In: Pre-Proceedings of the 6^{th} Thematic Workshop of the EU Network of Excellence DELOS: Digital Library Architectures - Peer-to-Peer, Grid, and Service-Orientation (DLA 2004), pages 37-48, Cagliari, Italy, June 2004, Edizioni Progetto Padova.

Data Stream Management and Digital Library Processes on Top of a Hyperdatabase and Grid Infrastructure

Manfred Wurz, Gert Brettlecker, and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology
Innrain 98 A–6020 Innsbruck Austria
[manfred.wurz|gert.brettlecker|heiko.schuldt@umit.at]

Abstract. Digital libraries in healthcare are hosting an inherently large collection of digital information. Especially in medical digital libraries, this information needs to be analyzed and processed in a timely manner. Sensor data streams, for instance, providing continuous information on patients have to be processed on-line in order to detect critical situations. This is done by combining existing services and operators into streaming processes. Since the individual processing steps are quite complex, is is important to efficiently make use of the resources in a distributed system by parallelizing operators and services. Grid infrastructures already support the efficient routing and distribution of service requests. In this paper, we present a novel information management infrastructure based on a hyperdatabase system that combines the process-based composition of services and operators needed for sensor data stream processing with advanced Grid features.

1 Introduction

Digital libraries in healthcare are increasingly hosting an inherently large and heterogeneous collection of digital information, like electronic journals, images, audios, videos, biosignals, three dimensional models, gene sequences, protein sequences, and even health records. Medical digital libraries therefore have to organize repositories managing this medical information [1] and to provide effective and efficient access to it. In addition, a central aspect is the collection, aggregation, and analysis of relevant information.

Due to the proliferation of sensor technology, the amount of continuously produced information (e.g., biosignals or videos) in medical digital libraries will significantly grow. These data streams need sophisticated processing support in order to guarantee that medically relevant information can be extracted and derived for further storage, but also for the on-line detection of critical situations. Biosignals, like a ECG recording, contain relevant information derived from the evaluation of characteristic parameters, e.g., the heart rate, and their deviance from average. In some cases, even the combination of different biosignals is needed for the extraction of relevant information, such as a comparison of heart rate and blood pressure. *Data stream management* (DSM) addresses the continuous process streaming data in real-time. Due to the streaming origin of parts of the information stored in medical digital libraries, the latter will significantly benefit from infrastructures incorporating DSM.

Due the service-orientation and the distributed nature of digital libraries (i.e., information is made available by means of services), *Grid infrastructures* are very well suited as basis for digital library applications. The composition of services and DSM operations can be realized by means of processes. The Grid then supports the efficient routing of service requests among different service providers. A very challenging aspect in process-based service composition on top of a Grid environment is that processes itself can be seen as services and therefore can be used within other processes again. This, in a way, adds recursive nature to processes and implements the well known composite pattern [2] for processes on the Grid. Moreover, also the runtime support for process execution can be considered as a special, inherently distributed Grid service.

In this paper, we introduce an integrated hyperdatabase and grid infrastructure that supports the processing of continuous data streams and that is able to distribute the processing of computationally expensive services within a Grid. By this, the requirements of efficiently processing continuous data that can be found in digital medical library applications can be seamlessly supported.

The paper is structured as follows. Section 2 describes a sample telemonitoring application in a digital healthcare library to motivate the need for a joint hyperdatabase and grid environment. In Section 3, we present a process-based approach to data stream management. The dynamic process parallelization by using Grid concepts is introduced in Section 4. Section 5 discusses related work and Section 6 concludes.

2 A Sample Application in a Digital Healthcare Library

In this section, we introduce a sample healthcare application to motivate the need for a flexible and reliable information management infrastructure that supports process management, data stream processing and management, and that provides Grid computing capabilities.

