Federated Information Management for virtual enterprises

Garita Rodriguez, C.O.

Citation for published version (APA):

General rights
It is not permitted to download or to forward/distribute the text or part of it without the consent of the author(s) and/or copyright holder(s), other than for strictly personal, individual use, unless the work is under an open content license (like Creative Commons).

Disclaimer/Complaints regulations
If you believe that digital publication of certain material infringes any of your rights or (privacy) interests, please let the Library know, stating your reasons. In case of a legitimate complaint, the Library will make the material inaccessible and/or remove it from the website. Please Ask the Library: https://uba.uva.nl/en/contact, or a letter to: Library of the University of Amsterdam, Secretariat, Singel 425, 1012 WP Amsterdam, The Netherlands. You will be contacted as soon as possible.
Chapter 4

Design and Implementation of the Federated Information Management System for VEs

4.1 Introduction

As identified during the requirement analysis phase of the DIMS, Virtual Enterprise members need to have access to up-to-date information that is physically distributed among different nodes in the VE network. This is necessary in order to support for instance, the basic data exchange operations among VE members, as well as more sophisticated functionalities regarding the coordination and monitoring of the tasks that are being independently executed by different VE members. Clearly, an advanced information management mechanism is needed to be designed and implemented in order to provide transparent access to VE distributed information. In principle, the design of this mechanism could be based on a single globally accepted VE schema, through which different VE nodes can make available their information to all the other nodes that represent actual or potential VE partners. Through this single global schema, enterprises would have access to an integrated view of the VE distributed information and the physical location details of the data would remain hidden.

However, due to the competitiveness of pre-existing enterprises and their proprietary information, it is not realistic to assume in the design of the information management platform for the VE network, that such a single global schema can be used to define all the information that is visible and exchanged by all partners. For instance, it is evident that the degree of trust among competitive enterprises in a VE is limited, and that each enterprise needs to precisely define the access rights and visibility levels on its local information for every other VE member. Besides the issues of trust and protection of sensitive information, there are many other factors that determine the information access rights among individual VE members, such as: the relationships with other VE members (e.g. producer, consumer, retailer) within a
given distributed production chain, the function that they play in the VE (e.g. coordinator, supervisor, regular partner), and the legal rights/obligations that they may have acquired through the VE contracts.

In order to address these and many other issues identified during the requirement analysis phase, the approach for VE information management adopted in the design of the DIMS is based on a federated database architecture [71, 11]. In particular, the design of the DIMS accomplished in the context of the ESPRIT PRODNET II project and described in this dissertation, introduces a “VCL integrated schema” that not only supports transparent access to distributed VE information, but at the same time it allows the proper definition of access rights among VE members through the creation and maintenance of export schema hierarchies defined on the local VCL database schema.

The main objective of this chapter is to describe the general design and implementation of the DIMS federated architecture and its internal components. In particular, the main DIMS elements that are analyzed in this chapter include:

a. The VCL federated schema that provides an integrated definition of both the local and the distributed VE information that can be accessed by end users and applications at each VE node.

b. The DIMS Export Schema Manager, which allows each VE member to create export schemas hierarchies that implement the proper access rights and visibility levels on the local information that needs to be shared with other VE members.

c. The Federated Query Processor, whose major goal is to handle the local and distributed queries issued on the VCL integrated schema, while respecting the access rights defined at each node through the export schemas. The DIMS federated query processor also incorporates a workflow-based mechanism to interoperate with the internal systems of the company in order to retrieve the most up-to-date information when required.

d. The DIMS Server Agent, which supports the interoperability between the internal DIMS elements and the other VCL components and external enterprise systems.

Additionally, this chapter illustrates the actual application of these generic DIMS components and mechanisms in the context of the general PRODNET VE demonstration scenario.

Consequently, the structure of this chapter is organized as follows. Section 4.2 presents the general approach for the design of the DIMS architecture and introduces its major internal components. Section 4.3 focuses on design of the VCL integrated schema and its high-level data structures (based on the requirement analysis phase described in Chapter 3). Section 4.4 provides a more detailed analysis of the concepts of roles and access rights definitions for VE partners, and how they are properly supported by the DIMS Export Schema Manager Tool (ESMT). Section 4.5 describes the specific tasks associated with the DIMS Federated Query Processor (FQP) component, and illustrates how they are assisted by internal VCL workflow management.
activities. Section 4.6 addresses more specific design and implementation details regarding the DIMS interoperable Server Agent component. Section 4.7 demonstrates the application of the aforementioned DIMS architectural components and mechanisms in several actual VE scenarios in PRODNET. Section 4.8 includes some possible directions for future work and extensions to the DIMS functionalities, focusing on the export schema management approach. Finally, Section 4.9 summarizes the main conclusions of this chapter.

4.2 General Approach for the Design of the DIMS

In the next sections, the details of the DIMS reference architecture and its main internal components are described.

4.2.1 The DIMS Three-tier Architecture

In addition to the traditional client-server applications, some multi-threaded applications are conveniently modeled using a three-tier architecture, also called client-agent-server architecture. In this architecture the client is only concerned with presentation services. The agent (or application server) processes the application logic for the client tier, hiding the underlying implementation and access details of the server tier and adding higher level support functionalities for the client. In this way, the server tier is designed in order to encapsulate lower-level implementation details of the data management services required by the agent tier.

In this sense, the DIMS design and implementation approach follows a three-tier architecture of this type (please see Figure 4.1). The client tier is represented by all the other VCL components that request DIMS services via a DIMS client library. The applications server (agent) is represented by the DIMS Server Agent, together with the other DIMS internal operational components. The DIMS Sever Agent acts as a client of an internal database server (i.e. Oracle), which in turn represents the server tier in this scenario. Bearing in mind this general three-tier architecture will help understanding the relationships among some DIMS components that will be addressed in forthcoming sections of this document.

Figure 4.1: General DIMS three-tier architecture.
4.2.2 The Internal DIMS Reference Architecture

This section focuses on the main components of the DIMS applications server tier introduced in the previous section. The general reference architecture of this tier embodies the following components, as depicted in Figure 4.2:

- VCL Integrated Schema.
- Export Schema Manager and Tool.
- Federated Query Processor.
- DIMS Server Agent.
- Internal DIMS Database Manager.
- DIMS Kernel Configurator.

A general description for every component is given in the paragraphs below.

The DIMS VCL Integrated Schema

As mentioned in Chapter 3, the DIMS component adopts a federated database approach in order to allow transparent access to the VE distributed information, with proper support for enterprise node autonomy and access rights. In particular, the design of the DIMS is based on a "VCL integrated schema" that provides a unified definition of both the local and the distributed VE information that can be accessed by end users and applications at each VE node. Other VCL components and external enterprise modules issue federated database queries on this schema through the DIMS server agent, which takes care of the interaction with the federated query processor (for federated and distributed queries), or with the local database schema (for certain kinds of queries that do not require advanced query processing). The VCL integrated schema is described in details in Section 4.3.

Export Schema Manager and Tool

The Export Schema Manager (ESM) is the component that encloses the functionality to create and maintain the hierarchy of export schemas that are defined on the VCL local schema, based on the visibility access that need to be specified for a given node. The ESM functionalities are used for instance, to create a basic VCL export schema, and then, to define VCL partner export schemas based on it. The ESM will ensure that the export schema hierarchy remains consistent, and that the schema definitions for every dependent partner export schema are properly created. The ESMT (Export Schema Manager Tool) developed for DIMS provides a user interface to support the definition and creation of the export schemas as part of the local information that is shared with other enterprises. This component is further described in Section 4.4.
The DIMS Federated Query Processor

The main objective of the Federated Query Processor (FQP) is to transparently support the access to data distributed over the nodes of the VE network, taking into account the specific visibility access rights (represented by export schemas) defined for every node. As will be described in Section 4.5, the federated query processing functionality of DIMS enables end users such as the VE Coordinator to query the privileged VE-related information for which the coordinator is authorized, while hiding the data location details. Furthermore, considering that the information in legacy systems and internal modules (e.g. ERP/PPC) of the enterprises is updated independently of the VCL operation, the federated query processor in the DIMS is also responsible for retrieving the most up-to-date generated data, in case it is required by the query issuer application or end user.

The DIMS Server Agent

The multi-user DIMS Server Agent corresponds to the heart of the “agent tier” of the DIMS architecture introduced in Section 4.2.1. Basically, the DIMS Server Agent is responsible for receiving and dispatching all the DIMS service requests issued by the other VCL modules. The Server Agent first determines the nature of the service requests and then triggers the activation of the involved DIMS internal components, for instance, the Federated Query Processor. The Server Agent also contains a library of services that support a part of the general operation of the VCL modules. As will be explained in details in Section 4.6, the Server Agent represents the gateway
to the internal DIMS architecture, which encapsulates all the specific information management services for the VCL modules. The DIMS agent tier is implemented using a sophisticated multithreading mechanism implemented in C++ through which multiple service requests can be processed concurrently. For instance, Figure 4.3 represents an example in which the DIMS agent dynamically creates multiple working threads depending on the incoming service requests from end-user applications or VCL modules (e.g. LCM, DBPMS, PCI components described in Chapter 1), or sends a thread message to the FQP server, which in turn instantiates multiple working threads if necessary.

**Internal DIMS Database Manager**

From the DIMS implementation point of view, it is highly desirable to utilize a good commercial existing DBMS in order to avoid the construction of a complete data management system platform from scratch (namely, the reinvention of the wheel). The DBMS used as the “construction ground” for the development of the DIMS is Oracle version 7.3. In the context of the general DIMS three-tier architecture, this component represents the server tier that provides the fundamental functionalities expected from a database management system including: transactions management, data storage and retrieval facility, stored procedures management, SQL support, interface libraries for front-end development environments, database triggers, etc. These functionalities provided by the Oracle Server are exploited by the DIMS component.

The Oracle Server is used by the DIMS internal components through a specific set of access mechanisms including [100]:

- ODBC driver. The facilities of the Microsoft ODBC driver for Oracle are used

![Diagram](image-url)  

**Figure 4.3:** Example of DIMS server multi-threading capabilities.
in most DIMS data access scenarios.

- Oracle Call Interface (OCI). OCI is a specific Oracle interface that allows developer users to directly embed Oracle calls into high-level applications. It provides a lower-level program call interface to an Oracle database.

- Oracle stored procedures and packages management. A stored procedure or function is an object that logically groups together a set of SQL (and PL/SQL) programming language statements to perform a specific task. Procedures and functions are stored in a database for continued use. Furthermore, a package is a group of related procedures and functions that are stored in the database for further use as a unit.

- Oracle embedded PL/SQL. PL/SQL is the Oracle procedural language extension of SQL. PL/SQL combines the simplicity and flexibility of SQL with the procedural functionality of a structured programming language, including for instance IF-THEN-ELSE, WHILE, and LOOP language instructions. It is also possible to embed PL/SQL blocks directly in the applications.

- Specific Oracle tools. For instance, the Oracle Data Loader functions were used by the DIMS server in order to load ASCII files containing the VE topology and configuration information.

In general, all of the above means to access the Oracle server were used to some extent for the DIMS implementation (see Figure 4.4). For every specific implementation case, the possibility to apply a specific mechanism was evaluated and the best choice was selected. For example, in some cases the use of ODBC interface (through Microsoft ODBC database classes) facilitated the implementation of the DIMS functions, but in some cases, a lower-level and more flexible mechanism such as the OCI was required to develop the required DIMS functionality. All the mechanisms were used at some point for some functions in the DIMS implementation.

It is also important to mention that all the low-level details to access these Oracle-specific tools remain hidden from the VCL components that use the functionality.
provided by DIMS, and therefore they do not access the Oracle server directly. The VCL modules do not need to know which internal DBMS is used for the DIMS kernel.

**DIMS Kernel Configurator**

The DIMS Kernel Configurator is an interface application that allows the user to specify configuration of certain DIMS operation parameters. The main parameters include:

- **DIMS users and access security definitions (accounts and passwords).** The local users of the enterprise that will have access to the DIMS applications need to be identified and access security needs to be reinforced. These users will be the general VCL users.

- **Communication port number of DIMS server.** The port number used by the DIMS service interface library must be specified (the details of the service interface mechanism will be described in Section 4.6).

- **Timeout duration for distributed queries.** When a distributed query is sent to a given node, there must be a timeout mechanism to decide how long the query processor must wait for the answer to arrive from the remote node.

In this section, the internal components of the DIMS have been briefly introduced in order to provide a global view of the system architecture. In the next sections of this chapter, the main DIMS components are individually described in more details.

### 4.3 Design of the DIMS VCL Integrated Schema

The analysis of the wide variety of shared and exchanged information among VE member nodes in order to achieve a comprehensive description and classification of this information, represents one of the main tasks tackled within the design of the DIMS. As described in Chapter 3, a step-wise approach has been followed for this analysis, including: (1) the definition of several distinct focus areas, (2) the categorization of the information according to different enterprise data exchange scenarios, and (3) the division of the enterprise information into three categories (Self, Acquaintance, and Virtual Enterprise) with corresponding public, restricted and private subdivisions.

Based on this extensive analysis of the information modeling requirements for the DIMS, it is possible to identify:

1. Which part of the information is generated, stored and maintained in one enterprise, and thus becomes local (i.e. partially the Self and partially the local part of the Virtual Enterprise information).

2. Which part of the information needs to be accessed from the other enterprises, and thus needs to be imported from other enterprises (i.e. partially the Acquaintance and partially some other enterprises’ information that is related to the Virtual Enterprise).
3. Which part of the local enterprise information needs to be shared with other enterprises, that thus needs to be exported to other enterprises (partially the Self information and partially some local part of the Virtual Enterprise information). However, the visibility levels of the enterprise information from external nodes must be carefully determined by every node in order to ensure its own autonomy and information privacy.

