An Overview of Souk Nets: A Component-based Paradigm for Data Source Integration[1].

William J. McIver, Jr.[*], Karim Keddara[**], Christian Och[*], Roger King[*], Clarence A. Ellis[**], John Todd[*], Nathan Getrich[*], Richard M. Osborne[*], Brian Temple[*].

University of Colorado
Database Research Laboratory[*] & Collaboration Technology Research Group[**]
Boulder, Colorado 80309-0430
contact: mciver@cs.colorado.edu

key words: data integration, interoperability, component-based software, mediators, beans.

Abstract

Construction of complex software systems with largely off-the-shelf components has become a reality with the wide availability and acceptance of component frameworks and distributed object systems. Component-based frameworks tailored specifically for the domain of database integration are lacking, however. To use an existing component framework, data integrators must construct custom components specialized to the tasks of the data integration problem at hand. This approach allows other components provided by the framework to be reused, but is overly tedious and requires the integrator to employ the programming paradigms assumed by the component framework for interconnection and intercommunication between components, and manipulation of data provided by them. This approach also does not lend itself to the use of components from other frameworks. A more desirable approach would be to use a framework that provides off-the-shelf data integration components, allows one to employ programming methods that are more natural to the domain of data integration, and can integrate components from other frameworks. This new framework should abstract away as much as possible the implementation details of integrating components from other frameworks. A vast amount of database research work can inform the factoring and construction of such a framework and its components. Souk is a language-independent, component-based paradigm for performing data integration in the context of common distributed object systems. It is designed to allow the rapid construction of data integration solutions from off-the-shelf components, and to allow flexible evolution. This extended abstract gives an overview of this paradigm and suggests how mediators can provide the containers for Souk Net-compliant components.

1. Background

This work addresses database, or data source, integration specifically through the use of a component-based framework and programming paradigm specifically designed to support the rapid construction of data integration solutions in the context of distributed object environments, and which can be easily adapted to evolve in the face of changes to the underlying local data sources and changing client requirements.

1. This work is supported by NSF grant 9806829 under the Experimental Systems Program.

The main goal of *database integration* is to create a *global database* which incorporates and encapsulates data from a community of discrete, possibly heterogeneous, *local databases*; and which can accept requests from clients to retrieve or update data managed by the community of local databases without regard to their location in a network of data sources, representation, data model, access methods, or languages required to properly specify the requests of the individual local databases [4]. Thus, in meeting this goal, the global database might be required to accept a query expression from a client, manage its translation into several sub-queries in different query languages or method invocations, retrieve the results, and convert them into the representation and data model required by the client. Furthermore, the global database is most likely constrained to respect the separate and independent design, control, and administration of the local databases with which it interacts in order to service client requests. For example, the global database may be required to adapt to independently initiated schema design changes to a local database, or the global database may not be able to implement global transaction semantics by accessing information about the transaction schedules of local data sources. This separate and independent design, control, and administration is called *local autonomy.* We will refer to local databases from now on using the more general term, *local data source,* to indicate that data to be integrated may not be managed by a proper database management system, but may be some other type of system, such as a file system or an application.

There are several dominant approaches to data source integration. First, the schemas from all local data sources are integrated into a *common data model* to form a *global schema*, and then queries in a corresponding *common query language* are performed against this global schema [2]. This approach becomes intractable when the number of local data sources becomes large. Second, the intractability of global schema integration can be dealt with by performing integration on only those parts of local schemas that are exported by local data sources [16]. Third, schema integration can be avoided altogether in lieu of a system that can accept requests in the form of global query language expressions or method invocations and translate them accordingly into the query languages or method invocations corresponding to the underlying local data sources required to satisfy the requests [7]. Finally, highly customized point solutions can be constructed based on the particularities of the local data sources to be integrated. These are costly and often not easy to adapt to evolution of the local data sources.

