Complex Situation Monitoring in a CORBA- and WWW-based Federation Architecture for Heterogenous Information Systems

Arne Koschel, Ralf Kramer
Forschungszentrum Informatik (FZI)
Haid–und–Neu–Str. 10–14, D–76131 Karlsruhe
{koschel, kramer}@fzi.de

1 Introduction

In many application areas, there is a need to access, to combine, and to exploit information residing in a range of autonomous, heterogeneous systems. Typically, these systems have been developed independently. They are often distributed accross networks, using different operating systems, are devolped in different programming languages etc. For example, decisions during the engineering of new products have to take into account previous designs, parts catalogues, offerings from other companies, etc. Such information may reside in different storage systems, e.g., file systems, relational databases, object-oriented databases, engineering data management systems, etc. Similarly, within workflow systems, data warehouses or comprehensive environmental information systems different information sources have to be taken into account. So, alltogether it is necessary for users of such systems to be able to access heterogenous information sources (data and non-data) but also often need to be informed automatically when new information which is relevant for their work becomes available. For example, in our environmental systems projects a civil servant has to be informed when a parameter measured in the air exceeds a certain threshhold. Thereby, detection of a critical situation requires monitoring of several data sources (e.g. a database recording measurement values and a database with limit values). Monitoring may is needed for rather complex situation detection, which requires, e.g., information access from several (also non-database) sources or even invocation of processing tools during checks for complex situations.

To deal with the integration of heterogenous, distributed information sources we have developed a federation architecture which follows the idea of federated DBMS [BHP92, SL90] for loosely coupled, potentially heterogeneous, and autonomous databases. We extend this idea in our federation architecture into more general information sources. Furthermore we introduced a complex situation monitoring unit for this environment. Our architecture is based on the following techniques and tools:

1. The Object Managment's Group Common Object Request Broker Architecture (CORBA), the open industrial standard for object-oriented, distributed information processing is used for the technical integration of information sources. CORBA gives a solid basis, since it already covers the heterogeneity of operating systems, programming languages and networks. However, CORBA covers the heterogeneity of information sources mainly by defining the concept of IDL-wrappers for them. These wrappers allow for the definition of syntactically uniform interfaces to information sources (using CORBA's Interface Definition Language, IDL).

2. World-Wide Web (WWW) techniques and tools (HTML, HTTP/CGI, Java) are used to enable clients without specific software tools to access information sources in WANs, especially beeing platform independent for the end user interface.

3. Active DBMS style ECA-like rules are used for complex situation monitoring and active information delivery. Our approach, fully described in [vBKK96b], allows for ECA-like rule definitions for arbitrary (not only data) information sources as event producers and it is also possible to revert to this sources (read access, tool invocation) during condition processing.

We assume the reader to be familiar with the concepts of CORBA, the WWW and active DBMS. For overviews on CORBA and CORBA implementations refer to [Adl95, Sig96, MZ95, KKTvB96, KL96].

For WWW technology and tools including Java and Java/Corba we just refer [Wor, Jav, HKNW96]. For an introduction to ECA-rule processing and the terminology used in active database systems refer to [WC96, DGG95].

Space limitations restrict a discussion of related work here. We discussed related work for Monitoring respective ECA-rules and CORBA in [vBKK96b, vBKK96a]. Work mentioned there includes [PV96, SLY+95, ZHKF95, Wid95]. ECA-rule processing for a federated database architecture is also described in [TC96].

The rest of the paper is organized as follows: In sec. 2 the federation architecture is described. This is followed by the complex situation monitoring in sec. 3, with implementation issues and examples in 4. Eventually in sec. 5 a conclusion and some outlook is given.

2 A Federation Architecture Using CORBA and WWW Technology

Our work for a federation architecture is based upon projects within the area of environmental information systems. Within our federation architecture [KKN+96, KNK+96], several information sources like databases, e.g. the meta-information system UDK [LSG96] and a limits value database, a geographical information system, processing tools, etc. can be accessed. End user navigation to the sources is provided using the UDK. Furthermore, the CORBA-based middleware layer allows to overcome some of the deficiencies of the World-Wide Web, e.g., the lack of combining the results of several DBMS queries in a single result HTML page. Because of the application demands, our system has primarily to work as an information system, our access to information sources is - except for internal intermediate data - read-only, thus without global transactions.