4. Since every enterprise needs to have access to both its local information and the information imported from other enterprises, thus the two parts of information need to be integrated into one coherent schema for the sake of enterprise's convenience of access and retrieval of distributed information. Then, this integrated information includes also part of the VE-Self and VE-Acquaintance information at this node.

Therefore, in order to support all the specified information management requirements, and the proper interoperation among VCLs in different nodes, a federated database architecture is designed for the DIMS component [71, 11, 65, 74]. The DIMS federated architecture approach has proven to adequately facilitate and support the sharing and exchange of distributed information between enterprises in VEs, while providing the necessary information visibility levels to ensure their own autonomy and information privacy. The design of the DIMS federated architecture has its roots in the PEEER federated database system, and it has been properly extended and adapted to cope with the specificities of the VE paradigm, as described in Chapter 3 (see also [169, 16]).

4.3.1 General DIMS Federated Approach

The design of DIMS is based on the definition of the VCL integrated schema, described in the next section, that is represented and handled in all nodes. Data can be imported/exported through this VCL schema, but the proper access rights are defined locally at every enterprise to precisely specify the rights of external nodes. Therefore, although the sharable data at every node can be accessed through the same schema structure representing the VCL information (as illustrated in Figure 4.5), DIMS properly preserves the federated information access and visibility constraints by means of well-determined export schema definitions.

The export schemas are specified for every other VE partner for every VE in which this node is involved. This addresses the fact that depending on the VE, an enterprise may play a different role and therefore may have a different access level in each VE. In other words, the information stored in a node about a particular VE is not made available with the same visibility level to all VE members (for instance, in one VE, the visibility can be defined differently for every node).

The concepts of roles, visibility levels, and access rights among VE partners will be addressed in details in Section 4.4. But, what is important to notice here is that regardless of these access rights considerations, the VCL schema itself remains the same along the network of cooperating nodes. As a result of this design decision, the nodes do not need to physically import the schema descriptions from each other,
in order to identify the structure of the data available in other nodes. Consequently, this design decision has simplified the adoption of the actual VCL platform and DIMS system for the involved enterprises.

4.3.2 The DIMS VCL Integrated Database Schema

The federated architecture of the DIMS handles the local, import, export and integrated schemas for the enterprise, while supporting the expected data location transparency for the user, the site autonomy, and the access security (associated to distributed query processing) among other requirements. As mentioned before, every enterprise needs to have access to both its local information and the information imported from other enterprises, and thus the two parts of information need to be integrated into one coherent schema in order to provide enterprises with seamless access to distributed VE information. Then clearly, always a part of this integrated schema represents the local schema (of which a part is also exported), and the other part represents some imported schemas at the node.

Based on the initial analysis and identification of the wide variety of information
described in the previous section, the **integrated object-oriented database schema** for enterprise information has been designed, and is depicted in Figure 4.5. This integrated object-oriented database schema represents a unified inheritance hierarchy (subtype/supertype) with the corresponding association relationships among different type (class) definitions. Here, the DIMS schema diagram is described in the style of the Object Modeling Technique (OMT) notation [139]. The notation used for this diagram is depicted at the bottom corner of Figure 4.5. Please notice that for simplicity reasons, this figure represents only partial high-level categories of the actual information handled within the VCL. This integrated schema has been developed taking as a reference the VE cooperation layer architecture of PRODNET (see Chapter 1).

The class that represents the integrating focal point in this schema is the “VCL class” (appearing in the middle of this schema in Figure 4.5). The comprehensive VCL information is basically composed of three major parts:

- **VE-related information** (represented at the bottom right side of this schema diagram)
- **Local VCL components’ information** (represented on the top left side of the diagram)
- **Directory information** (represented at the top right side of this schema diagram).

These main clusters of classes are briefly described below.

**Local VCL components information**

All the information that different components of the VCL need to handle in order to function properly and perform their activities is embodied in the VCL_Component class. These components consist of, for instance, the DBPMS, the LCM, the EDL_Mngr, STEP_Mngr, PCI_Mngr, the Human_Interface, and the Local_Configurator module. Every one of these VCL components is represented as one “class” in Figure 4.5, where clearly, the class for each particular component is defined as a specialization of the VCL_Component class.

**VE-related information**

As depicted in Figure 4.5, a given VCL node can be involved in any number of different VEs. The VE information (VE class) encompasses the information about the structure for these involvements, such as their members' composition (i.e. VE topology), identification of the VE coordinator, etc. as its attributes. Furthermore, it covers the detailed information about the partners of the VEs. The VE_Partner class represents the concept of the VE-member. A VE-member can be the enterprise itself (Self_Partner) or can be another partner (Other_Partner). As can be seen in the figure, every instance of class VE is associated with one instance of Self_Partner and with one or more instances of Other_Partner.
Furthermore, within each VE, for the enterprise itself (Self.Partner), a set of local enterprise information (e.g. ERP/PPC information) is managed and stored (Partner_Info) at the enterprise's VCL. This information encompasses different kinds of data at the enterprise that can be potentially shared with the other partners. The local information contains the ERP/PPC information, EDI-related data, STEP-related and DBPMS-related data. All this information is kept locally for every enterprise.

The information of other partners is captured through the association defined between the Other.Partner class and Partner_Info (has.partner.info association). Please notice that this information is not physically stored locally at this node for the other partners, rather this information is distributed over the corresponding other-partners' nodes. In other words, this part of information will be imported from other partners when there is a federated query at this node that needs to access it and if the node is authorized to access it. Now, from the point of view of the users and the applications issuing such queries, this association between Other.Partner and Partner_Info is treated as any other local relationship that is also represented in the integrated schema. This means that the DIMS provides transparency on data location, that supports the queries on other nodes' local information through the distributed/federated query processing on the integrated schema at every node. It is important to notice that with the design of DIMS integrated schema structure for the nodes, the schema for this node's Partner_Info and other nodes’ Partner_Info is the same. This design reflects the fact that the schema for what a company makes available to others, is in turn the same as the schema of the information that is imported from those companies.

The Other.Partner class is also associated with the Partner.Export.Schema. This class provides the facility to make the Partner_Info partially available to be exported to other partners. The Partner.Export.Schema class is introduced as a mechanism to support these different levels of visibility to local information for every other partner in the VE. Through this concept an export schema for every other partner is defined. The details of the export schema definition mechanisms are given later in this chapter.

Finally, there is some information that needs to be kept for all VE.Partners within one VE to facilitate the communication and interactions with them. This information corresponds to the VE-partners' VCL.Config class. For every VE-partner, a description of its VCL configuration is needed, in order to know its specific "VE platform-related" capabilities, and to determine the way in which the communications between the nodes can be held. For instance, the communication configuration establishes the means by which a node can be addressed (e.g. Internet, ftp, email) and the corresponding addresses. The VCL configuration also describes whether the node uses a STEP module, for example. For efficiency and reliability reasons, this information is kept locally at every node. Then, when it changes at a given node, the update-and-notification mechanism must propagate this information to others in order to inform the other nodes about these changes.

**Directory Information**

The descriptive information that every enterprise is willing to make publicly available, regardless of cooperating with them or not, is represented by the directory informa-
tion (Directory_Info) in VCL schema for both the enterprise itself (Self_Dir_Info) and for other enterprises (Others_Dir_Info), in which this enterprise is interested. The directory information can include a general description of the products, services, and activities of the company.

In terms of other research related to the VCL integrated schema, a comprehensive analysis of other approaches to integrate VE distributed information was presented in Chapter 2. In particular, the DIMS approach was also evaluated in that chapter against other VE information management infrastructures. Furthermore, other generic approaches of integrated schema management in the context of federated databases can be found in [63], where the emphasis is on a federated database approach with one integrating schema that uses an ODMG interface to different DBMSs. Also, in [138] and [3], other techniques are presented to define user-specific ‘views’ for supporting authorized database access based on the underlying database schemas. However, in contrast to the DIMS architecture, these generic approaches do not aim at the specific support for the VEs and their requirements. For instance, there is no notion of enterprise roles and visibility levels supported by hierarchies of export schemas, as will be explained in more details in the following section.

4.4 Federated Export Schema Management

This section presents how specific access rights on local information can be defined at each enterprise node through export schema definitions for every other VE partner. These export schema definitions are created and maintained by the DIMS Export Schema Manager Tool (ESMT). However, before addressing the detailed design and functionality of the ESMT, the underlying concepts of roles and visibility levels in VEs are introduced next.

4.4.1 The Concept of VE Partner Roles

As mentioned in Section 4.3.1, not all members of a VE need or can request and acquire the same access level to the local information of other enterprises. This is partly due to the fact that enterprises play different partner roles in different VEs. For instance, two given regular partners involved in the VE production chain may only require to exchange data about product descriptions, while the VE coordinator may need to have privileged access to more detailed information about internal production processes at several individual partner sites in order to monitor the progress of global VE activities. Furthermore, it cannot be denied that the level of trust among competitor enterprises in a VE is limited (even when they are collaborating towards a given goal), due to the risk of involuntarily disclosing sensitive know-how information to rival companies (see also [86, 153]). In addition, the access rights on local information may be determined by legal contracts or agreements among VE partners, which also need to be considered when deciding what information to make available and under which circumstances. Therefore, based on certain parameters or attributes, every enterprise needs to rely on a mechanism to precisely define the specific access
rights and visibility levels on its local information for every other VE partner. In the approach presented in this thesis, the varied parameters that will determine the level of access rights on local information for other VE partners, are merged into one main attribute called “Role”.

In related literature, different concepts of role in the context of VEs have been introduced. For instance, a role can be defined as an “access control group” [58], as a “named set of privileges” [118], or as a “task-oriented relationship” [80]. In this dissertation, a role is defined for every enterprise as a category (or class) that is associated to VE partners, depending on a combination of the determining characteristics mentioned above. Roles are used here in order to facilitate the proper definition of access rights of other VE partners on local information of an enterprise [65].

Furthermore, the notion of role usually implies some context that explains how roles interact and that determines certain relationships among them [107]. Based on a similar idea, an extended concept of role is handled in this chapter, where a complete hierarchy of roles is defined, representing the relationships among enterprises such as the coordinator, supervisor (subordinated to coordinator but enabled to monitor certain VE activities), and regular partners. The advantage of defining this hierarchy of roles is that then it is possible to associate to every “different category of functions” (i.e. “roles”) an individual definition of the access rights. Thus, the partner role concept can also be used to determine the position of each partner in a hierarchy of access rights definitions. In this way, both the description of the VE in terms of roles and associated privileges and its relation with the export schemas, remain relatively fixed as enterprises join/leave the VE or change the role within the VE. Also in [151] the importance of a flexible and dynamic support of roles in a shared environment is described, in relation to shared-space applications and programming environments.

Figure 4.6 shows two example role hierarchies that can be defined for a VE. In both cases, we assume that in the VE there are certain well-defined roles, such as the coordinator, supervisor, partner, client and supplier. In general, a role definition can be either based on (a restriction of) another role, or stand-alone. For instance in case (B) in Figure 4.6, the functions/responsibilities of the Supervisor are modeled as a direct restriction of the functions of the Coordinator, while in case (A) the functions/responsibilities of Supervisor and Coordinator are independent of each other. As such, in case (B), the access rights of the Supervisor role, are a restriction of the access rights of the Coordinator, which is the immediate predecessor in the hierarchy. Therefore, the export schema of the Supervisor is also a restriction of the export schema of the Coordinator as will be explained in the next section.

It is important to mention that the role assumed by an enterprise is established during the initialization/creation stage of the VE. Thus, according to the VE initialization/creation model described in [39], once the VE partners are identified and selected after some negotiation process, the VE structure (topology) is defined, as well as the role that each partner will play in the VE. This VE topology and role information is distributed among the VE members and the generated VE configuration is loaded at every partner site. Please notice that before this step, formal contracts need to be signed with individual partners, in order to formalize their agreement and determine their rights and obligations in the context of the new collaboration. Con-
tract terms can then be defined in terms of "supervision clauses" that can for instance enforce the rights of the VE coordinator in order to monitor the progress of a local production order at individual partner sites. The supervision clauses can be defined by the VE coordinator, based on commonly agreed VE contracts, and distributed to every VE partner [99, 152]. Each individual partner can in turn load this information, validate it, and then define the proper information access rights and visibility levels for the VE coordinator and other VE partners as follows.

### 4.4.2 Visibility Levels in VEs

As mentioned in the previous section, the level of visibility of each other VE partner on local information must be determined individually at each enterprise, based on the role and associated characteristics of the other partner enterprises. In other words, every node in the federation must decide what part of its local information to make available to every other partner within every particular VE in which the enterprise is involved. Therefore, the level of visibility and access rights that other partners have on the local VCL schema of an enterprise must be clearly determined [65].

In order to accomplish this objective, a federated database approach can be used in which every node can protect its autonomy and privacy by defining one detailed *individual export schema* based on its VCL local schema, for every other node with which it shares information. Every partner export schema represents a partner *visibility level* that defines the proper access rights of another node to this node's local data. For every class defined in the local schema, the individual export schema determines which instances (i.e. horizontal class partitioning) and which attributes (i.e. vertical class partitioning) of those instances will be made available to every other node (see also [149, 123]). Through this mechanism, different degrees of granularity can be preserved, ranging from a small atomic value of one specific attribute of a determined class instance, to the complete VCL information being accessible from another node.