2. Our approach

Our approach to data integration is orthogonal to those discussed above: construction of data integration solutions using a component-based framework, but using a programming or modeling paradigm that allows methods that are more natural to the domain of data integration. Current component-based frameworks force the data integrator to use tedious, imperative programming techniques and tedious procedures for binding and communicating with other components or objects in the framework. The complimentary and problematic nature of data integrator needs and distributed object environments is being recognized by industrial data integration providers such as Cohera [10]. The collection of components in our framework allow the modularization of functionality used in well-known approaches to data integration, such as view construction, set operations, and data transformations. There is considerable overlap in the issues that must be addressed in each of the existing data integration approaches, including schema integration and translation, query decomposition and query language translation, maintenance of consistency and dependency constraints across local data sources, and global transaction processing. Data sources can and are now being wrapped or constructed using mature distributed object and component-based software systems. These realities can be factored to produce a covering set of components, and a paradigm for communication and coordination between them. The result can be a covering set of

components which allow the data integrator to rapidly construct solutions taking one of the dominant approaches discussed above.

3. Modeling Data Integration Solutions

Szyperski defines software components as binary units of independent production, acquisition, and deployment that interact to form a functioning system [27]. Independence allows for multiple independent developers, and the ability to integrate at the binary level ensures robust integration. Our definition of components are units which define specialized, prefabricated functionality whose instances can be combined with other components to construct solutions to data integration problems involving multiple, distributed data sources. Data sources may or may not be traditional database management systems. Data sources in this context are assumed to be wrapped using a technology such as CORBA [22].

The overriding requirement for our paradigm is that it support programming approaches that are natural to the domain of data source integration. Carriero and Gelerner's [8] taxonomy of conceptual classes of parallel programming approaches provide a useful start here. We can envision the development of a data integration in terms of the *results* of the data integration process, the agenda of tasks that must be performed to achieve integration, or the *ensemble of specialists* required to perform data integration. We believe that these three paradigms are relevant classifications for the predominant data integration approaches described above. We do not suggest that each data integration approach falls squarely within one of these parallel programming approaches; rather, different aspects of each data integration approach are similar enough to be useful to us here.

Global database query languages provide result-oriented solutions. They allow a more "natural" means of expressing desired results solutions than do imperative programing languages. Schema integration and federation can be naturally articulated as an agenda of tasks that must be executed to achieve the desired integration. Customized data integration approaches and some aspects of the other data integration approaches require ensembles of specialists for certain parts of the solution.

Our work has been influenced in various ways by a number of research efforts involving modular or component-based data integration. These include *a la Carte* [12], *COMANDOS* [3], *Garlic* [25], *InterBase* [6], *MIND* [11], *Pegasus* [26], and *TSIMMIS* [14]. These projects variously addressed aspects of componentization of multidatabase systems from transaction management to query processing, the use of standard distributed object technologies, and the use of mediators [30] as the unit of componentization. InterBase, Garlic, and Pegasus provide fairly complete decompositions of the overall problem space of distributed data integration. Our work is focused in particular on tailoring current component-based software construction approaches, embodied in technologies such as *Enterprise Java Beans* [28], to the domain of distributed data integration. Mediators are to provide the containers for our components.

The paradigm we are developing is intended to support the three programming or modeling approaches of expressing: result-oriented solutions, agenda-based solutions, and solutions based on the use of ensembles of specialists. Just as important, our paradigm must also provide a bridge between existing component-based programming paradigms and environments, because the reality is that while technologies like CORBA provide "glue" for components in such systems, but once they are glued together a programming paradigm that is not natural to database programming is required to exchange and manipulate data that are exchanged between the components. For example, the tasks of locating object references and performing object binding is still relatively low-level, tedious and technology dependent. Data-oriented tasks, such as the submitting of queries and the retrieval of

result sets using technologies such as JDBC [31] must also be implemented in a way that is low-level, brittle with respect to the evolution of data integration solutions, and imperative, usually in great contrast to the underlying query expressions being passed from client to server. Component-based frameworks such as *Enterprise Java Beans* provide powerful mechanisms such as reflection and contracts for addressing these problems, but components therein must ultimately be manipulated in ways that are at odds (e.g. imperative and low-level) with conceptual models of distributed data integration solutions.

