(will be inserted by hand later)

Preprint pf: Hasselbring, W. (1997) "Federated Integration of Replicated Information within Hospitals". In: International Journal on Digital Libraries, 1 (3). pp. 192-208.
DOI 10.1007/s007990050016.

Special Issue on "Digital Libraries in Medicine"

Federated Integration of Replicated Information within Hospitals

Wilhelm Hasselbring

University of Dortmund, Department of Computer Science, Informatik 10, D-44221 Dortmund, Germany
e-mail: hasselbring@ls10.informatik.uni-dortmund.de

Received: 10 March 1997 / Revised: 5 June 1997 / Accepted: 12 July 1997

Abstract. A fundamental concern in hospital information systems is the integration of information across heterogeneous subsystems at the data level. Consistent data replication is a central problem to be solved in this domain. The specific requirements and problems for integration of information within hospitals are discussed and a software architecture which has been designed according to these requirements is presented.

The purpose of this paper is to study the problems and solutions of propagation of information updates across heterogeneous subsystems within hospitals. The general structure of the presented architecture is based on the reference architecture for federated database systems (Sheth and Larson 1990) and adapted to the specific demands on integration of replicated information within hospital information systems. This architecture is the basis for algorithms that restore the integrity of replicated information when changes occur. A prototype implementation is discussed.

Key words: Hospital information systems – Electronic medical records – Federated database systems – Data replication

1 Introduction

Digital libraries store materials in electronic format and manipulate large collections of those materials. However, the meaning of the term *digital library* has different meanings to different people as discussed in Fox et al. (1995).

In any case, hospitals offer an environment for deploying digital library techniques. The field relies heavily on visual observation, e.g. of magnetic resonance imaging (MRI), computed tomography (CT), or X-ray films. Hospitals generate complex databases of electronic medical records which are a potential research resource. Additionally, hospitals require complex information flow.

Correspondence to: W. Hasselbring

Most current work on digital libraries is concerned with offering access to collections of materials that were formerly stored in traditional libraries. Typical examples are journal articles and books. A central concern in this area is the retrieval and display of documents which meet the requirements of users who are searching within a library or within a collection of libraries for specific information. Therefore, most research on integration of heterogeneous information has focused on information access.

The purpose of a hospital information system (HIS) is to manage the information that health professionals need to perform their jobs effectively and efficiently (Shortliffe, Perreault, Fagan, and Wiederhold 1990). Integrated systems which satisfy all requirements on information processing in hospitals are not available; even if some vendors promise this. Also, from an economical perspective, it is desirable to install a number of applications, which effectively support the specific needs of the individual organizational units of a hospital. Typical examples are systems for patient registration, admission, discharge and transfer, appointment scheduling, management of laboratory tests as well as decision support for medical treatment. This situation naturally leads to a collection of heterogeneous subsystems scattered across the hospital. To effectively support the work in hospitals, it is necessary to integrate these subsystems avoiding multiple entry of the same information and inconsistencies among information that is stored in different subsystems. Integration is a decisive factor for the successful operation of a computer-based HIS (Ehlers, Schillings, and Pietrzyk 1992). The integration of data from various sources in the hospital produces a rich database supporting health professionals with their work. A modular system of interoperable and cooperating subsystems, which retain their autonomy as far as reasonable, is required.

As a small portion, Fig. 1 illustrates the overlapping areas of information relevant for individual subsystems in hospitals for a laboratory, a radiology and an administration subsystem. It is important to note that it is not the goal to provide access from all places in the hospital to all the information that is relevant for the hospital,

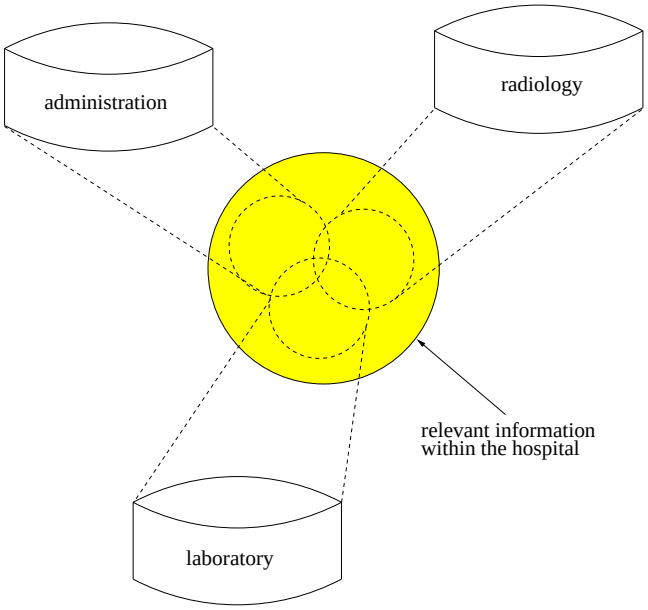

Fig. 1. The overlapping areas of information stored among subsystems in hospitals.

but to integrate the overlapping areas. The basic patient data such as name and birthday are in the overlapping area of all systems, but insurance information and therapy results are only relevant to the corresponding subsystems. Note, however, that the sizes of the areas in Fig. 1 are not proportional; only the structural segmentation is illustrated.

Even management information systems do not store *all* the information: relevant information from multiple sources is integrated and copied into so-called *data warehouses* (materialized views) for analytical processing (Inmon 1996; Zhuge, Garcia-Molina, Hammer, and Widom 1995). Consequently, such systems also do not require all the relevant information within the hospital.

A major need of HISs is, therefore, the integration of the overlapping areas of information stored among different and heterogeneous applications, even if they have been developed at different times, by different vendors and with different technologies. An open federation of autonomous but interworking systems should provide optimized support to the specific needs of the individual units by enabling different vendors to offer specialized applications and allowing the users to select the most effective solutions for their needs.

Furthermore, there will not be one 'right' database system for all components of HISs. In selecting the most appropriate sort of systems for components of a HIS, the technology that most completely satisfies the needs of the specific user groups should be selected. Multiple databases will be necessary since no single database structure is likely to be optimal for each of the various requirements on HISs. Each of the available software and hardware alternatives can have a place in the operation of a HIS. Mainframe facilities may remain appropriate for storage of large databases. Workstations have an advantage in memory-intensive operations such as graphics. Intermediate processing nodes will be required to transform information from dissimilar nodes and make it accessible.

To summarize: within hospitals, it is usually not required to use applications which offer access to all available information in the hospital. Therefore, the user requirements on information systems in hospitals are rather different to the requirements on electronic versions of traditional libraries. In HISs, exchange of information between autonomous subsystems is a central concern for integration of information.

Section 2 discusses some aspects of HISs that are relevant to the paper, before the current state of the art in connecting subsystems within hospitals through communication servers is discussed in Sect. 3. The general architecture of federated database systems is discussed in Sect. 4 and a federated software architecture for integration of replicated information within hospitals is presented in Sect. 5. Section 6 sketches the implementation of a prototype system, which is built to evaluate the presented architecture. Section 7 discusses related work and Sect. 8 draws some conclusions.

2 Hospital information systems

The different demands of different user groups of HISs — physicians, nurses, administrators, health care researchers, patients, etc. — are discussed in Ball and Collen (1992). These different demands lead to different requirements on HISs. For instance, physicians have a patient-oriented view to the information stored in medical records for treating patients, and administrators have a case-oriented view to the medical record for charging the treatment. It is unlikely that one integrated system will be able to satisfy all of them. Corresponding to the different demands of user groups, components of HISs are usually dedicated to such diverse tasks as nursing, laboratories, pharmacies, radiology units, patient monitoring, administration, decision support, medical research, etc.

A note concerning our terminology: a *Hospital Information System* or *HIS* is often regarded as a particular type of product that only focuses on patient registration, admission, discharge, transfer, and other administrative functions. HIS in the context of this paper is a broader concept of *Healthcare Information Systems* as a federation of autonomous information systems (which may include a traditional HIS). However, we use the term *Hospital Information System* because we restrict ourselves to *Healthcare Information Systems* in hospitals. We would call a traditional HIS a Hospital *Administration* Information System as one component of a HIS.

Electronic medical records are central components of HISs, in particular with respect to the integration of information. Section 2.1 discusses some concerns of electronic medical records and Sect. 2.2 discusses some standards for HISs and electronic medical records which are relevant to the integration of heterogeneous systems.

2.1 The electronic medical record

The purpose of electronic medical records is to store the information about patients that is generated by physicians, nurses, hospital administrators, etc. The information that originates from diagnosis and therapy is a central concern. Goals of digitizing medical records are, for instance, improving medical treatment of patients and the computerized evaluation of patient data to support research in medicine. Electronic medical records are not merely automated forms of today's paper-based medical records, but encompass the entire scope of health information in all media forms. Thus electronic medical records may include medical history, current medications, laboratory test results, X-ray films, etc.

The electronic medical record has several advantages over the conventional paper-based medical record, including:

- Patient information is available at several working places at the same time.
- The information is available within a short time. This is important in case of emergency.
- Acquisition of data may be improved by the use of advanced user interfaces.
- Reuse of results of medical operations is supported, even over the lifetime of a patient. This may relieve patients from being checked with the same medical operations several times.
- Medical research is supported. An application area is the control of the results of specific therapies (Ullrich, Hasselbring, Jahnke, Röser, and Christmann 1996).

However, the electronic medical record also has its disadvantages:

- It requires a larger initial investment than its paper counterpart because of hardware, software and training costs for the personnel.
- Capturing the physician-collected data for an electronic medical record can require a lot of time and effort: physicians often use a great deal of information to make one decision.
  New techniques to facilitate direct entry of such information (e.g. speech input) can reduce this problem.
- It is only possible to read and enter patient data where computer terminals are available.
  Mobile computing is a solution for this problem. For a discussion of problems and solutions of mobile computing refer to Imielinski and Badrinath (1994).
- Data security may be a problem when the system administration is not done carefully and responsibly. Access control will be discussed in Sect. 5.2.6.

2.2 Standards for hospital information systems

There exist several initiatives for standardizing healthcare systems (Bakker, Hammond, and Ball 1995; McDonald 1995). Standards are under development for transferring information across subsystems, codifying text, describing the content of the medical record, etc. The following subsections take a short look at four standardization initiatives which are related to our work: the HL7 protocol, CORBAmed of the Object Management Group, coding systems for medical information, and the European healthcare standards organization efforts.

2.2.1 The Health Level 7 protocol

The Health Level 7 (HL7) protocol has been designed to standardize the data transfer within hospitals (Hammond 1993). It is an application level protocol and so relates to level seven of the ISO/OSI-protocol hierarchy (Tanenbaum 1996), but does not fully concur with the ISO standard. HL7 covers various aspects of data exchange in HISs, e.g. admission, discharge and transfer of patients, as well as the exchange of analysis and treatment data. The HL7 standard represents hospital related transactions as standardized messages. HL7 is a de-facto standard for data exchange between commercial systems for hospitals (McDonald 1995).

HL7 defines messages as strings to be exchanged by subsystems. The messages themselves contain standardized information, but do not invoke specific methods at the destination. In Sect. 3, we will discuss how subsystems are usually connected within hospitals through communication servers. Some communication servers for hospitals support standard protocols such as HL7.

The current version of the HL7 protocol (2.2) does not cover all requirements of hospital applications. For instance, the exchange of images is not supported. ACR-NEMA DICOM (Bidgood and Horii 1992) is a standard for transmitting medical images, which can be used for this purpose.

2.2.2 CORBAmed of the Object Management Group

CORBA is the 'Common Object Request Broker Architecture' of the Object Management Group to standardize interoperability among heterogeneous hardware and software systems (Mowbray and Zahavi 1995). Simply stated, CORBA allows applications to communicate with one another no matter where they are located or who has designed them. CORBA defines the Interface Definition Language (IDL) and the Application Programming Interfaces (API) that enable client/server object interaction within a specific implementation of an Object Request Broker (ORB). CORBA also defines interoperability by specifying how ORBs from different vendors can interoperate.

The ORB is the middleware that establishes the client-server relationships between objects. Clients can transparently invoke a method on a server object, which can be on the same machine or across a network. The ORB intercepts the call and is responsible for finding an object that can accept the request, pass it the parameters, invoke its method, and return the results. The client does not have to be aware of where the object is located, its programming language, its operating system, or any

other system aspects that are not part of an object's interface.

CORBAmed is the Healthcare Special Interest Group for CORBA. In 1996, CORBAmed started the process of adopting standard interfaces for healthcare related objects by issuing a 'request for information' which requested the healthcare and information technology industry to give the OMG guidance in its upcoming standardization efforts for CORBAmed.

It can be expected that the CORBAmed services within the CORBA framework will be an important standard for the interoperation among subsystems in hospitals (Sokolowski 1997). However, in the current stage of standardization it is not clear what exactly CORBAmed will contribute to the challenge of integrating information within hospitals.

The HL7 Special Interest Group for Object Brokering Technologies is mapping the forthcoming version 2.3 of HL7 to the IDL of CORBA and version 3 of HL7 will be based on a structurally object-oriented model of the underlying data (Rishel and Quinn 1996). So, we can expect a combination of CORBAmed and HL7 in the future.