The left hand side of figure 1 illustrates a *telemonitoring system* which takes care of elderly patients suffering from chronic diseases (e.g., diabetes, heart diseases, or other age related problems like Alzheimer). This telemonitoring system is one of the information providers of the underlying medical digital libraries. Patients are equipped with an array of sensors, as for example the LifeShirt-System [3], that continuously measure the patient's body signals (e.g., ECG). Additionally, sensors integrated in the patient's home are detecting context information that describes what the patient is currently doing (e.g., if the patient is sleeping). This information is important to evaluate the medical meaning of vital signs — for example, the ECG signal has to be interpreted differently when a person is sleeping, compared to the case where she is active. In addition to medical monitoring, context information is also used to integrate a patient support system

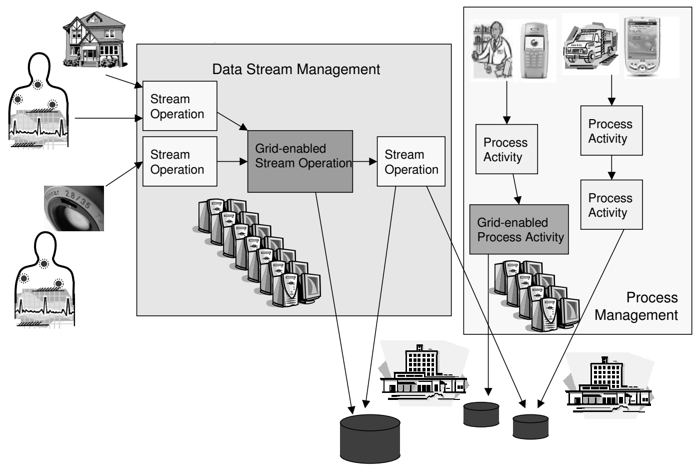

Fig. 1. Data Stream and Process Management in a Medical Digital Library

in this scenario. Patients can be remembered to turn off the oven or take their pills. In order to make use of the vast amount of sensor information, the incoming sensory data has to be processed in real-time. Medically relevant results may be stored in a digital library containing the patient's health record. Results with unknown characteristics are stored in repositories to support medical research. Critical results may request immediate intervention by the caregiver. In this case, appropriate processes (e.g., calling the emergency service or contacting a physician) have to be triggered.

Access to the contents of a medical digital library is supported by special services and user defined processes that combine several of these services (illustrated on the right hand side of figure 1). As described above, processes for contacting the caregiver (e.g., by sending a SMS to a mobile device of a physician), or even for triggering some rescue activities in case of critical situations have to be invoked if necessary. If the physician needs more detailed information or wants to request data on previous treatments or prescriptions, he has to be served with the data in a timely fashion. For all these purposes, appropriate processes have to be available (or have to be defined) and to be executed efficiently by the underlying infrastructure.

Our infrastructure for telemonitoring applications is based on a combined hyperdatabase system [4] and Grid environment [5]. It supports the definition and execution of processes on top of (web) services but also allows to implement continuously running processes for analyzing, processing, and managing data streams in real-time. Since processing data streams for evaluating the patient's health state requires the invocation of computationally intensive services, Grid concepts are exploited to support the distributed computation on top of heterogenous resources. Therefore, the different data streams coming from the various sensors of a patient are distributed within the Grid for parallel processing. Finally, the streams have to be joined in order to combine different sensor signals for rating medical relevance. The combination of process management and Grid concepts allows for the composition of existing services and for the efficient distribution of single service invocations within the Grid.

3 Data Stream Management for Medical Digital Libraries

In this section, we introduce an extended hyperdatabase system for the support and management of continuous data streams.

3.1 Challenges in Data Stream Management

The main challenges in *data stream management* (DSM) are imposed by the large number of sensors, components, devices, information systems and platforms connected by different network technologies, and by the vast amount of continuously generated data. For processing this data, existing systems and components are well in place and need to be incorporated into digital libraries. Reliability and provable correctness are new challenges that are of utmost importance particularly in healthcare applications, where failures may have perilous consequences. As described in Section 2, DSM has to interact with traditional process management in order to react to certain results (e.g., calling the ambulance) or to offer the user appropriate processes for the evaluation of DSM results. These challenges necessitate an infrastructure that combines the processing of data streams and process management, i.e., the possibility to combine services (conventional services as offered by digital libraries and services operating on data streams produced by sensors) and to execute composite services in a reliable way. Therefore, we propose an integrated information management infrastructure supporting user-defined processes, both conventional and processes performing DSM. *Hyperdatabase (HDB)* systems already provide an infrastructure for reliable process execution, which we will extend to enable DSM processes.