Our hypothesis is that Petri net graphs can provide a useful modeling tool for data integration in this context. They provide an abstract and formal way of modeling: data flow and control problems seen in data integration, concurrent and asynchronous behavior often extant and useful in distributed and database systems, and event detection and temporality. Petri nets offer many analysis techniques applicable to data integration, including reachability, coverability, and boundedness [23]. Variants of the general Petri net model have proven useful for the modeling of composite event detection and active database systems architectures [15, 19], and the specification and verification of distributed databases and software systems [5, 29]. The particular variant of Petri nets we use has the added benefit of being able to analyze and model dynamic changes to a network. We expect to apply these capabilities to model the handling of changes in data integration requirements and fault tolerance of a running data integration solution.

4. The Souk Network Specification Model

Our specification model for data integration is the *Souk Net (SNet)*, a variant of *WorkFlow Nets* (WoFNets) designed to model data integration process flows. WoFNets are a class of Colored Petri nets [23]. This class of Petri nets has been used by Ellis and Keddara to model workflow processes and dynamic change in them [13]. An *SNet* is a bi-partite net with two kinds of nodes, *places* and *transitions*. They are colored in the sense that the tokens in an SNet can be assigned color types to indicate what type of information they carry: data type, control type (e.g. parameters), or combinations of both. The distribution of tokens in the *SNet* represent its state. Tokens may be distributed over places and transitions.

Each transition in an SNet is associated with one of several types of data integration components. A component is executed when the associated transition *fires*. The association between transitions and components is injective (i.e. each instance of a component type is uniquely associated with a transition). Each transition has at least one input place and at least one output place. A transition's connectors are associated with the ports of its associated component through a total and injective labeling over port names.

Each SNet has a single entry place and a single exit place. A net is connected, and each node in the net is reachable from the entry and may lead to the exit place. The firing of an AFN starts the moment a token is injected into its entry place, and completes when a token is produced into its exit place. Below, we discuss the modeling elements of our variant of WoFNets.

4.1 The Modeling Elements

A *Souk process* (*S-process*) is a procedure where data and control objects are passed between *Souk-components* (*S-components*) according to a well-defined set of semantics to achieve or contribute to an overall set of data integration goals. Each S-process is defined in the context of a *container*. It will normally be the case that the container for an S-process is a *method* in the object-oriented sense, and

that collections of these methods will be implemented within a *mediator* or some other type of distributed object. Thus, each S-process constitutes all or part of an implementation of a *method* defined in the mediator's interface.

Each S-process defines a configuration of interconnected S-components and data places, and the flow of control and data among these network entities. An S-component (*figure* 1) may be one of several types, each type designed to perform specific operations on *data object sets* or control information. Operations on the elements of a data object set may be at either or both the semantic or structural levels. The flow of an S-process is specified by interconnecting its component using *connectors* (i.e. edges).

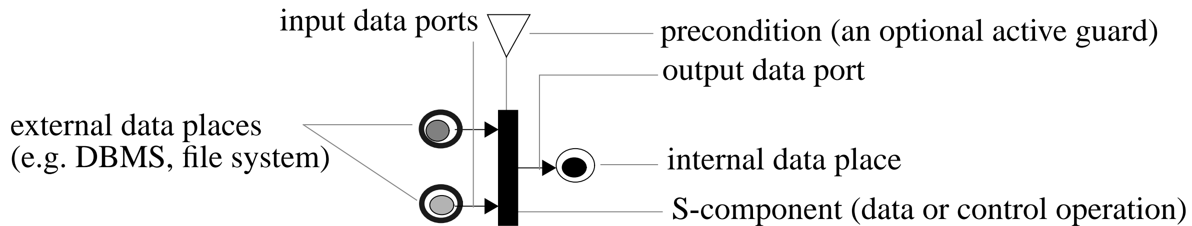

Figure 1. The basic structure of an S-component.

