc.).

**Siemens:** With Siemens PLM Teamcenter. MS SQL Server is used as database technology. Since 2009 DB2 is also possible. Teamcenter provides an open Service Oriented Architecture (SOA) interface.

**Dassault Systèmes:** With ENOVIA MatrixOne. The system provides an open SOA interface and as popular databases DB2, Oracle, and since 2009, the MS SQL server come to use.

**Oracle:** With Agile PLM. As database Oracle’s own database is used. With the Oracle Transparent Gateway for MS SQL Server the access to MS SQL Server is possible.

**PTC:** With Windchill. As databases Oracle or MS SQL Server are used. As an additional interface a SOA access is available via the Windchill Info Engine.

All PLM market leaders use databases as backend and partially provide SOA interfaces with which it is possible to access their systems. For the transmission of information from the service to the service users of a SOA, there exist standard internet protocols such as IIOP, DCOM, DCE, SNA, CORBA, or HTTP to SAP RFC. The protocols assure the complete and error-free transmission of any message. However, these protocols give no information about the content of the message and its structure. Therefore, the actual message in a SOA is once again wrapped in a web service protocol. Common protocols are REST, JavaScript Object Notation, Advanced Message Queuing Protocol and the XML-based protocol pair SOAP and WSDL.

The SOA approach enables open access to the systems, but has the disadvantage that there are differences among providers, in the protocols they use and in the message structure and style.

Furthermore, the solutions of the market leaders are often not usable for very small businesses, because the price is too high and the setup and administration is disproportional to the benefits. So, the only possibility for small businesses is to use their own solutions, small solutions from other vendors or standard office applications to store and manage their product-, customer-, accounting- and other corporate data.

Looking at standard office applications, product, customer and accounting data can be stored in spreadsheet programs like Microsoft Excel and Open Office Calc or in small databases like Microsoft Access. These storage facilities have in common with the big solutions that they are also relational and table-based. Another possible storage solution is the usage of self-developed solutions, but it is not possible to not make a statement about these, because they are
too customer-specific. However, small solutions from other vendors often offer an export option for the data in a table-based standard office format or in a low-level exchange format such as CSV (Comma Separated Values). CSV files consist of a row with header information and n lines with data. The header information is separated by a delimiter, such as comma, semicolon, etc.

3 Filing Types and Formats

After analyzing which systems or programs are used to store PDM and PLM information, now the different kinds of filing types and formats are classified. Therefore, we differentiate between a dynamic and a static filing.

3.1 Dynamic Filing

The dynamic filing is used within a program if the information is kept in the memory of the computer system to process it. Here, nowadays the approach of object orientation (OO) prevails. In doing so, information and methods are bundled together into one object. This offers the advantage of a clean structure and increases the maintainability and extensibility. The disadvantage of this filing type is its fleetingness, as the information is lost when turning off the computer system. Thus, the dynamic storage is suitable for the processing, using and displaying of information, but not for long-term storage.

3.2 Static Filing

Nowadays for long-term static filing relational or object-oriented databases or structured storage file formats such as XML, CSV, etc. are available. Relational databases are, as the studies in Chapter 2 show, widespread, standardized and have a single, common query language. Object-oriented databases have been in development for many years, but could not prevail in an industrial scale yet. Through years of development the relational databases have reached a stable, reliable and performant state. Unlike relational databases, the structured storage file formats are not suited for storing large amounts of information, as high-performance searching and efficient saving is not given. The strengths of the structured storage file formats can be found in the area of cross-system data exchange. Here, the data sets are smaller, a search is not necessary and structural information can be included with the data. Especially the XML format is emphasized which stores the information in a tree structure. With an XML schema further structural information can be provided. In contrast to XML-like formats the CSV format, with data in rows and columns, is built up like a database and can therefore fit into the group of databases-like or database formats.

Therefore, two major structural groups can be classified in static filing formats. First such formats that have a tabular structure and second those which have a tree-like structure. Certainly there can be found many other possible
static filing formats that are neither tabular structured, nor tree-like (for example once frequently used unstructured configuration files (*.ini)), but focusing on data source structures of practical relevance (see Chapter 2), these are almost exclusively tabular and tree-like structures.

4 Generic

On the input side of a generic connector for the Engineering Network Framework are the static storage formats and on the output side the dynamic storage formats. The connector transforms the information of the input formats into objects of the Engineering Network, then the objects are fleeting objects in memory and may be made persistent in the repository of Engineering Network by another connector. So for a generic connector, there are, in accordance with the above findings, on the input side, table-based data sources and data sources with a tree-like filing structure. Therefore, in the further design of the generic connector only these structures are considered. However, tabular and tree-like structures differ too much in their algorithmic processing, so that generics cannot be derived. Therefore, it has to be considered whether there is a possibility that one type of structure can be mapped by the other structural type and thus to reduce the two input formats to one universal format. For this purpose, the tabular and tree-like structures are examined in detail in the following two Chapters to determine any similarities or figure out characteristics.