2.2.3 Coding systems for medical vocabulary

Several initiatives for standardizing coding systems for medical vocabulary exist. The Systematized Nomenclature of Human and Veterinary Medicine (SNOMED) is a code structure which is widely accepted for describing pathological test results (Campbell and Musen 1992). It has a multi-axial coding structure for symptoms, diagnoses, procedures, etc. The Unified Medical Language System (UMLS) (Lindberg, Humphreys, and McGray 1993) contains a metathesaurus that links biomedical terminology, semantics, and formats of the major clinical coding and reference systems. It links medical terms (e.g. SNOMED) to so-called medical index subject headings (MeSH codes) and to each other. The UMLS also contains a specialist lexicon, a semantic network, and an information sources map. Together, these elements are intended to represent all of the codes, vocabularies, terms, and concepts to become a foundation for the medical informatics infrastructure (Lindberg, Humphreys, and McGray 1993).

These coding systems are used to encode medical information independent of any specific natural language. They can, for instance, be used to allow the exchange of medical records across different language areas.

2.2.4 The European Committee for Standardisation

Working Group 1 on 'Healthcare Information Modelling and Medical Records' of the European Committee for Standardisation CEN TC 251 recently defined a general architecture which is intended as a basis both for the comparison, evolution and integration of existing systems as well as for the planning and high-level design of new open and modular systems of healthcare organizations. To quote from the 'Healthcare Information System Architecture' pre-standard (CEN TC/251 Working Group 1 1997):

"[...] with respect to the definition of the architecture for a healthcare information system, it is enough to notice that, at a high level of abstraction, any healthcare organisation can be described by means of a federative model, as a set organisational components, mutually interacting for the effective delivery of services.

By refining this concept, it can be also realised that each organisational component of the healthcare enterprise has a certain level of autonomy and independence, in terms of information managed and activities supported, which can vary according to the specific organisational, clinical and logistical characteristics of the individual centre and, even within the same centre, according to the specific aspects of the individual units.

[...] Finally, it is important to stress that the architecture should not prescribe a certain organisational form, rather, it should allow system structures that may align to different organisational structures. A federated architecture does not dictate a specific organisational structure, it provides a flexibility for the structuring of systems so that the information technology infrastructure may follow the adopted organisational structure."

In CEN TC/251 Working Group 1 (1997), the proposed basic 'Conceptual Architectural Framework' is structured in three layers. We will relate these layers to the federated architecture which will be presented in Sect. 5.

3 Current state of the art: connecting subsystems within hospitals through communication servers

To connect heterogeneous subsystems in hospitals, communication servers are often deployed (Lange, Osada, Prokosch, and Hasselbring 1996). Figure 2 displays an example configuration of a HIS with a central communication server. In this configuration, a laboratory, a radiology, two wards, an administration, and a pharmacy application are connected by the server. The communication server enables the subsystems to send messages to each other. Each subsystem is connected to the communication server and sends messages only to this server. The communication server determines the receivers and forwards the messages. Hospital communication servers usually support standard protocols such as HL7 (see Sect. 2.2.1) and the translation across different protocols when forwarding messages.

The requirement for building complex systems that combine heterogeneous subsystems can be met at the low level of *interconnectivity* or at the higher level of *interoperability* (Pitoura, Bukhres, and Elmagarmid 1995). Interconnectivity simply supports system *communication*, while interoperability additionally supports systems to cooperate in the joint execution of tasks. A communication server primarily supports interconnectivity: the subsystems themselves

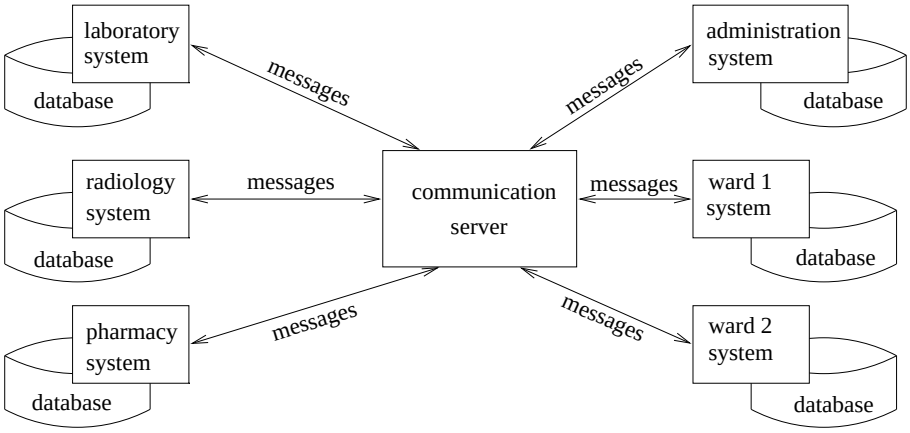

Fig. 2. A possible configuration for the integration of a distributed HIS through a communication server.

- need to know where to send which messages,
- must take the initiative to update replicas and send messages for this purpose, and
- must be aware to receive messages from other systems and store the message contents appropriately in their local data stores.

With an integration that is based on a communication server, it is *not* known at the integration level at which sites data actually is *stored*. It is only known that data is *exchanged*. A communication server does not know whether data is replicated or just needed temporarily by a client for answering a user query.

4 Federated database systems

A *database system* (DBS) consists of a database management system and one or more databases that it manages. As already discussed, data is usually distributed over a multitude of heterogeneous, autonomous DBSs within hospitals. These systems are often isolated and an exchange of data among them is not easy. These are the problems to be solved.

A federated DBS is an integration of such autonomous database systems, where both local applications and global applications accessing multiple database systems are supported (Sheth and Larson 1990). The work on federated DBSs focuses on three issues: autonomy, heterogeneity and distribution:

- Federated DBSs are characterized by a controlled and sometimes limited integration of autonomous DBSs. Often, there are conflicts between requirements of integration and autonomy.
- Causes for heterogeneity are different database management and operating systems utilized, as well as the design autonomy among component DBSs.
- In the case of federated DBSs, much of the distribution is due to the existence of individual DBSs before a federated DBS is built (legacy systems).

There exist several meanings of the terms *federated DBS*, *multidatabase system*, *distributed DBSs* etc. Below, we briefly explain our interpretation of these terms to put our work in context.

Federated DBSs may be distinguished as being *tightly* and *loosely* coupled (Sheth and Larson 1990). In the case of loosely coupled federated DBSs, each component site builds its own federated schema by integrating its local schema with the export schemas of some other component sites. Loose-coupling promotes integration and interoperability via *multidatabase query languages* which allow uniform access to all component DBSs (Tresch and Scholl 1994). It does not require the existence of an integrated schema, leaving many responsibilities, such as dealing with multiple representations of data and resolving semantic mismatch to the programmers of component DBSs. The schema architecture of tightly coupled federated DBSs will be discussed in Sect. 4.2. There is no centralized control in federated systems because the administrators of component DBSs control access to their local data.

In contrast to federated DBSs, *integrated* DBSs do not distinguish local and nonlocal users, because the component DBSs are not autonomous. Integrated DBSs which provide a single global schema to the applications are called *distributed* DBSs (Özsu and Valduriez 1991). Integrated DBSs provide the full functionality of DBSs (transaction and query processing, structured data organization etc).

Both federated and integrated DBSs are *multidatabase systems*. Multidatabase systems can be classified along the dimensions of autonomy, heterogeneity and distribution as shown in Fig. 3. Figure 4 displays the corresponding taxonomy. Other taxonomies and terminologies for multidatabase systems have been suggested (Bright, Hurson, and Pakzad 1992; Pitoura, Bukhres, and Elmagarmid 1995; Tresch and Scholl 1994, and others).

4.1 The general system architecture

Figure 5 displays the general system architecture of federated DBSs. In a federated DBS, both global applications and local applications are supported. The local applications remain autonomous, but must restrict their autonomy to some extent to participate in the federation. Global applications can access multiple local DBSs through the federation layer. The federation layer can

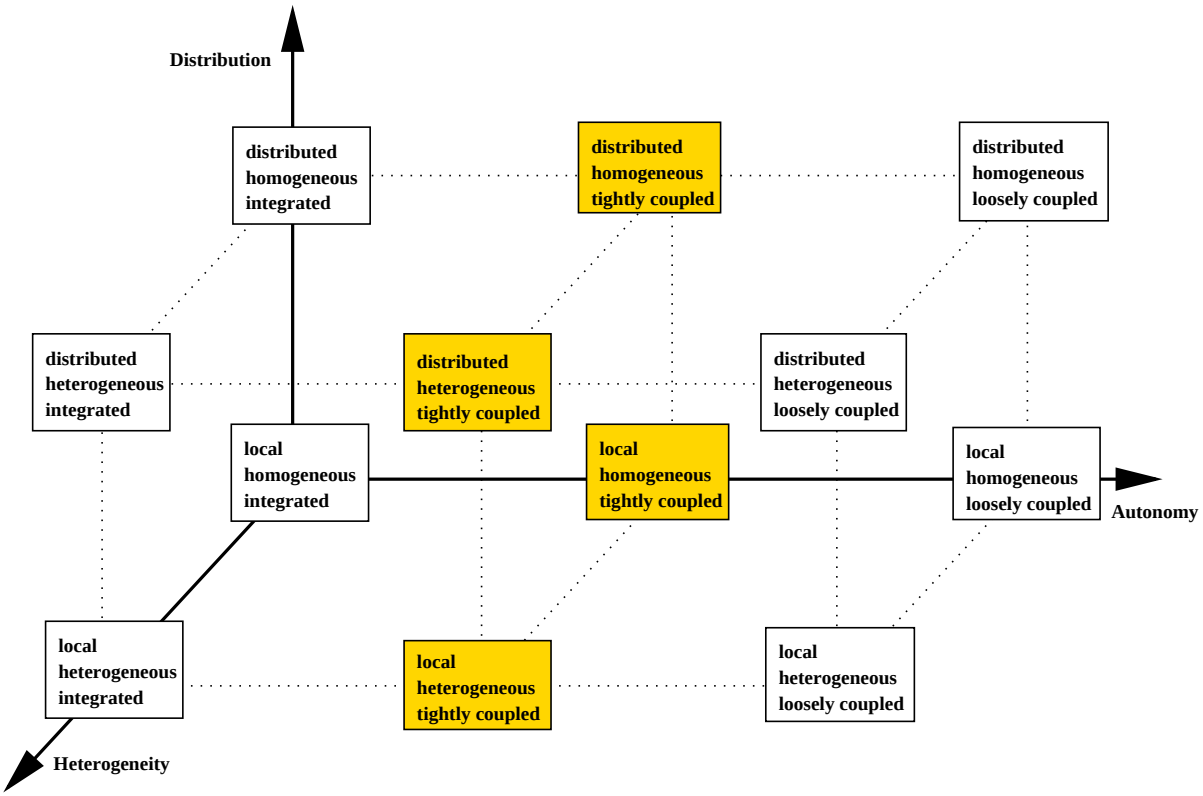

Fig. 3. Classification of multidatabase systems along the dimensions autonomy, heterogeneity and distribution (based on Özsu and Valduriez (1991)).

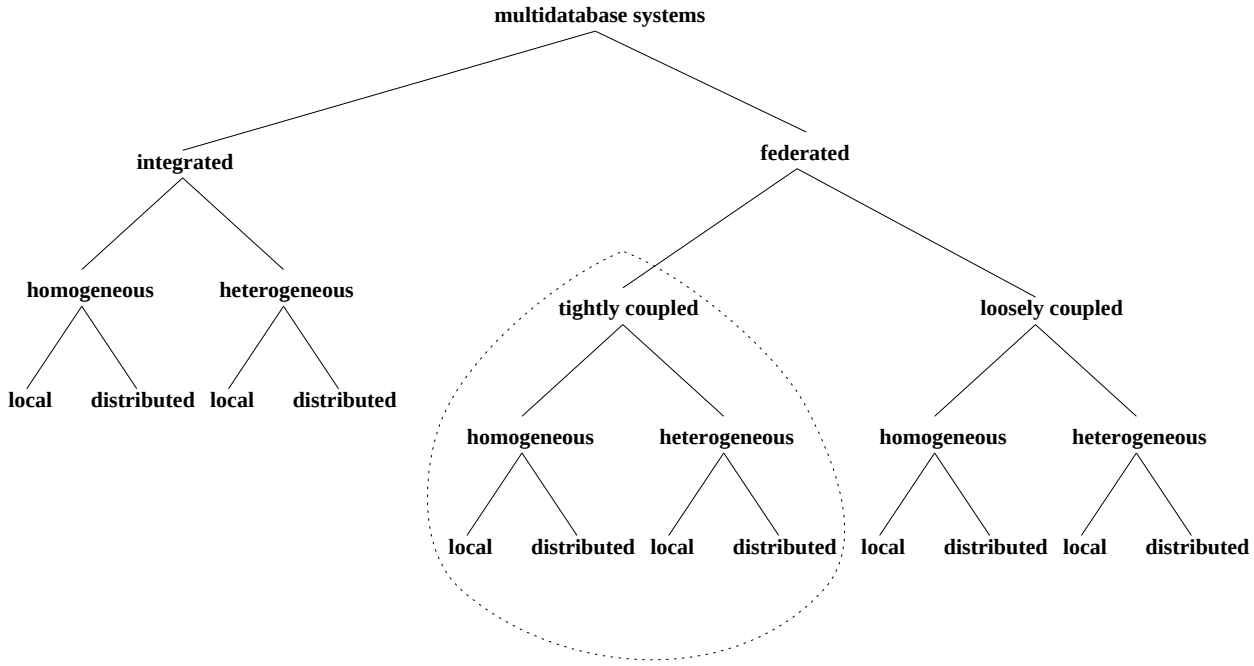

Fig. 4. A taxonomy of multidatabase systems. For HISs, tightly coupled federated DBSs are most appropriate (indicated in the dotted blob).