In the DIMS approach, the basic idea of defining individual export schemas on the local schema for every external "user", has been extended and generalized to the definition of a *hierarchy of export schemas* (i.e. a *hierarchy of visibility levels*) [71, 10]. This hierarchy allows the grouping and classification of common export schema characteristics based on the roles of enterprises in a VE, facilitating the task of individual

![Figure 4.6: Examples of VE role hierarchies.](image-url)
export schema definitions.

To explain a situation where this hierarchy becomes highly valuable and convenient for the VE creation and configuration tasks, let us assume that for a specific VE there are three different kinds of roles that a given enterprise can play: the coordinator, supervisor, and regular VE partner. Clearly, for every role, different information items must be made accessible from other nodes. For example, the VE coordinator needs to know some information, which is actually concealed to other VE partners. The support for a fine-grained visibility level mechanism is required in this case.

By using the described concept of hierarchical export schema definition, different visibility levels can be properly defined for every partner based on their role in the VE. Figure 4.7 represents an example of a general export schema hierarchy definition. At the first level of visibility it is needed to extract the data that corresponds to every given VE from the local VCL schema. Namely, a horizontal partitioning of information that chooses all objects related to one VE (e.g. VE-j in the figure) is achieved. Further, a second level of visibility is defined for every VE export schema, through a vertical partitioning that supports the proper information access rights for the different partner roles, i.e. coordinator, supervisor, and regular partner. For instance, the VE coordinator will obviously have more access visibility to other partners’ data than a simple regular VE partner due to its inherent control, monitoring, and possibly auditing responsibilities. Similarly, a supervisor in charge of the proper accomplishment of a specific subtask inside a VE, will have more access visibility to partners’ information than other regular VE partners, but certainly more limited than the coordinator. However, all these three kinds of export schemas are defined on top of the initial export schema (for a VE) as the base. Furthermore, at the third level of visibility, the definition of the export schema, even at the level of partners of the same VE, can still be different for every particular partner (e.g. for P₁, P₂, and Pₙ in Figure 4.7) since a node needs to exchange different information with every other VE member.

Clearly, when further layers for visibility levels become necessary to define, the export schema hierarchy can be extended and modified.

Each partner export schema is actually composed by a set of individual export definitions at the level of VCL schema classes, as will be described in the next section. Therefore, the partner export schemas correspond with export schema sets that are specified for each role. Through the external schema sets, the proper visibility levels

![Figure 4.7: Example of Export Schema hierarchy definition on VCL information.](image-url)
for the partners on the local schema of the enterprise are specified and maintained. Any change applied to an export schema will be perceived by the involved enterprises due to the reduction and/or extension of the visibility level when accessing the corresponding remote data. Depending on the trust or contractual relationship among enterprises, an update notification mechanism for this point could be implemented if required. For instance, a VE coordinator may want to be notified when a VE partner makes a change in the data set definitions for the internal production information that is exported to the coordinator. This notification could be automatically done by the DIMS component.

Finally, please notice that the export schema definitions based on the VCL schema in Figure 4.5, represent the derivations from that VCL's local schema in the federated database architecture of the DIMS. Also remember that no schemas need to be “physically imported” due to the fact that the schema structure is the same for both the exported and imported information at every VCL. In other words, as was explained in Chapter 3, the “importation” of the export schemas is automatically done by DIMS since there is a common unified integrated schema among all VCL nodes. Therefore, there is no need to explicitly handle import schema definitions that would be mapped into the integrated schema. Nevertheless, this does not mean that all the “information” is available to every given node, since the visibility levels carefully defined through their export schema definitions.

### 4.4.3 DIMS Federated Export Schema Manager

In order to properly manage the application of the hierarchy of export schemas described in the previous section, and to assist the end user with the sequence of steps for their definition, an “Export Schema Manager Tool” (ESMT) has been developed as part of the DIMS [11]. The ESMT is used for instance, to create the hierarchy of roles and the hierarchy of export schemas at each VE node. The ESMT will ensure that the export schema hierarchy remains consistent, and that the export schema for every VE partner is properly created and defined. Clearly, simple information access scenarios, which can also occur, are fully supported. For instance, even if no hierarchy is required and only basic independent export schemas for every partner are needed, the ESMT can support this functionality operating on the described schema in the same way. In this simplest case, independent roles can also be associated with the export schema sets; no further composition is required here.

The ESMT offers a graphical user interface which makes it a comprehensive, easy-to-use, and friendly tool, that is provided as a part of the VCL human interface. Therefore, the ESMT highlyfacilitates the definition of fine-grained access levels at each node for every VE partner.

Please notice that by using the ESMT to support the scenarios described in the previous section, proper access rights among enterprises can be defined in an easy and systematic way, and therefore, the necessary trust that enterprises need to participate in this kind of intricate collaborative environments is not only supported but also reinforced. This trust reinforcement is a key issue to guarantee the success of the VE.
**Internal Representation of Export Schemas**

The database definition in Figure 4.8 is designed to store the information that is handled by the ESMT of the DIMS. The database structures for export schema representation properly support the recursive definition of elements, as required by the export schema hierarchy.

Please notice that according to this figure, every Partner is associated with a Role. Every Role has as attributes: the role identifier, the general type of the role (e.g. coordinator, supervisor, regular), the name, and eventually the identification of the parent of the role in the hierarchy.

As mentioned in the previous section, the access rights are expressed in terms of the role that enterprises play within the VE. Thus, for every VE partner role, an external schema set (Export_Set) is defined, which at the end corresponds with the partner export schema. Through the Export_Set, the proper visibility levels for the partners on the local schema of the enterprise are specified. An Export_Set can be either a single or a dependent export set depending if they are based on other export sets or not. With this approach, the support for the general export schemas definition is provided, where no pre-defined export schema definitions are strictly required at the level of for instance, coordinators, supervisors, and partners, but rather any kind of other hierarchies can be defined and supported as necessary.

Furthermore, an Export_Set consists of a set of schemas, which in turn can be single schemas (EXP) or dependent schemas (Dependent-EXP) following this definition strategy. Every EXP schema has as attributes: the identification of the physical schema definition for the exact table (e.g. an Oracle view name), a list containing the attributes selected from that table (selectSpec), a condition on the specified attributes (whereSpec), and the original table on which the schema is defined (fromSpec). These three strings are used to generate the SQL statement, used to define the physical view in the local database system (i.e. Oracle). If a schema is a Dependent-EXP schema, then the fromSpec refers automatically to the corresponding EXP schema.

In order to illustrate the use of these schema definitions, some specific examples of instance diagrams for the schema classes are presented in Figure 4.9. Two database

![Diagram](https://example.com/diagram.png)

**Figure 4.8: Schema definitions for partner export schema management.**
tables (Order and Product) are shown, first with single EXP schemas \( (tv1, tv2) \) for each table, containing only the information about orders and products related to the VE identified by "ve12". The single EXP schema construction establishes the first visibility level, to filter only this information. Then, these schemas are associated with a single export schema set instance called "pv1", which represents the general export schema for "ve12". Based on this VE export schema, a VE coordinator export schema is defined. To do this, two dependent table schemas are defined \( (tv3, tv4) \), and associated to a dependent export schema set \( pv2 \). Please notice that the definition of the dependent table schemas hide those fields that were present in the previous visibility level, but that for example are not made available to the VE coordinator.

Another example is provided in the same figure, and in this case first a general partner export schema \( (tv5, tv6) \) is defined based on the general VE export schema \( (tv1, tv2) \). In this case, the fields which can be made accessible to any given partner

![Diagram of view hierarchy for regular VE partners](image-url)
are provided (in this example the order price information is removed in \textit{tv5} and \textit{tv6} definitions). On top of this generic partner export schema, a specific partner export schema is defined for Partner 1 in \textit{"vel12"}. For Partner 1, only the information related to the partner (client\text{Id} = 1) is made accessible through \textit{tv7} and \textit{tv8}. The specific partner export schema definitions provide examples of how the export schema hierarchy can be defined to support a flexible visibility level definition approach for every partner, based on its function in the VE, e.g. coordinator, regular partner, supervisor.

**Definition of VE Export Schemas using the ESMT**

In this section, a specific methodology for the definition of the VE Export Schemas using the Export Schema Manager Tool (ESMT) is presented. As mentioned in the previous section, the Export Schema, that will define visibility levels and access rights to other nodes' information, consists internally of a set of export table schemas (EXP) and dependent export table schemas (Dependent-EXP). The EXP refers to an export schema defined on one database table, and the Dependent-EXP is a schema defined on an EXP. Then, the Export Schema is assigned to a role, and every partner will be associated with one of the roles that have been defined.

The ESMT was developed for DIMS to support the definition and creation of the Export Schemas as part of the local information that is shared with other enterprises. It helps the user of VCL to define and create the export schemas, during the configuration phase of the VE. Also, it can be used during the execution phase of the VE, as access rights need to be modified due to for instance, agreement modifications or when defining access rights to new partners that have just joined the VE.

The main window of the ESMT interface contains a menu bar that enables the user to perform different operations, such as the definition of different schemas (see Figure 4.10). These operations will be illustrated below.

The VE Export Schemas and their relation with every node in the VE need to be defined in several steps. These specific steps and the actions associated to each of them using the ESMT are described next:

![Figure 4.10: Export Schema Manager interface.](attachment:image.png)
1. Define the Role Schema hierarchy. A complete hierarchy of the roles that every node can play in a VE must be created (as described in Section 4.4.1), including for instance, the three basic kinds of roles: coordinator, supervisor, and regular partner. The Create Role Schema option in the main ESMT window allows the user to create this role hierarchy schema, by specifying the type of role, the name that will identify it and its parent role in the hierarchy. As mentioned in Section 4.4.1, this hierarchy of roles can be automatically distributed by the VE coordinator and loaded by each partner during the VE configuration phase.

2. Create an EXP schema for every database table (Figure 4.11). The EXPs determine which instances (i.e. horizontal partitioning) will be available, for instance, to the Coordinator node. In this case, it is also necessary to define which database tables will be accessible by another node. In the ESMT, the Create EXP window supports the creation of the EXPs, and the user has to enter a unique name for the EXP name (these identifiers could be generated automatically). Then, the attributes that define the EXP need to be specified, i.e. the original table on which the schema is based on, the list of attributes selected from the table, and the conditions on the specified attributes. In principle, default EXPs could be created automatically for the whole VCL schema (e.g. an EXP could be created for all tables with no selected instances by default).

3. Define zero or more Dependent-EXPs for every EXP schema defined above. The Dependent-EXPs determine which attributes will be available for instance, to the nodes in the Supervisor level. The Create Dependent-EXP window supports the creation of the Dependent-EXPs when necessary. This operation is equivalent to the previous one, except that the base is not a table, but an EXP.

![Create EXP window](image)

Figure 4.11: Create enterprise EXP schema.
4. Define zero or more Dependent-EXPs for every Dependent-EXP defined above. This operation is equivalent to the previous one, except that the reference is not an EXP, but another Dependent-EXP. These schemas determine which attributes will be made available for instance, to the nodes in the Enterprise level.

5. Define the EXP/Dependent-EXP Set (Figure 4.12). This set groups the EXP and Dependent-EXPs that will specify the proper visibility levels for the different kind of roles. The Create EXP/Dependent-EXP set window supports the creation of the export schema set. An EXP/Dependent-EXP hierarchy (which represents the specific instance diagram for the export schema hierarchy) is displayed in order to show how EXP schemas are defined for each database table, and how the Dependent-EXPs are based on the EXPs. Also, the EXP/Dependent EXP Set is displayed showing how the set is taking form according to what the user has selected or removed from the hierarchy. Finally, the SQL description of the chosen database-table, EXP schema or Dependent-EXP schema is also displayed on screen.

6. Create the association Role/Export Schema. Each role in the hierarchy of roles (defined in Step 1) is associated with one EXP/Dependent-EXP Set (defined in Step 5), so that every defined role is able to access only predefined information. The advantage of associating every role with a EXP/Dependent-EXP Set is that then it is easier to define different visibility levels for every actual VE partner.

7. Assign an Export Schema to an Enterprise. This step associates an enterprise with one of the roles of the Role Schema hierarchy defined above, so that every

Figure 4.12: Create EXP/Dependent-EXP set.
node will be able to access specific information, according to the role that it plays in the VE. The Create Enterprise EXP Schema window supports the definition of the export schema for an enterprise. It allows the association of a specific enterprise (Enterprise ID) with a specific Role (Enterprise Role ID).

After having followed this sequence of steps accordingly for each VE partner, the access rights on local information would be properly defined at a given enterprise for all the other VE partners. Please also notice that since these steps are carried out independently and asynchronously at different VE enterprise nodes, there may be a need to notify other partners when the export schemas have been properly defined (although this is not needed if there is a fixed deadline agreed for this process among the VE members). If necessary, this “notification mechanism” could be done automatically by the DIMS as described in Section 4.4.2. Then, the end users or applications at different VE enterprise nodes can issue distributed queries on the VCL integrated schema described in Section 4.3.2, and the access rights defined at every node will be respected. The evaluation of the distributed queries is accomplished by the Federated Query Processing mechanism, described in the next section.

4.5 Federated Query Processing

The main objective of the federated query processing (FQP) in DIMS is to support seamless access to protected information that may be physically distributed over different nodes of the VE network [74]. The FQP component enables user applications (for instance, the DBPMS module), to query some proprietary VE-related information from other enterprises (for which the application is authorized), while hiding the physical data location details. Please notice that the federated queries are issued on the DIMS integrated VCL schema (described in Section 4.3.2), and that the authorized access on the local data is always controlled by the FQP mechanism through the export schema definitions described in Section 4.4.3. Furthermore, considering that the information stored in internal modules (e.g. ERP/PPC system) of the enterprises is constantly updated, the federated query processor in the DIMS can retrieve the most up-to-date generated data from these modules, when necessary.

