A Combined Hyperdatabase and Grid Infrastructure for Data Stream Management and Digital Library Processes

Manfred Wurz, Gert Brettlecker, and Heiko Schuldt

University for Health Sciences, Medical Informatics and Technology
Eduard-Wallnöfer–Zentrum 1
A–6060 Hall in Tyrol
Austria
[manfred.wurz|gert.brettlecker|heiko.schuldt@umit.at]

Abstract. Digital libraries in healthcare are hosting an inherently large and continually growing 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, it is important to efficiently make use of the resources in a distributed system by dynamically parallelizing operators and services. The Grid vision already considers 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 which consist of such digital artefacts. 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 ECG recordings, contain relevant information derived from the evaluation of characteristic parameters, e.g., the heart rate, and their deviation 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 processing of streaming data in real-time. *Hyperdatabases* [2], in turn, provide a flexible and reliable infrastructure for data stream management [3]. Therefore, because of the streaming origin of parts of the information stored in medical digital libraries, the latter will significantly benefit from hyperdatabase 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 has to support the efficient routing of service requests among different service providers. Considering the heterogenous and ever-changing nature of such environments, the need for dynamic binding of services during runtime is essential to achieve efficient resource usage [4]. Focusing on efficient routing and usage of available resources, it appears appealing to have services available that are able to split incoming requests and parallelize them according to the current status of the grid. In this paper, we introduce an approach to enable existing services to do so, without even changing existing and thoroughly tested functionality. By attaching two additional services, a *'Split Request'* and a *'Merge Result'* service, and by using an infrastructure component termed *'Dynamizer'*, a behavior as described can be realized.

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 [5] 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 introduces a sample telemonitoring application to show the need for a joint hyperdatabase and grid environment. Section 3 gives a brief overview on the hyperdatabase and grid infrastructure. In Section 4, we present a process-based approach to data stream management. The dynamic process parallelization by using grid concepts is introduced in Section 5. Section 6 discusses related work and Section 7 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.

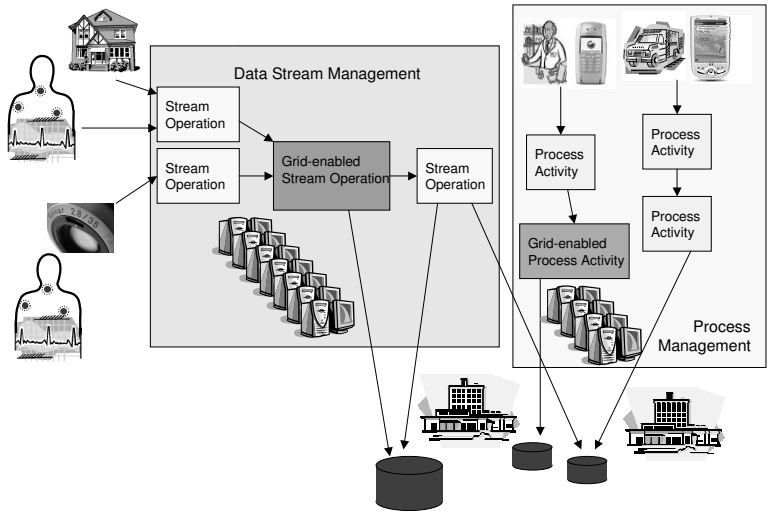

Fig. 1. Data Stream and Process Management in a Medical Digital Library

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 [6], 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 he is active. In addition to medical monitoring, context information is also used to integrate a patient support system 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, she has to be served with the data in a timely fashion. For all

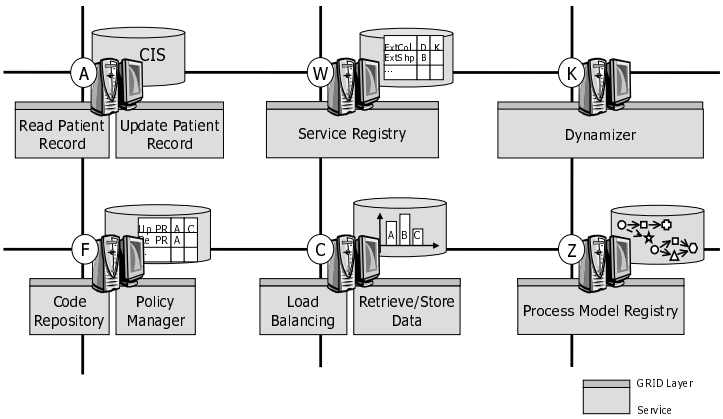

Fig. 2. Grid Infrastructure of a Medical Digital Library

these purposes, appropriate processes have to be available (or have to be defined) and to be executed efficiently by the underlying infrastructure.

