Report

Exploiting multidatabase technology for CIM

Author(s):
Norrie, Moira C.

Publication Date:
1994

Permanent Link:
https://doi.org/10.3929/ethz-a-000958665

Rights / License:
In Copyright - Non-Commercial Use Permitted
Exploiting Multidatabase Technology for CIM

M. C. Norrie, W. Schaad, H.-J. Schek, M. Wunderli

Department of Computer Science, ETH,
ETH-Zentrum, CH-8092 Zurich, Switzerland.
email: {norrie,schaad,schek,wunderli}@inf.ethz.ch

1st September 1994

Abstract

We describe an approach to the realisation of CIM systems through the coordination of autonomous component systems based on the exploitation of multidatabase technologies. Each component system is augmented by a CIM Agent which provides a coordination interface; the interface specifies globally important local objects in terms of a global semantic model. System-wide consistency is ensured through the management of global constraints by a central coordinator along with cooperative multi-level transaction schemes. The CIM Agents are responsible for monitoring local activities and notifying the central coordinator of updates relevant to global consistency; they are also responsible for initiating local actions necessary for the maintenance of system-wide consistency as delegated by the coordinator. In particular, we describe the CIM/Z system which is being developed as part of a Swiss collaborative project on the use of database technologies for CIM.

1 Introduction

A CIM system supports the various activities and users involved in manufacturing systems. Such activities include computer aided design (CAD), computer aided manufacturing (CAM), production planning, parts list management and document production. One approach to the realisation of CIM systems is to design and develop a system from scratch – however this has several disadvantages. Firstly, these systems are very large and by their very nature they must support a wide variety of specialist activities and users. It is difficult to produce a system that meets requirements in terms of functionality, convenience and performance across all groups of users. Secondly, many of the CIM activities are already supported by existing systems; for example, CAD systems are in wide-spread use. Thirdly, while a single system could support multiple views of the application reality, there is a
strong tendency with a single system that, to some extent, a global enterprise model be adopted. Since CIM activities will span many departments, it may be non-trivial to attain such a global model without imposing what might be seen as very restrictive practices on the various departments.

We take the view that a CIM system is primarily a form of cooperative working among component systems with the emphasis on coordination rather than integration. While there will be some global control to ensure system-wide consistency, existing local applications will be able to operate as before and, as far as possible, system coordination will be performed “behind the scenes” in such a way that global consistency is achieved with a minimum loss of autonomy. We therefore describe the system as “loosely-integrated” or “loosely-coupled”.

Further, it is preferable that a CIM system can utilise existing component systems and their data sets while catering for possible extensions in terms of new systems and functionality. We use the term ‘data sets’ rather than ‘databases’ since not all component systems will be database application systems. It is possible that some activities may be supported by application programs with long-term data stored in flat files.

The issues of global consistency in a distributed system have been investigated in the context of database systems where data is distributed over a number of separate databases. Depending on the objectives and architecture these systems have been classified variously as distributed [CP84], federated [HM85] or multidatabase [LA86] systems. Generally, cooperation among systems is achieved through a global data model and language and consistency is ensured by means of global constraint and transaction management. We propose an architecture that exploits such multidatabase technologies in the realisation of CIM systems.

The general aims of our work are twofold. Firstly, we are investigating how to adapt and combine state-of-the-art multidatabase technologies in real application systems and are using CIM as the driving force. Secondly, these technologies are “database technologies” and, as stated above, not all component systems will have database functionality. We therefore have to establish general principles for the augmentation of component systems by means of local agents such that the component system is enhanced with the required database functionality.

Our approach relates to ActMan [JRW88, JRW90], a project done at the University of Erlangen-Nürnberg, Germany. This project also investigated coordination between different CIM component systems. An important goal of our project is that we want to increase the local autonomy for the component systems, whereas, in ActMan, a tight coupling between the component systems was chosen. For example, ActMan uses a central, global database where globally important data is replicated. We avoid this approach and keep, whenever possible, all the data in the data management system of the component system. We then increase the local autonomy by using an advanced transaction model that does
not use the two-phase commit protocol as in ActMan. Further differences stem from the fact that both projects use different concepts for global constraint management.

Other related projects include the following. The concept of using an integration database was used in a project at IBM Germany, at the Heidelberg Scientific Center [Bro92, BH92]. The use of enhanced database support (e.g. by using complex objects) for CIM was investigated at IBM Almaden Research Center, San Jose [LB87]. The integration of CAD and Production Planning Systems by using update dependencies has been studied in a project at the University of Maryland [MRC91]. Achieving coordination through the use of a central, integrated document and order management system was investigated in the DOCMAN project at the WZL, Technical University of Aachen [EGW91].

Our work is being undertaken in the context of a collaborative project investigating the use of database technologies for CIM; this project involves both industrial partners and the CIM-research group at ETH Zurich at the Institute of Construction and Design Methods. The project concerns include the development of enhanced CIM component systems as well as the integration of existing ones. As part of the project we are developing a demonstrator system, CIM/Z, and it is this that we describe in this paper.

In chapter 2, we present a general overview of the CIM/Z system and introduce the general architecture and its coordination components – the global coordinator and the local CIM Agents. Details of the global coordinator and of the CIM Agents are given in chapters 3 and 4, respectively. Chapter 5 describes global transaction management. Concluding remarks are given in chapter 6.