Figure 1 presents the current version of this federation architecture that also takes into account most recent combinations of newly emerged technologies like Java/Java applets together with CORBA and "classical" WWW technology. The implementation state of system components differs, ramging from prototypes to fully implemented, stand-alone usable parts.

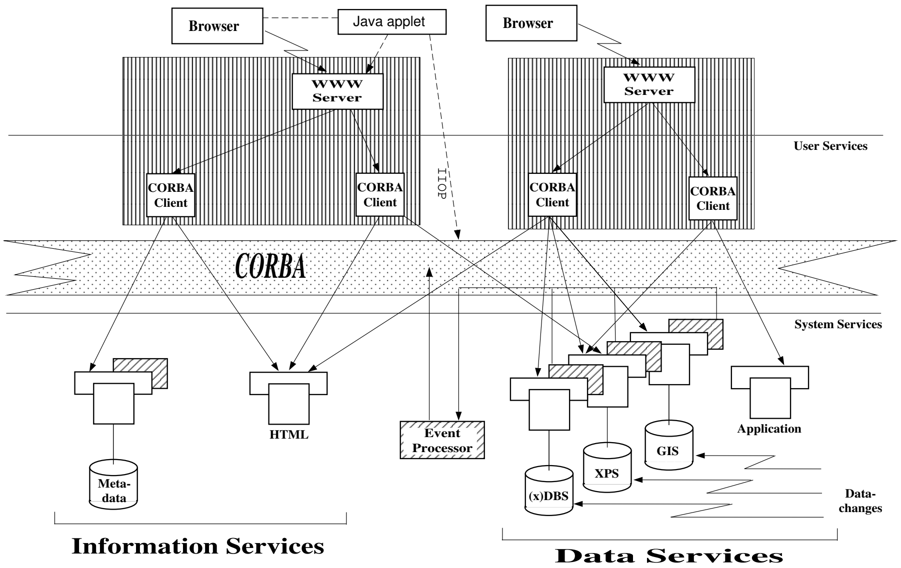

Figure 1: Federation Architecture incorporating the World-Wide Web, CORBA, and Java

Horizontally, we distinguish between user level and system level services. System level services provide basic functionality such as accessing databases or geographic informations systems and preparing HTML pages. Also non-data sources, like calculation applications, are integrated. Application level services basically combine several system level services into higher level services that are eventually made available to the user. Java applets communicate with their server site based counterparts using CORBA's stateful IIOP communication protocol.

Vertically, we distinguish between information and data services. Information services are based on

UDK meta data. They help the user to find relevant environmental data. Both information services and data services deliver results that in turn are either used as input for the generation of HTML pages or are directly presented in Java applets.

3 Complex Situation Monitoring Using CORBA and ECA-Rules

Integrated into the CORBA environment of our federation architecture is our approach for complex situation monitoring of the components, which we will describe here briefly. As stated, CORBA handles technical heterogeneity and encapsulates (event) information sources using wrappers (see also [vBKK96a]). The problem to handle semantic heterogeneity in informations source is only addressed by solutions that are already available. This is mainly the use of meta-data to locate and interpret available information (sources).

3.1 Approach

The key problem we address in our approach is how to support complex situation monitoring. In general, a situation is characterized by a combination of update events which occurred at some components, and a certain distributed state. Checking this state may require the invocation of processing tools or read access to other information sources. We distinguish two general approaches on how to perform situation monitoring:

- In the *pull approach,* components register update events in their local state such that they become accessible from the outside via a CORBA object wrapper for this component. A situation detector queries the components at regular time intervals in order to check whether a particular situation has occurred (possibly more than once). This includes checking state conditions and checking whether the required events have occurred. To avoid bottlenecks, different situation detectors might check for different kinds of situations.

- In the *push approach,* components signal update events directly to situation detectors. When a complex event (characterized by a combination of update events and possibly further events like completion of a time interval) occurs, the situation detector checks the state conditions and initiates appropriate actions. Note that this approach resembles event processing in active database systems.

As our application domain does not require actions to be taken immediately when a situation occurs, both approaches are applicable. The advantage of the second approach is that state checking is only initiated when the corresponding event has occurred. Furthermore, a possible advantage of the first approach, namely limiting the number of event signalling messages by setting time intervals, can also be transferred to the second approach by signalling several events within one message in case of a flood of events or by pre-filtering events in wrappers.

