RC23201 (W0405-025) May 5, 2004
Computer Science

IBM Research Report

An Information Integration Framework for Product Life Cycle Management of Diverse Data

Huong Morris, Simon Lee, Eric Shan, Sai Zeng
IBM Research Division
Almaden Research Center
650 Harry Road
San Jose, CA 95120-6099

An Information Integration Framework for Product Life Cycle Management of Diverse Data

Huong Morris (IBM Research, thm@us.ibm.com), Simon Lee (IBM Data Management),
Eric Shan, Sai Zeng (IBM Extreme Blue Interns)
IBM Almaden Research Center
650 Harry Rd
San Jose CA 95120

ABSTRACT

Automobile, aerospace and other industrial manufacturers have long depended on single vendor solutions to support their enterprise-wide engineering activities. Increased product complexity, distributed authoring environments and the needs for tighter team integration with partners and suppliers have created challenges and new opportunities for IT vendors to be able to integrate systems from multiple ISVs to work together as a coherent Enterprise PDM. Both design and manufacturing processes in Automobile, aerospace and manufacturing industries have evolved over the years to include a multitude of CAD/PDM systems for supporting their enterprise-wide engineering processes.

The efficient management of distributed information has become increasingly important as data is digitized and placed online. But this data comes in many forms and is under the management of diverse middleware components and applications. Users need integrated and on demand access to information, but the data itself is a moving target. Siloed systems are evolving independently and there are usually no budgets available for the integration task. With many companies operating with a focus on cost-cutting, the ability of an enterprise to align processes and information that support the efficient development, delivery, and management of a product over its useful life will determine who has the winning edge. Therefore, a variety of integration methodologies has emerged, and those methodologies must be carefully and selectively applied in order to achieve integration without unaffordable "tear up" of the systems.

This paper will describe in detail a case study and solution of an IBM Research project called Hedwig. Hedwig provides robust solutions in the space of Product Life Cycle Management (PLM). We focus on the several research issues including information federation, data mapping, synchronization and web services connections. We describe a working system that allows access to heterogeneous Product Data Management (PDM) systems that are used in the automotive and aerospace industries.

1. INTRODUCTION

Accessing heterogeneous data sources is increasingly necessary and difficult and there have been a number of studies on integration techniques [1-3]. The pressures of increasing competition have been forcing major corporations to reengineer their business organizations to achieve greater synergy and efficiency: this demands data integration. In the past, the strategic goal of these efforts was to develop enterprise-wide solutions that tied together information systems across business units within the organization. Still not wholly realized today, this goal of efficient management of distributed information has become progressively more difficult for several reasons: 1) the data volume is increasing due to increased digitization of sources, 2) it is coming from even more sources where it is under management of diverse middleware, and 3) virtual marketplaces and global partnerships are requiring integration efforts which stretch across the boundaries of previously siloed systems within individual corporations. According to the Aberdeen Group report in August 2003, many companies have adopted product life cycle (PLM) as a key strategy for driving competitive differentiation, performance, and profit optimization in the market. PLM is a business strategy intended to link all information, people, and processes associated with a product from initial concept through end-of-life disposal. In the past, PLM began with design and ended with manufacturing. These days PLM must start with market analysis and doesn't end even after sale: product warranty claims can be crucial in a time of small margin; product liability can destroy a company and post-sales service can be the business that really matters [4].

Information integration and management products, especially those that allow product lifecycle management for industries such as the automotive industry, have seen large growth in revenue for vendors in the past several years [5]. Interoperability barriers due to proprietary solutions, particularly in CAD, CAE, and PDM applications, and further exacerbated by each company's own set of requirements and lack of enforced standards have led to several endeavors in information integration. In this paper, we will identify four key areas concerning information integration as it applies to the automotive industry (and to the aerospace and electronics industries in similar ways).

In our project, Hedwig, a real-world IT solution was developed as a proof-of-concept for IBM Product Lifecycle Management customers chosen from the Automobile and Aerospace industries. Hedwig originated as an IBM Extreme Blue project http://www.ibm.com/extremeblue where top students from different universities come to IBM to work with professional researchers and developers to rapidly develop a case study and build a system solution based on real customer needs. Hedwig's goal was to study integration requirements and develop a prototype information integration system that leverages current IBM data management technologies combined with commercially available PDM systems. Specifically, it uses IBM's DB2 Content Manager (CM) to provide storage, management, and distribution of all types of text, digital and multimedia content across different applications. In addition, the project uses DB2 Information Integrator (DB2 II) and IBM Websphere Web Sevices products to federate and provide intelligent access to heterogeneous "third party" PDMs.