2 General Architecture

The main objective of the CIM/Z system is to coordinate the activities of the component systems to maintain system-wide consistency. This should be done in such a way that a user's view of a component system is, as far as possible, unaffected. It should be emphasised that it is not primarily the task of the system to enhance either local or global functionality – however extended functionality can be supported by the architecture and indeed the present system does provide some forms of extended local functionality. To make these distinctions clear, we can describe a three step approach to CIM systems: the first step is that of coordination, the second of enhanced local functionality and the third of global functionality. Coordination ensures consistency across component systems and therefore consistency across enterprise activities; this topic is the main focus of this paper. If a CIM system provides enhanced local system functionality, it means that a component system supports activities that were previously unsupported. For example, remote data may be visible to a local user through some form of import/export mechanism akin to that of federated database systems [HM85]; we call this schema enhancement. A second form of local extension is system enhancement whereby system functionality is enhanced: for example, certain constraint checking facilities may be performed by the global system on behalf of a
local system which cannot support such constraints. Providing global functionality means supporting global applications, such as global queries, which assume that users may have a global view of the data across the system; this is akin to supporting the logical notion of a single database which happens to be physically distributed across different systems [CP84]. Currently, such global applications are not supported in CIM/Z as this is considered not to be a major requirement. However note that the underlying principles and design of CIM/Z is such that global functionality could be added in the future.

The general architecture of CIM/Z is shown in figure 1. For simplification, we show only two component systems - a CAD system and a Parts List Management (PLM) system. Each of these component systems has a set of local users whose view remains unchanged by the incorporation of the system into the global CIM system.

![Figure 1: General Architecture of CIM/Z](image)

It is not assumed that a component system is a database application system. For example, the PLM system may simply be an application program written in a general-purpose programming language, such as C, and which stores data in files. This means that data description is internalised in an application program and that no transaction and recovery mechanisms are supported. At the other extreme, a PLM system could be implemented on top of a DBMS with full database functionality in terms of constraint, transaction and recovery management and with fully accessible data description. In general, we augment each component system with a CIM Agent and together they present a global database view of the component system with the required minimum functionality for global coordination.

The CAD system supports the design activity and will store information about the various design models consisting of parts and their assemblies. The PLM system manages information about the structure of the models. There is a dependency between the two
systems in that the parts referred to in the design assemblies must exist in the parts lists structures stored in the PLM. If a model is discontinued then we must ensure that designers cannot reference this model in future designs— and that we somehow inform designers that existing designs are no longer valid. This dependency between the two systems is a global consistency constraint and it is the main objective of the integrated system to ensure that global consistency is maintained.

To this end, the global coordinator must know about such dependencies and be informed of any updates within a component system that may result in a violation of a global constraint. It is the responsibility of a CIM Agent to monitor the activities of a component system and notify the global coordinator of updates relevant to the global consistency of the CIM system. The CIM Agent has a local schema which describes those parts of the local data set that are relevant to system-wide consistency; this defines the coordination interface of the component system in terms of globally important objects. Thus, in the case of a CAD system, the local schema may specify models and assemblies but would not include descriptions of geometrical data if this is not pertinent for global consistency. A local database system is used to store information required by the CIM Agent.

The global coordinator stores information about the various component systems, their schemas and the global constraints in a coordination repository. In addition, it is necessary for the coordinator to store information about the individual relationships between data objects of the component systems. For example, it must record the relationship between a particular model of the PLM system and a particular design of the CAD system. When the global coordinator is informed of local component changes, it delegates actions to be taken by the appropriate local CIM Agents in order that consistency be maintained.

Global transaction management is necessary to ensure that all information about changes and the corresponding coordinating activities reach the relevant parties and are acted upon. Having transactions is a prerequisite to the handling of more complex consistency constraints. Transactions are sequences of operations that are executed together in a consistent manner. They ensure atomic execution of all operations of the transactions, even if the transactions are distributed and different CIM component systems are involved. If a component system crashes or a communication line fails, the failures are detected and handled by appropriate protocols. Transactions are executed in isolation, even if they run in parallel. The effects of committed transactions are durable, even if there is a system crash afterwards. These properties are called the ACID-properties [BHG87].

Details of the operation of CIM/Z in terms of the global coordinator, the CIM Agents and transaction management are given in the following sections.
3 The Global Coordinator

The main task of the global coordinator is to ensure, with the help of the CIM Agents, the consistent state of the CIM system. In this section, we detail the operation of the coordination process. We use the example of a part list manager (PLM) and a CAD system. A CAD assembly may have one or more parts lists associated with it where these parts lists correspond to different stages of the design and manufacturing process. For example, an assembly may have both a construction parts list generated by the design process and a production parts list which specifies additional attributes and parts as required for the production process. We assume that a CAD assembly must have at least one parts list (PL) and a parts list belongs to exactly one CAD assembly.

In the case that a CAD assembly is deleted by a user of the CAD system there must be some sort of coordination to ensure that the associated parts lists are deleted in the PLM system. We shall explain the steps involved in the resulting coordination process with reference to figure 2.