Therefore, we concentrate on the push approach. In principle, it can be implemented by hard-coding event processing within component wrappers and situation detectors, or by providing some kind of event-condition-action (ECA) rules within the CORBA environment. Since only the latter alternative provides the flexibility required, we have chosen to provide an ECA-rule like service.

OMG also envisions CORBAservices and CORBAfacilities which can support such a service, e.g. event channels to pass events between CORBA objects. A rule facility was planned but has been declined for now. However, our work can be viewed as a proposal to provide ECA-rules for CORBA-based environments.

3.2 Rule Model and Execution Model

When designing a CORBA-based ECA-rule service a rule model and an execution model have to be developed. The rule model specifies the kinds of events, conditions, and actions provided by the system. The execution model specifies how rules are processed, i.e., event detection, condition evaluation, action execution, and the relationships among these steps. In defining these models, we can draw upon technology that has been developed for active database systems. This technology must however, be modified to meet the specific requirements of a distributed CORBA environment.

Feature	Design Choice	
Events	DB events:	triggers for updates
	CORBA meth.:	pre/post method
	CORBA env.:	ORB exceptions
	Composite:	conjunction, disjunction
Conditions	Predicates incl. CORBA method calls	
Actions	Sequence of CORBA method calls	
Coupling	Asynchronous decoupled processing	
Event bind.	DB events:	instance- or set-oriented
	CORBA ev.:	instance-oriented
Visible state	Actual state (cond. eval. / action exec.)	
Event cons.	Local consumption per rule	

Table 1: Rule Model and Execution Model

The rule model has to be aligned with the underlying data model. Within a CORBA environment, this data model is provided by the Interface Definition Language (IDL) which serves to specify the interfaces of the objects, i.e., both the method signatures and public attributes. Similar to active ooDBMS, primitive events include method invocation. Conditions are checked by means of predicates involving CORBA method calls and actions are performed by invoking methods (see Tab. 1).

Specific problems regarding the design of the rule model arise due to the distributed system and the autonomy of the components being integrated via CORBA. Distribution implies that there is no global time. Thus, events can only be temporally ordered according to the degree to which clocks can be synchronized. The event model has to reflect the fact that events cannot be completely ordered. At present, we avoid this problem by not supporting the detection of global temporal orderings in event composition. Later, we plan to incorporate the more elaborate approach of [Sch96] to distinguish between concurrent events and ordered events.

Autonomy and heterogeneity of the components implies that interesting events occur not only at the level of CORBA methods, but also within the components themselves. For example, an insert, update or delete to a relational table may be important, but not directly visible in the higher-level object invocation at the CORBA level which initiated the operation. Therefore, components must be able to raise such internal events. Our event model captures these events as abstract events which can be explicitly raised at the CORBA level.

The execution model has to observe the temporal relationship between events, conditions and actions. Within active database systems the execution model relies heavily on the notion of a transaction. Coupling modes specify whether actions are executed immediately, or deferred to the end of the same transactions, or decoupled to a separate transaction. In distributed applications similar to ours, we have to take into account, that not all components support transactions. Hence we must employ a simpler model based on immediate event signalling after completion of an atomic action: Events are signalled by a component either directly or, in case they occur inside the scope of a transaction, once the transaction has been completed. Action execution is then decoupled, with “exactly once” as default semantics, which may relax to “at most once” in less critical situations (e.g., information refreshment at regular time intervals).

During event detection, event occurences are bound to data items. Thus, data can be passed to condition checking and action execution. In our environment, these data items are either values or references to CORBA objects. Hence, it is possible to refer to these objects during rule processing. Because we cannot rely on transactional support, we must inspect the actual state of an object (as opposed to the state of the object when the event occured). In principle, our event binding strategy is instance oriented. However, a component may choose to collect several update events and signal them on the CORBA level as a single abstract event which is bound to the set of updated data instances, for example.

Table 1 summarizes our choices in the design of the rule model and execution model. Regarding the rule processing cycle, issues like event consumption, rule ordering or parallelism of rule execution have deliberately been left open in our architecture so that we may modify execution strategies in a flexible way, thus permitting us to experiment with different strategies.