Data places may be internal or external to an S-process. An *internal data place* is declared, created, and managed as part of an S-process. Internal data places are repositories that behave like Linda tuple spaces [8], except that they may only hold one token and the data contained therein may be complex objects. An *external data place* is a local data source whose existence and operation may be independent and autonomous of the S-process. A local data source can be a proper database management systems, a file system, or an application program. Each local data source is encapsulated by a *server object*. These server objects are assumed to be objects in some distributed environment such as CORBA[22] or DCOM[18]. Each local data source involved in an S-process is accessed through the *interface* of its associated *server object*. S-processes manipulate sets of data which originate from and are ultimately stored in local data sources.

S-processes make no assumptions about the structure of the data objects on which they operate. To support data integration data objects in our model must be able to represent arbitrarily complex data, including 1NF tuples, structurally complex objects, nested relations, text streams, and byte streams. We have, therefore, chosen as our *global data model* the data type system for set forth by CORBA IDL [22]. Thus, data objects may be of any type expressible in IDL and data object sets are IDL sequences having data objects as their elements.

An S-component is modeled using a black box approach with respect to the structure of the data object sets it processes. Room prevents us from discussing the intricacies of S-component firings here. It suffices to say that the general firing process of an S-component has three phases:

Phase 1: the S-component consumes one token from each of its input places.
Phase 2: the S-component performs its specified operation based on the information contained in the tokens it has consumed;
Phase 3: the S-component produces one new token for each of its output places.

S-components may be broadly classified as:

Filters: components that alter the membership of a data object set according to criteria specified by the modeler of the S-process. Filters support the database notion of *selection*. They do not alter the structure of their input data object sets. The filtering criteria are referred to as the *filtering guards* and are specified by a logical expression (e.g. in some predicate calculus or algebra). Each filter has one input data port and one output data port.

Transforms: components that perform specific structural or semantic (e.g. data value) transformations on a single input data object set. Transforms support the database notions of view construction and schema modification. Each transform has one single input data port and one single output data port.

Blenders: components that combine several data inputs into one single data output. The blending may be structural, elemental, or both. A structural blending is used to perform *view construction* over multiple data sources. Element-wise blendings are essentially set-based operations and support such operations as union, intersection, set difference, set division, Cartesian product, and join.

Controllers: components that manage the routing of tokens in an S-process. Controllers support the global database notion of global query decomposition, where a query is split into several sub-queries, each sent to a specific local data source. A controller may also be used to combine or decompose parameter values coming from different components.

A *data port* provides a conduit for transferring tokens between an S-component and one of its data places. An S-component transfers data object sets between its data places and itself using the methods **produce** and **consume**, which are defined for all *data ports* in attached to an S-component. If the methods **produce** or **consume** are to be invoked on a data port to or from an internal data place, a built-in implementation of these methods can be used.

To achieve *data object set* transfers between an S-component and an external data place, however, the implementations of the **produce** and **consume** methods must be overridden because we cannot assume which methods in the interface of the server object that wraps that external data place must be invoked to transfer data to and from it. This must be done for each data port attached to an external data place.

The usual Petri net semantics of token consumption will not usually apply for external data places, as the corpus of data in local data source will not likely be entirely removed during a query. Rather, data contained in the token that represents the contents of a local data source will either be copied, augmented or removed, but the token itself will remain. These special semantics for external data places allow us to retain the usual firing semantics for interior of an S-process since an S-component connected to an external data place needs only to receive a control token (e.g. a request involving the external data place) to be enabled. We simplify our graphical notation by using a double arrow for a data port if both updates and retrievals are possible from an external data place.