3 Architecture of our Hyperdatabase and Grid Digital Library Infrastructure

Our infrastructure for telemonitoring applications is based on a combination of a hyperdatabase system [7] and a service grid environment [8] as illustrated in Fig. 2. From hyperdatabases, we take the support for the definition and execution of processes on top of (web) services but also the possibility 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 dynamically 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.

In the following, Chapter 4 introduces Data Stream Management within our hyperdatabase infrastructure OSIRIS-SE whereas Chapter 5 discusses how grid standards and dynamic service composition are incorporated into our integrated hyperdatabase and grid infrastructure.

4 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.

4.1 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 have extended to enable DSM processes.

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

A *hyperdatabase (HDB)* [2] 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.) support reliable peer-to-peer execution of processes without global control, thereby supporting a high degree of availability and scalability, ii.) add transactional guarantees to the execution of processes [9], and iii.) apply decentralized process execution in areas of intermitted connectivity.

OSIRIS (Open Service Infrastructure for Reliable and Integrated process Support) [7] is a prototype of a hyperdatabase, that has been developed at ETH Zurich and that 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 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 [2, 10, 7].

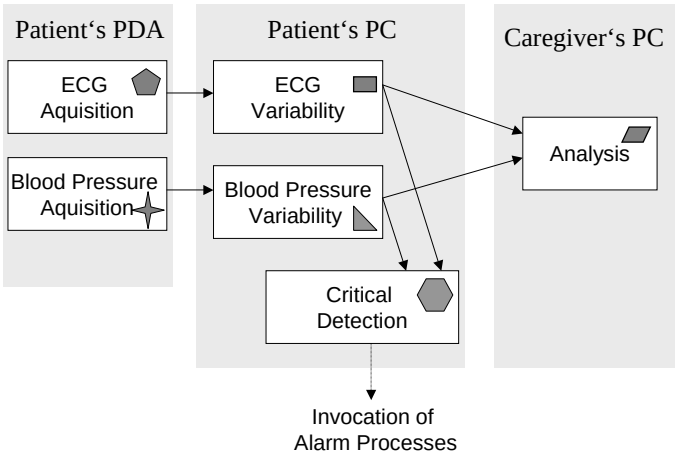

Fig. 3. Stream Process Processing ECG and Blood Pressure

4.3 OSIRIS-SE Infrastructure

HDB's have to be extended in order to enrich their benefits with the capabilities for DSM [3, 11]. This extended infrastructure is called *OSIRIS-SE* (OSIRIS Stream Enabled). 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 3 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. For this reason, operators have input and output *ports* for data streams. Sensors are the primary sources of data streams and can be considered as operators without incoming data streams. DSM is done by combining operators, similar to the combination of activities (service invocations) 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). Stream-processes are defined by users with graphical design tools based on tools used in traditional process management. Our process design tool O'GRAPE (OSIRIS GRAphical Process Editor) [12] is extended to design also stream-processes. Additionally to the process activation flow, which describes the links between activity execution needed for controlling stream-processes, stream-processes have a second flow, called *data flow*. Whereas the process activation flow of a stream process describes how the operators are activated, the data flow describes how the data stream ports of the operators are interconnected. The extended

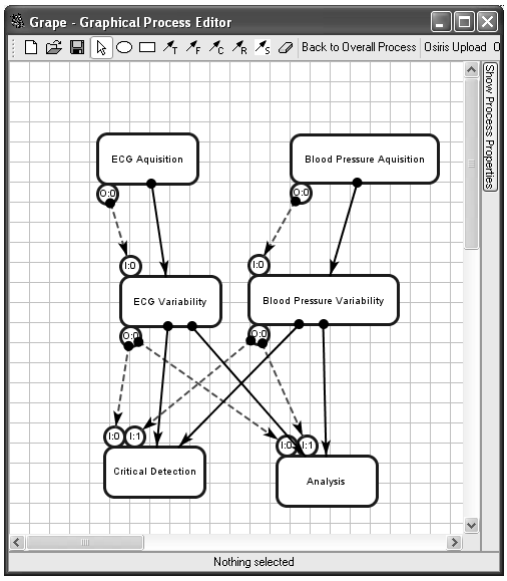

Fig. 4. Stream-Process Design with O'Grape

O'GRAPE tool for stream processes can also explicitly graphically model this data flow, which does not necessarily comply with the process activation flow. In contrast to the process activation flow, the data flow combines ports of operators and not process activities. Figure 4 shows the stream-process of Figure 3 in O'GRAPE. Ports of operators are modeled as circles attached to the process activities. The solid edges indicate the process activation flow and the dashed edges indicate the data flow between ports.