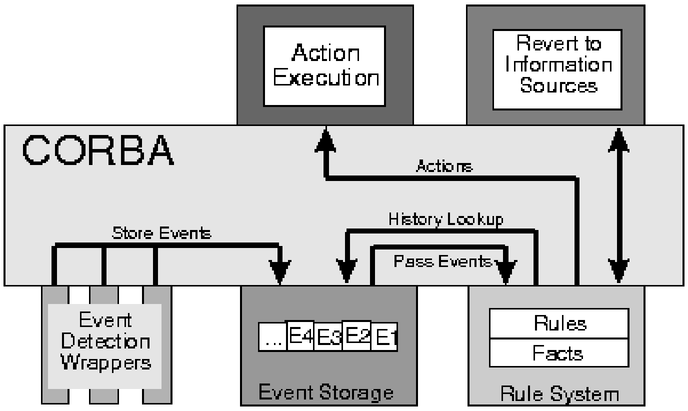

Figure 2: Functional Components for (Distributed) Event Processing

3.3 System Architecture

Our goal in developing the system architecture was to provide a flexible infrastructure for easy implementation of different execution strategies. Hence, we concentrated on the development of a set of basic components which can be replicated for enhanced parallelism to avoid bottlenecks and on which different execution models can be mapped.

The basic building blocks for any system for ECA-rule processing are primitive event detection, complex event composition and rule execution. The implementation and interaction of these building blocks depend on the particular execution model.

A specific problem in our environment arose from the fact that primitive events are detected at different components in the distributed system and have to be captured in a reliable way. Therefore, we have added a component for persistent event storage. A further benefit of this component is that it allows replication of the capturing of primitive events and the composition of complex events independently.

- In the first prototype, the detected events of all wrapped sources are passed on to event storage (see Fig. 2). We decided to use a (persistent) queue structure for reliable storage. Hence, we capture the event history by sequentially ordering the incoming events, thus allowing an easy first-in, first-out processing strategy to be used. Further, we define a uniform event structure, a canonical event model, as the "output stream" of this component, which makes further processing easier.

- Our rule component is responsible for complex event detection and rule processing (these two aspects could easily be separated, though). In order to be able to implement different execution models easily, we decided to use an expert system shell providing production rules. How we map ECA rules to production rules and how we revert to sources during condition checking and action execution is explained in detail in sec. 4.

Since all components in our architecture are wrapped using CORBA-IDL, we have the benefit of a (syntactically) uniform description. This allows easy integration of new components or replication of existing components.

4 Implementation Issues and Examples

In this section, we describe some implementation issues of the complex situation monitoring in more detail in order to give an impression of the overall system. For our current system, the CORBA implementations ORBeline [Pos94] and Orbix [ION95] have been used. Both have C++ language bindings, which also allow easy calls to procedures written in C. Because of space limitations we concentrate on the description of the rule processing strategy in sec. 4.1. Wrappers for event monitoring and access to information sources are more detailed described in [vBKK96b, vBKK96a]. We finish in sec. 4.2 with two application examples. The simpler one is used to illustrate an ECA-like rule and its processing in our system.

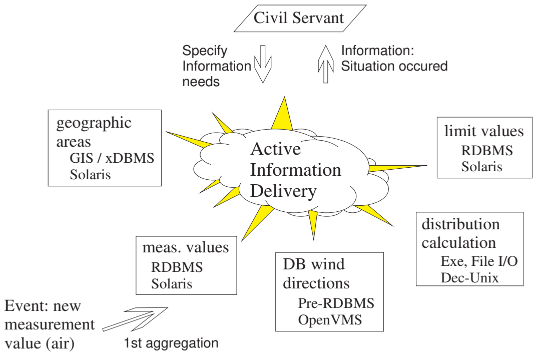

Figure 3: A Complex Situation

4.1 Rule Processing – Situation Detection

As already stated, we use a production-rule expert system shell (XPS) as basis for our rule processing unit. We decided to use the CLIPS system [CRD93a, CRD93b, Ril95], because the system is robust, written in C, and portable. The system is extensible by so-called user-defined C functions (and it is free of charge). We get a rule management system with the creation, deletion, etc. of rules, (almost) for free by using appropriate CLIPS functionality. CLIPS supports the usual form of production rules, which consist of a condition part and an action part. The condition part reacts to the existence of facts of a certain structure, in our case especially the existence of event facts.