3.2 Peer-to-Peer Process Execution in the Hyperdatabase OSIRIS

A *hyperdatabase (HDB)* [6] is an infrastructure that supports the definition and reliable execution of user-defined processes on top of distributed components using existing services. Characteristic features of HDB's are the possibility to i.) add transactional guarantees to the execution of processes [7], ii.) support reliable peer-to-peer execution of processes without global control, thereby supporting a high degree of availability and scalability, and iii.) apply decentralized process execution in areas of intermitted connectivity.

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) [4] is a prototype of a hyperdatabase, that has been developed at ETH Zurich and is used as a starting point of our joint HDB and Grid infrastructure. OSIRIS follows a novel architecture for distributed and decentralized process management. OSIRIS supports process execution in a peer-to-peer style based on locally replicated metadata, without contacting any central instance *(Peer-to-Peer Execution of Processes, P2PEP)*. With P2PEP, a component works off its part of a process and then directly migrates the instance data to nodes offering a suitable service for the next step(s) of the process according to its control flow specification. This is achieved by implementing two layers: the *HDB-layer*, a small software layer that is installed on each component providing a service

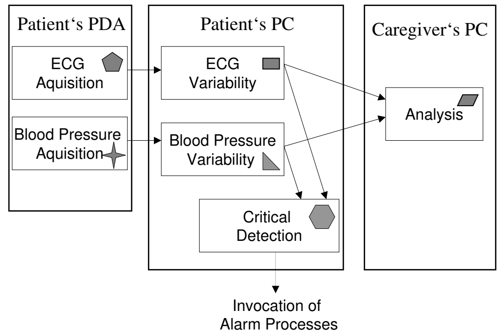

Fig. 2. Stream Process Processing ECG and Blood Pressure

and a set of global HDB repositories. These HDB repositories collect metadata on the processes to be executed, on the available components, and on their load. This meta information is smoothly distributed to the individual HDB layers – only metadata needed locally is actually replicated (e.g., only information on services and providers which might be invoked in some process are required at the local HDB layer of a component). More information on hyperdatabases and OSIRIS can be found in [6, 8, 4].

3.3 DSM enabled Extended Hyperdatabases Infrastructure

HDB's have to be extended in order to enrich their benefits with the capabilities for DSM [9]. We consider *stream-processes*, which perform continuous processing of data streams. The requirements for the execution of these stream processes are similar to those of conventional processes with respect to important aspects like distributed execution, load balancing, meta information distribution, or fault tolerance. Figure 2 illustrates a stream-process, which continuously processes patient's ECG and blood pressure. Sensor signals are recorded and preprocessed by patient's PDA, which is wirelessly connected to patient's PC. The PC does further processing and detects critical health conditions. Processed sensor information is continuously forwarded to the caregiver for further analysis.

Operators are the processing units of DSM. Operators perform stream operations on incoming data streams and produce outgoing data streams. Sensors are the primary sources of data streams. Sensors can be considered as operators without incoming data streams. DSM is done by combining operators, similar to the combination of activities in traditional process management. A stream-process is such a well defined set of logically linked operators continuously processing the selected input data streams, thereby producing results and having *side effects*. Side effects are effects on external systems imposed by processing results (e.g., feeding a digital library with medical relevant information gained by the stream process).

Based on the OSIRIS approach to fault-tolerant distributed peer-to-peer process execution, we need to distribute necessary meta information on stream processes for DSM in the same way this is done also for process management. This metadata contains the pieces of the global stream-process definition and a list of offered stream operators of components, which are subject for smooth distribution among the suitable components offering the corresponding stream operators. A stream-process is set up by sending an activation message to the HDB-layer of the component hosting a source operator (e.g., the component is attached to a sensor or has a data stream input). Due to locally available metadata, the local HDB-Layer knows the subsequent stream operator and components, which offer these operators and is able to make the routing decision. Then the component sends an activation message to the selected subsequent components and provides them with needed data streams.