The server objects that wrap external data places provide two fundamental ways of transferring data object sets across their interfaces, using *parameterized accessors* or *cursor-based accessors*. Parameterized accessors provide for the retrieval data object sets via *out* or *in out* parameters. Iterative accessors provide for the retrieval of data object sets, first, using an initialize method which returns a cursor, and then repeated invocations of an iterator method to extract each element of the data object set resulting from the initialize method invocation. Thus, the data object sets transferred

by parameterized accessors can be scalar values or sequences, and those transferred by iterative accessors are individual data objects which are aggregated into a sequence one at a time.

We give an example in *figure 2* of the handling of a global query, its decomposition into subqueries for each of the mediator's constituent local data sources, and the return of the results. This S-process accepts as input a token from place P_{in} in the form of a global query expression, *$query_g$*. S-component C_1 is a *transform* created such that it can parse and decompose *$query_g$* into query expressions in the dialects or languages required by external data sources EP_1 and EP_2. The resulting expressions are *$query_1$* and *$query_2$*. Once the tokens containing $query_1$ and $query_2$ reach data places P_1 and P_3 respectively, the *controller* components responsible for interacting with the external data sources, C_2 and C_3, are enabled and fire. S-component C_4, a *blender*, then combines the results, $result_1$ and $result_2$, in some way meaningful to the data integrator to produce $result_g$. If a global query involves only one external data place, component C_1 will generate a null query for the other external data place, a *no-op* that is ignored by the controller that receives it.

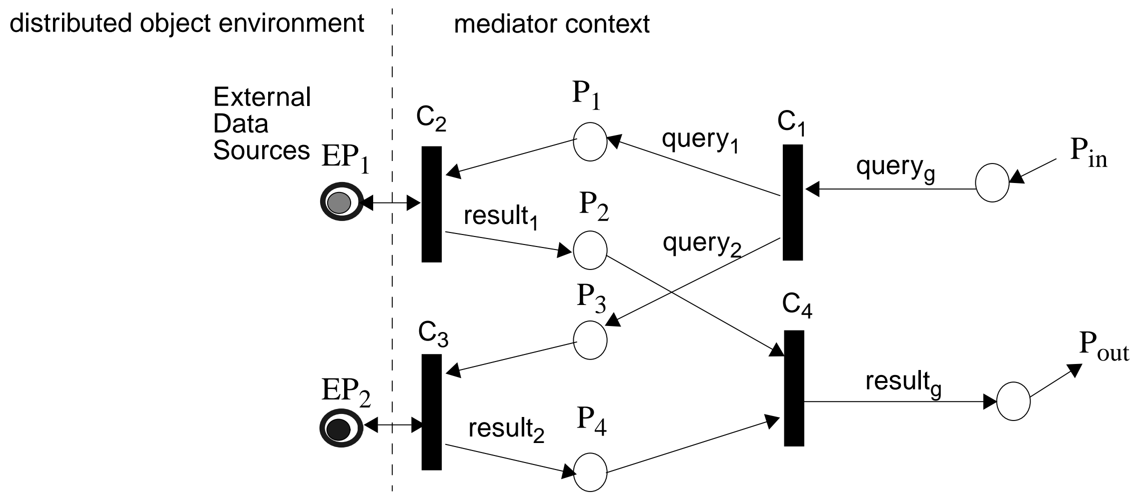

Figure 2. An example.

Each S-component may *observe events* and may perform *event notifications* [9, 24]. Rules are declared, in the form of an S-process, for execution when a component observes an event to which it has *subscribed* and that event indicates a specified *condition*. Events in our model support the notion of global database constraints and the specification of a course of action if constraints are violated. Rules may themselves subscribe to other events and perform event notifications to handle events that are triggered during the execution of the rules. An event subscription is represented as a *precondition* which adorns an S-component, depicted using a triangle. It should be noted that the use of a Petri net approach allows us to readily apply the efficient composite event detection approach developed by Gatziu and Dittrich [15].