![Figure 2: The Coordination Process]

The agent of the CAD system observes the deletion of an assembly. The CAD Agent retrieves the global name (object id) of this assembly from its local repository (2) and requests a confirmation for the CAD system’s operation from the global coordinator (3). The global coordinator checks the validity of the operation according to the global constraints stored in its repository (4). In our example, the constraint on the relationship between assemblies and parts lists will be violated since each PL must be related to exactly one assembly. The coordinator tries to resolve this inconsistency by sending a message to the PLM Agent (5) requesting the deletion of all PLs that were related to the assembly. The PLM Agent uses its local repository to translate the global names (object ids) of the PLs to the local names (6) and instructs the PLM to delete the PLs (7).
The above description is a simplification of the coordination action in that the requested deletion of PLs in the PLM is a logical deletion that may involve a number of local operations and possibly interactions with users of the PLM system. In such circumstances, interaction with a user of a component system may be required either for the purposes of authorisation or even to request a user to perform the necessary actions.

It is possible that a local system or its agent refuses to perform an action requested by the global coordinator. In that case, the global coordinator is responsible for the correct compensation of the updates already made in other component systems and has to send the initiating agent the order to undo its component system’s operation.

To ensure the atomic and isolated execution of all of the modifications, the operations are performed as part of a global transaction. The global coordinator controls the execution of such global transactions; we defer discussion of transaction management to chapter 5.

The general coordination process illustrated in figure 2 requires a global data model and language for communication between the global coordinator and the CIM Agents. In the Coordination View that data model is used to describe the globally important objects and their relationships. We shall first discuss the global data model and then go on to describe the Coordination View and its construction. We follow this with a description of the global coordinator’s repository and conclude the section with a discussion of the different classes of global constraints managed by the coordinator.

3.1 Global data model

The CIM component systems may have very different data models. A component system using a DBMS may use a relational, object-oriented, hierarchical or network data model. Other component systems may store their data in files and it may even not be clear what “data model” they are using. It is vital that the global coordinator views the data of all component systems in a uniform manner. This means that we require a global data model which

1. covers the semantic expressiveness of all data models used in the existing CIM component systems (and those that we envisage might be included in the future);
2. is capable of expressing constraints between the component systems.

In addition, we wanted a data model that appealed to our project partners from the engineering community in terms of ease of use, a rich graphical notation and availability of interactive design tools. To satisfy these requirements, we chose the modelling language NIAM/RIDL [VB82, Win89], which is supported by the tool RIDL* [?]. Currently, we are considering the transition from NIAM/RIDL to STEP/EXPRESS which is part of the upcoming ISO Standard 10303 [EXP92]. The two underlying data models have many
similarities and we are currently undertaking a detailed comparison and investigating the available design tools for STEP/EXPRESS.

NIAM/RIDI offers the features of semantic data models which are needed to describe the various constraints occurring in a coordinated CIM system. These features include the notion of abstract object types, relationships between them and classification constraints such as the *isa* relationship.

### 3.2 Coordination View

As stated in the introduction, it is not our goal to establish a centralized database which holds all of the data of the coordinated CIM component systems. We want the component systems to retain the highest possible degree of autonomy, and that it is reduced only when absolutely necessary for the maintenance of system-wide integrity.

An examination of the CIM component systems reveals that not all local data is important for system-wide integrity. In particular, the objects which do not take part in a global constraint are not relevant to global consistency. For example, a PLM system may store templates for reports which do not have any effect on global integrity and therefore the global coordinator does not need to know about them. We call the objects that are important for system-wide integrity *globally important* objects.

Note that we expect that a CIM component system handles local constraints *locally*. If this is not possible and we need the global coordinator to guarantee them the local constraints are regarded as global ones even if all related objects belong to the same component system.

For the definition of global constraints across component systems, a conceptual model of the globally important objects and the relationships between them is required. We call this conceptual model the Coordination View as it describes all of the entities pertinent to the coordination process.

Each component system has a conceptual model of its own globally important objects. This local conceptual model can be considered as a specification of the coordination interface of the component system. These coordination interfaces are incorporated into the Coordination View. Through the following example, we indicate how autonomous component systems are incorporated into the CIM system through the provision of coordination interfaces which are then integrated into the Coordination View. These steps have to be performed every time a new component system is added.

We will use the example of the PLM and the CAD system and assume that the PLM stores its data in a relational database and therefore uses the relational data model. It is assumed that the CAD system deals with objects stored in files which are seen by a user/CIM Agent as unstructured byte streams. This is illustrated in figure 3.
For each component system, the local data description is transformed into a schema of the global data model, NIAM/RIDL, describing the globally important objects and omitting objects which have no global relevance (figure 4). We will refer to these component schemas as the coordination schemas of the components.

The solid circles represent object types while boxes represent relationships between those object types. A dashed circle represents an attribute. For example, each CAD Assembly is represented by an object which has an attribute Name. Bars above relationship boxes are used to specify relationship cardinalities. For example, the placement of a pair of bars above the relationship between CAD Assembly and Name indicates that the relationship is one-to-one. The ‘v’ on the CAD Assembly’s edge indicates that every assembly must have such an attribute.
A bar over only one half of a relationship box indicates that a relationship instance is identified by that part of the relationship. For example, the PLM relationship between Part and Parts List has a bar over the Parts List side of the relationship box. This means that a particular part list object identifies a particular instance of such a relationship. In other words, each part list can be associated with at most one part (the root part). The absence of a bar on the Part side of the relationship box indicates that there is no such restriction on part objects and a part may therefore be associated with any number of part lists. In the case that a relationship has no such restrictions and is many-to-many, a single bar is placed over the whole of the relationship box.