A simple example of a federated query is the query issued by a human operator at an enterprise who requests some information from other VE members. Another federated query example can be related to the advanced VE coordination module, that requires to access information from many enterprises. For instance, the coordinator enterprise in the VE needs to access certain information (spread over many nodes), so that it can monitor the status and performance of VE nodes. Thus, the coordinator enterprise (represented by for instance the DBPMS coordinator module) asks its own DIMS for this information. For this purpose, the DIMS provides a set of high-level functions based on the VCL interface standard for module interoperation, so that the enterprise DBPMS does not need to worry about low-level database access mechanism/language details and/or the location of the data required to be accessed.

In order to explain the internal mechanisms and functionality of the FQP component of DIMS, it is necessary to first classify different kinds of queries that it processes.
In the most general case, two kinds of federated queries may arrive at the DIMS of an enterprise, i.e. an internal query: a query arriving from the PPC or other VCL modules in the node, such as DBPMS; or an external query: a query arriving from another enterprise's DIMS. Furthermore, there are two forms of internal queries that need to be handled. The internal query can be either a local query, which will be answered without acquiring any information from other enterprises, or a distributed query, which involves the retrieval of information from one or a specific set of other enterprises. In either case, the original federated query is processed against the DIMS integrated schema, through which the local and imported data is seamlessly accessed from any higher-level application interacting with the DIMS.

Please notice that in order to identify if the requested data at the query receiver node has been properly granted to the query issuer, the FQP component must work with data definitions gathered and configured through the ESMT. Namely, FQP controls the access to the local data using the export schema definitions generated by using ESMT, as described in Section 4.4.3.

### 4.5.1 Federated Query Processing Tasks

The processing of general federated queries can be summarized as follows: when the query arrives at the DIMS, it is analyzed and decomposed into a set of single-node subqueries, each of which needs to be sent to only one site (VE node) to be processed. After that, the results of the sub-queries are gathered and merged into the final result. If necessary, the FQP module interacts with the corresponding internal enterprise modules e.g. ERP/PPC, to retrieve up-to-date local production data, during this process.

As will be described later in this section, given the complexity associated with the FQP subtasks and the need to involve several VCL modules to provide the query results, the FQP process heavily relies on the workflow management features offered by the LCM module of VCL.

In order to better understand the internal operation of the FQP module in DIMS (and also its relationship with the ESMT), Figure 4.13 describes the main subtasks of the federated query processing. These subtasks are described next, starting with the query processing steps that take place at the query issuer node (Node A in Figure 4.13):

1. Query Reformulation. The FQP module provides a high-level function library that contains both general and application-dependent functions for the DIMS client applications. Besides those functions, a friendly user interface has been developed, which enables the end-user to issue certain queries by selecting menu items. In general, the query requesters do not need to worry about the low-level database access language and details, when using the DIMS high-level functions. When one of these library functions is called, the first step in the FQP is to parse it and reformulate it into an internal query format, depending on the parameters specified for every function.
4.5. Federated Query Processing

![Diagram of Federated Query Processing](image)

AC 1: Workflow activities for sending/receiving DIMS to DIMS query message
AC 2: Workflow activities for retrieving data from legacy system

Figure 4.13: Federated Query Processing subtasks and interactions.

2. Query Decomposition. In this step, the reformulated query is analyzed to determine the specific VE partners involved in the original query. This can be done by analyzing the input parameters of the DIMS access services, such as the high-level functions and generic distributed/federated queries described in Chapter 3. The incoming query is then decomposed into a set of simpler subqueries, so that each subquery involves the retrieval of data from only one VE node at a time. Namely, each subquery needs to be sent to only one corresponding partner to be processed locally at that side. Some subqueries may need to be processed locally at the node of the application that originated the federated query, or may need to be sent to other nodes. It is also important to point out again that the FQP mechanism operates on the integrated schema described in Section 4.3.2. In PRODNET, all the nodes have agreed on this common integrated schema, and as a result, it is not necessary to keep extra information about different import schema structures at each node. Therefore, there is no transformation needed to be applied on the external queries before they are sent.

3. Subquery Transmission. Here, the subqueries that need to be sent out to remote nodes (external subqueries) are properly handled and transmitted. Furthermore, the external subqueries are coded according to an internal DIMS message structure, which contains: the type of message (subquery in this case), the query id, the id of the sender node, the id of the query receiver node, and the query itself.
4. **Local Subquery Evaluation.** The local queries in DIMS are evaluated on enterprise data that is temporarily stored in the DIMS database. Depending on the information request (a query parameter) the end user application can decide if the query should be evaluated against the data that is stored in the local DIMS database at the point of evaluation, or if the up-to-date data must be pulled out from the PPC. In the first case, it is assumed that the PPC pushes the information periodically when it is necessary and that therefore, the data in the DIMS database is up to date. In the second case, the local subquery evaluation step must wait until the information is pulled from the PPC or until a time-out occurs. Please notice that the DIMS properly handles both PPC push/pull mechanisms depending on the parameters specified by the query issuer application. Different workflow plans will be executed in each case, as will be explained later in this section. Furthermore, the DIMS can also be configured to remove unused data from the local database at some point during the execution of the workflow plans. For example, as the last step of a given VCL workflow plan execution, the LCM can notify the DIMS to delete some data that the PPC may have temporarily stored for the processing of a given workflow event.

5. **Pull PPC Data.** The DIMS can communicate with the local data sources of the enterprise, through a specific application programming interface (API). The functions in the developed API are executed within a workflow plan (controlled by the LCM module) that allows the retrieval of data from the internal database system. The API functions convert the internal results into the common data format defined by the VCL interoperability approach. In this way, any internal system of an enterprise can interoperate with the VCL components. Consequently, when it is necessary to apply the "PPC pull" strategy described in the previous step, the DIMS can get the up-to-date data from enterprise's local production system through the specific API, and store it in its internal database temporarily during the processing of the query. As mentioned before, after this step the modified external subquery or local subquery can be executed on the data stored temporarily in the DIMS database.

6. **Subquery result merge.** In this step, the data results for local and/or distributed subqueries are gathered and interpreted. This step must wait for the results associated with the external queries that may have been transmitted to another VCL node. When all the expected results have arrived or a time-out event is triggered, the results are merged and returned to the originating application. The mechanism through which the data is returned, can be synchronous or asynchronous as will be detailed out later in this chapter.

Furthermore, the query processing steps that take place at the query receiver nodes (e.g. Node B in Figure 4.13) are described next:

1. **Subquery arrival.** An internal DIMS "dispatcher" thread is always active and listening for incoming queries from other DIMSs. When an external query is received, a new thread is launched in order to initiate the local query processing
at the receiver node. The arriving external query is actually contained in a DIMS message according to the format described in the subquery transmission step for the query issuer node.

2. Query rewriting against Export Schema Definitions. The correct processing and evaluation of the subquery at the receiver node is crucial from the point of view of secure data access. There is a number of steps that must be followed in order to convert the incoming subquery into a local query that can be properly evaluated. As described earlier, the VCL schema definition is the same in all nodes. Therefore, any node can issue a query against the whole VCL schema. But clearly, the access rights of every node to the data that it can actually import from a second node, are precisely specified in the export (individual) schema defined for it in the target node. Therefore, the arriving query will be evaluated against the corresponding export schema. However, before this query evaluation step, the query specification first needs to be rewritten according to the particular export schema definition. For example, the class names of the individual views that ultimately implement the export schemas, will be definitely different than the actual class names specified in the incoming subquery. In other words, since names of VCL integrated schema classes are the ones used by the incoming query, these names need to be replaced by the names specified through the corresponding local export schema. For this purpose, the FQP mechanism must have access to the information generated by the Export Schema Manager Tool. After the rewriting step, the query can be evaluated locally, and all the visibility access constraints are preserved.

3. Pull PPC data. This step is exactly the same as in the query issuer node (please see corresponding step above).

4. Subquery result transmission. Once the result of the evaluation of the external query is received, it must be transmitted back to the query issuer node as a DIMS message. The format of a DIMS query-result message contains: the type of message (query answer in this case), the query id (the same that arrived with the DIMS query message), the id of the receiver node (the query sender), the id of the query receiver node as message sender, and the result of the query itself.

The DIMS FQP also implements the mechanisms to properly handle the errors and exceptions that might eventually occur during the query processing steps. For example, the DIMS timeout mechanism properly handles PPC delays when updating data, as well as delays in other nodes when resolving an external query. In the case of a failure in a single site or the communication links, some data will be unreachable to the end user module. However, the end users should be allowed to access the partial information, which was successfully retrieved. In the DIMS federated query processing approach, the end user is informed about the incompleteness of the data and the incomplete query results are provided to him. Afterwards, the requesting module or user can issue another query to try to access the unreachable part of information again.
Other approaches to the federated query processor design and development can be found in [174] and [95]. In these works, the query processing mechanism is designed and developed to support a general multi-database architecture where there is a central interoperable layer to handle global queries, and a component layer at each database system participating in the federation. However, this general approach does not support all the VE environment requirements and peculiarities that have been described in Chapter 3.

It should be mentioned that in the PRODNET project the DIMS-to-DIMS communication involved in some of these steps is not done directly. For the sake of uniformity of interactions, and to ensure the security and authenticity of information exchange among enterprises, the DIMS-to-DIMS communication is performed via two other VCL modules, namely, the local coordinator module LCM (developed in PRODNET by the New University of Lisbon) and the communication interface module PCI (developed by the ESTEC software company). The strategy used for interactions among DIMSs is consistent with the general VCL architecture, while at the same time the PCI advanced communication functionality is exploited. As such, the inter-DIMS messages among enterprises are always embedded within the PCI messages.

Please also notice, that all the messages exchanged between VCL nodes are properly encrypted and secured by the PCI communication component of the VCL, applying algorithms such as the DES, RSA, and digital signature for message authentication (see [119] for more details). The communication protocol parameters for exchanging low-level messages are also handled by this VCL component (e.g. using TCP/IP communication parameters). The PCI is also in charge of handling very large messages, which may for example contain the results from the evaluation of a federated query. This component determines the best way to send large amounts of information along the network, embedded in VCL messages. Thus, the DIMS federated query processor exploits all these PCI features when passing the inter-DIMS messages.

4.5.2 Workflow Support for DIMS

As mentioned in the previous section, the DIMS in PRODNET does not use its own "dedicated" mechanism to communicate either with the legacy systems or with remote DIMSs. Instead, the DIMS follows the common VCL approach, that exploits the facilities of the VCL communicator (PCI) and the VCL workflow manager components (LCM). Due to the involved complexity associated with FQP, the workflow management facilities of LCM are used in order to coordinate the operations performed by multiple internal and external VCL components. To achieve this purpose in PRODNET, close inter-relations are created between the DIMS and LCM modules of VCL as described in the next section.

Workflow support for federated query processing

There are several specific workflow plans designed for DIMS in collaboration with the LCM developers to support the FQP mechanism, including plans for: (1) send-
4.5. Federated Query Processing

...ing/receiving DIMS-to-DIMS query messages; and (2) DIMS data retrieval from the enterprise’s ERP/PPC legacy system. These workflow plans are referenced as AC1 and AC2 respectively in Figure 4.13. In the first case, sending DIMS-to-DIMS query messages can be generalized as sending/receiving messages between two enterprises in the VE (e.g. EDIFACT messages). Clearly, within each enterprise the specific activities in these workflow plans can be defined and configured differently depending on the business processes and procedures applied at every enterprise. The second workflow plan is defined for data retrieval from the legacy systems. Namely, it enables the DIMS to get data from legacy systems of the enterprise.

Figure 4.14 represents an event-trace diagram that illustrates the complex sequence of actions that need to take place among VCL components during the combined execution of these FQP workflow plans. In particular, the figure represents the flow of activities involved in the processing of a DIMS federated query initiated by the DBPMS. This particular query involves an inquiry about the status of the requested orders, from the involved partners in a VE. The diagram shows the major steps in the activity flows for the part of the query processing where the coordinator node (Enterprise A) sends a subquery to one of the remote nodes (Enterprise B), and requests the most recently updated data (order status) from its PPC.

The sequences in this diagram can be summarized in the following list. First, the steps at the sender node where the query is generated (Enterprise A) include:

- DIMS processes the federated query and calls VCL.SendExternalQuery core...
service with datakey of the message $k_1$. This core service aims at sending an inter-DIMS subquery message, embedded in the PCI message format, to one remote node.

- LCM calls PCI.DeliverMessage, to send the message which is identified by DIMS to remote node Enterprise B.
- PCI requests the message content from DIMS by calling DIMS.GetPciMsg-Content service and builds the PCI message.
- PCI sends the message to the remote node and calls LCM_ServiceAnswer.
- LCM calls DIMS_ServiceAnswer to inform that the inter-DIMS message is sent.

Furthermore, at the remote node (Enterprise B) the steps include:

- PCI receives the message and puts the content of the message in DIMS by calling DIMS.PutPciMsgContent service.
- PCI asks LCM the VCL.RecognisingMessage service with the datakey of the coming message, $k_2$
- LCM interprets that the message is inter-DIMS message and calls DIMS.ReceiveMessage.
- DIMS processes the message in its database and invokes the VCL.SendInternal-DimsQuery core service to ask the PPC to update the requested data in the DIMS.
- LCM asks PPC.PutData to perform the DIMS request.
- PPC stores/updates data by using DIMS internal services and calls LCM_ServiceAnswer to inform DIMS about the updated data.
- LCM calls DIMS_ServiceAnswer so that DIMS knows that PPC finished the requested task.
- DIMS evaluates the query and embeds the result in the inter-DIMS message and calls LCM.SendDimsQueryResult core service with the datakey of the message, $k_3$. This core service aims at sending the answer of external query to the query issuer (Enterprise A).