Our extended infrastructure also allows for load balancing during the execution of stream processes. Therefore, the distribution of metadata on the load of components that are able to host stream operators needs to be published. This load information is used to choose the best component during the stream-process activation. In case of high load, the overloaded component is able to transfer a running stream operator to a component with less load. When stream operations are affected that accumulate an internal state during their execution, this state has to be managed and transferred to the new host. Due to this fact, components make a backup of internal state of running stream-operators at a regular basis. Information about the backup location address is metadata, which is also smoothly distributed.

The previous techniques are also responsible to allow for sophisticated failure handling. In case a component hosting a stream operator fails, components hosting preceding parts of the same stream-process will recognize the failure because the transmission of their outgoing streams is no longer acknowledged. The infrastructure distinguishes between four failure cases. First, the failed component recovers within a certain timeout, then processing is continued in the state before the failure. This is possible since output queues of preceding components are used to buffer the data streams until they are acknowledged. Second, the failed component does not recover within the timeout period. In this case, the preceding component is in a similar situation as during the setup phase of the process. The component has to find suitable components that are able to perform subsequent stream operators. Due to local metadata, the new component is able to find the backup location and to load the old internal state for the continuation of stream processing. If the failed component recovers after the timeout, it has to be informed that its workload moved and that it is no longer in charge. Third, the failed component does not recover and there is no other suitable component. In this case, the stream-process may have an alternative processing branch (defined in the streaming process), which is now activated by the preceding component. Fourth, there is no recovery and no possibility to continue stream processing. If so, a conventional process can be invoked to handle the failure situation (e.g., calling an administrator to fix the problem).

This extended HDB system is capable of supporting telemonitoring applications by providing integrated process and data stream management in peer-to-peer style. Furthermore, it allows to seamlessly cooperate with digital libraries, e.g., by making use of the services that are provided to access information.

4 Digital Libraries on the Grid

An important challenge when dealing with service composition, especially with computationally complex services, is the efficient routing of service requests among a set of providers. OGSA (Open Grid Services Architecture) [10] compliant Grid systems are rapidly emerging and are widely accepted. These Grid systems provide support to efficiently invoke and use individual services in the Grid in a request/reply style. However, they do not support service composition and process execution. In contrast, the focus of state-of-the-art process support systems is not at all or only marginally oriented towards a tight integration into a Grid Environment.

4.1 Bringing Service Composition to the Grid

Although OSIRIS, the starting point of our integrated *DSM* and *Grid Infrastructure*, is quite powerful in doing distributed process management, it does not yet follow OGSA or WS-RF [11], the de facto standard for Grid Environments. It does also not make use of the enhanced features offered in the globus toolkit [5] (the reference implementation of OGSA) like, for example, resource management and security. In our current work, we aim to bring support for service composition to the Grid, which is done by extracting some of the ideas that can be found in OSIRIS, and integrate those with current standards and services which have recently emerged in the Grid Community. This will result in a set of new OGSA compliant services enhancing current Grid Infrastructures with the ability of recursive process composition.

There are several possibilities to decompose an application into smaller parts that can then be executed in parallel. The most important ones are master/slave type of applications, as well as the divide and conquer or branch and bound paradigms. The applicability of these paradigm of course strongly depends on the semantics of the application to be parallelized. Especially the master/slave paradigm is very suitable to Grid-enable applications [12], and is therefore widely used. In case of master/slave parallelization, the main prerequisites are: i) few or *no communication* among the sub parts ii) work is dividable among *identical* sub parts iii) work can be dis- and reassembled in a central point iv) work can be parameterized and parallelized and does not need serial iterative processing.

Since the potential for master/slave parallelization can be found in several applications, we have started to apply this paradigm to enhance the efficiency, the creation, and the ease-of-use of services in the Grid. Using the master/slave paradigm, applications developers can focus on the implementation of the problem specific subparts of the service as well as on the split into and merge of parallel subparts, but they do *not* need to care about the distribuiton of subparts. This is particularly important since the latter requires dynamic information on the currently available resources which is not available at build-time, when the services, their split and merge are defined.