Hedwig was successful in demonstrating progress towards two central goals. The first was to obtain efficient access to third party PDMs without excessive query processing load. We return results while maintaining the inherent relationships within the data. For instance, a query on a specific car model will send requests to diverse PDM data stores, because a whole vehicle often relies on subsystems from different partners that use diverse PDM systems. Data is returned from multiple sources, but the parent-child relationship of those parts needs to be retained where they exist. An overall hierarchy is maintained, while the sub-hierarchies maintain their native structure. Maintaining as much structure as possible in the top-level data model (CM) is of course desirable, as opposed to a flat structure, which might have for example, nuts and bolts next to the frame drawings and the final car design. The second goal is to standardize the connection framework so that unforeseen processes and services can be incorporated in the future: this also enhances global scalability and portability. This is enabled through use of web services.

We will approach this paper by describing (through an informal case study) our research and development to build a simple but highly illustrative federation infrastructure. We will highlight some of the approaches we took in each of these areas as well as future approaches and modifications needed to enable a scalable and efficient federation infrastructure. We will describe our experience with the rapid integration of several databases and their applications and recommend a framework for future information integration. We will also address the several accompanying research issues and discuss their projected impact on the overall architecture.

2. DESIGN APPROACH AND CONSIDERATIONS

2.1. Overall federation infrastructure

Large automobile and aerospace enterprises especially incur high costs from maintaining and using disparate CAD and PDM systems, long delays due to the transfer of information and manual reentry into destination systems, and quality problems caused by the movement of data across media boundaries [6]. We aimed to design and build a federation infrastructure that would alleviate many of these problems in the industry by allowing and managing the bi-directional flow of data through distributed heterogeneous systems and environments in an automatic way.

This federated infrastructure allowed those backend sources to be accessed as if they were from a single source with a common data model. Users would make XML Path Language (XPath http://www.w3.org/TR/2003/WD-xpath20-20031112) requests through a browser to the integration platform, where the queries are automatically rewritten based on the contents of the central data management system (discussed in the next section) and the query manager and then converted to backend-specific semantics and sent through the connectors (discussed in Section 2.3). Results from the backend sources would be appropriately transformed, mapped, and reassembled through the connectors and central management system and returned to the user in a consistent and unified format. Indeed, this infrastructure focused on providing end users data visibility to the distributed data sources, with interoperability between heterogeneous environments occurring only on a small scale. To this end, the single-access-point query and navigation features served well.

The major roadblock in this infrastructure is that the involved data sources are rarely all immediate sources. Though the above federation platform allows even systems outside the enterprise to be directly connected, many trading partners/suppliers (and even different divisions within a corporation) wish to maintain strict control and data integrity over their data sources and applications. Interoperability and automated data transfer become the keys to success in the automotive industry. Each OEM, especially big automobile companies, would like to maintain its specific PDM systems, such as IBM's Enovia and EDS' Metaphase, while its Tier 1, Tier 2, etc. suppliers manage their own data management systems and/or applications.

This integration platform can be transformed into an engineering hub, which provides a portal for data visibility to end users and connection mechanisms to immediate data sources as before. At a higher level, however, they allow for a whole network of engineering hubs, one that stretches across all facets of a virtual enterprise and connects business partners together across logical divisions within a company, company intranets, and geophysical barriers, as shown in Figure 1. These hubs must support decentralized, message-oriented communication between one another using standardized, XML-wrapped data units. The integration hubs and engines must therefore serve human as well as application requests distributed across multiple hubs, requiring query managers to query across the network to determine an appropriate horizon in which results are returned and assembled.