Finally, the steps that take place at the node Enterprise A, when the query results are sent back, are:

- PCI receives the message and puts the content of message in DIMS by calling the DIMS.PutPciMsgContent service.
- PCI asks LCM VCL.RecognisingMessage service with the datakey of the message, $k_4$. 
• LCM interprets that the message is inter-DIMS message and calls DIMS-ReceiveMessage.

• DIMS parses the message in its database, processes the result and if all the results have arrived then the DIMS returns the answer.

Data Availability Notifications for the VE coordinator

As another example of workflow management applications to support DIMS-related functionalities, let us consider the case in which local systems need to inform the VE coordinator component about the availability of new specific local data. Namely, since the production-related order information within a VE is generated individually in different VE nodes depending on their own local production planning, the VE coordinator needs to be informed about the availability of such information in every VE node. Figure 4.15 represents the approach followed by DIMS, which allows the legacy systems of the VE node to inform or notify the VE coordinator about the availability of new production information.

As described in the example below, the PPC of Enterprise A stores the notification in DIMS and informs LCM (that asks PCI) to send it to the coordinator Enterprise B. Enterprise B receives the notification and the VE coordinator module (DBPMS) will pickup the notification message. After this process, the VE coordinator starts asking federated queries to the node’s DIMS in order to gather the data necessary for the supervision of the VE nodes.

The general steps of informing the VE coordinator are provided in the following list to have a better understanding of this diagram. First, let us consider the steps that take place at Enterprise A:

![Diagram of VCL Module interactions for informing VE coordinator.](image-url)
When the production information of VE order is available in PPC, PPC calls DIMS_PutDbpmsNotification.

After DIMS stores the PPC notification given in the input parameter of the DIMS_PutDbpmsNotification service, it answers this request with the datakey (k1) of the notification in the DIMS database and also the “location of the coordinator”.

If the coordinator is the local node, PPC invokes the VCL_InformLocalDbpms while passing the DIMS datakey (k1). Otherwise, it invokes the VCL_InformRemoteDbpms (in Figure 4.15, the coordinator is a remote node).

- **VCL_InformRemoteNotification**
  * LCM receives the VCL_InformRemoteDbpms core service and calls the PCI_DeliverMessage, to send the notification message to the coordinator node (Enterprise B).
  * PCI requests the message content from DIMS by calling the DIMS_GetPciMsgContent service with key k1 and builds the PCI message.
  * PCI sends the message to the remote node and calls LCM_ServiceAnswer.
  * LCM calls PPC_ServiceAnswer to inform that the notification is sent.

- **VCL_InformRemoteNotification** (not shown in the figure)
  * LCM calls DBPMS_InformNotification.
  * DBPMS asks DIMS to get the PPC notification by calling the DIMS_GetDbpmsNotification service.
  * DIMS returns the notification data structure to DBPMS.

Consequently, the corresponding steps at Enterprise B (VE Coordinator) include:

- PCI receives the message and puts the content of the message in DIMS by calling the DIMS_PutPciMsgContent service.
- PCI asks LCM the VCL_RecognisingMessage service with the datakey of the coming message, k2.
- LCM recognises the message and calls the DBPMS_InformNotification service.
- DBPMS asks DBPMS to get the notification that arrived in the PCI message by calling the DIMS_GetDbpmsNotification.
- DIMS returns the notification data structure to DBPMS.

Other scenarios regarding the application of workflow management techniques to support the DIMS information management tasks have been described in [76, 9].

Finally, it can be mentioned that in the reference VCL architecture, the LCM component actually works very closely with the DIMS component, and together they
constitute a *kernel for information handling and coordination*. In other words, the relationship between DIMS and LCM is in fact "symbiotic", in the sense that they establish a mutually beneficial cooperative relationship in order to support different VCL functionalities. Namely, the DIMS provides LCM with specific information management functions that support the proper flow of data between the VCL components involved in the execution of a given workflow plan.

### 4.5.3 FQP End-user Interface

Besides the FQP services that the DIMS offers for other VCL components and internal enterprise modules, a user interface is also developed that demonstrates the advanced functionality of FQP and how the FQP works in conjunction with the Export Schema Manager. This user-friendly interface allows the end users to form a query by selecting query parameter items from a dialog window. For example, one of the main users of the federated query processing is the person in charge of the VE coordination and monitoring. However, any other user can send different requests to the DIMS FQP component by using this interface. Figure 4.16 shows a snapshot of the FQP interface through which the end user can retrieve different data fields of specific production orders from several VE partners within a given VE.

### 4.6 DIMS Internal Kernel Implementation

This section addresses more specific design and implementation details regarding the DIMS Server Agent that was introduced in Section 4.2.2.
4.6.1 The DIMS Server Agent

The Server Agent is the gateway to the internal DIMS architecture, which encapsulates all the specific information management services for the VCL modules (here, "module" refers to the system implemented for each VCL component). The Server Agent can be seen as a bi-directional gateway, in the sense that it also provides a mechanism that allows internal DIMS components to reach the service interface of other VCL modules when required. For example, when internal DIMS components require a specific service from another VCL module, the request will be canalized through the DIMS Server Agent, so that both the in-going and out-going service request functionality between the DIMS and other VCL modules is encapsulated in the Server Agent.

To support this interoperation, a mechanism must be defined through which the DIMS services are made available to be requested by the other VCL internal modules (as well as ERP/PPC and DBPMS). The mechanism used by DIMS complies with the common mechanism that has been defined in the PRODNET project, in order to achieve the required level of interoperability among all the VCL modules. Here, we need to take into account that the VCL modules in PRODNET have been developed independently, and that different development tools and implementation languages have been adopted by different partners of the consortium. Even though all the modules have been developed for the Windows NT operating system, there is still a very wide range of possibilities for development environments and tools as represented in Figure 4.17. Since every partner has specific reasons or preferences for one or another environment, a common interoperation mechanism has been defined and agreed by the consortium. In this section, the implementation of this common interoperation mechanism.

![Diagram](image)

Figure 4.17: Diversity of implementation environments and interoperation mechanisms used by the development teams of PRODNET modules.
In order to reach the desired level of interoperability between the DIMS and the other VCL modules, both DIMS and the other VCL modules should be extended (wrapped) with some sort of common interoperation layer, through which services can be reciprocally requested and answered. This layer couples with the associated heterogeneity problems among modules that are autonomous and independently developed. There are some considerations for the design of such a layer, for instance:

- Dynamic Integration. No compilation process should be required when the DIMS is installed in a company, or even when it is coupled to other modules.

- Bi-directional interoperation. This means that the VCL modules should be able to access the DIMS services, and vice versa.

Taking these points into account, the interoperation layer is actually composed of two main parts (see Figure 4.18): the VCL Module Interoperation Layer (developed for each individual VCL module) and the DIMS Interoperation Layer. Each of these layers consists in turn of two major components: the client interface component and the interoperation server (or proxy) component. Please also notice that the DIMS Interoperation Layer actually represents a part of the DIMS Server Agent interoperation functionality described before in this section. In order to understand this model, it is important to keep in mind that the interoperation between the VCL module and the DIMS is managed by a dual client-server interaction. In other words, each interoperation layer needs to act both as client and server of the other layer simultaneously. For instance, the DIMS is able to request services from any VE Cooperation Layer Module (here called LM), via the LM client interface (which is dynamically embedded in the DIMS interoperation layer). The LM client will in turn contact the LM interoperation server that will carry out the service request (at the LM side). Similarly, the LM needs to be able to request services from the DIMS via the DIMS client, which is dynamically embedded in the LM interoperation layer. In this way, the DIMS client will in turn contact the DIMS interoperation server (proxy) that will carry out the service request, as shown in Figure 4.18.

In the case of PRODNET, all LMs and DIMS client interfaces are provided as DLLs that are linked to the corresponding main application. Each DLL supports the interface to specific services that will be implemented in the associated interoperation server. The LM client DLLs must be dynamically loaded by the DIMS in order to enable the sending of messages from DIMS to that LM; similarly, the LM must load the DIMS client DLL in order for the LM to be able to send messages to the DIMS. The DIMS and each LM also have to implement a communication mechanism in order to receive messages from other modules through the corresponding client interface. The protocol or mechanism adopted to support the communication between the client interfaces and respective interoperation servers is totally dependent of the LM itself and the PRODNET approach does not impose any constraints about this.

**General Functions for Interoperation among VCL modules**

As it was already mentioned, the integration of DIMS and LMs via interoperation layers involves a bi-directional "message exchange". Namely, the DIMS needs to request
services (and get the service result or answer) from the LMs, and conversely, the LMs need to request services (and also get the service answer) from the DIMS. Furthermore, please notice that the communication mechanism to implement the functions provided in the client DLLs to request internal services, can be implemented in either synchronous or asynchronous manner. In the synchronous approach, the requesting client application program will not proceed with its execution until the request is fully processed at the server application side, and the service answer is returned as parameters of the service request itself. The service requests can also be satisfied asynchronously, which means that the issuing application program will send the request and will be “released” to do other tasks while the requested service is being carried out at the server application side. Once the service request is accomplished at the receiver server side, the answer is sent (at some point) to the issuing application via a specific function. Both approaches are supported by the general PRODNET model and the DIMS implementation, however in this chapter the asynchronous approach is assumed and described since it is the most commonly used approach by the PRODNET VCL components.

In order to support the asynchronous approach, a pre-requisite for each LM (and the DIMS as well) is the inclusion, in the corresponding LM client interface, of a pair of services that are required for the bilateral interoperation mechanism. These interface services are included in the client LM DLLs.

The basic declaration for both of the LM interface services is as follows:

\[<LM\_ID>_ServiceRequest\] (parameters)
\[<LM\_ID>_ServiceAnswer\] (parameters)

In these declarations, \(LM\_ID\) is a unique identifier for each LM, i.e. an acronym used to uniquely represent each LM within a certain enterprise environment. The \(LM\_ID\) is just a string of characters, for instance, it may be instantiated as: “Dims”, “Lcm”, “Step”, etc. For both of the request/answer functions, the parameters comply
with a generic type definition that allows the transmission of elements of all the necessary parameter types, as will be described in the next section. Furthermore, the LM DLL is named as VCL<\texttt{LM_ID}>.dll, and will be distributed together with a header file (VCL<\texttt{LM_ID}>.h), containing all the data type and service definitions that are required in order to use the corresponding library file. Furthermore, all the VCL modules will use a common header file (VCLDefs.h), that contains the common definitions that are shared among all the VCL modules.

In order to support the DIMS service request/answer mechanisms, the DIMS client interface provides two services that comply with the same criteria established for the LM client interfaces as described above. Namely, the DIMS client interface provides the following two functions:

- DIMS_ServiceRequest (parameters)
- DIMS_ServiceAnswer (parameters)

Both services will be available to the LMs through a header file called VCLDims.h and a DLL named VCLDims.dll.

A basic interaction scenario of the general DIMS-LMs integration model using the service request/answer functions is depicted in Figure 4.19. For instance, let us suppose that an LM needs to request a DIMS service. Then the LM will asynchronously call the DIMS_ServiceRequest function of the DIMS client interface, which is distributed through the VCLDims.dll library file. After this invocation, the LM is released to continue with its regular execution, and the request is transparently transferred to the DIMS interoperation server at the DIMS interoperation layer side. Then, the DIMS Server Agent will contact the internal DIMS components that need to be activated in order to satisfy the service request. When the DIMS service request is fulfilled, the answer is sent to the LM via the <\texttt{LM_ID}>.ServiceAnswer function of the LM client interface that is included in the VCL<\texttt{LM_ID}>.dll library file. This LM client interface will in turn seamlessly contact the LM interoperation server, which will eventually trigger LM specific functionalities in order to process the service answer. It is also possible that the DIMS requests a service from another LM, in which case an equivalent sequence of steps will be followed, achieving therefore the desired full-duplex asynchronous communication between the DIMS and all other LMs.

**Parameters of Service Request/Answer Functions**

The parameters of the service request/service answer functions are specified using two main predefined types: a \textit{token} parameter type (VclTokenPtr), and a linked list of \textit{VCL parameters} of an generic VCL parameter type (VclParametersList), which supports all possible parameter types that are required to be exchanged among VCL modules. In other words, the service function declarations for DIMS (considering only the asynchronous approach) are specified as follows:

- int DIMS_ServiceRequest(VclTokenPtr token, VclParametersList inputParams);
- int DIMS_ServiceAnswer(VclTokenPtr token, VclParametersList outputParams);
The token parameter type supports the context definition for the execution of the service request, and specifies among other fields:

- A unique service request identifier (used for workflow management purposes).
- The identifier of the VCL “core service” (i.e., specific VCL workflow plan) in which context the specific service request is being issued.
- The identifier for the VE and the VE enterprise node where the service request is issued.
- The identifier of the specific “module service” that is being requested from the corresponding module.
- A token creation timestamp, among other fields.

Furthermore, the nodes of the VCL parameters list allow the specification of the actual parameters that the specific service being requested (or answered) needs. For the VCL parameter list nodes, a generic VCL data type has been defined from which a large set of specific data types can be derived and used in any module service. For instance, the union type definition in the C language depicted in Figure 4.20, illustrates some of the possible data types that have been defined for a VCL parameter list node.