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.

Figure 5 illustrates the architecture of OSIRIS-SE. The process repository keeps definitions of stream processes and traditional processes. Locally needed pieces of the global process definition are published for replication on each corresponding node. Additionally, the service repository offers a list of available stream operators of components, which are also 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). By making use of locally available metadata, the local HDB-Layer knows the subsequent stream operator and components in the process activation flow, which offer these operators and is able to make the routing decision. Then, the component sends an activation message to the selected subsequent component(s) and provides them with needed data streams.

Special treatment is needed for join operators, which have more than one preceding operation (e.g., the analysis operator of Figure 3 which combines ECG variability

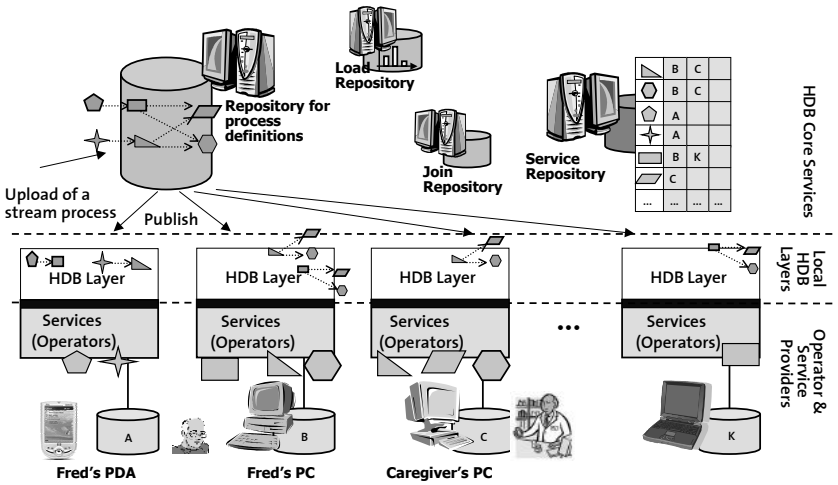

Fig. 5. Architecture of OSIRIS-SE

with blood pressure variability of the patient). Due to the fact that preceding operations may run on different components, the infrastructure has to agree on one distinguished preceding operator. The component hosting the distinguished operator has unique responsibility for process routing, as needed for activation, load balancing, and failure handling. The routing decision is published as metadata via the join repository to non-distinguished components that are used in parallel.

OSIRIS-SE 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 instance to a component with less load. This is called *operator-migration*. 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. Therefore, components make a backup of internal state of running stream-operators at a regular coordinated basis controlled by OSIRIS-SE. Information about the backup location address is metadata, which is also smoothly distributed via the operator backup repository.

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:

1. The failed component recovers within a certain timeout (temporary failure). 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.

2. The failed component does not recover within the timeout period (permanent failure). 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. In addition to normal activation, operator migration is needed to initialize the newly generated operator instance with the existing operator backup. 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.
3. 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.
4. 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).

More on reliability of DSM with OSIRIS-SE can be found in [11]. OSIRIS-SE is capable of supporting telemonitoring applications by providing reliable 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.

5 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) [13] compliant grid systems are rapidly emerging and are widely accepted. These grid systems provide support for the efficient invocation and usage of individual services in the grid in a request/reply style. However, they do neither support service composition nor 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. In what follows, we introduce an approach that combines (data stream) processes and (service) grid environments.

5.1 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 [14], the de facto standard for grid environments. It does also not make use of the enhanced features offered in the globus toolkit [8] (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

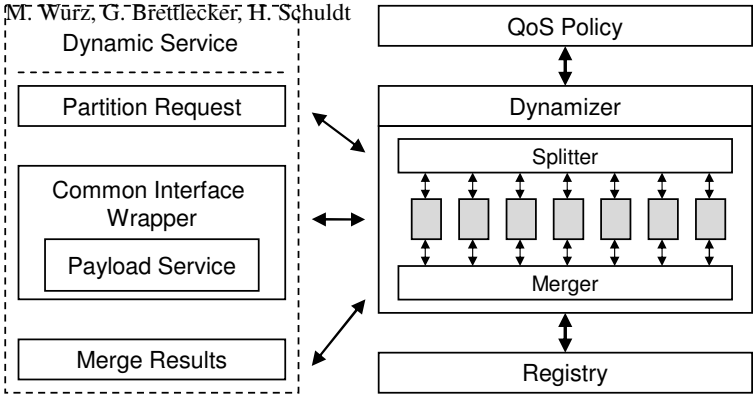

Fig. 6. Architectural Overview: Dynamic Parallelization of Grid-enabled Web Services