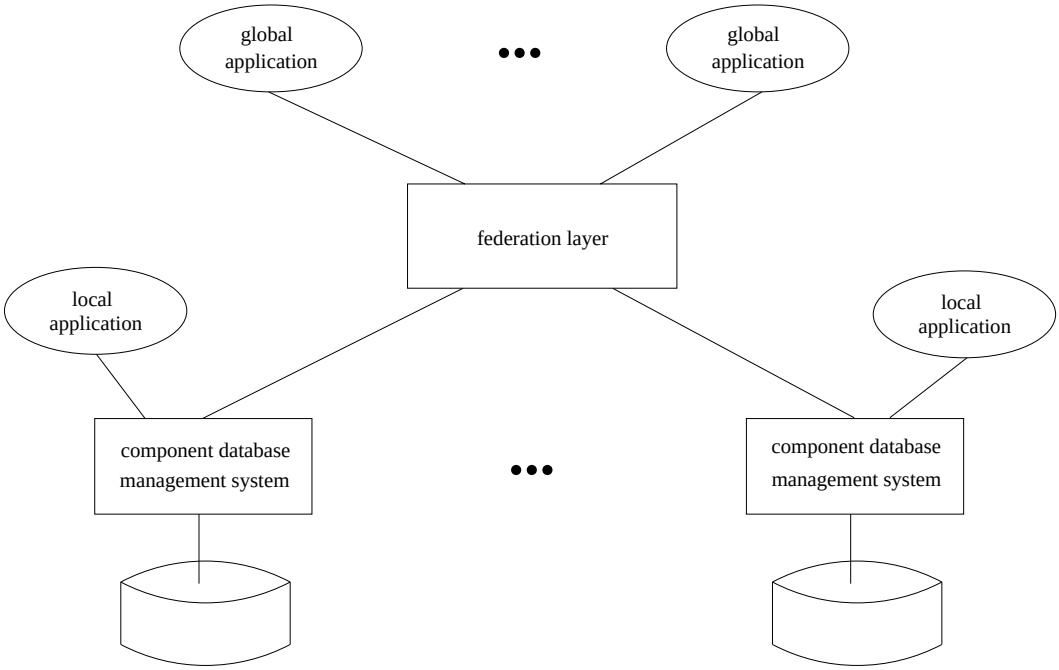

Fig. 5. General system architecture of federated DBSs.

also control global integrity constraints such as data value dependencies across multiple component DBSs.

4.2 The schema architecture

For federated DBSs, the traditional three-level schema architecture (Date 1995) must be extended to support the dimensions of distribution, heterogeneity, and autonomy.

The generally accepted reference architecture for schemas in tightly coupled federated DBSs is presented in Sheth and Larson (1990) and, in the same form, in Pitoura et al. (1995) where approaches to object-orientation in multidatabase systems are surveyed. The diagram in Fig. 6 displays this schema architecture which presents, apart from the dots that indicate repetition, one possible configuration of a federated database system. The different schema types are:

Local schema: A local schema is the conceptual schema of a component DBS which is expressed in the (native) data model of that component DBS.

Component schema: A component schema is a local schema transformed into the (common) data model of the federation layer.

Export schema: An export schema is derived from a component schema and defines an interface to the local data that is made available to the federation. Thus, only exported schema elements and their data are accessible by the federation layer.

Federated schema: A federated schema is the result of the integration of multiple export schemas, and thus provides a uniform interface for global applications.

External schema: An external schema is a specific view on a federated schema or on a local schema. External schemas may base on a specific data model different from the common data model. Basically, external schemas serve as specific interfaces for applications (local or global).

4.3 The processing architecture

The edges between the schemas in Fig. 6 correspond to software processors as indicated in the right hand column of Fig. 6.

Transforming processors translate commands from a source language to a target language. Their purpose is to provide *data model transparency* through hiding differences in query languages and data formats.

Filtering processors constrain the commands and associated data that can be passed to another processor.

A constructing processor transforms commands on the federated schema into commands on one or more export schemas: constructing processors and federated schemas support the distribution feature of a federated DBS. A constructing processor integrates the information from the export schemas.

As discussed in Sheth and Larson (1990), several options in the schema and processing architecture are available, some of which are:

- Any number of external schemas can be defined, each with its own filtering processor.
- Any number of federated schemas can be defined, each with its own constructing processor.

A tightly coupled federated DBS with multiple federations allows the tailoring of the use of the federated

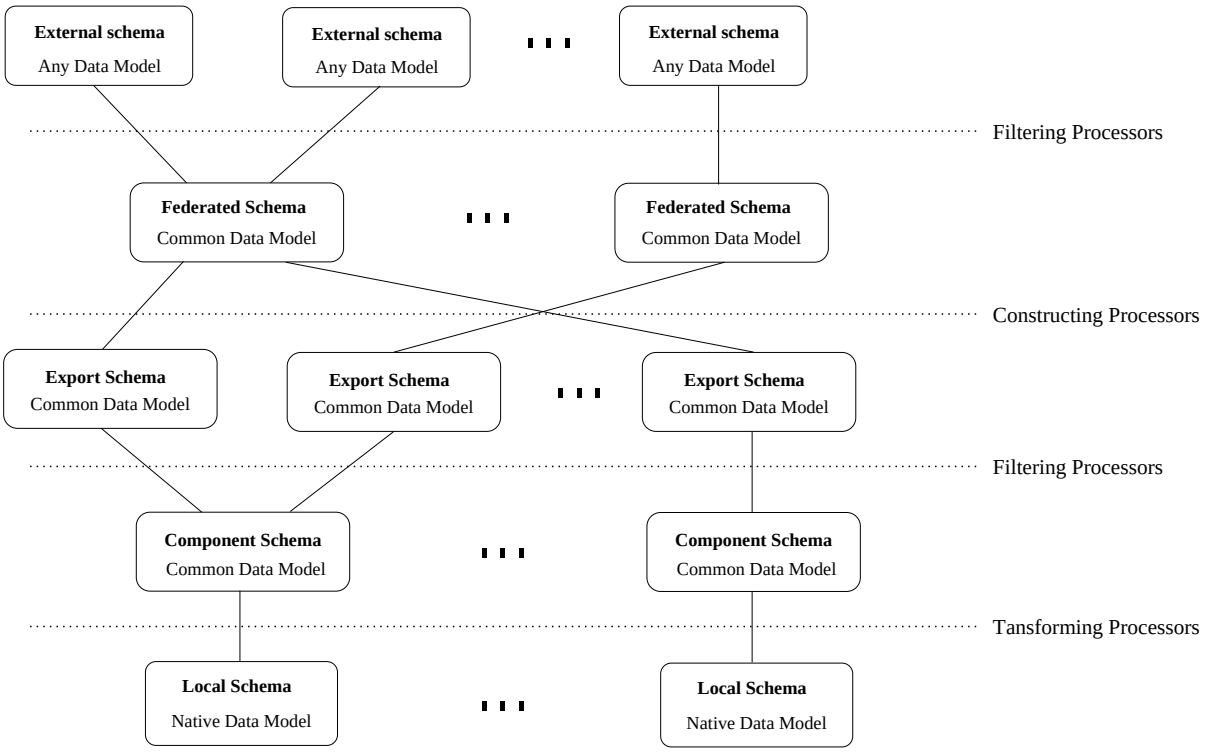

Fig. 6. The 5-level schema architecture as presented in Sheth and Larson (1990) and annotated with the corresponding processor types.

DBS with respect to multiple classes of global federation users with different data access requirements.
- Schemas on all levels, except the local and federated schemas, are optional and may be combined into a single schema of another level.
- A component DBS can participate in more than one federation and continue the operation of local applications.

Note, that a schema architecture which consists of just one federated schema and some local schemas concurs with the 5-level schema architecture of Sheth and Larson (1990). The other levels contain no schemas in this case.

These constraints are not defined formally. In Sect. 5, a refined model for our tightly coupled federated DBS architecture will be presented in a more formal way. Our architecture requires schema integration and consist of component DBSs that can (and usually do) operate independently but also participate in a federation to make their local data sharable. Component DBSs must grant permission to access the data they manage and, therefore, restrict their autonomy to some extent.

With a tightly coupled federated DBS whose data integration is on the basis of schema integration, the federation layer is capable of supporting subsystems to interoperate. Instead of enabling the subsystems with a communication server to send messages for information exchange (Sect. 3), the exchange of information can be accomplished through updates of replicated data in subsystems by the federation layer as will be discussed in Sect. 5.

5 A federated software architecture for hospital information systems

This section presents our federated software architecture which has been designed according to the specific requirements of integrating replicated information among heterogeneous components of HISs. The following subsections present an extended schema architecture and the associated algorithms that restore the integrity of replicated information when changes occur. The general system architecture coincides with the architecture as presented in Fig. 5.

5.1 Extending the schema architecture

It is rather obvious that the reference schema architecture of Sheth and Larson (1990) has been designed primarily to support global access to the component DBSs, only secondarily to support integrity control. However, it is a good approach for resolving the syntactical and semantical conflicts among heterogeneous schemas and for integrating the schemas. Therefore, we extend this reference schema architecture by import, export and import/export distinction for *public* schemas to adequately support the algorithms for changing replicated information (Hasselbring 1997a; Hasselbring 1997b). This distinction does not exist in the reference architecture of Sheth and Larson (1990), which is displayed in Fig. 6.

Figure 7 displays a *meta* model for this extended schema architecture for federated DBSs using the Unified Modeling Language (UML) notation for class diagrams

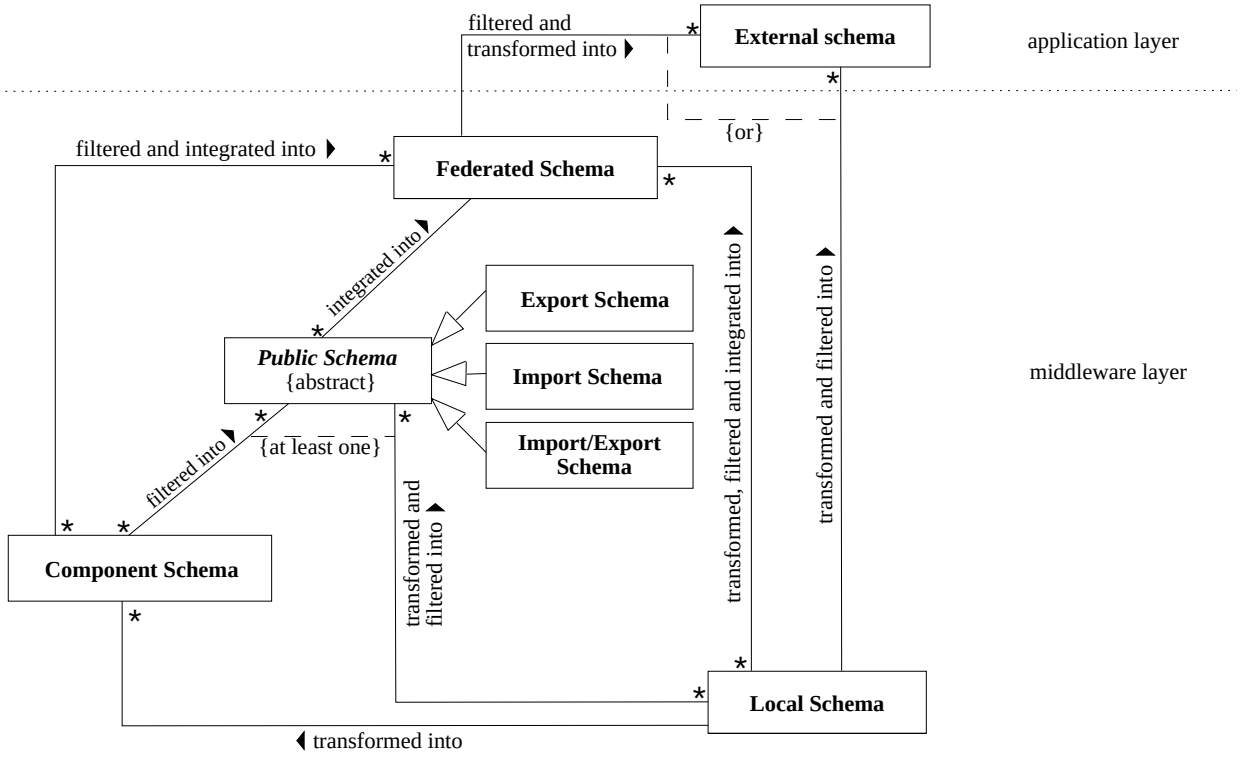

Fig. 7. Modeling the extended 5-level schema architecture as a UML class diagram (Rational Software Corporation 1997).