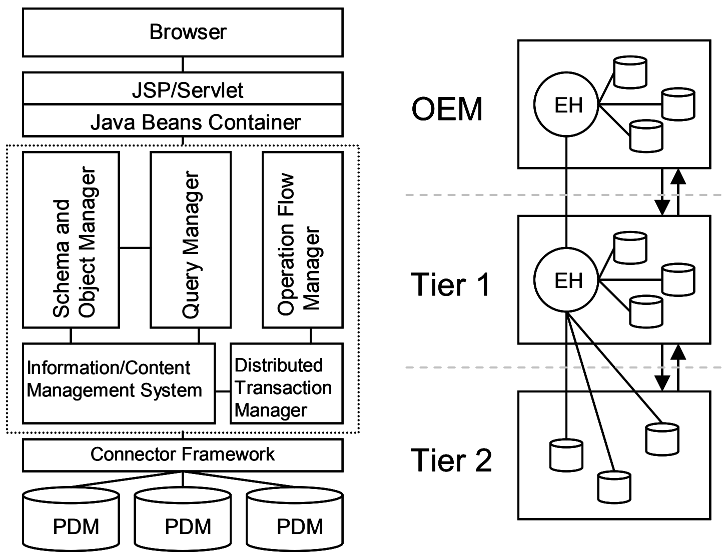

Figure 1: Federation Infrastructure (left) and Global network of 'engineering hub' (right)

2.2. Intermediate data management

The data path from the backend data sources to the end user generally involves up to two major intermediate levels of data management. Most traditionally, developers have used only a single intermediate level, employing the approach of data consolidation, or placement, where data is physically consolidated into a single, local repository in advance of using the data to service user queries [7]. This consolidation typically takes place in an information warehouse, which provides periodic data extracting services [8]. Recent endeavors, however, have leaned towards a second approach of data federation, where the data is dynamically and directly retrieved on every request. Several complex hybrids of the two approaches involving data marts and warehouses alongside federation have also been introduced [9].

The approach we took was to use a mature content management system such as the IBM DB2 Content Manager product, as an intermediate step in the federation operation. Therefore, each query request by the user would pass through the management system before reaching the backend sources, and each set of results received from the backend sources would be stored and assembled in the intermediate system before being returned to the user. By putting an advanced data management system at such a high level in the data path (near the integration engine), we realized several benefits. This management system partly acted as a data warehouse in consolidating the data locally within the context of a single, unified schema. On the other hand, it served more as a cache than a warehouse in functionality by storing the binary data and metadata associated with recently requests locally. Furthermore, it provided advanced management of metadata, including storage of backend information (i.e. where the data originated) on a per result basis.

These benefits together enable huge gains in performance; large binary data sets, which put immense strain on the network when transported, are moved to this local “cache” only once or when the data has both been updated and requested by the user instead of over periodic intervals (user may retrieve outdated data) or blindly fetched on each request as would occur in a purely federation-based infrastructure. Similarly, backend information allows queries on certain result sets to be sent directly to the involved backend sources. These all help to reduce per query latency (which is of greater importance than overall throughput). The underlying principle of this is that the central cache or warehouse should involve a highly evolved and intelligent content management system to handle caching, performance, business analysis and other functionality. Placing this system so near the integration engine also substantially aids it in distributed query planning, data source registering and mapping, and synchronization protocols.

2.3. Connector Implementation

Information management systems require a combination of middleware to support integration of diverse software systems [10]. These middleware technologies implement *connectors*, which are used to provide data transport and possibly data mapping and transformation between the backend data sources and the core integration platform. Database Oriented Middleware, which facilitates communication with databases through means such as ODBC/JDBC calls; Message Oriented Middleware, which uses queuing software to move information point to point; and Message Broker Middleware, which offers advanced routing and mapping mechanisms in message brokering between several different applications, are a few of the types of middleware used as the underlying technology for data transport through connectors.

We used Database Oriented Middleware (JDBC API) and a subset of the backend interface API to build two connectors to two separate PDM systems. Since the automotive industry’s needs involved a rapid integration of several disparate systems, we focused on achieving:

- *Extensibility* – We desired that additional connectors be simple and straightforward to implement. This required that every connector class have access to a standardized set of backend-specific user and privileges information, including PDM schema, location, and server name.
- *Maintainability* – While we wired the semantics mapping into the connectors themselves, we cleanly separated out the smallest subset of backend-specific code to allow for clearly meaningful function “templates” to implement and occasionally update. Furthermore, this solution can be easily integrated into IBM DB2 Information Integrator, which enables flexible changes and updates to the semantics mapping to support organizationally structured data retrievals.