4.2 The Frameworks Architecture and Use

To ease the creation of services for tomorrow's Grid Infrastructures, we are currently developing a generic framework to handle master/slave applications where a single master process controls the distribution of work to a set of identically operating slave processes. This framework is designed to accept ordinary Web/Grid Services as destinations for calls, as well as composite services. The framework enables application developers to port new master/slave type of applications to the Grid by just implementing a very limited set of *application focused methods*, and declare the so implemented classes as available to the framework in a deployment descriptor file. The framework takes care about all the *infrastructure related functionality* like marshaling and unmarshaling, communications, failure handling and distributed invocation of services depending on availability and performance considerations. Among this functionality, the framework dynamically takes care about the level of parallelization based on the current status of the Grid, availability of nodes, and QoS restrictions.

The framework developed is based on GT3 [5]. The core part consists of a set of classes building the central master and slave services. These are OGSA-compliant Grid Services [10] bundled with corresponding stubs and some supporting classes for specialized exceptions and encapsulating the input and output parameters passed around. The work left to the application programmer is to implement abstract methods which are responsible for the application specific part, in particular methods for splitting, merging, and the actual application logic. The ones for splitting and merging used in the master service, and the calculative method is used to concretely specify what the slave has to do. In addition, a Web Service deployment descriptor (WSDD) has to be written, as specified by the Axis framework [13], which GT3 is partly based on. At runtime, the framework determines which slaves to use, out of the set of all slaves registered to provide the appropriate service. This is done by accessing an *IndexService* available in the Grid. The request is then forwarded to all the slaves, after being divided into subtasks. This is shown in the upper right corner of figure 3 where the service depicted as cross is provided by a set of slave services executing in parallel.

The current implementation can easily be adopted to more sophisticated distribution mechanisms based on the Service Data Elements (SDE's) [5] provided by each Grid Service. There might be more specialized implementations that distribute to slaves based on current workload, cost or other metrics available. After having distributed he work, the *MasterService* registers for notifications from the slaves and waits for results. After all slaves have returned, the Master Service generates the final result by merging the results of the subparts and returns the completed result to the requestor. An important aspect here is to provide sophisticated failure handling that allows the Master Service to re-distribute requests when slaves have failed during the execution of their subpart. On the slaves side, in addition to the implementation of the actual application logic, a deployment descriptor is needed that specifies where to register this particular slave service.

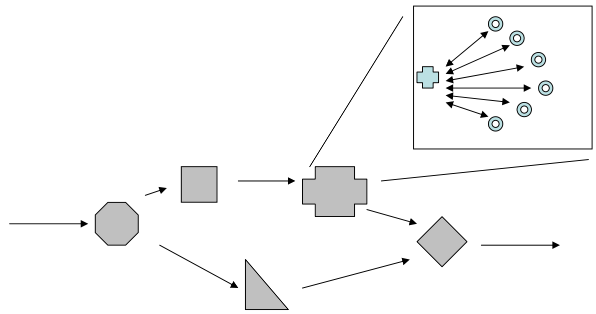

Fig. 3. Process containing a dynamically acting Grid Node

In the scenario described in section 2, there is one master Grid Service which accepts streamed data from the patients life vest and ECG. This service acts, from the point of the process management system, as an ordinary step in the process chain. However, in the background it re-directs the data stream to the slaves available in the system and checks the data against local replicas of digital libraries holding characteristic pathologic and non pathologic data. The time intensive comparison of the streamed data with entries in the digital library is done in a distributed way on the Grid. The slaves report the result of their search back to the master who is then able to store the data for further usage and to trigger subsequent services or processes when needed (e.g., in critical situations).

4.3 From Master/Slave to Process Execution

A *MasterService* can generally be seen as a Grid Service that controls the execution and dataflow among a set of several services whose availability, number and distribution is only known during runtime and subject to frequent changes. Since from the point of view of the OSIRIS process execution engine, it acts just as any other operator or service, the dynamics of request distribution as well as the distribution pattern itself is transparent to the process execution engine. Figure 3 illustrates a process schema as executed by OSIRIS including a dynamically acting Grid Node. One step in this process, shown as a cross, is dispatching the request to various nodes in the Grid and awaits their feedback. The process execution engine is not aware of this dispatching behind the scenes. This leads to the more general idea that the *MasterService* can be seen as a Process Execution Service itself, calling arbitrary Grid Services — either in parallel, sequentially or in any other pattern available to the system.