(Rational Software Corporation 1997; Fowler and Scott 1997). In this model, some of the constraints and options for the architecture, which are discussed in Sect. 4, are defined by means of the *cardinalities* at the associations. The distinct classes of *public* schemas replace the export schemas in the reference architecture of Sheth and Larson (1990).

The diagram in Fig. 6 (apart from the dots that indicate repetition) can be regarded as an instance of the model in Fig. 7. The model in Fig. 7 is a *meta model* for schemas and their associations. The class diagrams of UML are very similar to the class diagrams of its predecessor OMT (Rumbaugh, Michael, William, Frederick, and William 1991).

Specifying an import schema in our architecture is a subscription to change notifications for the corresponding data. Export schemas specify data to be exported to other systems. Import/export schemas define data to be both imported and exported. The schema types determine the change algorithms for integration of replicated information as will be discussed below.

To explain the diagram in Fig. 7: Rectangles are the UML symbols for classes. In UML, cardinalities for associations are specified through numerical ranges at the association links. The default cardinality is 1. If the cardinality specification comprises a single star, then it denotes the unlimited non-negative integer range (zero or more). The arrows attached to the association names indicate the direction for reading the names which are annotations to associations (called *name direction*) (Rational Software Corporation 1997).

The association between local schema and component schema in Fig. 7 specifies that each component schema is transformed from exactly one local schema, but each local schema can be transformed into multiple component schemas when the corresponding component DBS participates in more than one federation.

A constraint in UML is a semantic relationship among model elements that specifies conditions which must be maintained (Rational Software Corporation 1997). A constraint represents semantic information attached to a model element syntactically enclosed in braces. The predefined or-constraints indicate situations in which only one of several potential associations may be instantiated at one time for any single object. This is shown as a dashed line connecting two or more associations, all of which must have a class in common, with the constraint {or} labeling the dashed line. Any instance of the class may only participate in at most one of the associations at one time. This is simply a particular use of the constraint notation.

Each *Public Schema* is filtered from at least one component or local schema and integrated into exactly one federated schema. Each external schema is derived from either one federated or one local schema, etc. External schemas which are directly derived from local schemas are used for local applications.

Inheritance is shown in UML as a solid-line path from the sub-class to the super-class, with a large hollow triangle at the end of the path where it meets the super-class (Rational Software Corporation 1997).

In Fig. 7, *Public Schema* is an abstract class (Meyer 1997). The concrete classes Export Schema, Import Schema, and Import/Export Schema inherit all associations from *Public Schema*. There will be no instances (schemas) of the abstract class *Public Schema* in an instantiated schema architecture.

As indicated in Sect. 2.2.4, Working Group 1 of the European Committee for Standardisation proposes to structure HISs in three layers (CEN TC/251 Working Group 1 1997), to which we intend to relate our schema architecture:

Application layer: The *application* layer consists of a set of applications individually supporting the specific requirements and functionalities in the various units of the hospital.

Middleware layer: The *middleware* layer relates to the operation of the hospital as a whole, and is therefore responsible for supporting the functional and information interworking both of the individual applications and of the definition and management of information and procedures which are of relevance to the hospital. It supports the cooperation of the different applications.

In order to comply with these requirements, the overall HIS is to be described as a federation of information systems, individually responsible for the support of the functional areas and individual organizational units (CEN TC/251 Working Group 1 1997).

Bitways layer: The *bitways* layer is the basic technological platform for the physical connection and interaction of all components of the system. It provides the basic technological infrastructure.

In the right hand column of Fig. 7, the first two layers are related to our schema architecture to place it into the context of the European pre-standard for the 'Healthcare Information System Architecture' (CEN TC/251 Working Group 1 1997). As the basic technological infrastructure, the bitways layer is not related to the schema architecture. However, the bitways layer is *used* by the federation layer to carry out the integration of distributed subsystems, but this is not relevant at the schema level. It is remarkable that the 'Healthcare Information System Architecture' assigns the healthcare-related data to the middleware layer and not to the application layer (CEN TC/251 Working Group 1 1997).

5.2 Change algorithms

The schema architecture is the basis for algorithms that restore the integrity of replicated information when changes occur. For generality, we use the term *change* for insertion, deletion and update of data. Below, a change algorithm with one master copy for data items and a change algorithm with multiple master copies for data items are motivated and discussed. In the sequel, the specification of change propagation and the detection of changes are discussed.

5.2.1 Change algorithm with one master copy for data items

As discussed in Stead et al. (1992), each datum in a distributed DBS for electronic medical records (and, consequently, in a HIS) should have only one *master copy* at which changes are allowed. Note, however, that such a master (server) can cooperate with multiple clients that intend to modify the datum. The restriction to one master copy does not imply a restriction for data entry from just one location within a hospital. A system that is the client in one situation may be the server in another situation provided only one system is the master for a particular datum. A datum may be allowed to reside in multiple places (replica), but changes must be handled through the master who forwards the changes to all places where copies of this datum exist. The federated schema relates elements of export and import schemas to each other, in which each element relates exactly one export element to one or more import elements. This constraint should be enforced by the integration tools.

Figure 8 illustrates an example scenario for changing replicas. Component DBS 1 exports a datum through an export schema. Data about the same real world phenomenon is stored in component DBSs 2, 3 and 4. The latter three component DBSs import this datum through some import schemas. Component DBSs 2 and 3 share the same import schema. To integrate the semantic replication of the same real world phenomenon, the federated schema relates the appropriate parts of export schema 1 to the corresponding parts of import schemas 2 and 3. A change event in component DBS 1 on an exported data item triggers corresponding change operations of the replicas which are imported by the other three component DBSs. The dotted lines illustrate the data flow.

Figure 9 illustrates the passing of information up and down in the schema architecture by means of a more detailed example. The schemas in this example are more detailed than in the scenario in Fig. 8, but Fig. 9 shows fewer schemas. On the lower left corner in Fig. 9, the local relational schema of a simple patient administration system consisting of two tables is indicated. This local schema is mapped to the corresponding component schema in the canonical data model of the federation layer, which could be the object model of the ODMG-93 standard (Cattell 1996). Here, the foreign key relation between Patient and Invoice is transformed to an object-oriented relationship. The patient administration system exports the basic patient data through an export schema via the federated schema to the object-oriented research system as indicated by the dashed arrows. On the other side, the object-oriented research system exports information about medical materials to the relational patient administration system. Here, the research system stores *used materials*, which are transformed via the federated schema to *materials on invoices* for the administration system. When transmitting material, the patient identification is attached to identify the corresponding patient. For simplicity, we assume that a patient is registered in the administration system before the treatment starts for which materials are entered into the research system. We

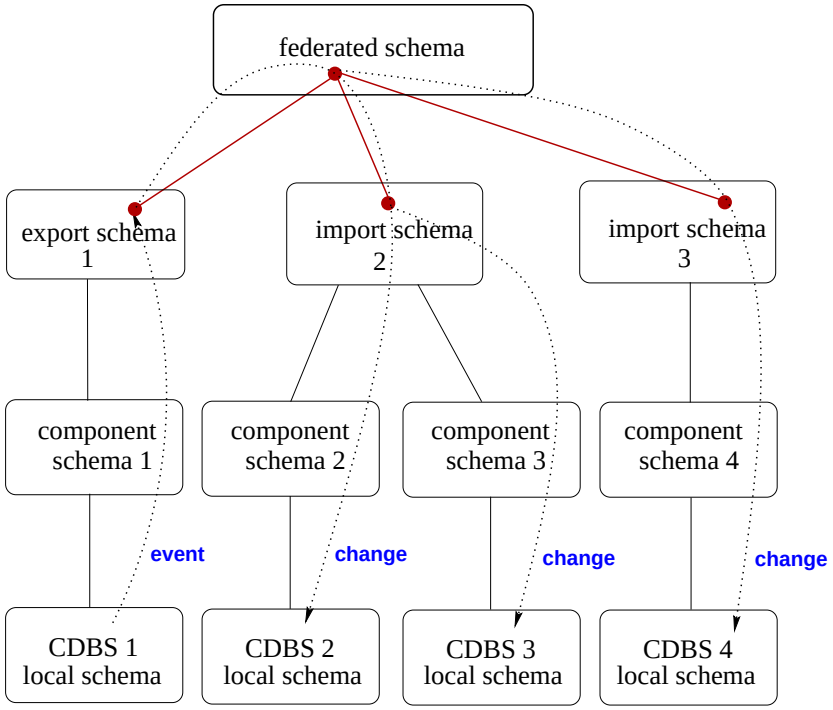

Fig. 8. An example scenario for changing replicas. The model in Fig. 7 is the *meta model* for the schemas and their associations in this scenario. The dotted lines illustrate the data flow.

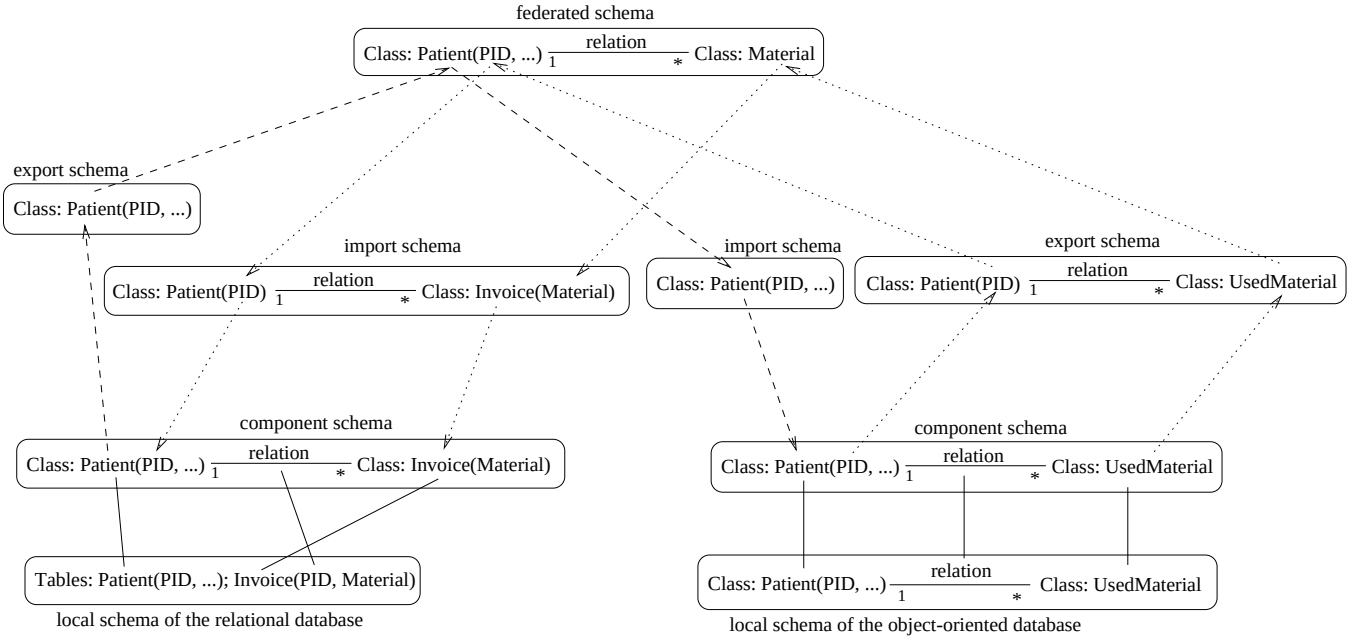

Fig. 9. An example for exporting basic patient data by a relational patient administration system to an object-oriented research system (dashed arrows) and for exporting information about used materials by an object-oriented research system to the relational patient administration system (dotted arrows).

will present more details about the schema architecture in Sect. 6.

5.2.2 Change algorithm with multiple master copies for data items

There exists the need to integrate pre-existing legacy database and file systems into HISs. Typically, these legacy information systems have evolved over many years and play a crucial role in the day-to-day information processing of the hospital. They are often difficult to modify and virtually impossible to rewrite.