Thus our attention turned more towards the framework model in which to implement the connector code on the integration platform side instead of the underlying transport mechanisms leading to the backend side. In our framework, we designed a set of generic *PDMDatastore* classes that served as the foundation (and entry point to all the system/engine logic) for all connector implementation. This provided utility access functions to the central content management system as well as common data retrieval methods (involving JDBC requests) to the backend sources. By providing this set of functions, it became significantly easier to implement the backend-specific connectors without concern about inconsistencies with other components in the system. Each backend-specific connector extended the parent *PDMDatastore* classes.

Thus to construct a connector to Enovia VPM, developers would only need to implement two main functions in *PDMDatastoreVPM*. The 'searchFederation()' function supported open XPath queries from the user to retrieve part information from the backend, and therefore would require developers to 1) convert a query string parameter to backend queries/API calls and 2) query and reconstruct the results from the backend into a system result set object. The 'navigateAssembly()' function supports user navigation through entire assembly structures by following parent-child pointers, and would consequently require developers to 1) query for children parts using a parent part's unique part number (where a child is one level down from a parent in the assembly hierarchy), 2) acquire the attributes from each child to pass on using a system data object, and 3) passing the object on to perform the linking logic. This connection framework worked exceedingly well as the second connector took only a few days to add to the integration platform.

Indeed, we intend to extend this connector framework across other types of middleware. As discussed earlier, however, the other side of integration in the automotive industry demands greater extents of interoperability. The tight coupling between our integration platform and backend sources played a dominant role in terms of interoperability in even a simplistic connection effort. The backend application environment required us to downgrade our own database applications when the project began. There was a constant threat of backend engineers rebuilding their database tables or upgrading their systems and possibly their interfaces. And while the data modeling process is always a tremendous effort in any integration endeavor, our reliance on the backend models often hampered forward progress on the platform side. To resolve these problems we have already begun to support connector transport using Web Services Middleware. This loosely-coupled technology will promote interoperability; this allows service provides and requesters to operate on different environments using any language by using platform-independent and standard network protocols with TCP/IP, transport protocols with HTTP, message format with SOAP, and description mechanisms with WSDL. This will also significantly increase productivity as it is relatively easy to generate SOAP wrappers and generate WSDL documents to cast applications as web services for connecting legacy applications. Furthermore, the flexibility of XML Schema and XSLT templates allow for dynamic interpretation of semantics and some freedom in message and data mapping and transformation.

2.4. Relationship Management

In most cases, users desiring integration of their data stores do not simply want to retrieve independent fragments of data, but to navigate through and visualize entire structures of data based on relationships. One enormous barrier for the automotive industry is retrieving the entire bill of materials (BOM) of certain products. For instance, an airplane can be broken down into several subcomponents, such as the engines, wings, fuselage, and cockpit. Each of these can be subdivided into several subcomponents. The entire assembly of a plane can reach thousands of levels of subassemblies consisting of millions of parts. To enable BOM retrieval, developers would typically hard-wire this functionality into integration platforms in an ad-hoc fashion.

Changes to the backend data stores or additions of heterogeneous data sources would require a huge revamping process.

Our prototype used the central content management system to rebuild the inherent backend relationships for navigation and display by the front end user. The concept of *folders* and *links* were supported by the management system, where items (that had a folder component) could contain other items through links. Thus for an automobile, any part's subassembly of parts was linked to it by placing the appropriate subassembly parts in the correct folders (parent parts). At every step in a user's navigation through an assembly structure, the request (such as for an engine's subparts) is handled by the 'navigateAssembly()' function from above. Thus, the subparts are fetched through some means of computation, such as a SQL statement or sequence of API calls, linked to the parent part, and displayed to the user in a hierarchical format. While this hierarchical folder structure is conducive to storing most any type of relationship, we hope to extend this further by relying on new components to handle relationship management.

The main component is the *Relationship Definition & Computation Templates* (RDCT) database, which allows for the definition of relationships that the system can support. There are several types of complex relationships that a user can define. Modeling languages, such as the Unified Modeling Language (UML), formally specify many such relationship classes, such as association, aggregation, and composition. We decided, however, to shield the user from these abstractions of relationships and provide them with the flexibility of defining any sort of relationship in a conceptually intuitive and visually explicit manner using XML scripts. For example, the subassembly relation of an automobile could simply be defined as a 'subassembly' relationship. Similarly, an executive who is navigating a certain project's responsibility and management chains could have those associations represented as 'coordinator' and 'manager' relationships.