4.1 Table-like Structures

A tabular structure, as it can be found in one of the databases from Chapter 2, consists of tables between which relationships of the following kinds can exist:

1:1 Relation This type of relationship assigns exactly one entity to another entity accurately. (e.g.: A husband has exactly one wife. See Figure 2.)

1:n (n:1) Relation This Relationship assigns an entity on one side to 0, 1 or more entities on the other side. (e.g.: a museum has n artworks. See Figure 3)

n:m Relation The n:m relationship assigns several entities to multiple other entities (e.g.: A professor serves n students, a student attends lectures at m professors. See Figure 4)

![Figure 2: 1:1 Relation](image)

The relationships in the tables of a database are implemented by foreign keys that refer to the respective associated records. For a 1:1 relationship an
additional column can be inserted into one of the two tables, which carries the reference to the other table. It has to be ensured that any reference occurs only once. In the 1:n (n:1) relationship, the table on the n-side gets an additional column that contains a reference to the other table. For n:m relationships a new table is created that contains value pairs which reference the two tables that are related. One of the columns carries the references to table n, the other column carries the references to table m. In Figure 5 a sample database is illustrated with five tables, a 1:1, 1:n and n:m relationship. Typically, the relationships in a database have no direction because they can be read forward, backward or in both directions. Since the labels in Figure 5 express only one way of the relationship (for the opposite direction there is not enough room in the graph), the relationship arrows are directed and point in the direction of the label.

If looking at the structures of a database from the perspective of object orientation, the tables can be viewed as the type definitions and the rows of tables are the instance. If one interprets the data as a graph, the tables are the nodes and the relationships the edges. Thus, the tables in Figure 5 result in an undirected typegraph. Undirected, because in the database it is not defined in which direction a relationship is traversed (a Captain has n crew members and n crew members have a Captain). The undirected typegraph can be split into $2^n$ directed typegraphs, where n is the number of edges, if one sets a reading direction for each edge. In Figures 6 and 8 the database schemata from Figures 5 and 7 are shown as a directed graph.
Figure 5: Example for relations between the tables of a database

<table>
<thead>
<tr>
<th>SpaceshipNr</th>
<th>SpaceshipName</th>
<th>NrCaptain</th>
</tr>
</thead>
<tbody>
<tr>
<td>NCC-2893</td>
<td>U.S.S. Stargazer</td>
<td>2</td>
</tr>
<tr>
<td>NCC-74656</td>
<td>U.S.S. Voyager</td>
<td>3</td>
</tr>
<tr>
<td>NCC-1701</td>
<td>U.S.S. Voyager</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NrCaptain</th>
<th>Captain</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>James T. Kirk</td>
</tr>
<tr>
<td>2</td>
<td>Jean-Luc Picard</td>
</tr>
<tr>
<td>3</td>
<td>Kathryn Janeway</td>
</tr>
</tbody>
</table>

Every shuttle of a spaceship can be used by n crew members

Every crew member can use m shuttles of a spaceship

Figure 6: Cyclic, directed, typegraph of the database in Figure 5
A special feature that makes a database very flexible, is the fact that relationships can be cyclic. There are open cycles that are directly visible in the database schema and hidden cycles, which are only visible on closer examination of the relationship.

Open cycles: Open cycles are cycles that arise when tables are set in circular relation to each other. A table A references another table B and B, in turn, references A. An example is shown in Figure 7. Here, the three entities are related to each other with a 1:1 relation so that a cycle is created. In Figure 8 this database is visualized as a graph and the open cycle can be seen particularly well. However, it has to be be said that this case is questionable from the perspective of database normalization, since one of these relationships constitutes an information redundancy in the database. The same facts can be represented with only two relationships. Nevertheless, a database allows this design. In practice, a database that contains open cycles in this way will rarely be found.

Hidden cycles: A hidden cycle is concealed in Figure 5 in the n:m relationship and will be visible if the relationship is split in a 1:n and m:1 relationship and a reading direction is set for the edges. Thus, the cyclic, directed typegraph shown in Figure 6 comes into existence. Such hidden cycles are
a standard element of a database and do not contradict the principles of a normalized database.

Therefore it can be noted that a database is structurally subject to an undirected typegraph, which can contain both open and hidden cycles and that can be converted in one of $2^n$ potential directed typegraphs by setting a reading direction. In the further course it will now be investigated whether a similar graph is hidden behind tree-like structures. If this would be the case, one structural type could be mapped to the other and this would simplify the design of a generic connector significantly.