The component coordination schemas are integrated into the Coordination View. For the purposes of coordination, it is not necessary for the coordinator to know about the structure of objects; rather it is only necessary that it should know of the existence of objects and their relationship to objects in other component systems. For this reason, descriptions of object attributes are excluded from the Coordination View. However, the object attributes are included in the component schemas since these form the basis not only for the specification of global constraints but also for data exchange between component systems.

For example, the creation of an object in one component system might require the creation of a corresponding object in a second component system. A description of the new object in terms of some of its attributes may be required by the second component system in order that it can create its new object with the appropriate attribute values. The coordinator needs to know only of the action to be taken when informed by the first component system of the creation of its object. It will simply pass on the object description received to the second component system without any interpretation of the structure. It only requires that the corresponding component schemas have compatible descriptions of the relevant object structures.

![Figure 5: Coordination View](image)

The global constraints between component systems must also be modelled in the Coordination View. For example, in figure 5, we introduce a relationship between the global object types Parts List and CAD Assembly to represent the global constraint that each assembly must have one or more associated parts lists. The NIAM notation specifies that the relationship is total on both Parts List and CAD Assembly and that it is many-to-one. In other words, every parts list must be related to a CAD Assembly and vice versa, and a
parts list is related to one assembly whereas an assembly can have many parts lists.

3.3 The Coordinator Repository

The global coordinator stores information necessary for the coordination process in its repository database. Since this information is not as such CIM data but rather a description of the CIM system and its data, we refer to it as metadata and it is described by the Coordination View. This metadata includes:

- Information about the object types that occur in the component schemas. Generally, an object type will correspond to an instance of a construct of the component system’s data model. For example, an object type will correspond to a particular relation if its component system is relational or an ‘object class’ if it is object-oriented.

- For each object type, the set of object ids of the objects of that type. The coordinator is provided with these object ids by the CIM Agents. A CIM Agent stores the mapping between the component system specific identification of an object (TID, filename and path, OID etc.) and the object id used in the coordination system.

- The descriptions of the global constraints; in some cases this may include relationships between individual object ids to represent a relationship which spans component systems.

Figure 6 shows the part of the Coordination View describing the object types, the constraints in which they are involved, and the component systems (i.e. component systems) to which they belong.

The solid circles represent coordination entities while boxes represent relationships between those entities. As before, the dashed circles represent attributes. For example, each component system is represented by a component system entity which has an attribute ID. The relationship between Subsystem and ID is one-to-one. The ‘ν’ on the component system edge indicates that every component system must have such an attribute.

Each type of globally important object is represented by an Object_type entity and each such entity is related to exactly one component system. The single bar on the Object_type side of the relationship between Object_type and Subsystem indicates that the relationship is one-to-many in that a particular object type is associated with one component system, but a component system can be associated with many object types. Each Object_type has many ObjectIDs associated with it, where an ObjectID denotes a specific global object.

Each global constraint is represented by a constraint entity and is classified as either a relationship constraint (Rel_constr), a subtyping constraint (ISA_constr) or a cardinality constraint (Card_constr). A cardinality constraint is associated with a single object type
whereas relationship and subtyping constraints involve two object types. These various forms of global constraints are described in the following subsection.

3.4 Global Constraints

The forms of global constraints supported in the current version of CIM/Z are those which can be expressed in NIAM/RIDL. As stated above, there are three basic forms of constraints - relationship constraints, subtyping constraints and cardinality constraints - and we now provide a brief description of each of these. We also indicate the direction of future work to provide additional forms of constraint through an extended modelling language.

Relationship constraints place restrictions on the extension of a relationship between two (or more) object types. Part of the example of the previous subsection is shown again in figure 7: A CAD assembly must have at least one, and possibly many, parts lists whereas a parts list belongs to exactly one CAD assembly.

The relationship constraints of NIAM/RIDL could be extended to provide fully flexible cardinality restrictions. For example, in figure 8, we extend the NIAM/RIDL notation to express cardinality constraints in terms of a minimum and maximum value associated with object types participating in the relationship. The bar on the left side, together with the totality indicator $\forall$, indicates that each object of the associated type must participate
in exactly one instance of this relationship. The cardinality constraint 4:6 on the right specifies that objects of the associated type must occur in a minimum of 4 and a maximum of 6 relationship instances. A similar proposal can be found in [NH89].

A second class of constraint deals with classification structures based on subtyping relationships. In figure 9 we show three object types – Drawings, CAD Drawings and Sketches and indicate the component systems to which the objects of these types belong. Both CAD Drawings and Sketches are specialisations of Drawings. This means that every object in, for example, CAD Drawings must also be in Drawings. Then it is the task of the global coordinator to enforce this constraint by sending a request to the agent of the Directory Service to create a new object whenever it is informed of the creation of a CAD Drawing object by the CAD Agent. The arc labelled X indicates that the specialisations are disjoint – no object can belong to both CAD Drawings and a Sketches.