Suppose in the example shown in *figure 2*, we are required to update an ADDRESS attribute value in EP_2 whenever an ADDRESS update is sent to EP_1. This event subscription and corresponding rule can be specified as shown in *figure 3*. The event subscription is the expression written over the triangle. It specifies that whenever component C_1, in figure 2, generates a *$query_1$* token that is an update, this rule should be fired. The *guard* on component C5 of the rule tests if the update is to ADDRESS. If it is, an ADDRESS update is generated for EP2 and sent to data place P_4 to replace the no-op token that would have been generated by C_1. The S-process in *figure 2* then resumes with

the new query token that has been placed in P_4. If the *guard* is false, then the rule ends.

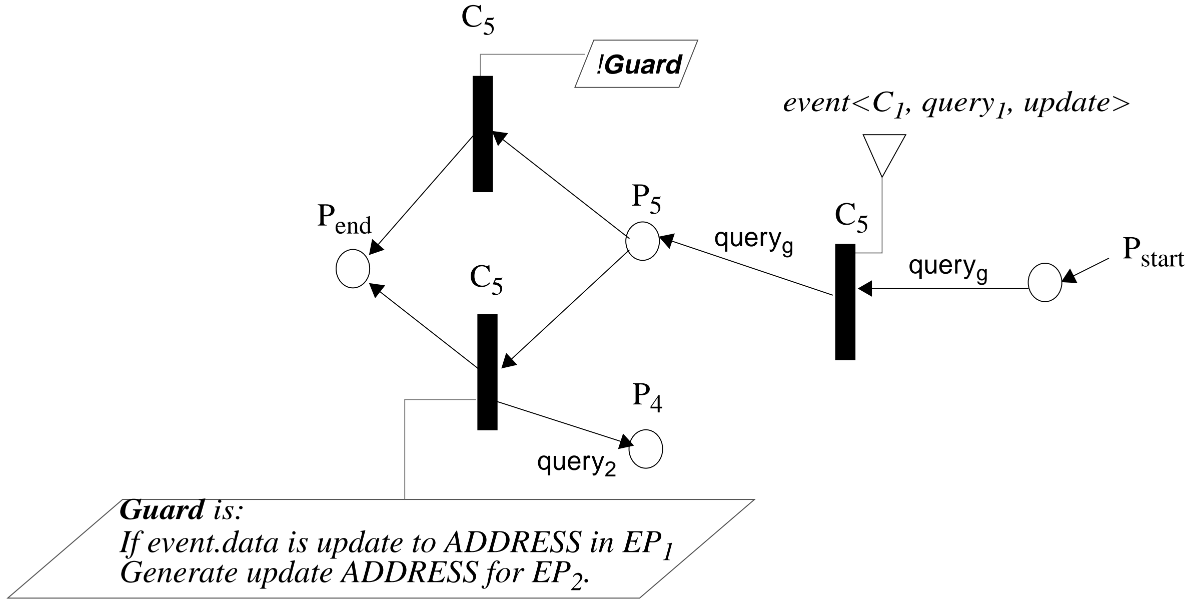

Figure 3. Handling update constraints in the SoukNet paradigm.

Modular design is further supported in Souk Networks through the use of *Souk-macros* (*S-macro*). An S-macro is an S-component that references another S-process. We call this type of S-process an *S-subprocess*. The S-subprocess is executed whenever the macro starts. The S-process may in turn contain S-macros and so on. Thus, arbitrary process nestings can be achieved.

5. Conclusions

We have presented a new component-based paradigm that is designed to support a more natural programming paradigm for data integration than provided by existing component-based frameworks and the use of distributed objects external to our framework. The programming paradigm is supported through the use of a special variant of Petri nets, called *SoukNets*. The SoukNet model contains special types of *transitions* called *S-components* which modularize common or custom data integration operations, such as query translation and decomposition, set operations, and view construction. Data to be integrated are transferred among components in our framework as *tokens*. Control information such as query expressions and parameters are passed in tokens as well. We support a global data model which can be used to represent data coming from local data sources of virtually any complexity and structure. Local data sources are assumed in our paradigm to be wrapped by distributed objects. These are represented as special types of places called *external data places*. Transitions which interact with external data places use *overridden* versions of the data transfer methods **produce** and **consume** built in to the *SoukNet* model. The ability to override the normal Petri net token consumption and production semantics to account for the specifics of interfaces provided by external data places allows our model be applied to the integration of data and services provided by objects resident in distributed object environments such as CORBA and DCOM.