Through the agreed parameter-passing mechanism, the DIMS can offer high-level services to other VCL modules. For instance, to support the VE monitoring and coordination, specific DIMS services are used in PRODNET by the DBPMS module. One such example is when the DIMS needs to receive specific information related to for instance: the requested (purchase) orders, requested order items, and internal production orders, that have been assigned to a given set of the VE partners (these
4.6. DIMS Internal Kernel Implementation

```c
union VclGenericParameter {
    VclResultCondition vclResultCondition;
    VclMessageContent vclMessageContent;
    VclLogInformation vclLogInformation;
    VclEnterpriseListNode vclEnterpriseListNode;
    PciDeliveryConditions pciDeliveryConditions;
    PciSecretKey pciSecretKey;
    PciAuditInfoList pciAuditInfoList;
    PciAuditInfoBuffer pciAuditInfoBuffer;
    DimsDataKey dimsDataKey;
    StepDataExchange stepDataExchange;
    AttachedFileNode attachedFileNode;
    DimsQuerySpec dimsQuerySpec;
    LcmMessageContext lcmMessageContext;
    DbpmsRequestedOrderTree dbpmsRequestedOrderTree;
    DbpmsRequestedOrder dbpmsRequestedOrder;
    DbpmsRequestedItem dbpmsRequestedItem;
    DbpmsProductionOrder dbpmsProductionOrder;
    DbpmsDbp dbpmsDbp;
    DbpmsDp dbpmsDp;
    DbpmsSupervisionClause dbpmsSupervisionClause;
    VclCompleteOrder vclCompleteOrder;
};
/* default */
Opaque opaque;
```

Figure 4.20: Partial definition of VCL parameter list node structure.

Data structures will be further described in Section 4.7.1 of this chapter. A partial definition for the requested order type is included in Figure 4.21.

By properly filling the VCL token and parameter structures of the DIMS service request function, the VCL modules as well as other internal enterprise systems, can use this kind of high-level functionality involving the retrieval of distributed information that is physically stored along the VE network. In PRODNET, the DIMS offered many specific services for different VCL and PPC modules involving a large variety of data types, as will be described in Section 4.6.3 of this chapter.

The detailed description of the files and type declarations for the services and data structures mentioned in this section is outside the scope of this chapter. However, they have been properly documented in [76].

Remote Procedure Call Implementation

The two DIMS DLL functions to answer/request DIMS services have been implemented using a specific Remote Procedure Call (RPC) mechanism [150]. This mechanism avoids the development of complicated low-level communication code that would be necessary in order to cope with the interoperability among DIMS and the other VCL modules.

Furthermore, the RPC mechanism clients can make a function call that is actually executed in the server, which does not need to be running in the same machine as the client. All the parameter-passing details are managed internally by the RPC. As any
typedef
{
  TimeStamp plannedStartDateOfProduction;
  TimeStamp plannedEndDateOfProduction;
  TimeStamp startDateOfProduction;
  TimeStamp endDateOfProduction;
  TimeStamp deliveryDate;
  float priceTotal;
  DbpmsRoStatus orderStatus;
  float orderProgress;
  TimeStamp lastFeedback;
  unsigned int manufacturingLeadTime;
  Identifier32 code;
  Identifier32 name;
  Identifier32 officialName;
  Identifier64 description;
  Identifier32 operationResponsibility;
  Identifier32 operationAuthority;
  ...
} DbpmsRequestedOrder;

Figure 4.21: Sample definition for DBPMS requested order structure.

regular function, the remote function may receive input/output parameters of any built-in or user-defined data type.

Therefore, the two DIMS services included in the DIMS DLL actually perform a call to a remote procedure for which the implementation resides in the DIMS server. The DIMS DLL acts in this sense as an RPC client that connects to the RPC server (in the DIMS server).

Finally, it must be mentioned that the reliability and the performance results using RPC for the DIMS implementation were satisfactory considering the PRODNET demonstration scenarios (these scenarios will be described in Section 4.7.2). In other words, the mechanism proved to be robust, efficient, and convenient to implement the PRODNET interoperation approach in the DIMS case. For more information on how to use RPC, please see for instance [150].

4.6.2 DIMS Implementation Environment and Tools

Regarding the specific implementation environment and tools that were used in the DIMS development, the system was implemented on Windows NT using Microsoft Visual C++ (MVC++), Professional Edition, version 5.0 [101, 170, 134]. The MVC++ tools used to support the DIMS implementation include: Microsoft Foundation Classes (MFC), MFC Database classes, Microsoft ODBC driver for Oracle, and RPC support tools. The Oracle tools used in the DIMS implementation include the Oracle Enterprise Manager Utilities and the Oracle Data Loader, among others [100, 106]. Finally, it can also be mentioned that the tools and mechanisms mentioned in this section were successfully used during the development phase of the DIMS, and contributed to a satisfactory level of performance and robustness of the final system.
4.6. DIMS Internal Kernel Implementation

4.6.3 Summary of DIMS Implemented Services

There is a wide variety of DIMS **high-level** services that were developed and made available for other VCL and internal enterprise modules, based on the interoperation mechanism described earlier in Section 4.6. In PRODNET, the DIMS offered a total of (roughly) fifty services for different VCL modules involving many varied parameter data types. In Table 4.1, a brief classification and summary of these DIMS services is provided.

A short description of each of these services is included in Appendix C of this thesis. Besides these high-level services, other kinds of lower-level data access services (e.g. through ODBC or Oracle package functions) could be used if necessary, in some particular scenarios. Nevertheless, the use of these low-level services is discouraged for

<table>
<thead>
<tr>
<th>1. Support services for LCM</th>
<th>4. Support services for EDI</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.1. Service DeleteDataElements</td>
<td>4.1. Services for sending/receiving orders</td>
</tr>
<tr>
<td>1.2. Service AttachFilesRef</td>
<td>4.1.1. Service PutOrder</td>
</tr>
<tr>
<td>1.3. Service GetDBP</td>
<td>4.1.2. Service PutOrderLines</td>
</tr>
<tr>
<td>2. Support services for DBPMS</td>
<td>4.2. DIMS services for EDI blob services</td>
</tr>
<tr>
<td>2.1. DIMS support for DBPMS workflow</td>
<td>4.1.4. Service GetOrderLines</td>
</tr>
<tr>
<td>2.1.1. Services for DBPMS registers requests</td>
<td>4.1.5. Service FetchOrderLine</td>
</tr>
<tr>
<td>2.1.1.1. Service GetRequestedOrderTree</td>
<td>4.2. DIMS services for EDI blob services</td>
</tr>
<tr>
<td>2.1.1.2. Service GetRequestedOrder</td>
<td>4.2.1. Service PutBlob</td>
</tr>
<tr>
<td>2.1.1.3. Service GetRequestedItem</td>
<td>4.2.2. Service GetBlob</td>
</tr>
<tr>
<td>2.1.1.4. Service GetProductionOrder</td>
<td>4.3. Services for EDI CONDRA services</td>
</tr>
<tr>
<td>2.1.2. Services for querying DBPMS registers</td>
<td>4.3.1. Service GetStepDataExchange</td>
</tr>
<tr>
<td>2.1.2.1. Service GetResultRequestedOrderTree</td>
<td>4.3.2. Service PutStepDataExchange</td>
</tr>
<tr>
<td>2.1.2.2. Service GetResultRequestedOrder</td>
<td>4.3.3. Service GetStepDataElements</td>
</tr>
<tr>
<td>2.1.2.3. Service GetResultRequestedItem</td>
<td>4.3.4. Service GetStepDataElements</td>
</tr>
<tr>
<td>2.1.2.4. Service GetResultProductionOrder</td>
<td>5. Support services for STEP</td>
</tr>
<tr>
<td>2.1.3. DIMS services for Informing DBPMS</td>
<td>5.1. Service StepDataExchange</td>
</tr>
<tr>
<td>2.1.3.1. Service GetDBPMSNotification</td>
<td>5.2. Service StepDataExchange</td>
</tr>
<tr>
<td>2.2. Services for Dbpms Supervision Clauses</td>
<td>5.3. Service PutAttachedFileNames</td>
</tr>
<tr>
<td>2.2.1. Service Put.Sup.Clause</td>
<td>5.4. Service GetAttachedFileNames</td>
</tr>
<tr>
<td>2.3. Services for getting DP information</td>
<td>6.1. Services for PCI message mgmt.</td>
</tr>
<tr>
<td>2.3.1. Service GetDPId</td>
<td>6.1.1. Service PutPciMessageContent</td>
</tr>
<tr>
<td>2.3.2. Service GetDBPMS</td>
<td>6.1.2. Service GetPciMessageContent</td>
</tr>
<tr>
<td>3. Support services for PPC</td>
<td>6.2. Services for PCI enterprise node</td>
</tr>
<tr>
<td>3.1. Services for PPC purchase orders</td>
<td>6.2.1. Service PutVEPartner</td>
</tr>
<tr>
<td>3.1.1. Service PutVcOrder</td>
<td>6.2.2. Service GetVEPartner</td>
</tr>
<tr>
<td>3.1.2. Service GetVcOrder</td>
<td>6.2.3. Service PutPrivateKey</td>
</tr>
<tr>
<td>3.2. Services for PPC generic &quot;blobs&quot;</td>
<td>6.2.4. Service GetPrivateKey</td>
</tr>
<tr>
<td>3.2.1. Service GetPFCMessageContent</td>
<td>6.2.5. Service PutPublicKey</td>
</tr>
<tr>
<td>3.2.2. Service PutPFCMessageContent</td>
<td>6.2.6. Service GetPublicKey</td>
</tr>
<tr>
<td>3.3. Services for PPC/Informing DBPMS</td>
<td>7. Services for VE configuration</td>
</tr>
<tr>
<td>3.3.1. Service PutDBPMSNotification</td>
<td>7.1. Service DIMS Load VE Configuration</td>
</tr>
<tr>
<td>3.4. Services for PPC putting DBPMS data</td>
<td></td>
</tr>
<tr>
<td>3.4.1. Service PutReqOrdTree</td>
<td></td>
</tr>
<tr>
<td>3.4.2. Service PutRequestedOrder</td>
<td></td>
</tr>
<tr>
<td>3.4.3. Service PutRequestedItem</td>
<td></td>
</tr>
<tr>
<td>3.4.4. Service PutProductionOrder</td>
<td></td>
</tr>
</tbody>
</table>

Table 4.1: List of DIMS services implemented in PRODNET.
the reasons of functional generality, data access independence, and compatibility with the workflow management approach that provides coordination among VCL modules.

For a full description of the parameters and data types associated with these services, please see the UvA Technical Report on the implementation of the PRODNET DIMS [76]. This technical report also includes the complete definitions of the DIMS internal database schema, the DIMS Oracle stored procedures and packages, the DIMS interface header files, and the DIMS end-user installation manual.

In order to show an example of the run-time interoperation of DIMS with other VCL components, and the on-line execution of some of the DIMS services listed in Table 4.1, Figure 4.22 and Figure 4.23 are provided in this section. Namely, Figure 4.22 illustrates a run-time service request call (for the PutVclOrder service) from the ERP/PPC module to the DIMS component of VCL, in order to store the information related to a purchase order that is being sent to another VE partner enterprise.

Furthermore, Figure 4.23 represents an example of the run-time module interaction between the DIMS and the STEP components. In this case, the STEP module requests the DIMS service PutStepDataExchange in order to store the STEP information that is required to send an EDI/CONDRA message to another enterprise, which needs to analyze the technical information about a given product. Please notice that these module interactions strictly follow the interoperation approach described in Section 4.6.

4.7 Application of DIMS Approach in PRODNET

In this section, the application of the main DIMS architecture components described in this paper is demonstrated in the context of a real PRODNET VE scenario. For this purpose, the main features of the VCL integrated schema, the Export Schema Manager Tool, and the Federated Query Processor will be specifically applied to support the VE coordination and monitoring tasks associated with the DBPMS module.
in PRODNET. In addition, the DIMS support for the basic VE operation and the protected exchange of distributed information among regular partners will also be demonstrated.

Therefore, Section 4.7.1 introduces the DIMS information modeling and data access requirements associated with the support for VE coordination and management of Distributed Business Processes (DBPs). Further, the application of the FQP and ESM components to support these VE coordination requirements is demonstrated in Section 4.7.2, within the context of the general PRODNET VE scenario.

Please notice that all the DIMS components and functionalities presented in this section were fully implemented within the CO-IM group of the University of Amsterdam, and were properly tested and integrated with the other components of the PRODNET architecture developed in conjunction with the other European and Latin American partners of the PRODNET consortium.

### 4.7.1 DIMS Support for VE Coordination and Monitoring

This section first introduces the specific information modeling and management requirements to support DBP management in VEs. After this analysis, the general design of the DIMS integrated schema and high-level functionalities to support these needs, are also presented.

#### Application of Distributed Business Processes for VE Coordination

The monitoring and coordination of the VE tasks and activities is a crucial issue during the operation phase of the VE life cycle. Given the fact that the VE establishes

![Image](image_url)

**Figure 4.23:** Example of run-time interoperation between STEP and DIMS modules.
a set of global goals involving several partners, which in principle can be completely autonomous and independent, there must be a well-defined mechanism to coordinate different partners activities towards these global goals. In PRODNET, the DBPMS module, strongly supported by the DIMS and the LCM modules, accomplishes this VE coordination task. The DBPMS module provides a set of advanced functionalities that have been identified and developed in PRODNET to tackle some specific problems involved in the coordination of VE distributed activities [99].