Using CLIPS as its basis, the rule processing unit of our system currently works as follows: As usual in CORBA, we encapsulate CLIPS by using a CORBA wrapper. This CLIPS wrapper contains a CORBA-method, which is called by the event storage component (see Fig. 2). This method maps an incoming event to a uniform event fact structure. CLIPS provides so-called fact templates for such structures, which are similar to C `struct` definitions. An event fact contains the usual primitive event information like type of event, creation time, parameters etc. This event fact structure is now written into the CLIPS fact list.

With the incoming event fact in the fact list, we have the basis for the event part of an ECA rule. We can now define CLIPS rules which react to the existence of a new event fact. These rules check whether an ECA-rule matching the event fact exists by testing the above mentioned primitive event information. For the ECA-rule condition part, we use further CLIPS rules which gather event information from event facts and subject them to the usual comparisons, use predefined functions on them, etc. For this purpose, CLIPS provides a broad range of mechanisms. As an important addition to the predefined mechanisms, we provide user-defined CLIPS functions, which we use to call our information source read wrappers. These wrappers return their results to the fact list, so rules can again act on these facts. Finally, when all rule conditions have been satisfied, we invoke CORBA methods in the form of user defined C functions for the ECA-rule action part. Hence, by combining all these CLIPS production rules for event dectection, condition checking, and action execution, we obtain the implementation of an ECA-rule.

Since we are able to preserve the current state of the processing of one ECA-rule during intermediate processing of new events, we can also detect complex events by defining CLIPS production rules which define a finite state machine.

4.2 Application Examples

To give an idea of how system usage proceeds, we give two examples here. Figure 3 shows a complex application scenario. A new measurement value arrises in a databse, giving the event. During condition checking a GIS has to be queried for the location area of the measurement value, a processing tool has to be invoked for a spreading calculation and results from the calculation hav to be compared against limit values from another database. Finally, if limit values are exceed, a mail to inform the user (action) has to be send.

Part of these scenario give in fig. 4 an example on processing in our system. Consider for the processing in our system the arrival of a new ozone measurement value x in a measurement database DB-1. This ozone value has to be compared with a limit value y for ozone levels stored in database DB-2. If the ozone value exceeds its limit value, an alarm has to be sent (perhaps via email).

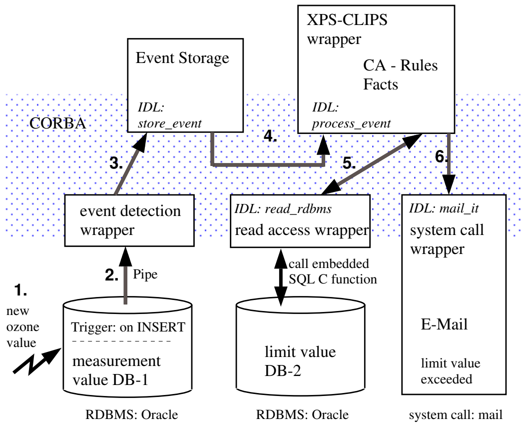

Figure 4: ECA like Rule Processing for the Example Situation

An ECA-rule for this situation looks (in pseudo code) as follows:

ECA rule:
ON [insertion of new ozone measurement value x in DB-1]
IF [$x \geq (y =$ limit value read f. DB-2)]
DO [method call for alarm]

The processing of the situation in our system is demonstrated in Figure 4. The figure shows five components involved: the two databases and their CORBA wrappers, the event storage, the rule processing unit, and a wrapper for a system call to a mail program. Processing proceeds as follows:

1. A new ozone measurement value arrives. It is inserted into DB-1 via an external application.

2. The trigger for ozone values recognizes the `INSERT` event and passes the value and other description parameters via an Oracle pipe to the event detection wrapper.

3. The event detection wrapper, which is a CORBA object as well, inspects the Oracle pipe and passes the event information to the event storage. This storage is another CORBA object, which collects the events from several sources. It provides an IDL method for storing the incoming events. This method is called synchronously by the event detection wrapper.