There is, therefore, a need to provide techniques not only for accessing the data which is locked inside these systems from newer systems, but also for providing a strategy which allows the gradual migration of the systems to new platforms and architectures. A *smooth* mi-

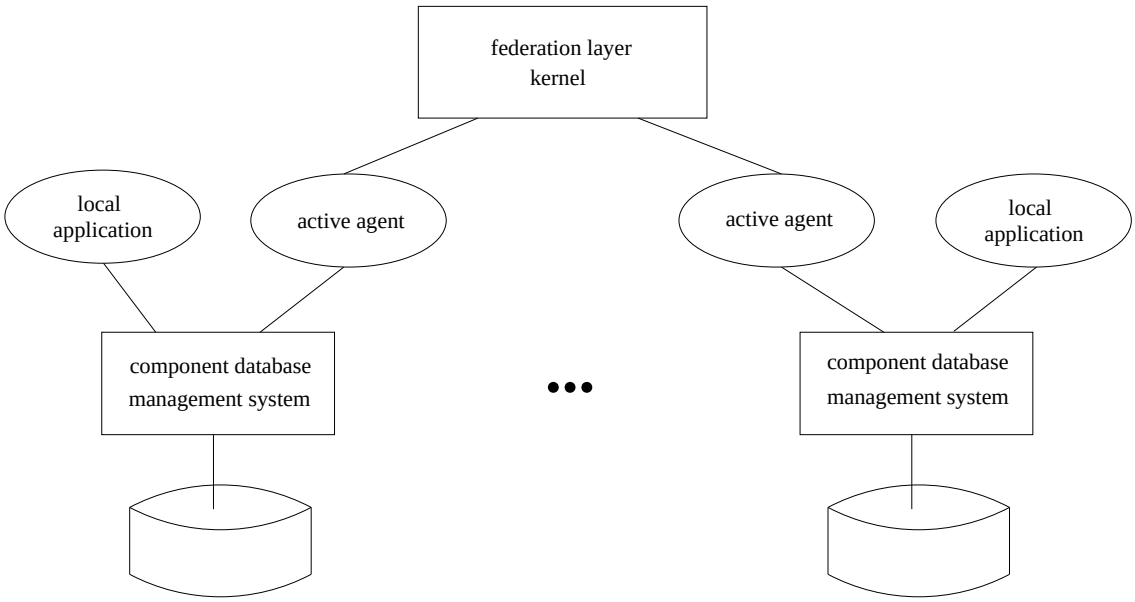

Fig. 10. Active agents in our architecture. Global applications are not displayed in this figure, but they could be connected to the federation layer as shown in Fig. 5.

gration from legacy systems to modern information systems can be accomplished with federated DBSs (Radeke and Scholl 1995).

To integrate replicated information across legacy systems, it cannot be expected that only one master copy for each datum exists at which changes are allowed, because legacy systems usually store the data in their own repositories where the data items must be considered as master copies. To incorporate such situations in which multiple master copies for specific data items are needed, the import/export schemas can be used in our architecture. An import/export schema specifies that the corresponding data items are imported as well as exported.

The difference to a combination of an import with an export schema is that data which is changed by the federation layer does not trigger additional changes to be propagated by the federation layer. Only changes by local applications trigger change events to be propagated by the federation layer.

This mechanism avoids endless loops of changes by the federation layer when the same information is exported as well as imported by multiple component DBSs. However, import/export schemas should only be used when multiple master copies for specific data items are required. For import/export schemas we do not have the constraint that only one data source is allowed.

5.2.3 Specification of change propagation

For the specification of change mechanisms, *agents* connected to the component DBSs are introduced as *active* DBSs (The ACT-NET Consortium 1996; Widom and Ceri 1996). Figure 10 illustrates this division of labor between kernel and agents. The local database management systems of the component DBSs consider the active agents as local applications.

This approach allows a 'separation of concerns' between the federation kernel and the component agents. Separation of concerns is an important principle for software engineering (Ghezzi, Jazayeri, and Mandrioli 1991). The responsibility for monitoring and announcing changes in component DBSs is delegated from the kernel of the federation layer to the agents for the individual component DBSs.

This way, the kernel of the federation layer sees the component DBSs as active DBSs. An active DBS is an extended conventional DBS which has the capability to monitor predefined situations (situations of interest) and to react with defined actions (Widom and Ceri 1996). Such re-active behavior is generally expressed by the so-called *event-condition-action* rules (ECA rules) which define what to do if a certain situation occurs in the DBS.

The production rule paradigm — which originated in the field of Artificial Intelligence with expert system rule languages — has been generalized for *active* DBSs to ECA rules. These are of the form:

on *event*
if *condition*
then *action*

This allows rules to be triggered by events such as database operations, by occurrences of database states, by transitions between states, etc. The triggering mechanism is the addition to the production rule paradigm of

Artificial Intelligence. When the triggering event occurs, the condition is evaluated; if the condition is satisfied, the action is executed.

ECA rules are a promising principle not only for integrity enforcement in single, centralized DBSs, but also for federated DBSs (Türker and Conrad 1997). The active rule mechanism can be considered as a communication mechanism between the component DBSs and the federation layer. Therefore, it is rather straightforward to use ECA rules to specify integrity constraints for replicas and actions which have to be executed in case of potential integrity violations.

Let *ExportSchemas* denote the set of export schemas, *ImportSchemas* denote the set of import schemas and *ImExSchemas* denote the set of import/export schemas. The change mechanisms for our architecture are specified as follows:

∀ ES: *ExportSchemas* ∪ *ImExSchemas*:
  on *event*(ES)
  if ∃ IS: *ImportSchemas* ∪ *ImExSchemas* |
      dependence (ES, IS)
  then -- *change dependent values:*
    ∀ IS ∈ *ImportSchemas* ∪ *ImExSchemas* |
      dependence (ES, IS) : *change* (IS)

Note, however, that this is only a superficial specification of the general mechanisms. For a detailed specification, it would be necessary to specify the structure of the schemas and the functions *event*, *dependence* and *change* which operate on the schemas. The *change* function must not raise events on *ImExSchemas*. A detailed and exhaustive formal specification is beyond the scope of the present paper which focuses on the overall system architecture and the general mechanisms of the associated algorithms.

The execution of rules consists of two phases. In the first phase, which is triggered by the occurrence of the corresponding event, the condition of the rule is evaluated. If the evaluation yields true, the second phase, which is the execution of the action part of the rule, is started.

Both, condition evaluation and action execution, are performed in transaction boundaries. These transactions are called *triggered transactions* whereas the transaction in which the event occurs is called *triggering transaction*. Coupling modes between triggering and triggered transactions determine when the triggered transactions are executed (The ACT-NET Consortium 1996):

Immediate: the triggered transactions are executed as nested sub-transactions (Breitbart, Garcia-Molina, and Silberschatz 1992) directly after the event has been signaled.

Deferred: the triggered transactions are executed at the end of the triggering transaction, but before it commits.

Decoupled: the triggered transactions are started as separate (top-level) transactions.

For our approach, the decoupled mode is most reasonable, as we should not restrict the autonomy of component DBSs more than necessary.

In HIS, there occur predominantly insertions of new information; modifications of existing information occur very seldom. Therefore, a weaker consistency criterion is acceptable: you rarely see outdated information that has been updated somewhere else. You only see new inserted information later on. Therefore, it is reasonable to execute the change operations in separate transactions in this environment. Furthermore, immediate and deferred coupling would restrict the autonomy substantially: the subsystems would be forced to wait until the triggered transactions commit.

However, there are also important cases, usually referred to as 'merge' operations that must be dealt with. For example, a trauma patient who is imaged immediately without being registered and identified to an information system. The acquired data is usually linked to a default patient identification. At a later point in time this information has to be merged with the correct patient records. In legacy systems with redundant data entry, improper identification of patients often occurs. This also requires modification operations. Such merge operations require tighter synchronization and we consider to use some kind of (interactive) tools that apply *immediate* coupling for these update operations which do not occur as frequently as insertions.

5.2.4 Concerns of rule processing

Rule processing is subject to infinite loops, that is, rules may trigger one another indefinitely. In general, it is an undecidable problem to determine in advance whether rules are guaranteed to terminate, although conservative algorithms have been proposed that warn the rule programmer when looping is possible. Methods for statically analyzing sets of active database rules to determine if the rules are guaranteed to terminate etc. are discussed in Aiken et al. (1995).

A prevention against infinite loops in our architecture is the prohibition of cycles in dependencies among import and export schemas via component and federated schemas.

A run-time solution to detecting and preventing infinite loops is to provide a rule triggering limit (Widom and Ceri 1996). In this case, the number of rules executed during rule processing is monitored; if the limit is reached, rule processing is terminated. Another mechanism is to detect whether the same rule is triggered a second time with the same parameters.

5.2.5 Detecting changes by the active agents

How do the agents find out about changes to data? To solve this problem, a balance between autonomy and integration must be found. Some approaches are:

- Some DBSs offer active mechanisms such as *triggers* to detect and announce changes (Widom and Ceri 1996). With the availability of active mechanisms, local applications do not have to be changed: triggers are assigned to monitor changes of exported data.

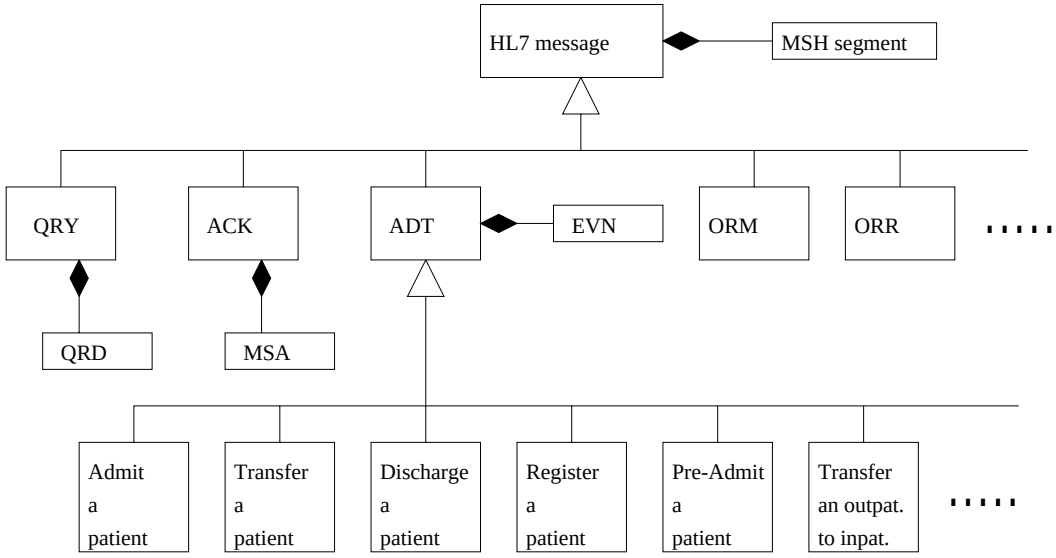

Fig. 11. An extract of the data structure of version 2.2 of HL7 messages modeled as a UML class diagram. Filled diamonds indicate part-of relations (*non-shared* aggregation). UML allows to specify *shared* aggregation, too (Rational Software Corporation 1997). The corresponding OMT model is presented in Hasselbring and Kröber (1997).

- If a component DBS does not support such detecting techniques, *polling* techniques can be deployed:
  - The evaluation of system data can be used to detect the specific operations. For instance, the transactions log file can be monitored (Eliassen and Karlsen 1991).
  - Changes can be detected by comparing data snapshots. Keys can be used to efficiently compute the changes, as described in Labio and Garcia-Molina (1996).
- In client/server systems, an interface between application and server can be used to analyze the client requests and announce detected changes (Kudrass, Loew, and Buchmann 1996).
- If the component DBS is an object-oriented DBS, the stored objects can be modified by an overriding technique (Schmitt and Saake 1995). Any critical method will have to be refined by adding operations that announce changes. This approach restricts the autonomy of local applications, since the local applications are changed.

An additional approach in a hospital setting is the following:

- Wrapping HL7-messages: A HL7 message is a string, which contains mandatory and optional segments (see also Sect. 2.2.1). These segments consist of several fields. The syntax of version 2.2 of HL7 messages is defined informally in HL7 Group (1994). To gain an insight into the structure of the HL7 message types, we analyzed the informal description of HL7 from HL7 Group (1994) and constructed a class diagram for the data structure of HL7 messages (Hasselbring and Kröber 1997). An extract of this model is displayed in Fig. 11. This model can be used as the basis for the component schema of the corresponding component DBS, which could be specified using, e.g., the object definition language of ODMG-93 (Cattell 1996) (see also Sect. 6.2). The corresponding agent would intercept the HL7 messages from the subsystem and announce changes when they are detected. The forthcoming version 3 of HL7 will be accompanied with a *structurally* object-oriented data model (Rishel and Quinn 1996). Version 3 of HL7 is not really object-oriented in the sense that it does not specify any method for these objects. In any case, this will simplify the task of wrapping HL7-messages. However, given the syntactic and semantic ambiguity of the existing HL7 specification and implementations it is difficult to predict how successful this approach will be in practice, but HL7 interfaces are more readily available in HISs than the other methods mentioned.

5.2.6 Access control

Access control is an important concern in hospitals. We briefly discuss how access control can be integrated into our architecture.

Two important features of the schema architecture are how autonomy is preserved and how access control is managed (Sheth and Larson 1990). Autonomy is preserved by dividing the administrative control among some component DBS administrators and a federation DBS administrator. The local, component and public schemas are controlled by the component DBS administrators. The federation DBS administrator defines and manages the federated schemas and the external schemas related to the federated schemas. Access control can be managed by filtering processors associated with public schemas under control of the component DBS adminis-

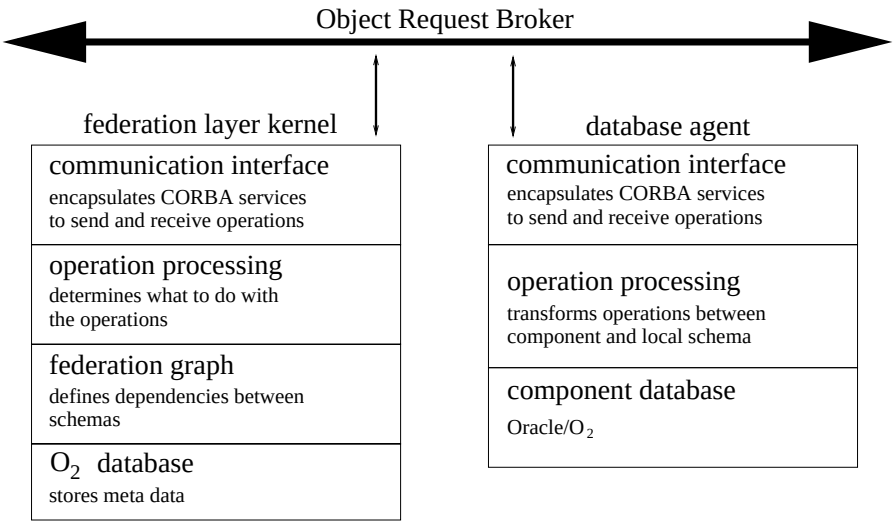

Fig. 12. The layers in the CORBA-based architecture of our prototype implementation.

trators. Note, however, that one person can take on the role of several DBS administrators.