The relationship definition script also includes the list of backend data sources that support this relationship as well as associated mechanisms for computing the relationship. For instance, these computation scripts could include anything from simple SQL statements to the calling sequence and input/output parameters of web services invocations through compiled proxy code or proprietary API invocations, all of which can be automatically interpreted by an XML engine to allow for dynamic computation at every step of navigation across all predefined relationships. (As an example, while navigating through a car's subassembly, we may be interested in navigating the responsibility chain of the managers overseeing a certain subassembly's design.) Thus, each new relationship is contained within its own definition script that can be easily extensible to any number of connected backend systems. In this way, the system has the flexibility to allow for adding, removing, or modifying relationships versus having to dig into the guts of the integration platform. These relationship assembly structures are managed within session beans that can be persisted or transported in XML format, allowing further flexibility in data exchange, this time based on dynamic user specifications and data navigation.

3. ARCHITECTURE

Figure 2 shows the overall Hedwig architecture.

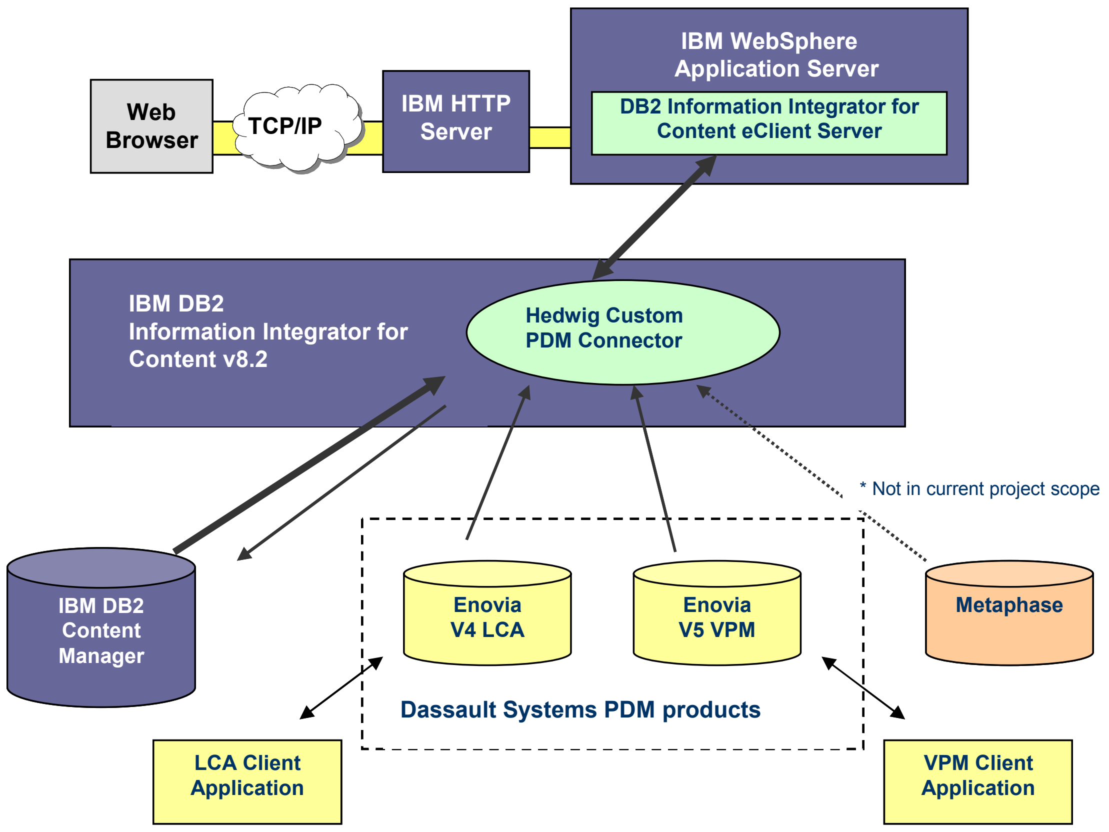
IBM WebSphere Application Server
DB2 Information Integrator for Content eClient Server
Web Browser
TCP/IP
IBM HTTP Server
IBM DB2 Information Integrator for Content v8.2
Hedwig Custom PDM Connector
* Not in current project scope
IBM DB2 Content Manager
Enovia V4 LCA
Enovia V5 VPM
Metaphase
Dassault Systems PDM products
LCA Client Application
VPM Client Application