SoukNet articulates a model and a modeling paradigm not a specific database programming language. It is a meta-language. We are in the process of developing a database programming and

integration language called *COIL* which is an instance of the SoukNet meta-language [20].

SoukNets are based on *WoFNets* which are designed to be adaptive [13]. That is, they can be used to analyze and model dynamic changes to a Petri net. We expect to apply these capabilities to model the handling of changes in data integration requirements and fault tolerance of a running data integration solution. Finally, we are addressing the modeling in *SoukNet* of global transactions, environment-wide mediator coordination [1], and the specification of inter-mediator and inter-object behavioral constraints [17].

References

1. Jean-Marc Andreoli and Francois Pacull. “A Quick Overview of the CLF.” Xerox Research Centre Europe. Grenoble, France. (Web site). December 10, 1997.

2. C. Batini, M. Lenzerini, and S.B. Navathe. “A Comparative Analysis f Methodologies for Database Schema Integration.” ACM Computing Surveys. Vol 18, No. 4, December 1986. pp. 324-364.

3. Elisa, Bertino, Mauro Negri, and Licia Sbattella. “An Overview of the Comandos Integration System.” in *Object-Oriented Multidatabase Systems: A Solution for Advanced Applications*. Omran A. Bukhres and Ahmed K. Elmagarmid (eds). Prentice Hall. 1996.

4. Athman Bouguettaya, Boualem Benatallah, and Ahmed Elmagarmid. “An Overview of Multidatabase Systems: Past and Present.” in Management of Heterogeneous and Autonomous Database Systems. Ahmed Elmagarmid, Marek Rusinkiewicz, Amit Sheth editors. Morgan Kaufmann. 1999.

5. G. Bruno and G. Marchetto. “Process-translatable Petri nets for the rapid prototyping of process control systems.” IEEE Transactions on Software Engineering. Vol 12, No. 2. February 1986.

6. Omran A. Bukhres, Jiansan Chen, Weimin Du, Ahmed K. Elmagarmid, Rob Pezzoli. “InterBase: An Execution Environment for Heterogeneous Software Systems.” *IEEE Computer.* 26(8): 57-69 (1993) .

7. O. Bukhres, A. K. Elmagarmid, and E. Kuhn. Advanced Languages for Multidatabase Systems. Chapter in “Object-Oriented Multidatabase Systems”, A.K. Elmagarmid, O. Bukhres (eds), Prentice-Hall. 1996

8. Nicholas Carriero and David Gelertner. “How to Write Parallel Programs: A Guide to the Perplexed.” ACM Computing Surveys. Vol 21, No 3. September 1989. pp 323-357.

9. Antonio Carzaniga, Elisabetta Di Nitto, David S. Rosenblum and Alexander L. Wolf . “Issues in Supporting Event-Based Architectural Styles.” Third International Software Architecture Workshop, Orlando, Florida, November 1998, pp. 17-20.

10. Cohera Corporation. “ ‘Comming to Terms’ with Distributed Computing.” http://www.cohera.com. 1999.

11. A. Dogac, C.Dengi, and M.T. Öszu. “Distributed Object Computing Platforms.” *Communications of the ACM*. September 1998, Vol. 41. No. 9.

12. Pamela Drew, Roger King, Dennis Heimbigner: A Toolkit for the Incremental Implementation of Heterogeneous Database Management Systems. VLDB Journal 1(2): 241-284 (1992).

13. Clarence Ellis and Karim Keddara. “ML-DEWS: A Meta-Language to Support Dynamic Evolution of Workflow System.” to appear in the Journal of CSCW Special Issue on Adaptive Workflow. 1999.