The architecture must permit control of access to data by both category of data and category of user. In some cases, it will be necessary to control access at the individual datum and user level. In addition to category of user and data it appears to be reasonable that users have varying rights depending on the clinical context, i.e. their relationship to the patient and the circumstances of the case. For example, psychiatric history or HIV status may be appropriately accessed by a class of users in restricted contexts but not in general.

6 A prototype implementation

A prototype system has been developed to evaluate the architecture with the associated change algorithms, which have been presented in Sect. 5. The prototype integrates a system which controls the results of therapies in angiopathy (Ullrich, Hasselbring, Jahnke, Röser, and Christmann 1996) with a system which manages patient data and supports charging the treatment. The therapy control system has been implemented in cooperation of the University of Dortmund and the University Hospital Wuppertal (Ullrich, Hasselbring, Jahnke, Röser, and Christmann 1996) with the object-oriented database system O_2 (Bancilhon, Delobel, and Kanellakis 1992). The patient data management system has been implemented with the relational database system Oracle (Bronzite 1989).

As the infrastructure to overcome distribution and heterogeneity on the operating system level, we use the Chorus Cool CORBA implementation (Jacquemot, Jensen, and Carrez 1995). Figure 12 illustrates the layers in the general architecture of the prototype implementation which is based on an Object Request Broker according to the CORBA architecture (Mowbray and Zahavi 1995) (see also Sect. 2.2.2). At the bottom, the meta data of the federation layer (federation graph etc.) are stored in O_2 (Bancilhon, Delobel, and Kanellakis 1992).

6.1 The CORBA-based communication interfaces

The communication interfaces encapsulate CORBA services to send and receive operations which restore the integrity of replicated information when changes occur in a component DBS (see Fig. 12). The transferred data objects contain specifications of the change operations to be transferred between the federation layer and the (active) database agents. The dependencies specified in the schema architecture determine the destinations for the operation objects. Figures 13 and 14 illustrate the architecture of the communication interface, which uses two basic mechanisms:

Operation Objects: For each type of operation a concrete class is defined through inheritance from the abstract class Operation which specifies a uniform interface for all operation types (see Fig. 13). Each concrete class specifies the specific structure for the specifications of one kind of change operations to be exchanged via object instances of this class. The communication interface itself is independent of this specific structure.
The methods FromSeq and ToSeq (Fig. 13) are needed to transform C++ structures into CORBA sequences and vice versa. The Clone method is needed to obtain copies of the objects (see below).

Operation Handler: To receive operations of a concrete type, it is necessary to provide corresponding operation handlers to process the operation in an appropriate way (see Fig. 13). On receipt of an operation object, the communication interface uses copies of *prototype* objects for operation/handler pairs, which are managed by the class Pool (see Fig. 14). The handler is responsible for processing the associated operation.

This mechanism, which makes the communication interface independent of the concrete operation classes, is based to the design pattern *Prototype Factory* (Gamma, Helm, Johnson, and Vlissides 1994, pages 117ff).

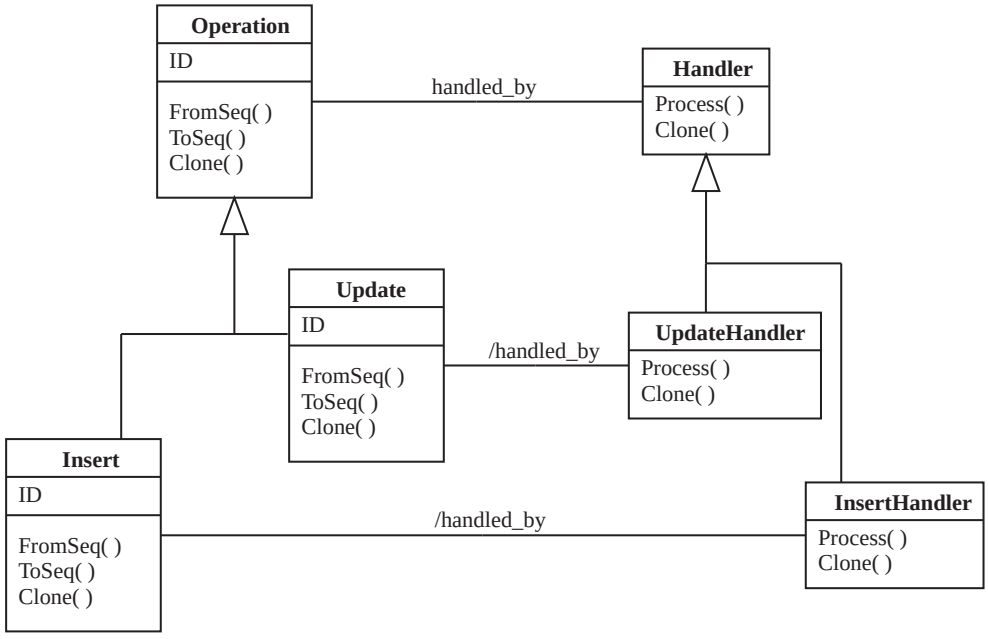

Fig. 13. Operations and their handlers in UML notation. The symbol '/' at the lower 'handled_by' associations indicates their inheritance relationship to the corresponding upper association.

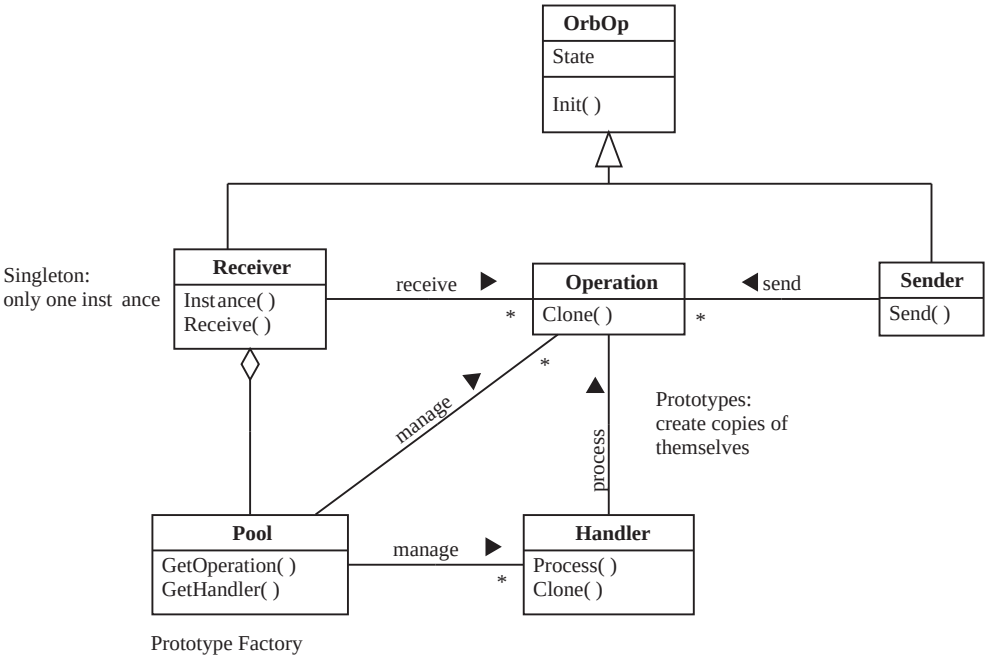

Fig. 14. The general architecture of the communication interface in UML notation. The applied design patterns *Singleton* and *Prototype Factory* are indicated.

Formerly, we used the design pattern *Abstract Factory* (Gamma, Helm, Johnson, and Vlissides 1994, pages 87ff) for this purpose, but with the *Abstract Factory* pattern, the communication interface needs to know the names of the concrete classes yielding a less flexible design (Hasselbring and Ziesche 1997). Design patterns can be viewed as abstract descriptions of simple frameworks that facilitate reuse of software architectures (Gamma, Helm, Johnson, and Vlissides 1994). Another design pattern used in Fig. 14 is called *Singleton* (Gamma, Helm, Johnson, and Vlissides 1994, pages 127ff). Each CORBA object (a C++ program) contains exactly one C++ object instance of the class Receiver, because each database agent is accessed as a CORBA object.

The classes Sender and Receiver inherit some general methods for using the Object Request Broker from the abstract class OrbOp (see Fig. 14). The return values of the methods for sending and receiving operations are either the local object identifiers of changed objects or error codes to indicate the failure of a change operation.

It could be possible that the agent for a component DBS is unable to accept a change operation due to various reasons (out of memory, unavailability of the local database management system, network error, violation of local integrity constraints, etc.). These situations are

reported to the federation layer via the return values of the corresponding method calls. If an error occurs, the federation layer keeps the failed operation and, depending on the error type, informs the system administrator and/or manages the resulting inconsistency (Getta and Maciaszek 1995).

6.2 The canonical data model for the federation graph

The canonical data model is the data model for the middleware layer in Fig. 7. In Saltor et al. (1991), it is shown that functional and object-oriented data models are most appropriate as a canonical data model. Their expressive power allows to cover other data models and eases the mapping. The advantage of object-oriented data models as canonical data models is due to the extensible nature and support of data abstraction and encapsulation which allow to overcome heterogeneity through the provision of abstract data types.

In our project, the object model of the ODMG-93 standard (Cattell 1996) for object-oriented databases is used as the basis for the canonical data model. The ODMG-93 standard includes languages for object definition (ODL), object manipulation (OML) and object query (OQL). Figure 15 displays an extract of the data model for our canonical data model in UML notation.

To explain Fig. 15: If we remove the classes Variable and Object with their associations from this class diagram, we obtain a simplified version of the ODMG-93 object model. A Type is either a PointerType, SimpleType, ComplexType or CollectionType (specialization through inheritance). A PointerType is related to a Type at which it points (target_type). SimpleTypes are integers, floats, characters or Booleans (atomic values). ComplexTypes may be compared with records in Pascal: they contain some Attributes with possibly different Types (value_type). CollectionTypes are sets, bags, lists or arrays where all the elements have the same type (element_type). Multiple PointerTypes, CollectionTypes and Attributes can be associated to the same abstract Type (specified by the cardinalities at the corresponding associations).

Now let us take a look at the extensions to the ODMG-93 object model in Fig. 15: Each Type in a schema is related to *proxy objects* for Objects (instances) of this Type that are stored in the local databases. As illustrated in Fig. 16, these proxy objects contain only the object identifiers for the local objects. This is necessary, because the federation layer stores local and global object identifiers for replicas and the relations among them (Schmitt and Saake 1995). Fig. 16 is a simplified illustration of an instance of the UML model in Fig. 15. On insertion of a new data item, the federation layer generates a new global object identifier and relates it to the local object identifiers of the master copies and the replicas. Therefore, as one of the extensions to ODMG-93, each object of class Type is related to the corresponding instances of class Object. The same_as associations relate local object identifiers in the component schemas via the global object identifier in the federated schema to each other.

The relations between Variables and Objects in Fig. 15 are needed when ComplexTypes are replicated. There is a correspondence between ComplexType/Attribute and Object/Variable, but this constraint is not specified in the UML diagram in Fig. 15. Notice that UML is a semi-formal notation which does not allow to specify all possible constraints explicitly. The import and export schemas and their associations to the federated and component schemas are not displayed in Figs. 15 and 16. A comprehensive discussion of the federation graph can be found in Project Group FOKIS (1997).

To summarize: The purpose of a canonical data model, which is an instance of the UML model in Fig. 15, is to specify the relationships among the schemas (intensional overlapping) and the objects (extensional overlapping) which are stored in the component DBSs.

A note on concerns of performance with respect to the transfer of multimedia data: the federation graph only stores object identifiers and the relations among them to determine the destinations of operation objects. The operation objects may contain large multimedia data to be transferred. To determine the destinations of this data, it is *not* necessary to touch the large multimedia components, only the small keys. As soon as the federation layer knows the destinations, it can forward the change operation with the multimedia data. The processing of the schema architecture is independent of those data values to be transferred. Another optimization is the use of the *Proxy* design pattern (Gamma, Helm, Johnson, and Vlissides 1994, pages 207ff) for the multimedia data values. The proxy objects contain only references (e.g. host and file name or access path to a picture archiving system) to the large data values which are stored externally to the federation layer. The original multimedia data is only touched when it is actually needed for transfer. However, it may be necessary to convert the format of the multimedia values in a heterogeneous environment.