The global coordinator may also be used in a way that enhances the capabilities of a component system. Consider the case in which we wish to specify a lower and/or upper bound on the number of objects of a particular type. This could be useful in case of a stock control system where the inventory level of a particular part must lie between certain limits to guarantee contiguous production while minimising storage costs. In figure 10, we show how such cardinality constraints are modelled in the Coordination View. By the inclusion of this cardinality constraint specification in the Coordinator View, responsibility for the management of this constraint will be passed to the combined working of the CIM Agent and the global coordinator.
In addition to the representation of global constraints, the coordinator repository must record the actions to be taken on the violations of such constraints. For example, it might associate with the constraint of figure 7 that a violation of this constraint resulting from the deletion of a particular CAD assembly should invoke actions to delete all of the associated parts lists. Then the coordinator would send messages to the appropriate CIM Agents requesting them to initiate local operations to delete the relevant parts list objects. This would all be performed within a global transaction to ensure that either all of the local deletions are performed, or, the original deletion of the CAD assembly would be undone and the user initiating the CAD delete operation would be informed of the failure of the delete and some compensating local action would be performed under the auspices of the local CIM Agent.

The global constraints expressed in NIAM/RIDL deal with dependencies between object types and their instances and not with dependencies between specific objects or properties of specific objects. An example of a dependency between specific objects is a relationship instance between two particular objects which is specified as fixed and therefore cannot be
updated. As an example of a constraint between object properties, it may be necessary to express the fact that a measurement property of one object depends on a measurement property of another object. Such constraints require a more general constraint language and at present cannot be expressed in the Coordination View. Work in this area is in progress.

4 CIM Agents

A CIM Agent provides the coordination interface for a component system. It specifies which objects of the component system are visible to the global coordinator, monitors the activities of the component system and requests, from the global coordinator, confirmation for any action of the component system that effects its global objects and therefore may violate a global consistency constraint. In addition, a CIM Agent is responsible for reacting to actions delegated by the global coordinator to restore global consistency in the case of constraint violations. The general architecture of such an agent is shown in figure 11.

![Figure 11: General Concept of an Agent](image)

All CIM Agents specify the coordination interface in terms of the global data model used by the global coordinator and described in the previous section. The process of integration to produce the Coordination View and the definition of the integrity constraints is all done in the context of that single data model which is the basis of the communication language between a CIM Agent and the global coordinator. As illustrated in figure 11, a CIM Agent must also be able to communicate with the component system and also with its own database (the agent’s repository). Thus a CIM Agent is trilingual and one of the major functions of the agent is to map between the various models and languages used.

To perform the coordination task, a CIM Agent has to take over the role of a “spy”. It has to monitor the operations that are executed by its component system and, according to the agent’s coordination interface, must recognise which operations are of global relevance. For
example, to be consistent with the bill of material objects stored in the parts list manager, it is important to know when a new part or subassembly is added to, or deleted from, an assembly in the CAD system. In this case, the CIM Agent must be aware of calls to operations to create, assemble, disassemble or delete parts and subassemblies.

In the remainder of this section we want to discuss the CIM Agent’s monitoring role in more detail. Subsection 4.1 gives an overview of general variants of monitoring approaches; subsection 4.2 describes the monitoring aspects of a specific CIM Agent built for the CAD system Pro/ENGINEER [Par93b].

4.1 Monitoring Variants for CIM Agents

We can classify the monitoring scenarios into two categories depending upon whether or not a component system uses a DBMS. Possible architectural variants for these categories are illustrated in figures 12 and 13 and will be discussed in the remainder of this subsection. Note that we will generally describe the variants in terms of operations on database objects; typically these objects will be tuples in the case of relational systems.

If a CIM Agent has to monitor a component system which uses a DBMS to store its data, it can use the support that contemporary DBMSs provide for monitoring the database state. In the case of a client-server architecture (figure 12 a), a CIM Agent could be implemented as a transparent layer between the database clients and their server. That is, the clients assume direct communication with the server and the server assumes direct communication with the client but, in fact, there is an additional layer between them which can obtain knowledge about every change made to the database. This information, together with the knowledge about the semantics of the database schema used by the component system, is used by the CIM Agent to report to the coordinator about insertions, deletions and updates on objects relevant to global consistency. The Sybase Open Server [Syb92] is an example of a system which provides a framework to support such a layer.

If the DBMS supports triggers on updates to database objects, the interface layer can be omitted and direct calls to the CIM Agent can be inserted into the database (figure 12 b). Access restrictions to a component system’s data may impose some implementation problems but in general this concept is feasible and powerful enough to perform the required monitoring task. Example implementations of such trigger mechanisms can be found in the Ingres [Ing91] and Sybase [Syb93] relational DBMSs.

The task to monitor a component system’s actions is more complicated in the case that a component system is not using a DBMS as its data store. A rather primitive way to perform the monitoring is to identify objects with files and periodically check either some form of time stamp of the last write or, if possible, the file contents at the file system level. It may also be possible to determine updates on globally important objects from an examination of trail files which are supplied by the component system. One problem
with this approach is that it is heavily system version dependent. Figure 13 a) illustrates this approach.