14. H. Garcia-Molina , Y. Papakonstantinou , D. Quass , A. Rajaraman , Y. Sagiv , J. Ullman , V. Vassalos, J. Widom . “The TSIMMIS approach to mediation: Data models and Languages.” Journal of Intelligent Information Systems. 1997.

15. Stella Gatziu and Klaus R. Dittrich. “Detecting Composite Events in Active Database Systems Using Petri Nets.” in *Proceedings of the 4th International Workshop on Research Issues in Data Engineering: Active Database Systems*. Houston, Texas. February 1994.

16. D. Heimbigner and D. McLeod. A federated architecture for information management. ACM Trans. on Office Information Systems, 3(3), pages 253-278. July 1985.

17. P. Inverardi and A.L. Wolf. Formal Specification and Analysis of Software Architectures Using the Chemical Abstract Machine Model. IEEE Transactions on Software Engineering. Vol. 21, No. 4, April 1995.

18. Microsoft Corporation. Distributed Component Object Model Protocol -- DCOM/1.0. http://www.microsoft.com. January 1998.

19. Waseem Naqvi and Mohamed T. Ibrahim. “REFLEX Active Database Model: Application of Petri-Nets.” in *Proceedings of 4th International Database and Expert Systems Applications Conference (DEXA’93)*. Prague, Czech Republic. September 1993.

20. W. J. McIver, Jr., R. King, R.M. Osborne, and C. Och. The COIL Project: A Common Object Interconnection Language to Support Database Integration and Evolution. Proceedings fo the Third International Baltic Workshop on Databases and Information Systems. Riga, Latvia. April 15-17, 1998.

21. Gary Nutt. *Operating Systems: A Modern Prespective*. Addison-Wesley. 1997.

22. Object Management Group. *The Common Object Request Broker: Architecture and Specification*. July 7, 1998.

23. James L. Peterson. *Petri Net Theory and The Modeling of Systems.* Prentice-Hall: Englewood Cliffs, NJ. 1981.

24. David S. Rosenblum and Alexander L. Wolf . “A Design Framework for Internet-Scale Event Observation and Notification.” 6th European Software Engineering Conference (held jointly with SIGSOFT ’97: Fifth International Symposium on the Foundations of Software Engineering), Lecture Notes in Computer Science 1301, Springer, Berlin, 1997, pp. 344-360.

25. Mary Tork Roth, Manish Arya, Laura M. Haas, Michael J. Carey, William F. Cody, Ronald Fagin, Peter M. Schwarz, Joachim Thomas II, Edward L. Wimmers. “The Garlic Project.” *Proceedings of SIGMOD Conference 1996*. p. 557.

26. Ming-Chien Shan, Rafi Ahmed, Jim Davis, Weimin Du, William Kent. “Pegasus: A Heterogeneous Information Management System.” *Modern Database Systems*: *The Object Model, Interoperability, and Beyond*. Won Kim (Ed.). ACM Press and Addison-Wesley. 1995. pp. 664-682.

27. Szyperski, C. Component Software: Beyond Object-Oriented Programming. *Addison-Wesley*. 1999.

28. Anne Thomas. “Enterprise Javabeans Technology: Server Component Model for the Java Platform.” *Patricia Seybold Group (Prepared for Sun Microsystems, Inc.)*. December 1998. (www.javasoft.com).

29. Klaus Voss. “Using Predicate/Transition-Nets to Model and Analyze Distributed Database Systems.” IEEE Transactions on Software Engineering. Vol. SE-6, No. 6, November 1980.

30. Gio Wiederhold. "Mediators in the architecture of future information systems." IEEE Computer. Vol 25, No. 3. 1992.

31. Seth White, Maydene Fisher, R. G. G. Cattell, Graham Hamilton, Mark Hapner. *JDBC[tm] API Tutorial and Reference, Second Edition: Universal Data Access for the Java[tm] 2 Platform*. Addison Wesley Longman. 1999.