Figure 2: Overall Hedwig Architecture

It works as follows:

1. The user logs in using the IBM DB2 Content Manager (CM) browser, called eClient. [13].
2. The user queries for a part or assembly. Hedwig transforms the query to Xpath http://www.w3.org/TR/2003/WD-xpath20-20031112 and then passes it to IBM DB2 Information Integrator's (DB2 II) federation layer engine.
3. The federation layer then uses the custom connectors and logic to each of the PDM systems to process the query. The connectors thus reach out to the disparate data sources and combine and store this information into CM for building complex relationships and assembly structures. The federated data model is detailed in the next Section 4 while the connector design and implementation is discussed in Section 5.

4. FEDERATED DATA MODEL

The federation layer operates on a single canonical data model in CM. In the long run we will build an enhanced generic model that may include some features of backend-specific data models. Even at this stage, CM provides clear value add because of the richness of its data model, its ability to cache a whole design or sub-design while being worked on, and the ability to warehouse snapshots of various designs for versioning and archival purposes. Limitations of space in this paper allow only a brief description of the canonical data model.

Typical final products in these industries, such as passenger cars or commercial airplanes, are composed of hundreds of thousands or significantly more parts that are assembled into a hierarchical structure, sometimes with hundreds of levels. We need to maintain, evolve and version this hierarchy throughout a product's lifecycle and inherit any hierarchical structure where possible from the third party PDMs.

4.1. CM Data Modeling

The required CM data modeling is accomplished in three major steps. In the first step – data identification – the full set of data is gathered, analyzed and represented. We have two exemplary but very different PDM data sources to study – the first one is VPM (Virtual Prototype Management) and the second one is LCA (Lifecycle Application) [14]. The next step – data categorization and hierarchical relationship construction – helps to develop the structure of our data model. We use UML to represent the data model at this stage. In the last step – part structure CM data modeling – we 'convert' the data we gathered, and diagrammed previously, into a CM data model [11].

In CM, an *item type* is a template for defining and later locating like items. An item type consists of both system- and user-defined attributes. An attribute stores data or values that describe a characteristic or property of an item. An item is an instance of any item type. During data modeling, an item type can be further classified as an item, resource item, document, and document part. Items are those things that can be described completely by a set of attributes. Items are similar to a row in a database. Documents provide the template to model multi-part documents with related contents. This template saves the effort to create similar data models from scratch.

When creating an item, the item's behavior can be identified by a descriptive attribute called a *semantic type*. This semantic type helps to distinguish the usage and the purpose of different items, which belong to the same document item type. Forming relationship between items can be done by *links*. A link is a directional relationship between the source item and the target item. CM offers more building blocks than those described above however this is simple and usable set..

A part instance contains zero or more part instances. In CM, a document item with semantic type as folder allows inclusion of a variable number of document items. Based on this similarity analysis, an item type as document with semantic type as folder is selected to represent part instance. The meaningful meta-information to describe a part, such as part number, name, creation date, etc. are defined as a set of attributes in this selected item type. Each part instance has documents attached to it. In CM, the document item type provides the template to allow users to define metadata for the document.

4. CONNECTOR DESIGN

For the Hedwig Project, our team built two custom connectors, one to Enovia LCA and another to Enovia VPM. As mentioned above in Section2.3, we designed the connector framework in a manner to specifically support two main goals of Extensibility and Maintainability.

In terms of extensibility, we designed the framework model as shown in Figure 3 below. Each backend-specific class extends the basic PDMDatastore class, which serves as the foundation for all connector implementation. This provides utility access functions to the central repository that maintains the repository's consistency as well as generic utility functions for common data retrievals, such as through JDBC calls. By providing this set of utility functions, it becomes significantly easier to implement the backend-specific classes without concern about inconsistencies with other components in the system.