An alternative for component systems with a programming interface is to use the interface to automatically notify the CIM Agent of the component system’s local actions (Figure 13 b). With this information based on the level of the logical data model of the component system, a CIM Agent can easily decide on the global relevance of an operation. This approach was employed to build a CIM Agent for a CAD system, Pro/ENGINEER, using the developer’s package of Pro/ENGINEER, Pro/DEVELOP [Par93a]. We describe this particular CIM Agent in more detail in the next subsection.
4.2 A CIM Agent for the CAD system Pro/ENGINEER

We now describe in detail the interface between a CIM Agent and the component system Pro/ENGINEER. This interface is less flexible with respect to its implementation than those between the CIM Agent and its repository and between the CIM Agent and the global coordinator since it involves interfacing to an existing system. We note that Pro/ENGINEER does not use a DBMS to store its data.

The objects managed by this CAD system are assemblies, parts and drawings. Initially, a user either calls a search/retrieve operation to load an existing object or he creates a new object. An object in memory can be modified by performing a sequence of update operations. There are also operations to query an object and to create reports about its components. On completion of the task, the user can call a store operation to save the changes to the CAD object.

A record of user operations is stored in a trail file; a short sequence of a Pro/ENGINEER trail file is shown in figure 14. Every operation called is recorded including all of its parameters. We see that first a part was created with the name ‘test’. Then a feature was added to the part, and so on until, finally, the part was stored successfully.

![Figure 14: Pro/ENGINEER Trail File](image)

So the CIM Agent for Pro/ENGINEER principally could use the trail file and the fact, that every part, drawing and assembly is stored in a separate file, to get the necessary
information for its coordination task. But fortunately Pro/ENGINEER offers a programming interface which is more stable than the format of the trail file and allows for a better observation of the CAD system’s actions.

Through its programming interface, Pro/ENGINEER can be enhanced with procedures that are triggered whenever the associated operation is executed. Such a notification procedure can be installed so that it is called either before or after the execution of the associated operation; it sends a notification message to the CAD Agent with the name and the parameters of the operation. The agent receives the message, records it in its repository and evaluates it. The agent expresses the operation and its parameters in terms of the global data model ready for communication to the global coordinator.

However a CIM Agent has not only to monitor the operations executed in the component systems, but also to respond to actions delegated by the global coordinator. The operations and objects of the global coordinator’s request must be mapped into the language of the component system. Examples of Pro/ENGINEER operations are given in Figure 15.

| pro_export_file_from_pro        | export operation |
| pro_read_file_to_pro            | import operation |
| prodb_retrieve_object           | reads an object  |
| prodb_save_object               | save an object   |
| prodb_erase_object              | erase an object  |
| prodb_delete_feature            | delete a feature from a part or assembly |
| prodb_disassemble_member        | removes a member from an assembly |
| prodb_write_attribute           | write a user defined attribute |
| prodb_get_object_info           | retrieves info about current object |
| prodb_get_object_ptr            | obtains pointer to object by name |
| prodb_was_object_modified       | shows whether object was modified |
| prodb_find_nobject_depend       | finds dependent objects |
| prodb_first_member              | gets first member of assembly |
| prodb_next_member               | gets next member of assembly |

Figure 15: Operations of Pro/ENGINEER

As an example, consider the case where the global coordinator sends a message to the CAD Agent that a part has to be removed from an assembly. The agent then searches for the local names of the part and the assembly in its repository. It calls a read operation for this assembly to load it into the memory and a disassemble function to remove the part from the assembly. Finally, the modified assembly has to be stored. Some information is saved in the repository in order that a user can be informed about the changes when the assembly is accessed the next time.

There are other Pro/ENGINEER operations to change geometry, to transform objects, to query details about properties, about relationships, and so on. All of these operations make
up the programming interface of Pro/ENGINEER.

In general, CAD systems are complex systems and because they are used for very different engineering tasks, they have to be flexible and extensible. In almost all cases, the manufacturers or third party suppliers have developed additional modules to these systems and for doing this, a programming interface had to be provided, e.g. GII/CATGEO in CATIA [GII88, CAT92] or Pro/DEVELOP in Pro/ENGINEER [Par93a]. These programming interfaces are system dependent but the techniques used to realise the CIM Agent for Pro/ENGINEER and to do the coordination task are not.

In the following section we describe how global transaction management works with respect to the coordination task.

5 Transaction Management

In this section, we examine mechanisms used for transaction management in the CIM/Z system. We look first at atomicity (recovery) and later at the concurrency control problem.

5.1 Atomicity

Atomic transaction execution is one of the prerequisites that a change done in one component system is propagated to the other component systems. There are two main parts to providing atomicity of system-wide updates. Firstly, component systems without transaction support must be enhanced via their agents to provide some minimum level of support required for global atomicity. Secondly, a global transaction mechanism must be provided to ensure atomicity across component systems.

Component systems without a DBMS normally do not support transactions. Some of these systems write all operations and their parameters to trail-files. If the system crashes, these files are read after the restart and the operations are re-executed. This helps to avoid the loss of data locally but does not guarantee its consistency if other component systems have also been involved. There is no guarantee for global atomicity. The question is what support should be provided locally for a component system without transaction management in order that it is possible to have atomicity and durability for global transactions. The CIM Agent must take over the role of maintaining a log of local operations and have some means of initiating some form of undo and redo actions in the local component system. Logging is performed in the agent’s repository which ensures the stability of logs. What we now have to describe is how the component systems are synchronised with the CIM Agents.