4.2 Tree-like Structures

A tree-like structure consists of a root node, multiple child nodes and leaves. Here, the root, the nodes and the leaves are connected by strictly hierarchical relationships. Generalized a tree has nodes and relationships between them. The relationships apply to the fact that they are always between nodes of level $n$ and $n+1$. Relationships between nodes on the same level are not permitted. In addition, the nodes are unique. Tree-like structures are used, as described above, e.g. in XML formats. In computer science, one uses these structures not only for static storage, but also in programs. Unlike the use for static storage it is not uncommon that the nodes maintain horizontal relations. They allow to quickly browse the tree structure and simplify the access to individual nodes. This type of relationship, however, can not be serialized directly, because it allows for one cycles, which can not be serialized easily and for another it can
lead to a double filing of objects. Therefore most serialization libraries and tools deny to serialize cycle-prone object structures. For the static storage formats, this means that tree-like structures with horizontal relationships can be excluded from consideration.

In the following, it will be examined by an example XML structure, whether there is a hidden graph behind the tree-like structure as discovered in the tabular structures.

In Figure 9 an excerpt from an XML serialized Engineering Network Model instance can be viewed. The objects in the excerpt are serialized instances of their types. The type definitions can be stored either in the XML schema or class definitions. Among other things, the Engineering Network Model consists of several subtypes such as the EOTypes or DataTypes, which are kept in lists.

```xml
<?xml version="1.0" encoding="UTF-8"?>
<ENModel xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <ID>4fc1f7aa-da94-4c24-afdb-6c8fa7fd7bb</ID>
  <Name>Engineering Networks</Name>
  <DataTypes>
    + <ENDataType>
    + <ENDataType>
  </DataTypes>
  <EOTypes>
    - <EOType>
      <ID>14d74518-e101-4f9d-bee8-1c9f4a8ab519</ID>
      <Name>Cities</Name>
      <Description><no description set></Description>
      <AccessControlList/>
      <Instances/>
      <RelationshipsIDs/>
      - <Attributes>
        + <ENAttributeType>
      </Attributes>
    </EOType>
  </EOTypes>
</ENModel>
```

Figure 9: Excerpt from an XML serialized Engineering Network Model

If one views the holding of child objects from an abstract modeling point of view, a father-child relationship exists here. An ENModel has n possible sub-objects of type EOType. The fact that there exists a list between the model and its sub-objects that is also visible in the XML code, plays no role on the modeling level. If you illustrate the XML serialized Engineering Network Model instance as a graph (see Figure 10), this gives a much smaller graph than it would have been expected from the XML code. All lists are contained in the 1:n relationships implicitly and need not be stated separately in the graph.

Furthermore, from the hierarchical structure of the XML results a non-cyclic directed graph. Because of the absence of type information in an XML file, it is,
in contrast to the graph of a database, an instance graph. Then the typegraph results from the type information related to this instance (from XML schema or classes). With reference to the findings of the examination of the tabular structures and despite the large differences at first glance, that means that a common structure does exist.

4.3 Unification

In the previous two Chapters (4.1 and 4.2) it could be shown that a database is based on an undirected cyclic type graph (hidden & open) that by setting a reading direction can be decomposed in one of $2^n$ directed cyclic graphs. Furthermore, a tree-like structure as found in XML is based on a non-cyclic directed graph, for that, in the case of XML and the existence of a XML schema or a class definition, a typegraph exists. Since a non-cyclic directed graph can be represented by a cyclic directed graph (a cyclic graph with no cycle), all tree-like structures can be represented by a tabular structure.

A transformation function that transforms a tree-like structure into a tabular structure is given as follows:

1. For all node types create exactly one table

2. For each relationship between node types t and t+1 add a column for a foreign key to the table that corresponds to the node type t+1 and create a 1:n relationship with the primary key column of table t on the 1 side and the foreign key column of table t+1 on the n side.

So, now it is shown that it is sufficient for a generic connector to allow a
connection to a tabular format. That makes the implementation much more easier.

5 Summary

For the Engineering Network Framework various connectors are required to access different data sources. To simplify the development of connectors and to save work, the commonalities of the connectors to create a common code base have been figured out. But this turned out to be very difficult. Therefore, starting at a basic level the concept for a generic connector was developed, as presented in this paper.

In this paper, it is shown, that today’s most relevant data sources in the field of PDM/PLM are either based on table- or tree-like structures. Subsequently it is pointed out that a graph is beneath both kinds of data source structures. Beneath the table-like structures we found a cyclic, non-directed typegraph and beneath the tree-like structures we found a non-cyclic directed typegraph. With this knowledge, it could be proved that the tree-like structures are a subset of the table-like structures and a transformation rule to transform the tree-like structures into a table-like structure could be provided.

With these results, it is now possible to implement a generic connector that subsumes the commonalities of both structure types. Of course, the generic connector will not be able to connect to every data source. Most data sources need some more customization. But the generic connector provides a good foundation for further customizations and reduces the implementation efforts a lot.

References

In der Reihe VPE White Paper erschienen