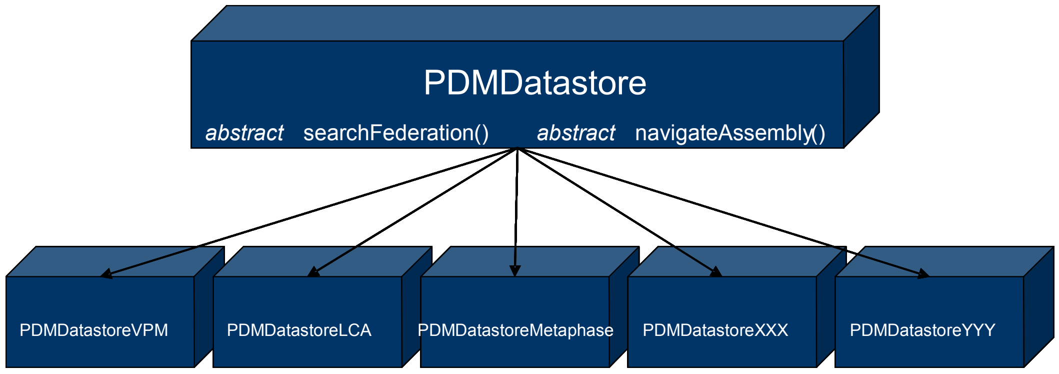

Figure 2. The PDMDatastore framework and extension classes

Moreover, the PDMDatastore parent class enforces the methods that must be implemented to provide the appropriate functionality by providing a short list of abstract methods. The two most critical functions are shown in Figure 3. The 'search Federation ()' function supports open XPath queries from the user to retrieve part information from distributed backend data management systems. The 'navigateAssembly ()' function supports user navigation through entire assembly structures by following parent-child pointers.

5. CONCLUSION

There remain several research issues that will be further investigated in building a robust infrastructure for information integration.

- Synchronization: It is important to find efficient, practical, and functional means for ensuring that CM (the central data repository) is adequately synchronized with each backend data management systems. As of now, we use an approach of refreshing the relevant parts of CM's contents. Presently, users are expected to only perform 'search' operations and retrieve documents. In this data-brokering environment, our solution serves well. Eventually, we can move to a more complete data warehouse model. There are several ways to achieve more advanced synchronization including using built-in hooks or triggers at both ends and then propagating these data connector listeners as supported by IBM's Websphere Web Services products.

- Standard semantic mapping mechanisms: As the reader will see from the above, there is still a manual mapping process that is required between the central data model and the backend-specific data models. With the additional functionality of allowing for navigable assembly structures and BOM representations, further mapping must be done. These new mappings should not be wired into the connector code, but specified at the federation layer level using standardized formats, GUI, and interpreting engines. Schema mapping and integration tools would be a valuable addition.

- Performance: performance is a very crucial part of the system. As the integration stretches to a broader role in product lifecycle management (e.g., to include customer support), we will need to further consider latency as well as throughput. Our current prototype does considerable data access the first time data is accessed, but more advanced proactive caching and partial-update schemes can be developed.

6. REFERENCES

1. **"Query Reformulation for Dynamic Information Integration",** Yigal Arens, Craig A. Knoblock and Wei-Min Shen, J. Intell. Inf. Syst. 6(2/3): 99-130 (1996).
2. **"Information integration – Distributed access and data consolidation"**, B. Devlin, IBM White paper, 2003.
3. **"Composing Mappings Among Data Sources"**, Jayant Madhavan and Alon Y. Halevy, VLDB 2003: 572-583.
4. **"2004: Era of Warranty and Earlier Warning"**, Kevin Mixer, AMR Research, January 2003 report.
5. **"The Product Life Cycle Management Applications Report"**, 2002-2007, AMR Research Report.
6. **"Product Data Management Interoperability"**, AIGA Research Report to NIST, AIAG-NIST 2003.
7. **"Information Integration – Distributed access and data consolidation"**, B. Devlin, IBM White Paper, 2003.
8. **"Information Warehouse: An Introduction"**, Publication No.GC26-4876, IBM 1992.
9. **"Information Integration – Extending the data warehouse"**, B. Devlin, IBM White Paper, 2003.
10. **"A New Class of Software",** M. Wagner, Open Systems Today, No. 113, 1992.
11. **"Modeling Your Data in Content Manager Version 8"**, IBM Content Manager for Multiplatforms, May 2003.
12. **"Data Federation with IBM DB2 Information Integrator V8.1"**, Redbook, SG247052, October 2003
13. **IBM DB2 Content Manager V8 Implementation on DB2 Universal Database: A Primer**, by Chen et al, May 2003
14. **Enovia LCA Documentation**, Version 5, Release 11, Dassault Systemes, 2003.