In the case that the global coordinator detects from the change request that a global constraint has been violated it may delegate actions to component systems that reestablish
global consistency. It is possible that either the global coordinator or a component system refuses to propagate the necessary actions to restore consistency. In this case, the transaction is aborted and the appropriate status messages returned to the agent which has several possible reactions to a transaction abort. If the agent was notified of the operation before execution in the component system, then the operation may be rejected and the status messages are returned to the user. If the operation has already been completed, then the agent can compensate it. The prerequisite for this is that the agent always logs the information necessary to perform the compensation before the operation is executed. In some cases, the local users may prefer that the operation is executed in order that work is not lost and that consistency problems are deferred until the data in the other component systems can be adjusted. In this case, the agent marks the data as invalid.

If a component system is using a DBMS, the same DBMS may be used to store the agent’s repository. A modification performed on both objects in the component system and objects in the repository can be performed inside a single transaction thereby ensuring local atomicity. Additionally, the use of the component system’s DBMS for the agent’s repository avoids the replication of data. In the case where all component systems use a DBMS, the problem of building transaction management on top of existing DBMSs corresponds to the multi-database problem [BGMS92, BST92].

We now describe the global transaction mechanism by returning to the parts list example introduced in chapter 3. Assume a global integrity constraint that any modification of an assembly in a CAD component system has to be sent to the Parts List Management system (PLM). If a change occurs in the CAD system, the CAD Agent initiates a global transaction and sends a change request to the global coordinator; this may occur when the CAD model is stored.

To illustrate how such a change may be detected by an agent, we return to the case of the CAD Agent for Pro/ENGINEER as described in section 4.2. We assume that an assembly structure of the CAD system is also stored in the repository of the CAD Agent. The CAD Agent retrieves the actual assembly structure from the CAD system after a change and compares it to the version stored in its repository using the operations shown in figure 15. For example, in Pro/ENGINEER, a modification of an object can be detected with the `prodb_was_object_modified` operation. Also, the structure of an assembly can be navigated using the `prodb_first_member` and `prodb_next_member` operations to retrieve the component objects. Then the CAD Agent notifies the global coordinator of all differences found.

Consider the situation where a CIM Agent sends a change request to the global coordinator and then the system aborts. After restart, the CIM Agent has to check the log stored in its repository. If there is an entry about the start of a change operation, but there is no entry about the completion of this operation, then the abort must have happened during the execution of the operation. It is possible that a global transaction had already been started in which case a `begin-of-transaction` would have been written to the log. The CIM Agents together with the coordinator can determine the state of the global transaction
and, using a global commit protocol, terminate the global transaction. If the transaction
was completed successfully, the CAD Agent has to ensure that the CAD operation also
completes. This can be done by checking the version of the assembly, performing an
integrity test and, if necessary, re-executing some parts of the operation. If the transaction
was aborted, then the CAD Agent has to undo the local changes by going back to the old
version or, as an alternative, restarting the global transaction and completing the operation.

We have built a prototype of such a CAD Agent with transaction support for the CAD sys-
tem Pro/ENGINEER [Pfi94]. The CAD Agent itself uses a repository based on a relational
database system.

5.2 Concurrency Control

The second aspect of transaction management is concurrency control. How can different
transactions executing in parallel be isolated? The local transaction management of a
CIM component system can be incorporated into the global transaction management. A
subtransaction of a globally distributed transaction can be executed as an ordinary trans-
action in the local system. The subtransaction will be isolated from other concurrently
executed local transactions and, after the local commit, all modifications will be durable.
However, the local transaction management does not ensure isolation of global transactions
[BGMS92]. We have investigated component systems that use a DBMS and have de-
veloped concepts to integrate the local transaction management into the global transaction
management. We first show how local transactions can influence the correctness of global
transactions and then discuss how conflicts can be detected.

Conflict serializability is an accepted correctness criterion for concurrent transaction exe-
cutions [BHGW87]. A schedule is conflict serializable if the dependency graph showing the
conflicts between transactions has no cycles. If two global transactions (GT) are executed
concurrently then the coordinator has to check the serialization orders of the subtrans-
actions. It has to avoid the situation where in one component system the serialization order
is GT\(_1\) before GT\(_2\) and in the second component system the order is inverted. In figure
16, we show a schedule in which a local transaction (LT) of DBMS\(_2\) runs between the two
global transactions.

In the schedule, there is a conflict between GT\(_2\) and LT and another conflict between LT
and GT\(_1\). If we now examine the dependencies we see that there is an indirect conflict
between GT\(_1\) and GT\(_2\) in addition to the direct conflict between GT\(_1\) and GT\(_2\). This
schedule is not serializable.

The problem is that neither the coordinator alone nor the local DBMS can detect the
conflicts. The local DBMS does not know if a transaction is part of a global transaction;
it can only ensure that the transactions in its DBMS are serializable. The coordinator, on
the other hand, knows nothing about local transactions. It can only detect direct conflicts, but not indirect conflicts between transactions.

In the CIM/Z system, we get control over local transactions [SS03] by using a CIM Agent for every component system. It may be possible to force the component systems to send all the local transactions to the CIM Agent rather than directly to the DBMS. If globally important objects are modified, then the local transaction is expanded to a global transaction. If it aborts, then the local transaction execution is also aborted. The CIM Agent analyses the transactions and, if there is no conflict with previously executed global transactions, forwards them to the DBMS. Results and messages are passed back to the component system.