4. From event storage, the event is passed to the CORBA wrapper for CLIPS. The event parameters are stored in a CLIPS fact structure and a fact is created. For our example, this fact includes slots for a type (RDBMS event), a source (DB-1), an operation (`INSERT`), an attribute value (measurement value), etc. A CLIPS rule reacts to this information and creates a fact indicating that the event part of the example ECA rule is fulfilled.

5. The next CLIPS rules reacting to this newly detected event comprise the condition part of the ECA-rule. The first one issues a call to the RDBMS read access wrapper of DB-2, querying the limit value for ozone levels and passing it back to CLIPS, again creating a fact with this information. This fact is checked by another CLIPS rule, which actually checks the ECA-rule condition *new ozone value* $\geq$ *ozone limit value*.

6. If this condition is satisfied, the action this rule takes is a method invocation in the system call wrapper, which issues a mail command to inform the user.

5 Conclusions, Current, and Future Work

In this paper we described a federation architecture for heterogenous, distributed information systems, based upon CORBA and WWW technology. We furthermore described an ECA-like rule processing unit for CORBA environments, as part of the whole architecture. Using the combination of CORBA and CLIPS for the ECA-like rules, we provided a flexible environment, allowing the implementation of higher-level rule processing in an effective way.

Our current work concentrates on two aspects. The first is to fill the architecture with more information sources also developing the prototype parts (e.g. the GIS part) further. Second, based especially on the work described in [vBKK96b, vBKK96a] a Configurable Set of Services for Active Functionality in CORBA-based Heterogenous Information Systems is proposed by us. It will be especially suited for situation detection in such CORBA-based information systems, and will include configurations ranging from pure event monitoring services up to ECA rules for complex situation monitoring. For this environment it will be tested, up to which extend active DBMS like ECA parameters and semantics are supportable, different service set architectures will be provided, and protocols between services will be exchangable. The focus is to provide this in and for a CORBA based environment, thus build upon functionality provided by CORBA (ORB and Services), but taking also restrictions arrising from this as given, e.g. allowing not for all active DBMS like ECA parameters and supporting possibily (almost) no transaction capabilities.

References

[Adl95] Richard M. Adler. Emerging Standards for Component Software. *IEEE Computer*, March 1995.

[BHP92] M. W. Bright, A. R. Hurson, and S. H. Pakzad. A taxonomy and current issues in multidatabase systems. *IEEE Computer*, pages 50–60, March 1992.

[CRD93a] C. Culbert, G. Riley, and B. Donnell. *CLIPS Reference Manual Volume 1, Basic Programming Guide, CLIPS Version 6.0.* Software Technology Branch, Lyndon B. Johnson Space Center, NASA, 1993.

[CRD93b] C. Culbert, G. Riley, and B. Donnell. *CLIPS Reference Manual Volume 2, Advanced Programming Guide, CLIPS Version 6.0.* Software Technology Branch, Lyndon B. Johnson Space Center, NASA, 1993.

[DGG95] K.R. Dittrich, S. Gatziu, and A. Geppert, editors. The active database management system manifesto: A rulebase of ADBMS features. In T. Sellis, editor, *Proc. Rules in Database Systems, Second International Workshop, RIDS'95, Glyfada, Athens, Greece*, number 985 in Lecture Notes in Computer Science, pages 3–17, Berlin, Germany, September 1995. ACT-NET, Springer.

[HKNW96] D. Hoffmann, A. Koschel, R. Nikolai, and M. Wipf. Integration von CORBA und JAVA - Aktuelle Projekte und praktische Erfahrungen. In R. Kramer, R. Nikolai, and C. Rolker, editors, *Java-Evaluierung für das UIS Baden-Württemberg – Phase 1*, number 5/96. Forschungszentrum Informatik (FZI), Karlsruhe, Germany, May 1996.

[ION95] IONA Technologies Ltd. *Orbix: Programmer's Guide.* 8-34 Percy Place, Dublin 4, IRELAND, 1995.

[Jav] JavaSoft. Homepage. http://www.javasoft.com/.

[KKN+96] Arne Koschel, Ralf Kramer, Ralf Nikolai, Wilhelm Hagg, and Joachim Wiesel. A Federation Architecture for an Environmental Informaton System incorporating GIS, the World-Wide Web, and CORBA. In *Third International Conference/Workshop Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA, January 1996.