The VE monitoring and coordination approach in PRODNET is based on a reference model for Distributed Business Process representation. This model has been used in order to properly represent the abstract concepts of VE, VE member, and its associated complex production chain, in such a way that the coordination and monitoring of VE tasks is adequately facilitated and supported [99]. The model is designed and developed at the Federal University of Santa Catarina in Brazil, based on some concepts and definitions specified in [20], among other sources. The complete description of the DBPMS operations and DBP data models are outside the scope and the aim of this chapter. However, in the next paragraphs, those aspects of the DBPMS data model that require support by the DIMS functionality, are analyzed. In particular, the following concepts are defined for the DBPMS:

- Business Process (BP): structure that models the set or pool of VEs within a particular enterprise node. Several VEs can be managed inside a given enterprise node (one enterprise can be involved in many VEs simultaneously).

- Distributed Business Process (DBP): represents a specific VE instance, irrespectively of the particular service or products being addressed. A DBP is associated with the corresponding VE partners (Domain Processes) described below.

- Domain Process (DP): represents a specific VE member enterprise (i.e. a given VE partner). A DP is also associated with a set of zero or more requested orders e.g. specific product purchase orders.

- Requested Order (RO): represents a specific Requested Order from another VE member. A RO is composed of (a set of) Requested Items. The RO contains detailed commercial specification related to the purchase order. The purchase order could correspond to any product or service being provided by a given company.

- Requested Item (RI): represents a specific Requested Item (or line) associated with one RO. The RI contains descriptive information about the different items of the purchase order. An RI is in turn composed of a set of Production Orders.

- Production Order (PO): contains the detailed description of an internal production order to be supervised. It contains specific internal production chain information that is specific to the ERP/PPC of every company.

- VE-Intra: contains extra intra-organizational parameters used for an analysis of the progress evolution of each production order in a given VE.
In reality, the coherent support and management of the above concepts results in a flexible approach to cover the complicated distributed production chain interactions involved in manufacturing VE scenarios. Every VE coordinator node has one or more instances of the described data model. In this way, the VE coordinator, e.g. the DBPMS system, can actually ask questions about the orders and monitor their status, progress, and the other information related to other partners in a given VE.

**DBP Distributed Information Management Requirements for DIMS**

In order to properly support the distributed business process model described in the previous section, a set of distributed data structures and data access functions must be developed. It must be noticed that the DBP information needed to be handled by the VE coordinator, is actually spread along the network of the VE member enterprises. In other words, in fact the requested order information with its associated requested items, production orders and VE-intra parameters, represents a distributed "requested order tree", that is constantly updated at the local site of each VE partner fulfilling the requested order. When a given VE coordinator needs to monitor some of the information contained in this tree, it must be provided with the most up-to-date information. This means that the DIMS must implement an access mechanism in such a way that the data stored in the internal module of a remote node, can be seamlessly queried, and accessed by the coordinator node. An important issue to keep in mind in this process is that in the access to any information associated with the requested order tree needed for the advanced coordination modules, such as the DBPMS in this case, the physical location of the distributed data shall remain hidden and only known to the DIMS module.

For instance, the VE coordinator must be able to ask the following kind of queries on the DBP model involving several VE members:

- Get all the DBP identifiers, i.e. which VEs are being coordinated at this node.
- Get all the DP identifiers associated with a given DBP, i.e. which partners are involved in a given VE.
- Get all the RO identifiers for a given DBP.
- Get the RO (including status/progress information) for a given RO in a DBP.
- Get all the RO tree information for a given RO in a DBP.
- Get the most up-to-date RI information in a DBP regularly.
- Get the most up-to-date PO information in a DBP regularly.

At the same time, while the distributed queries are issued, data access rights for this kind of data retrieval must be always supported by the information management component. Please notice that as mentioned previously in Section 4.4, the VE coordinator will obviously have more access visibility to the data stored in a given enterprise, than a simple regular VE partner, due to the control, monitoring, and
possibly auditing responsibilities (and access rights) of the VE coordinator. Also, we need to remember that the visibility levels on the local enterprise information must be individually defined and configured for every other partner in every VE.

The PRODNET DIMS not only supports the transparent access to the complete DBP distributed information model, but at the same time, it supports the autonomy and independence of every involved enterprise. For example, the DIMS VCL integrated schema, previously introduced in Section 4.3.2, also considers the integration of the distributed information of the DBP, and provides the DBPMS with a unified and coherent schema that hides all the details of the physical distribution of data, when issuing the database queries. A partial and simplified description of the DIMS integrated schema designed for DBPMS support is illustrated in Figure 4.24. This part of the DBPMS integrated schema basically represents the information related to the data structures described for the DBP model in Section 4.7.1. Namely, the diagram clearly illustrates the relationship between DP and DBP, and the composition of RO, RI, PO, and intra-VE structures as a requested order tree.

Please notice that by allowing the DBPMS module (or other end-users) to execute queries and other specific high-level data access functions on this integrated schema, the ultimate goal of monitoring and coordination of distributed requested order trees in a VE is accomplished. Furthermore, the federated query mechanism will take care of retrieving the proper up-to-date data from the corresponding remote PPC system, when required. In addition, the corresponding export schemas and access rights will be defined through the ESMT and will be considered during the FQP tasks. In the next section, several example scenarios are given regarding the application of

Figure 4.24: Partial high-level DIMS integrated schema for DBPMS.
both FQP and ESMT mechanisms to support general VE operation and coordination activities. These scenarios rely on the DBP models and operations described above.

### 4.7.2 DIMS FQP/ESMT Example Scenarios

As mentioned in Chapter 3, a big live demonstration scenario was prepared for the PRODNET project and presented at the Pro-VE '99 international conference in October 1999 in Porto (Portugal) [43]. This scenario involved the functionality of all the VCL modules of PRODNET. In the case of the DIMS, several specific demonstration cases were also developed to show the main functionalities of its FQP and ESMT modules within the PRODNET context. The following sections represent some of such scenario cases. In order to describe these cases, let us first recall the main PRODNET demonstration scenario.

**PRODNET Demonstration Scenario**

Figure 4.25 represents a general overview of the PRODNET intercontinental VE scenario to be used as a reference for the DIMS demonstration cases. It presents the VE purchase orders that were requested and received on each node (see also [43]). Please remember that in this VE (corresponding to the VE2 described in Chapter 3), Enterprise 1 is a bicycle producer that requires bicycle pedals and bicycle frames to supply its internal production. Therefore, it issues a purchase order to Enterprise 2 for each kind of product. In order to attend the order for bicycle pedals, Enterprise 2 needs pedal moulds and some raw material (in this case PVC resin), which are ordered from Enterprises 4 and 3 respectively, which are located in Brazil (Enterprises 1 and 2 are located in Europe). The bicycle frames are produced and supplied by Enterprise 2 itself (for more details about this scenario, please see Chapter 3).

For the DIMS demonstration, we use the same data applied in the general PRODNET scenario. As shown in Table 4.2, Enterprise 2 locally stores the production information of bicycle frames. Enterprise 3 the production information of PVC resin, and Enterprise 4 the production information of pedal moulds. Further, let us bear in mind that Enterprise 2 is the coordinator in this scenario.

![Figure 4.25: Purchase orders flow in PRODNET demonstration scenario.](image)
DIMS Case 1 – Definition of Access Rights for VE Partners

The main goal of this case is to demonstrate the definition and creation of the export schemas at a given VE node. As explained in Section 4.4.3, this definition takes place at every enterprise and is part of the Enterprise Configuration/Reconfiguration step.

As mentioned in Section 4.4.1, it is not desirable that all the enterprises involved in a VE have the same access rights to information of a specific node. For instance, an enterprise with a role of regular partner should not have access to the internal information managed by an enterprise with coordinator role. In this specific demonstration case, Enterprise 2 will then define restrained access rights on its local information for the enterprises with a “Regular Partner” role, i.e. Enterprise 3 and Enterprise 4.

The main steps in the general process to define the access rights are illustrated in Figure 4.26 and briefly described next (please notice that these steps are consistent with the methodology described in Section 4.4.3):

1. Define the role hierarchy. For this first step, we assume that the basic Role Schema hierarchy has been already created as described in Section 4.4.1, and defined as in Figure 4.26. This role hierarchy just defines the Coordinator and the Regular Partner roles.

2. Definition of export schemas. This step includes the definition of EXPs for the database tables of the DBPMS integrated schema described in Section 4.7.1. These tables include the production orders (Dbpms.ProductionOrder), the requested orders (Dbpms-RequestedOrder) and the requested items tables (Dbpms-RequestedItem). Figure 4.26 also depicts the detailed definition of the export schema for the production order table, namely the ExpProdOrder1. The other export schemas are defined in a similar way for the other tables, as specified in the EXP hierarchy. The ExpProdOrder1 definition determines that all the attributes (‘*’) of the local Dbpms.ProductionOrder table in Enterprise 2 can be made available, for those rows associated with VE1 and regarding the information about pedals (see conditions VE = VE1, Name = ‘Pedals’ in Step 2).

3. Define the EXP Set. The third step is the creation of the export schema set. This EXP Set groups the EXPs and/or the Dependent-EXPs that will specify

<table>
<thead>
<tr>
<th>Enterprise</th>
<th>Name</th>
<th>StartDate</th>
<th>EndDate</th>
<th>Delivery</th>
<th>Requested</th>
<th>Produced</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 2</td>
<td>Frame</td>
<td>1-Apr-99</td>
<td>1-Nov-99</td>
<td>3-Nov-99</td>
<td>100</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Enterprise 2</td>
<td>Pedal</td>
<td>2-Aug-99</td>
<td>5-Nov-99</td>
<td>8-Nov-99</td>
<td>200</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>b) Enterprise 3</td>
<td>PVCResin</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>70</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>c) Enterprise 4</td>
<td>PedalMould</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 4.2: Sample of Local production information at every VE member.
the proper visibility level that Enterprise 2 wants to have for enterprises with a
given role (to be defined next).

4. Associate Exp Set with Role. In this step, the export schema set ExpSet1 is
associated with the role for Regular Partner.

5. Associate VE partner enterprise with Role. At last, it is necessary to specify
which is the role that Enterprise 3 plays in the Virtual Enterprise; in this case,
it corresponds with the Regular Partner role. The same must be specified for
Enterprise 4. In this case, both nodes have access to the same information.

Through the steps defined above, this scenario case demonstrates how the ESMT
component of the DIMS can be used at the VE setup time (initialization) to give
specific information visibility rights to other partners.

**DIMS Case 2 – FQP Reinforcement of VE-Partners Access Rights**

In this DIMS demonstration case, it will be shown how the FQP mechanism works
together with the ESMT access rights definitions specified in Case 1. Please notice
that according to the previous definitions, both Regular Partners (Enterprise 3 and
4) have exactly the same access rights on the local information of the Coordinator.
Also, notice that these rights do not grant full access to the local information, but
only to the part of it related to the pedal product in this specific VE.

In this second case, the regular partners in the VE will ask information that is
distributed over the VE network. For example, let us assume that Enterprise 3 is
expecting a delay in its local production plan, and therefore it would like to postpone its delivery date of the PVC resin. For this reason, it needs to ask Enterprise 2 for some production information about the pedal, such as the delivery date in Enterprise 2, and also needs to check it against its own production information, so that it can compare the two delivery dates. Then, it could propose a new delivery date for its local production considering the delivery date of the pedal in Enterprise B. Enterprise 4 may encounter a similar situation at some point of its mould production process, and may as well issue distributed queries involving the retrieval information from other remote nodes.

In order to validate the definition of the export schemas at Enterprise 2 for Enterprise 3 and Enterprise 4, the DIMS Browsing Interface of the Federated Query Processor can be used to execute certain specific queries on the VCL integrated schema. For example, any of the Regular Partners (e.g. Enterprise 3 and Enterprise 4) can ask the query GetProductionOrder to gather the production information about the orders in the VE. Thus, the DIMS at Enterprise 3 (or Enterprise 4), after sending the query to the other enterprises, and after collecting, processing and merging the results, will present the final results through the Browsing Interface.

In this case, Enterprise 3 (or Enterprise 4), as a Regular Partner node, will not get all the information from Enterprise 2, since this enterprise has defined the access rights in such a way that only the information of Pedals will be presented, as it is shown in Table 4.3b.

However, if the Coordinator node (i.e. Enterprise 2) asks the same query, then this node will be able to see all the local and distributed VE information, since the Coordinator does not have any restricted access, as it is shown in Table 4.3a. This result is consistent with the fact that the DBPMS module of the Coordinator enterprise may need to gather production order information from all the VE partners, which are involved in producing a VE order, in order to check if there is any delay in the delivery date.

Please notice that all the queries issued by the end users and modules are applied on the DIMS integrated schema defined for DBPMS, and that the FQP properly collects and merges the partial results of the sub-queries (sent to all partners), before returning the final result to the coordinator as described in Section 4.5.1.

<table>
<thead>
<tr>
<th>EnterprisedId</th>
<th>Name</th>
<th>StartDate</th>
<th>EndDate</th>
<th>Delivery</th>
<th>Requested</th>
<th>Produced</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 2</td>
<td>Frame</td>
<td>1-Apr-99</td>
<td>1-Nov-99</td>
<td>3-Nov-99</td>
<td>100</td>
<td>50</td>
<td>10</td>
</tr>
<tr>
<td>Enterprise 2</td>
<td>Pedal</td>
<td>2-Aug-99</td>
<td>5-Nov-99</td>
<td>8-Nov-99</td>
<td>200</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Enterprise 3</td>
<td>PVCResin</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>70</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Enterprise 4</td>
<td>PedalMould</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>EnterprisedId</th>
<th>Name</th>
<th>StartDate</th>
<th>EndDate</th>
<th>Delivery</th>
<th>Requested</th>
<th>Produced</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 2</td>
<td>Pedal</td>
<td>2-Aug-99</td>
<td>5-Nov-99</td>
<td>8-Nov-99</td>
<td>200</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Enterprise 3</td>
<td>PVCResin</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>70</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Enterprise 4</td>
<td>PedalMould</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 4.3: Final results for Demonstration Case 2.
DIMS Case 3 – Dynamic Modification of Access Rights for VE Partners

The aim of this scenario case is to show how the access rights for an enterprise can be changed at any time during the VE operation, due to for instance, changes in bilateral agreements or contracts among other reasons.