This prototype implementation serves to evaluate the presented architecture with the associated algorithms that restore the integrity of replicated information when changes occur. To save space, the present paper emphasizes on the conceptions, thus only a coarse overview of this implementation is presented. For a more detailed discussion of using CORBA and ODMG-93 for implementing a federated DBS refer to Sander and Hasselbring (1996) and to Project Group FOKIS (1997).

7 Related work

Some standards which are related to our work have been discussed in Sect. 2.2. In some related approaches, the techniques for federated DBSs are being deployed in the application domains of HISs (Roantree, Hickey, Crilly, Cardiff, and Murphy 1996) and digital libraries (Schatz, Mischo, Cole, Hardin, Bishop, and Chen 1996). For instance, Schatz et al. (1996) propose to build digital libraries as repositories of indexed multiple-source collec-

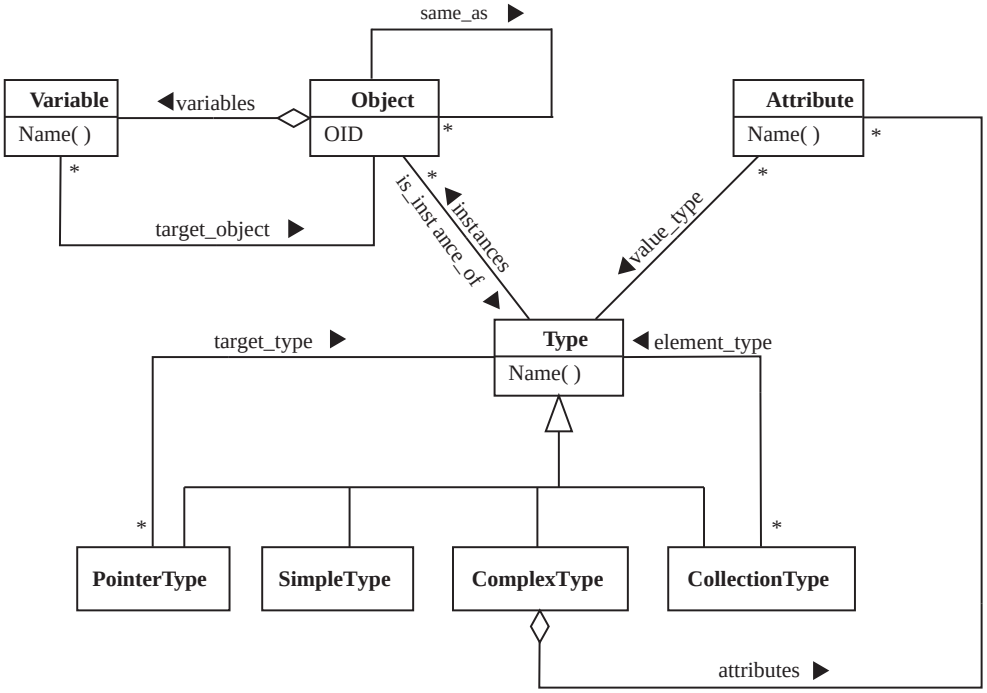

Fig. 15. An extract of the data model for our canonical data model which is based on ODMG-93 (Cattell 1996).

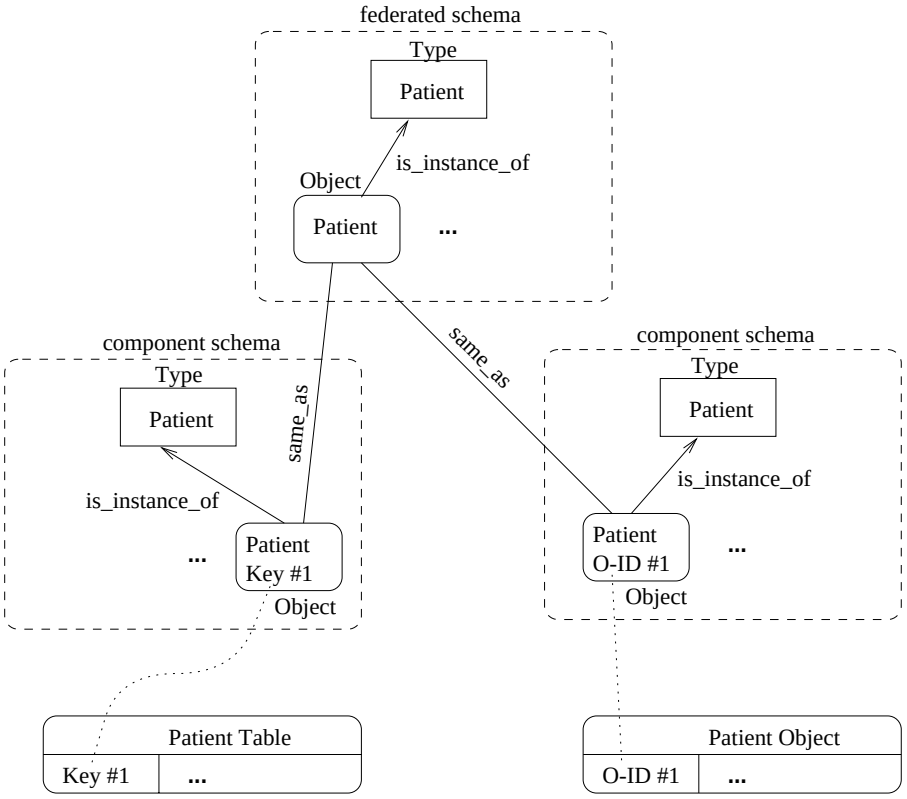

Fig. 16. Types and object instances. Objects of class Type are displayed as rectangles and objects of class Object are displayed as boxes with rounded corners. This is a simplified illustration of an instance of the UML model in Fig. 15.

tions and *federating* them by searching the material via multiple views of a single virtual collection. Roantree et al. (1996) discuss the management of changes to the structure of federated DBSs. However, these approaches do not discuss integrity control for replicated information. Other approaches aim at integrating information within a hospital unit, viz. radiology (Wong and Huang 1996), or at integrating the hospital with external organizations (Wiederhold, Bilello, Sarathy, and Qian 1996). Below we discuss some related work concerning replicated data in distributed DBSs.

Replicated data is employed in distributed DBSs to enhance data availability and performance: multiple copies of some data items are maintained, typically on

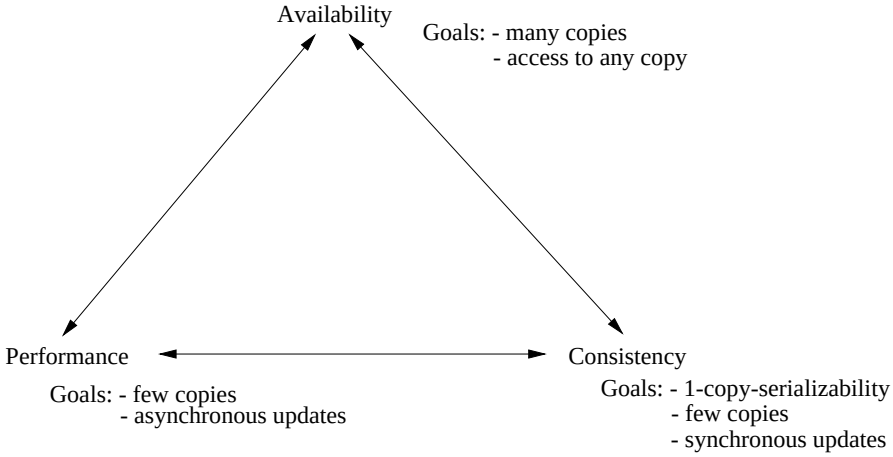

Fig. 17. The conflicting goals for data replication.

separate sites, so that the data item can be retrieved even if some copies of the data item cannot be accessed due to system failures (Guerraoui and Schiper 1997). However, this benefit of data availability is only realized at the cost of elaborate algorithms that hide the underlying complexity of maintaining multiple copies of a single data item. The difficulty lies in keeping the copies consistent with each other while at the same time maximizing the data availability and performance. The algorithms which address these problems are called *replica control algorithms* (Bernstein, Hadzilacos, and Goodman 1987).

With failures, however, writing all copies within a transaction can cause indefinite blocking, which is unacceptable in practice. Hence the write-all approach can be modified to write all copies available to the transaction coordinator. Unavailable replicas receive changes on a deferred basis. The most commonly known protocol of this genre is the *primary copy protocol*. A two-phased commit protocol is required to guarantee a consistent view of the replica to obtain 1-copy-serializability (Bernstein, Hadzilacos, and Goodman 1987). To some extent, the basic principle of this protocol can be compared to our change algorithm with one master copy for data items, but we do not guarantee a consistent view of the replica since the changes are not executed in transaction boundaries. Also, for distributed DBSs it has been suggested to replace the 1-copy-serializability with, e.g., ϵ-serializability to allow asynchronous updates (Pu and Leff 1991). Temporary inconsistencies in replicas may be seen by queries with this asynchronous approach.

The integration of replicated information across autonomous subsystems within hospitals is only attainable by weakening the autonomy requirements of component DBSs. Therefore, a way has to be found for introducing global integrity maintenance without restricting local autonomy too much. Our approach preserves a high degree of local autonomy by applying mechanisms of active databases on the global level of integrity maintenance through decoupling triggered and triggering transactions.

Decoupling of triggered and triggering transaction of change operations is reasonable in HISs, because therein predominantly insertions of new information occur: you rarely see outdated information that has been updated somewhere else, you only see new inserted information later on. Therefore, the weaker consistency is acceptable in this environment. Additionally, *lazy* replication algorithms that asynchronously propagate replica changes to other subsystems *after* the updating transaction commits are less deadlock prone than *eager* replication algorithms that propagate replica changes *before* the updating transaction commits, because the transactions have shorter duration (Gray, Helland, O'Neil, and Shasha 1996). Figure 17 illustrates the conflicting goals in data replication. It is necessary to find compromises between these goals.

8 Conclusions

A distributed HIS is a complex *system of systems* which requires a well designed organization at the software architecture level. For digital information that is needed in hospitals, it is a major requirement to integrate the replicas of information about the same real world phenomenon which are stored in dissimilar and autonomous subsystems. Therefore, the requirements on integration of information for electronic medical records which are stored in HISs are rather different to the requirements on electronic versions of traditional libraries. However, the materials stored in electronic medical records can be regarded as digital libraries of patient-related information (Fox, Akscyn, Furutu, and Leggett 1995).

This paper presents our approach to federated integration of replicated information within hospitals where *tight* coupling is emphasized. An architecture which is based on the reference architecture for federated database systems (Sheth and Larson 1990) and adapted to the specific demands on integration of replicated information is presented. This architecture is the basis for associated algorithms that restore the integrity of replicated information when changes occur. The change algorithms are based on the schema architecture. This approach keeps these algorithms simple and the analysis of the dependencies within the schema architecture can be used to detect possibly infinite loops of change propagation or deadlocks.

The schema architecture is extended to support change algorithms with one or multiple master copies for data items. Multiple master copies for data items should be avoided (Gray, Helland, O'Neil, and Shasha 1996; Stead, Wiederhold, Gardner, Hammond, and Margolies 1992), but sometimes legacy systems have to be integrated which store the data in their own repositories. However, a federated architecture supports a smooth migration from legacy systems to modern information systems which do not require multiple master copies.

A prototype implementation of the presented software architecture which uses CORBA as the infrastructure to overcome distribution and heterogeneity on the operating system level, and ODMG-93 as a basis for the canonical data model is outlined. The use of design patterns for structuring the architecture is discussed.

Replication of information is the problem to be solved and not the suggested solution for the problems in todays HISs. We do not suggest replicating as much data as possible, but to integrate the replicated data across autonomous (legacy) systems which are already in use in the hospital. Here, you cannot avoid the replication of information without re-engineering the local systems. However, there exist also some advantages of a distributed architecture for HISs with data replication:

- Efficient data retrieval by reading local or close copies of data.
- Data availability in the presence of failures of individual subsystems: the failure of one subsystem in the network does not necessarily take the entire system down as long as no interaction with components which are out of order is required.
- Good effectiveness through a problem-oriented division of labor between subsystems.
- High scalability and flexibility.
- Good price/performance ratio.
- Low dependence on a single vendor.

Problems to be solved with distributed heterogeneous architectures are:

- Concurrency and security control is more difficult than in a centralized system.
- The effort for administration may be very costly, because it is necessary to manage several dissimilar systems.

In contrast to the current state of the art in connecting subsystems within hospitals through communication servers, a tightly coupled federated DBS whose data integration is on the basis of schema integration is capable of supporting subsystems to interoperate. Instead of enabling the subsystems to send messages for information exchange, the exchange of information is accomplished through updates of replicated data in subsystems by the federation layer, which *knows* the dependencies among replicas. This approach allows to analyze and optimize the data flow within hospitals.

Acknowledgement. The author would like to thank the members of our project group 'FOKIS' Andreas Dinsch, Mario Ellebrecht, Xi Gao, Sven Gerding, Betül Ilikli, Djamel Kheldoun, Patrick Koehne, Mischa Lohweber, Ulf Radmacher, Karl-Heinz Schulte, Dilber Yavuz, and Peter Ziesche for carrying out the prototype implementation, as well as Klaus Alfert and Stefan Sander for the cooperation in this project. The comments on drafts of this paper by Klaus Alfert, Ernst-Erich Doberkat and the anonymous referees were a valuable source to improve the paper.