This Process Execution Services can be deployed to the Grid as highly dynamic components. The distribution pattern of an algorithm can be determined at runtime based on some QoS information provided through the caller or can be hard wired to a special distribution pattern.

In order to avoid a centralized Process Execution Service that could lead to a single point of failure, we are currently integrating the distributed process execution engine described in OSIRIS. In OSIRIS, the execution plan for a process (determined by the control flow) is, prior to its invocation, split up into several execution steps. Each step consists of a service invocation, and information of all successors. This allows to move the control from a centralized component to the responsibility of each node participating in the process. Therefore, this approach is much more robust to the failure of single nodes execution and triggering the next step is up and running) than centralized solutions.

5 Related Work

5.1 Data Stream Management

DSM aspects are addressed by various projects like NiagaraCQ [14], STREAM [15], and COUGAR [16]. The main focus of these projects is on query optimization and approximate query results and data provided by sensor networks. Aurora [17] allows for user defined query processing by placing and connecting operators in a query plan. Aurora is a single node architecture, where a centralized scheduler determines which operator to run. Extensions like Aurora* and *Medusa* [18] also address DSM in distributed environments. TelegraphCQ [19] is a DSM project with special focus on adaptive query processing. Fjords allow for inter-module communication between an extensible set of operators enabling static and streaming data sources. Flux [20] provides load balancing and fault tolerance. PeerCQ [21] is a system that offers a decentralized peer-to-peer approach supporting continual queries running in a network of peers. The DFuse [22] framework supports distributed data fusion. Compared to other projects in this field, our infrastructure offers two unique characteristics. Firstly, dynamic peer-to-peer process execution where local execution is possible without centralized control. Secondly, the combination of DSM and transactional process management enables sophisticated failure handling.

5.2 Grid Infrastructure

The master/slave paradigm is commonly agreed as valuable asset for the development of Grid applications [12]. The master-worker tool [23] provides the possibility to integrate applications in the Grid by implementing a small number of user-defined functions concentrating on the applications main purpose. It is applied to complex problems from the field of numerical optimization [24]. While it is tightly integrated into a former Grid environment, the Globus Toolkit 2, our approach uses more recently emerged technologies and focuses on evolving into a more generally useable distributed process execution engine.

A similar approach is taken in AppLeS Master-Worker Application Template (AMWAT) [25] where the main emphasis is on scheduling issues and a workflow model to select the best locations for the master and worker services. Other Approaches focusing on other task-parallel models can be found in [26, 27] for the divide-and-conquer distribution pattern, and [28] for branch-and-bound.

In [29], BPEL4WS, the Business Process Execution Language for Web Services [30] is evaluated for the use within transactional business processes on the

Grid. The authors point out that the usage of single, non-orchestrated Web Services is limited, and that there is a need for reliable and coordinated process execution on the Grid.

6 Conclusion and Outlook

The proliferation of ubiquitous computing and the huge amount of existing information sources is leading towards a world where sophisticated information management is becoming a crucial requirement. A digital library for medical applications not only has to manage discrete data, it has also to support the acquisition, processing, and storage of streaming information that is continuously produced by sensors. Essentially, both streaming and non-streaming processes and applications have to be supported. Moreover, due to the complex processing operators that are used within stream processes, the distribution of work is a major requirement to efficiently process continuous data streams. By exploiting the features of a Grid infrastructure, subparts can be executed in parallel by making use of the resources that are available at run-time. As a paradigm for the distribution of work within the Grid, we have integrated a master/slave type of interaction into a stream-enabled HDB system.

Based on this extended HDB system, we are currently building a comprehensive infrastructure that jointly addresses process-based service composition and streaming processes, and that is enriched by features from an existing Grid infrastructure. In terms of the distribution paradigms supported, we are currently extending the master/slave type of distribution to allow for arbitrary execution plans. The goal is to define a generic, distributed and OGSA compliant process execution engine. This engine has to support different control flow specifications for composite services that are controlled by the Grid-enabled Process Execution Services so that it can be exploited for process-based applications on top of medical digital libraries.