[KKTvB96] Arne Koschel, Ralf Kramer, Dietmar Theobald, and Günter v. Bültzingsloewen. Evaluation and Application of CORBA implementations. In *ECOOP'96 Workshop: Putting Distributed Objects to Work, 10th European Conference on Object-Oriented Programming*, Linz, Austria, July 1996.

[KL96] Arne Koschel and Philipp Leibfried. Experiences with the CORBA-Implementation DSOM. In *Proc. International Conference on Distributed Platforms, ICDP'96, Industrial Stream.* IFIP, IEEE, February 1996.

[KNK+96] Ralf Kramer, Ralf Nikolai, Andree Keitel, Rudolf Legat, and Konrad Zirm. Enhancing the Environmental Data Catalogue UDK for the World-Wide Web. In *Proc. 10th International Symposium Umweltinformatik'96*, Hannover, Germany, September 1996.

[LSG96] H. Lessing, W. Swoboda, and O. Günther. UDK: A European Environmental Data Catalogue. In *Third International Conference/Workshop Integrating GIS and Environmental Modeling*, Santa Fe, New Mexico, USA, January 1996.

[MZ95] T. J. Mowbray and R. Zahavi. *The Essential CORBA*. John Wiley & Sons, Inc., New York, U.S.A, 1995.

[Pos94] PostModern Computing. *ORBeline User's Guide*. PostModern Computing Technologies, Inc., 1897 Landings Drive, Mountain View, California 94043 U.S.A, 1994.

[PV96] N. Pissinou and K. Vanapipat. Active Database Rules in Distributed Database Systems. *International Journal of Computer Systems*, 11(1):35–44, January 1996.

[Ril95] G. Riley. What are Expert Systems, What is CLIPS. In *CLIPS World Wide Web Home Page*. http://www.jsc.nasa.gov/ clips/CLIPS.html, 1995.

[Sch96] S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. PhD thesis, Selwyn College, University of Cambridge, University of Cambridge, Computer Lab, Cambridge, United Kingdom, 1996.

[Sig96] J. Sigel. *CORBA Fundamentals and Programming*. John Wiley & Sons, Inc., New York, USA, 1996.

[SL90] Amit P. Sheth and James A. Larson. Federated database systems for managing distributed, heterogenous, and autonomous databases. *ACM Computing Surveys*, 22(3):183–236, September 1990.

[SLY+95] S. Su, H. Lam, T. Yu, S. Lee, and J. Arroyo. On Bridging and Extending OMG/IDL and STEP/EXPRESS for Achieving Information Sharing and System Interoperability . In *Fifth Annual Express User Group Infernational Conference (EUG '95)*, Grenoble, France, October 1995.

[TC96] C. Türker and S. Conrad. Using Active Mechanisms for Global Integrity Maintenance in Federated Database Systems. In S. Conrad, M. Höding, S. Janssen, and G. Saake, editors, *1. Workshop föderierte Datenbanken*, Magdeburg, Germany, April 1996.

[vBKK96a] Günter von Bültzingsloewen, Arne Koschel, and Ralf Kramer. Accept Heterogeneity: An Event Monitoring Service for CORBA-based Heterogeneous Information Systems. submitted for publication, September 1996.

[vBKK96b] Günter von Bültzingsloewen, Arne Koschel, and Ralf Kramer. Active Information Delivery in a CORBA-based Distributed Information System. In Karl Aberer and Abdelsalam Helal, editors, *Proc. First IFCIS International Conference on Cooperative Information Systems (CoopIS'96)*, pages 218–227, Brussels, Belgium, June 1996. IFCIS, IEEE Computer Society Press, Los Alamitos, California.

[WC96] J. Widom and S. Ceri, editors. *Active Database Systems: Triggers and Rules for Advanced Database Processing*. The Morgan Kaufman Series in Data Management Systems. Morgan Kaufmann Publishers, Inc., San Francisco, California, U.S.A, 1996.

[Wid95] Jennifer Widom. Research Problems in Data warehousing. In *Proceedings of 4th Int. Conference of Information and Knowledge Management (CIKM'95)*, November 1995.

[Wor] World Wide Web Consortium. Homepage. http://www.w3.org/pub/WWW.

[ZHKF95] G. Zhou, R. Hull, R. King, and J. Franchitti. Supporting Data Integration and Warehousing Using H2O. *Data Engineering*, 18(2):29–40, June 1995.