References

Aiken, A., J. Hellerstein, and J. Widom (1995, March). Static analysis techniques for predicting the behavior of active database rules. *ACM Transactions on Database Systems 20*(1), 3–41.

Bakker, A., W. Hammond, and M. Ball (1995). Summary report of observations, conclusions and recommendation of the IMIA WG 10 Hospital Information Systems Working Group Conference, Durham, NC, August 1994. *International Journal of Bio-Medical Computing 39*(1), 11–15.

Ball, M. and M. Collen (Eds.) (1992). *Aspects of the computer-based patient record.* New York Berlin Heidelberg: Springer-Verlag.

Bancilhon, F., C. Delobel, and P. Kanellakis (1992). *Building an Object-Oriented Database System: The Story of O_2.* San Francisco: Morgan Kaufmann.

Bernstein, P., V. Hadzilacos, and N. Goodman (1987). *Concurrency control and recovery in database systems.* Reading: Addison-Wesley.

Bidgood, W. and S. Horii (1992). Introduction to the ACR-NEMA DICOM standard. *Radiographics 12*, 345–355.

Breitbart, Y., H. Garcia-Molina, and A. Silberschatz (1992). Overview of multidatabase transaction management. *VLDB Journal 1*(2), 181–293.

Bright, M., A. Hurson, and S. Pakzad (1992). A taxonomy and current issues in multidatabase systems. *IEEE Computer 25*(3), 50–60.

Bronzite, M. (1989). *Introduction to Oracle.* London: McGraw-Hill.

Campbell, K. and M. Musen (1992). Representation of clinical data using SNOMED III and conceptual graphs. In M. Frisse (Ed.), *Proc. Sixteenth Annual Symposium on Computer Applications in Medical Care*, Baltimore, MD, pp. 354–358. London: McGraw-Hill.

Cattell, R. (Ed.) (1996). *The Object Database Standard: ODMG-93, Release 1.2.* San Francisco: Morgan Kaufmann.

CEN TC/251 Working Group 1 (1997, January). Healthcare information system architecture part 1 (HISA): healthcare middleware layer. European prestandard, CEN European Committee for Standardisation. (available from `www.imc.exec.nhs.uk:8000/tc251/wg1`).

Date, C. (1995). *An introduction to database systems* (6th ed.). Reading: Addison-Wesley.

Ehlers, C.-T., H. Schillings, and P. Pietrzyk (1992). HIS and integration. In A. Bakker, C.-T. Ehlers, J. Bryant, and W. Hammond (Eds.), *Hospital Information Systems: Scope – Design – Architecture*, pp. 49–56. Amsterdam: North-Holland.

Eliassen, F. and R. Karlsen (1991, December). Interoperability and object identity. *ACM SIGMOD Record 20*(4), 25–29.

Fowler, M. and K. Scott (1997). *UML Distilled: Applying the Standard Object Modeling Language.* Object Technology Series. Reading: Addison-Wesley.

Fox, E., R. Akscyn, R. Furutu, and J. Leggett (1995, April). Introduction to the special issue on digital libraries. *Communications of the ACM 38*(4), 23–28.

Gamma, E., R. Helm, R. Johnson, and J. Vlissides (1994). *Design Patterns – Elements of Reusable Object-Oriented Software.* Reading: Addison Wesley.

Getta, J. and L. Maciaszek (1995). Management of inconsistent information in federated systems. In M. Papazoglou (Ed.), *Proc. 14th International Conference on Object-Oriented and Entity-Relationship Modeling (OOER'95)*, Volume 1021 of *Lecture*

Notes in Computer Science, Gold Coast, Australia, pp. 412–423. Berlin Heidelberg New York: Springer-Verlag.

Ghezzi, C., M. Jazayeri, and D. Mandrioli (1991). *Fundamentals of Software Engineering*. Englewood Cliffs: Prentice Hall.

Gray, J., P. Helland, P. O'Neil, and D. Shasha (1996, June). The dangers of replication and a solution. *SIGMOD Record 25*(2), 173–182. (Proc. ACM SIGMOD International Conference on Management of Data).

Guerraoui, R. and A. Schiper (1997, April). Software-based replication for fault tolerance. *IEEE Software 30*(4), 68–74.

Hammond, W. (1993). Health Level 7: A protocol for the interchange of healthcare data. In G. D. Moor, C. McDonald, and J. van Goor (Eds.), *Progress in Standardization in Health Care Informatics*, pp. 144–148. Amsterdam: IOS Press.

Hasselbring, W. (1997a, June). Extending the schema architecture of federated database systems for replicating information in hospital information systems. In S. Conrad, W. Hasselbring, A. Heuer, and G. Saake (Eds.), *Engineering Federated Database Systems EFDBS'97 (Proc. International CAiSE'97 Workshop)*, Barcelona, Spain, Computer Science Preprint 6/1997, University of Magdeburg, pp. 33–44.

Hasselbring, W. (1997b, September). A federated schema-based middleware architecture for hospital information systems. In *Proc. Conference on Architectural Concepts for Hospital Information Systems (New Technologies in Hospital Information Systems)*, Ulm, Germany. Amsterdam: IOS Press.

Hasselbring, W. and A. Kröber (1997). Combining OMT with a prototyping approach. *The Journal of Systems and Software*. (to appear).

Hasselbring, W. and P. Ziesche (1997, September). The use of design patterns in the development of a federated hospital information system with CORBA. In *Proc. 'Verteilte Objekte in Organisationen' (Mobis '97)*, Bamberg. Rundbrief Informationssystem-Architekturen. (in German).

HL7 Group (1994, December). Health level seven: an application protocol for electronic data exchange in healthcare environments, version 2.2. Technical report, Health Level Seven, Inc., Ann Arbor, USA.

Imielinski, T. and B. Badrinath (1994, October). Mobile wireless computing: Challenges in data management. *Communications of the ACM 37*(10), 18–28.

Inmon, W. (1996, November). The data warehouse and data mining. *Communications of the ACM 39*(11), 49–50.

Jacquemot, C., P. S. Jensen, and S. Carrez (1995). CHORUS/COOL: CHORUS object oriented technology. In *Object-Based Parallel and Distributed Computation (OBPDC '95)*, Volume 1107 of *Lecture Notes in Computer Science*, pp. 187–204. Berlin Heidelberg New York: Springer-Verlag.

Kudrass, T., A. Loew, and A. Buchmann (1996, June). Active object-relational mediators. In *Proc. First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, Brussels, Belgium, pp. 228–239. Piscataway: IEEE Computer Society Press.

Labio, W. and H. Garcia-Molina (1996, September). Efficient snapshot differential algorithms for data warehousing. In *Proc. 22th International Conference on Very Large Data Bases*, Bombay, India, pp. 63–74. San Francisco: Morgan Kaufmann.

Lange, M., N. Osada, H.-U. Prokosch, and W. Hasselbring (1996, September). An approach to classify and compare communication servers. In *Abstracts of the 41. GMDS-Jahrestagung*, Bonn, Germany. (in German).

Lindberg, D., B. Humphreys, and A. McGray (1993). The unified medical language system. *Methods of Information in Medicine 32*, 281–291.

McDonald, C. (1995). News on U.S. health informatics standards. *M.D. Computing 12*(3), 180–186.

Meyer, B. (1997). *Object-oriented Software Construction* (second ed.). Englewood Cliffs: Prentice Hall.

Mowbray, T. and R. Zahavi (1995). *The Essential CORBA: Systems Integration Using Distributed Objects*. New York: Wiley.

Özsu, M. T. and P. Valduriez (1991). Distributed database systems: where are we now? *IEEE Computer 24*(8), 68–78.

Pitoura, E., O. Bukhres, and A. Elmagarmid (1995, June). Object orientation in multidatabase systems. *ACM Computing Surveys 27*(2), 141–195.

Project Group FOKIS (1997, August). A federated object-oriented hospital information system. Project Report, University of Dortmund, Dept. Computer Science. (in German).

Pu, C. and A. Leff (1991, June). Replica control in distributed systems: an asynchronous approach. *ACM SIGMOD Record 20*(2), 377–386.

Radeke, E. and M. Scholl (1995, March). Functionality for object migration among distributed, heterogeneous, autonomous database systems. In *Proc. 5th International Workshop on Research Issues in Data Engineering: Distributed Object Management (RIDE-DOM '95)*, Taipei, Taiwan, pp. 58–66. Piscataway: IEEE Computer Society Press.

Rational Software Corporation (1997, January). The Unified Modeling Language. Documentation Set Version 1.0, Santa Clara, CA. (available from `www.rational.com`).

Rishel, W. and J. Quinn (1996, March). Software components, the clinical workstation and healthcare networks: How HL7 is helping you get there. In *Proc. Healthcare Information and Management Systems Society's Annual Conference*, Atlanta, Georgia.

Roantree, M., P. Hickey, A. Crilly, J. Cardiff, and J. Murphy (1996, October). Metadata modelling for healthcare applications in a federated database system. In O. Spaniol, C. Linnhoff-Popien, and B. Meyer (Eds.), *Trends in Distributed Systems: CORBA and Beyond, International Workshop TreDS '96*, Volume 1161 of *Lecture Notes in Computer Science*, Aachen, Germany, pp. 71–83. Berlin Heidelberg New York: Springer-Verlag.

Rumbaugh, J., B. Michael, P. William, E. Frederick, and L. William (1991). *Object-Oriented Modelling and Design*. Englewood Cliffs: Prentice Hall.

Saltor, F., M. Castellanos, and M. García-Solaco (1991). Suitability of data models as canonical models for federated databases. *SIGMOD Record 20*(4), 44–48.

Sander, S. and W. Hasselbring (1996, December). Realizing a federated database system based on the standards CORBA and ODMG-93. In W. Hasselbring (Ed.), *Abstracts 2. Workshop "Föderierte Datenbanken"*, pp. 45–50. Software-Technik Memo Nr. 90, University of Dortmund. (in German).

Schatz, B., W. Mischo, T. Cole, J. Hardin, A. Bishop, and H. Chen (1996, May). Federating diverse collections of scientific literature. *IEEE Computer 29*(5), 28–36.

Schmitt, I. and G. Saake (1995). Managing Object Identity in Federated Database Systems. In M. Papazoglou (Ed.), *Proc. 14th International Conference on Object-Oriented and Entity-Relationship Modeling (OOER'95)*, Volume 1021 of *Lecture Notes in Computer Science*, Gold Coast, Australia, pp. 400–411. Berlin Heidelberg New York: Springer-Verlag.

Sheth, A. and J. Larson (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Computing Surveys 22*(3), 183–236.

Shortliffe, E., L. Perreault, L. Fagan, and G. Wiederhold (Eds.) (1990). *Medical informatics: computer applications in health care*. Reading: Addison-Wesley.

Sokolowski, R. (1997, January). Integration services for clinical data exchange. OMG/CORBAmed DTF White Paper 97-01-01, Kurzweil Applied Intelligence. (available as `ftp.omg.org/pub/archives/docs/corbamed/97-01-01.pdf`).

Stead, W., G. Wiederhold, R. Gardner, W. Hammond, and D. Margolies (1992). Database systems for computer-based patient records. In M. Ball and M. Collen (Eds.), *Aspects of the computer-based patient record*, pp. 83–98. New York Berlin Heidelberg: Springer-Verlag.

Tanenbaum, A. (1996). *Computer Networks* (third ed.). Englewood Cliffs: Prentice Hall.

The ACT-NET Consortium (1996, September). The active database management system manifesto: a rulebase of ADBMS features. *SIGMOD Record 25*(3), 40–49.

Tresch, M. and M. Scholl (1994, September). A classification of multi-database languages. In *Proc. 3rd International Conference on Parallel and Distributed Information Systems*, Austin, TX, pp. 195–202. Piscataway: IEEE Computer Society Press.

Türker, C. and S. Conrad (1997, July). Towards Maintaining Integrity of Federated Databases. In *Proc. 3rd International Workshop on Information Technology (BIWIT'97)*, Bayonne, France.

Ullrich, I., W. Hasselbring, T. Jahnke, A. Röser, and A. Christmann (1996, September). An object-oriented system for therapy control in angiopathy. In *Abstracts of the 41. GMDS-Jahrestagung*, Bonn, Germany. (in German).

Widom, J. and S. Ceri (Eds.) (1996). *Active Database Systems – Triggers and Rules For Advanced Database Processing.* San Francisco: Morgan Kaufmann.

Wiederhold, G., M. Bilello, V. Sarathy, and X. Qian (1996, October). A security mediator for health care information. In *Proc. 1996 AMIA Conference*, Washington, DC, pp. 120–124.

Wong, S. and H. Huang (1996, July). A hospital integrated framework for multimodal image base management. *IEEE Trans. Systems, Man, and Cybernetics 26*(4), 455–469.

Zhuge, Y., H. Garcia-Molina, J. Hammer, and J. Widom (1995, June). View maintenance in a warehousing environment. *SIGMOD Record 24*(2), 316–327. (Proc. ACM SIGMOD International Conference on Management of Data).

This article was processed by the author using the LaTeX style file *cljour2* from Springer-Verlag.