References

1. Haux, R., Kulikovski, C.: Digital Libraries and Medicine. Yearbook of Medical Informatics (2001)
2. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
3. VivoMetrics: VivoMetrics – Continuous Ambulatory Monitoring. http://www.vivometrics.com/site/system.html (2003)
4. Schuler, C., et al.: Peer-to-Peer Process Execution with OSIRIS. In: Proceedings of ICSOC 2003, Trento, Italy, Springer LNCS, Vol. 2910 (2003) 483–498
5. The Globus Alliance: The Globus Toolkit Version 3. http://www-unix.globus.org/toolkit/ (2003)
6. Schek, H.J., Böhm, K., Grabs, T., Röhm, U., Schuldt, H., Weber, R.: Hyperdatabases. In: Proc. of WISE Conf., Hong Kong, China (2000) 28–40
7. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation for Transactional Processes. ACM Transactions on Database Systems **27** (2002) 63–116

8. Schek, H.J., Schuldt, H., Schuler, C., Weber, R.: Infrastructure for Information Spaces. In: Proc. of ADBIS Conf., Bratislava, Slovakia (2002) 22–36
9. Brettlecker, G., Schuldt, H., Schatz, R.: Hyperdatabases for Peer–to–Peer Data Stream Management. In: Proc. of ICWS Conf., San Diego, USA (2004) to appear
10. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002)
11. Sabbah, D.: Bringing Grid & Web Services Together. Presentation, IBM Software Group (2004) http://www.globus.org/wsrf/sabbah_wsrf.pdf.
12. Foster, I., Kesselman, C., eds.: The Grid 2, Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (2004)
13. Apache WebServices Project: AXIS. http://ws.apache.org/axis/ (2004)
14. Chen, J., et al.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: Proc. of SIGMOD Conf., Dallas, TX, USA (2000) 379–390
15. Babcock, B., Babu, S., Datar, M., Motwani, R., Widom, J.: Models and Issues in Data Stream Systems. In: Proc. of PODS Conf., Madison, WI, USA (2002) 1–16
16. Yao, Y., Gehrke, J.: Query Processing for Sensor Networks. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
17. Carney, D., et al.: Monitoring Streams - A New Class of Data Management Applications. In: Proc. of VLDB Conf., Hong Kong, China (2002) 215–226
18. Cherniack, M., et al.: Scalable Distributed Stream Processing. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
19. Chandrasekaran, S., et al.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
20. Shah, M., et al.: Flux: An adaptive partitioning operator for continuous query systems. In: Proc. of ICDE Conf., Bangalore, India (2003)
21. Gedik, B., et al.: PeerCQ:A Decentralized and Self-Configuring Peer-to-Peer Information Monitoring System. In: Proc. of Distributed Computing Systems Conf., Providence, RI, USA (2003) 490–499
22. Kumar, R., et al.: DFuse: a framework for distributed data fusion. In: Proc. of SensSys Conf., Los Angeles, CA, USA (2003) 114–125
23. Linderoth, J., et al.: An enabling framework for master-worker applications on the computational grid. In: 9th IEEE Int. Symp. on High Performance Dist. Comp., Los Alamitos, CA, IEE Computer Society Press (2000) 43–50
24. Anstreicher, K., et al.: Solving large quadratic assignment problems on computational grids. In: Mathematical Programming 91(3). (2002) 563–588
25. Shao, G.: Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources. PhD thesis, University of California - San Diego (2001)
26. Foster, I.: Automatic generation of self-scheduling programs. In: IEEE Transactions on Parallel and Distributed Systems 2(1). (1991) 68–78
27. v. Nieuwpoort, R., et al.: Efficient load balancing for wide-area divide-and-conquer applications. In: 8th ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming. (2001) 34–43
28. Iamnitchi, A., et al.: A Problem-specific Fault-tolerance Mechanism for Asynchronous Distributed Systems. In: Int'l Conference on Parallel Processing. (2000)
29. Leymann, F., Güntzel, K.: The Business Grid: Providing Transactional Business Processes via Grid Services. In: Proc. of ICSOC 2003, (Springer)
30. Andrews, T., et al.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. BEA, IBM, Microsoft, SP, Siebel. (2003)