We adopted this approach for some component systems based on the DBMS Sybase where we investigated how transactions can be controlled from the CIM Agent without modifying the component system. In this context, we investigated the important issue of how to detect conflicts between transactions. We will describe our method using Sybase and SQL, but note that the method applies generally to other forms of component systems.

Sybase is a relational DBMS and thus the operations visible in the CIM Agent are SQL-operations. It is known that Sybase, and many other DBMSs, use locking protocols and acquire locks on database pages. But for the CIM Agent it is normally not possible to access such information. The only way to detect conflicts is by comparing the predicates of the SQL-operations. We employ a strict two-phase locking protocol and, for every operation executed, we acquire a lock based on the predicate. The granularity we chose for locking is sets of tuples. If the lock cannot be granted, because some other transaction already holds an incompatible lock, then the operation execution is delayed until it is possible to obtain the lock. The following example with two transactions TA1 and TA2 demonstrates the locking scheme.

TA1: UPDATE A SET a=5 WHERE b>3
TA\textsubscript{2}: SELECT * FROM A,B WHERE A.b>6 AND A.b<9 AND A.a=B.c AND B.c<4

TA\textsubscript{1} has to acquire an UPDATE-lock on A for all tuples with b>3. TA\textsubscript{2} has to acquire a SELECT-lock on A for all tuples with b>6 and b<9. On B all tuples are locked with c<4. Without knowledge of the tuple-values, the join predicate A.a=B.c does not help in further restricting the tuple-set. The lock manager cannot use this information. Thus it may lock more tuples than necessary, but importantly, it does not miss any conflict.

We have implemented an SQL scanner and parser that retrieves the predicates of the operations in a normalised form [Wal93]. With this, we call a lock manager that is able to handle such predicates [May93]. If necessary, the lock manager puts a transaction in a wait-queue. It ensures that deadlocks are detected and if necessary some transactions are aborted. Before the operations can be passed forward to the DBMS for execution, the recovery manager [Flü94] stores on a stable log any information necessary to redo or undo the operations. Finally, the operations are sent to the DBMS, processed there and all the results are returned, through the CIM Agent, to the application.

What we are doing here is applying database technology to CIM systems. But in some cases we used techniques, and even extended techniques, that have been proposed from the research community but have not yet been used in commercial systems. An important goal of this work is to enhance the autonomy of the component systems. Rather than using strict two-phase locking (2PL) together with two-phase commit (2PC), which would severely restrict the local autonomy, we use open-nested transactions [WS91, BSW88]. Open nested-transactions have the big advantage, that subtransactions can be committed early, thereby allowing other transactions to access the objects in the database. Because in some cases committed subtransactions have to be compensated if the global transaction aborts, the system has to ensure that compensation will be possible. This is done by using multi-level transaction management [Wei91]. By exploiting the semantics of the transactions, the global transaction manager decides what subtransactions are compatible and can be executed together. We have extended this idea so that we can have a mixed execution of global and local transactions [SS93, WDSS93, SWS91]. This is necessary if we want to use this concept for existing applications where we have local transactions with semantics unknown to the transaction manager.

Currently, we are investigating the overhead costs due to the additional transaction management in the CIM Agent [Hel94]. Experiments are underway to show when we have benefits using open-nested transactions instead of the traditional flat transaction management using 2PL together with 2PC. The first measurement results are promising but a detailed analysis has yet to be completed. The next step is to realise a collection of important example scenarios from the project’s industry partners to demonstrate the coordination of different CIM component systems.
6 Conclusions

We have described how multidatabase technologies can be adapted and combined to achieve CIM systems through the coordination of existing component systems. A component system which does not have full DBMS functionality is provided with the minimum level of database functionality required for coordination by means of a CIM Agent. This agent is also responsible for monitoring local activity and notifying a global coordinator of any modifications that may affect system-wide consistency. The global coordinator stores information on component systems, global constraints and relationships between individual objects of component systems. This information is used to detect global inconsistencies and delegate actions to the local component systems to restore consistency.

Communication between the global coordinator and the CIM Agents is based on an agreed global schema, called the Coordination View, expressed in terms of a semantic data model. This Coordination View is an integration of the coordination interfaces of the component systems where a coordination interface, also expressed in terms of the semantic data model, describes those local objects which are relevant to global consistency.

Critical to the operation of the global coordinator and the CIM Agents is some form of global transaction management to ensure that no system failures will result in global inconsistency. The CIM/Z system employs the ideas of open-nested transactions and multilevel transaction management which are less restrictive than the traditional two-phase commit protocols.

Proof of concept has been obtained by preliminary work on the CIM/Z system. This work has been based on component systems with full relational database functionality and also some without any database functionality.

Acknowledgements

The work described in this paper is part of the project “Databases for CIM” funded by KWF (Swiss Federal Commission for the Advancement of Scientific Research). We thank our partners on this project for their contributions to the project as a whole and the CIM/Z system in particular; they are ABB (Asea Brown Boveri), Sulzer and the CIM-research group of Prof Flemming of the Mechanical Engineering Department at ETH.

References