Following an approach similar to the export schema definition process illustrated in Case 1, Enterprise 2 can later limit the access rights for any other enterprise. For example, let us assume that now Enterprise 2 does not want to allow Enterprise 3 to see all the information about pedal, but only some part, since Enterprise 3 does not need to know the content of some fields related to specific internal production processes. For this reason, Enterprise 2 will make changes in the information visibility rights defined for Enterprise 3 by modifying the Export Schema associated to Enterprise 3. Please notice that the changes on the visibility levels are only for Enterprise 3 but not for the other enterprises with a Regular Partner role (e.g. Enterprise 4 will keep the same access rights).

The process to change the access rights is very similar to the previous case, except that at the first step instead of creating an “EXP” schema, the “Dependent-EXPs” (see Section 4.4.3) need to be defined for the existing EXPs of the production orders (ExpProdOrder1), the requested orders (ExpReqOrder1) and the requested items (ExpReqItem1). For example, the Dependent-EXP for the EXP ExpProdOrder1 in Enterprise 2 determines that only some attributes (e.g. EnterpriseId, Name, Description, Delivery Date and Produced Quantity) can be made available to Enterprise 3. As such, when Enterprise 3 asks through the DIMS Browsing Interface for the Get-ProductionOrder query, to gather the production information about the orders in the VE, only the specified attributes related with ‘Pedals’ will be received from Enterprise 2 (see Table 4.4). In other words, the information for the StartDateOfProduct, EndDateOfProduct, RequestedQuantity, PricePerUnit attributes for Enterprise 3 will not be accessible. At the same time, Enterprise 4 is still able to access the entire set of information.

The DIMS scenario cases presented in the previous sections, demonstrate how the ESMT tool of the DIMS can be used at the VE creation and set up phase to define specific information visibility rights to other partners, and also show how the information access rights for other partners can be dynamically changed during the

<table>
<thead>
<tr>
<th>Enterpriseld</th>
<th>Name</th>
<th>StartDate</th>
<th>EndDate</th>
<th>Delivery</th>
<th>Requested</th>
<th>Produced</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 2</td>
<td>Pedal</td>
<td>-</td>
<td>-</td>
<td>8-Nov-99</td>
<td>200</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Enterprise 3</td>
<td>PVCResin</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>70</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Enterprise 4</td>
<td>PedalMould</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Enterpriseld</th>
<th>Name</th>
<th>StartDate</th>
<th>EndDate</th>
<th>Delivery</th>
<th>Requested</th>
<th>Produced</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Enterprise 2</td>
<td>Pedal</td>
<td>2-Aug-99</td>
<td>5-Nov-99</td>
<td>8-Nov-99</td>
<td>200</td>
<td>100</td>
<td>2</td>
</tr>
<tr>
<td>Enterprise 3</td>
<td>PVCResin</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>70</td>
<td>40</td>
<td>1</td>
</tr>
<tr>
<td>Enterprise 4</td>
<td>PedalMould</td>
<td>1-Sep-99</td>
<td>8-Oct-99</td>
<td>3-Oct-99</td>
<td>1</td>
<td>1</td>
<td>2000</td>
</tr>
</tbody>
</table>

Table 4.4: Final results for Demonstration Case 3.
VE operation time.

Please also notice that the information retrieval in all these scenarios takes place according to the FQP steps and workflow activities described in Section 4.5, thorough which several VCL modules (e.g. LCM, PCI, DIMS, DBPMS) and external modules such as the ERP/PPC, work together in order to satisfy a federated query issued on the DIMS integrated schema.

4.8 Extensions and Future Work

Although the research work presented in this chapter can be extended in many directions, this section principally focuses on some possible extensions for the ESMT design and implementation. In particular, the concept of export schema templates, and a mechanism to automatically create export schemas based on VE contract and supervision clauses information, are introduced in this section in the context of the PRODNET approach. Further extensions will also be suggested in Chapters 5 and 6 of this thesis.

Please notice that in general, issues related to the formalization of the interactions among VE partners in terms of contracts and supervision clauses represent an active field of research. For example, regarding other approaches related to VE contract management, in [88] an approach towards automation of contract match making is described, where standard form contracts and standard contract clauses are proposed. In [122], contracts are used to specify the roles and relationships among "objects", such as enterprise objects. However, the approach presented in this chapter exhibits unique features towards the automatic reinforcement of the contract and supervision clauses via the definition of fine-grained access rights based on export schema definitions.

4.8.1 Export Schema Templates

Besides the concept of role defined in Section 4.4.1, there are other mechanisms that could simplify the task of export schema definition for a specific VE partner. For instance, the concept of "export-schema-set template" can be introduced. Such templates represent a predetermined VCL export schema set, which will be used to create new export schema sets with similar characteristics. The idea of the template is to capture the general descriptions of certain VE member roles; for example, a default regular VE partner, or a default VE coordinator. In this way, a set of templates (built-in default descriptions of general export schema sets) are pre-defined and stored in DIMS, and the creation of new export schema sets can be carried out based on these profiles. A constructor function for an export schema set template would include a list of parameters that would be used to create the particular export schema set instance. The concept of template suggested here is similar to the use of class templates handled in object-oriented programming languages such as C++ [126].

As a simple example of template definition and instantiation, please see Figure 4.27. In this figure, a template for export schema sets is defined, where it is
possible to specify parameters for the VE identifier and the client (VE member) identifier. If many export schema sets present a given "pattern" such as the characteristics defined for this template, then the template can be used many times to create any number of export schema instances. Please notice that a template could be based on other template definitions, giving place to the concept of dependent templates.

Other applications of the concept of "templates" and "roles" applied to VE modeling can be found in [122]. In that approach the template refers for instance to the specification of the common features of a collection of objects of a particular type e.g. enterprises, so that other object collections can be instantiated using it. These other definitions of roles and templates are related to our suggested approach; however the emphasis in that work is on formally modeling the general characteristics, behavior and interactions among abstract entities involved in a VE, while our emphasis is on the specific application of these concepts to the definition and support of the visibility levels for information exchange among the VE partners.

### 4.8.2 Automatic Creation of Export Schemas

At the VE creation time, the task of defining initial export schemas for every VE partner can be automated to a great extent in some specific cases. Based on the functionality offered by the ESM of DIMS and on VE contract-related information, a specific procedure can be developed that takes advantage of this information and that automatically generates the export schemas for every VE partner. In order to see how this could be supported, it is necessary to reference some of the steps involved in the VE creation phase, as described in [39].

Please notice that the definition of access rights based on the concept of roles given in Section 4.4, corresponds in reality to the definition of the proper export schema set for every other partner. Also, notice that the supervision clauses that are distributed by the VE coordinator (see Section 4.4.1), can be formally and precisely modeled by data structures, and they can be distributed for example, as a text file that can be parsed locally at every node. For instance, the data associated with a supervision

---

```
<table>
<thead>
<tr>
<th>Export Schema Set</th>
<th>Template definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EXP Schema template)</td>
<td></td>
</tr>
<tr>
<td>tableName: Order</td>
<td>templateName: T1</td>
</tr>
<tr>
<td>selectSpec: *</td>
<td>fromSpec: Order</td>
</tr>
<tr>
<td>whereSpec: ve='VEID' and client = 'CLIENTID'</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>InstanceTemplate (TS1, &quot;ve12&quot;, &quot;client1&quot;)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(EXP Schema template)</td>
</tr>
<tr>
<td>tableName: Product</td>
</tr>
<tr>
<td>selectSpec: *</td>
</tr>
<tr>
<td>whereSpec: id in (select productID from T1)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Export Schema Set Instance</th>
</tr>
</thead>
<tbody>
<tr>
<td>(Single Schema Set Template)</td>
</tr>
<tr>
<td>Parameters: &lt;VEID&gt;, &lt;CLIENTID&gt;</td>
</tr>
<tr>
<td>templateSetName: TS1</td>
</tr>
</tbody>
</table>

| (EXP Schema) |                         |
| tableName: Order | physicalViewName: tv1    |
| selectSpec: *    | fromSpec: Order          |
| whereSpec: ve="ve12" and client = "client1" | |

| (EXP Schema) |                         |
| tableName: Product | physicalViewName: tv2    |
| selectSpec: *      | fromSpec: Product        |
| whereSpec: id in (select productID from tv1) | |

| (EXP Schema) |                         |
| tableName: Product | PCLViewID: pv1           |
```

Figure 4.27: Export Schema Set template instantiation example.
clause for a specific production order could indicate the information that needs to be made available to the VE coordinator (see also [99]). This information can include the identifier of the production order, the real and planned dates for starting and ending the production, the delivery date, the status of the order production (e.g. in progress, delayed, completed), etc. Therefore, the supervision clause for a production order can be modeled by a C-like structure as shown in Figure 4.28.

In general, if the data structures describing the supervision clauses are commonly defined in advance and are well-known to all partners of a given VE, it is possible to create a set of predefined export schema templates (see previous section) for specific types of supervision clauses. These templates can be instantiated with parameters which values are extracted from the supervision clauses' information. For instance, if the supervision clause specifications are distributed as a text file, then this file can be parsed and processed to determine the parameters for the construction functions associated to specific “export schema templates”. Subsequently, when the templates are instantiated, the actual export schema sets for every VE partner are created automatically. Once the export schema sets are defined, the operation phase of the VE can start since the proper visibility levels for information are already properly defined according to the contract and supervision clauses. Clearly, if it becomes necessary, a human operator can modify the generated export schemas at any time during the VE life cycle. This feature is mandatory to be provided for VE evolution.

4.9 Conclusions

In order to support the wide variety of distributed information management requirements identified after the extensive analysis of the VE paradigm, the federated architecture of the DIMS has proven to properly support the cooperative information sharing and exchange, the enterprise autonomy, and the visibility levels and access rights for exchanged data among the VE partners. In this way, the general federated database architecture concepts and principles have been specifically tailored in the DIMS design to handle the complex interoperability and information management requirements set by industrial manufacturing VEs and their associated SMEs.

In this chapter, the major components of the DIMS architecture were described in details, namely: the federated VCL integrated schema, the DIMS Export Schema

```c
typedef struct
{
    Identifier VEId;
    Identifier VEPartnerId;
    Identifier identifierRequestedOrder;
    Boolean requiresDeliveryDate;
    Boolean requiresPlannedStartDate;
    Boolean requiresRealStartDate;
    ...
} RequestedOrderSupervisionClause;
```

Figure 4.28: Partial data structure definitions for requested-order supervision clauses.
Manager Tool (ESMT), the Federated Query Processor (FQP) and the DIMS Server Agent component. In particular, it was described how these components of DIMS together support the secured import/export of information among the federated nodes in virtual enterprises. Some conclusions about these components follow below:

- The VCL integrated schema of the DIMS federated architecture properly achieves the common representation and integration of both local and distributed VE information, while supporting the expected data location transparency for end user, the site autonomy, and the access security among other requirements.

- The Export Schema Manager Tool properly supports the definition of fine-grained visibility levels and access rights defined locally at every node to determine which other VE partners are allowed to access which part of the local information. The export schema hierarchy used by ESMT adequately supports the definition of information access rights based on the partner roles and relationships established in the VE through legal contracts or bilateral agreements among VE partner enterprises. In addition, the ESMT incorporates advanced user interface graphic elements and provides a comprehensive and friendly environment for the end users.

- The federated query processing mechanism developed for the DIMS provides controlled access to data spread over the nodes of the VE network, with proper respect of the exported data definitions (export schemas) and the hierarchy of roles played by each VE node. The FQP is a generic DIMS feature that proved to adequately support the complex management of distributed business process information involving several VE enterprises, which is required for VE coordination and monitoring purposes.

- The DIMS Server Agent component defines and implements an interoperability approach, which properly supports the interaction between the DIMS kernel and the other VCL modules and enterprise internal systems. Since the DIMS Server Agent was implemented using Remote Procedure Call capabilities, it provides a high degree of flexibility for future extensions, and at the same time it allows the invocation of the DIMS services from physically distributed machines.

The application of these components in real case scenarios was presented, considering the general PRODNET VE demonstration scenario. Please notice that all the DIMS components and functionalities presented in this chapter were fully developed at the CO-IM group of the University of Amsterdam, and were also tested and successfully integrated with the other VCL components.

It can also be mentioned that the combination of the approaches for federated/distributed database information management and workflow management technology substantially contributed to tackle the complex interactions among and inside the VCL nodes to support the VE functionalities. For example, a few specific scenarios were described in this chapter, addressing how the implementation of the DIMS
federated architecture can benefit from workflow plan specifications. Conversely, the workflow management engine can also benefit from the distributed information management services offered by systems such as the DIMS.

Furthermore, the current DIMS implementation provides support for a large number of services for different VCL components and internal enterprise system. Therefore, the DIMS acts as a real data backbone supporting the diverse VCL functionalities, and ultimately supporting the global VE operation itself.

In addition, some directions in terms of future work regarding the ESMT component were introduced in this chapter, including the management of export schema templates, and the automatic creation of enterprise export schemas. These extensions would greatly facilitate the task of defining individual export schemas for each VE partner.

Finally, it can be mentioned that the implemented DIMS module satisfies the large set of information management requirements that were identified within the context of the PRODNET project and its target SMEs, and provides a solid platform that can also be extended in order to address future VE life-cycle support enhancements in addition to the current reference scenarios.