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 paradigms 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 [15], and is therefore widely used. In case of master/slave parallelization, the main prerequisites are:

- few or *no communication* among the sub parts
- work is dividable among *identical* sub parts
- work can be dis- and reassembled in a central point
- 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, application 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 take care about the distribution 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 are defined, as well as the way calls to these services are dynamically split and merged.

5.2 The Frameworks Architecture and Use

To ease the creation of services for tomorrow's grid infrastructures, we developed a generic framework to handle master/slave applications where a single master process, labeled 'Dynamizer' in Figure 6, 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 enhancing a pre-existing web service with two additional services, one to split/partition an incoming request into parts for parallel execution, and one to merge and reintegrate sub-results to meet the original request. The service enhanced in such a kind is then named a 'Dynamic Service'.

The overall architecture can be seen in Figure 6 and described as follows: The box in the center of the left hand side, labeled 'Payload Service', represents the actual service. It is responsible for providing application semantics, e.g., a complex computation or a database lookup. This is usually a piece of business logic that has existed beforehand, which is now supposed to be opened to the grid and enabled for parallel execution. To achieve this goal, it is surrounded by another box, labeled 'Common Interface Wrapper', which encapsulates the 'Payload Service' and enhances it with a common interface.

On top, 'Partition Request' encapsulates knowledge on how incoming parameters for the 'Payload Service' have to be partitioned, so that the original request can be decomposed into numerous new sub-requests. Each of these sub-requests can than be distributed on the grid, and be processed by other instances of the originally targeted service. The box at the bottom ('Merge Results') integrates (partial) results returned from the grid to provide the original service requester with a consolidated result. It can therefore be seen as the reverse operation to the 'Partition Request' service. The combination of these elements is referred to as 'Dynamic Service'.

To find the instances of the originally targeted service (e.g., services where the functional description equals the one of the 'Payload Service'), a registry is used (depicted in the lower right corner of Figure 6) . This registry provides information on which services are available, how they can be accessed, and what their properties are (in terms of CPU load, connection bandwidth, access restrictions, etc).

The 'Dynamizer', depicted on the right hand side, makes use of the services mentioned above. It glues together the previously described services by making the parallel calls and by coordinating incoming results. It has also to provide appropriate failure handling. It is, in contrast to 'Partition Request' and 'Merge Results', application independent and generally usable. The 'Dynamizer' can interact with all services that adhere to a common interface, as ensured by the 'Common Interface Wrapper'. It can be integrated in environments able to call and orchestrate services, or it can be packaged and deployed together with specific services.

To make the best possible use of the 'Dynamizer', the user can send a description of the desired service quality along with the mandatory parameters needed to process the request. In this QoS (Quality of Service) policy, the user can, for example, describe whether the request should be optimized in terms of speed (select high performance nodes, and partition the input parameters accordingly), in terms of bandwidth (try to keep network usage low) or if it should aim for best accuracy (important for iterative approaches or database queries, where there is an option to use different data sources). Since these specifications can be contradictory, adding preferences to rank the users requirements is important.

To better illustrate the functionality of the 'Dynamizer' regarding the user specified QoS policy, we reconsider the scenario from Section 2: A physician wants to use the

digital library provided to gain some additional data on the patient she got sensor data from. In the QoS policy file, she specifies that she only wants to have data that can be viewed on her mobile device, a smart-phone, and as a second preference to have her call optimized in terms of speed since treatment for this patient is urgent. The 'Dynamizer' has only three services at hand, which are able to deliver data preprocessed for viewing it on mobile devices, all of them on not very fast computers, or alternatively access to 5 computationally powerful main frames in the hospitals the patient was treated before, but without the capability to render their output for mobile devices. The algorithms needed to reconcile the user specifications, the details of the QoS description language and how to integrate this best with our existing implementation is currently under investigation.

The framework developed is based on web services. The core part consists of a set of classes building the 'Dynamizer' and the implementations of application specific slave services. These can be evolved to be OGSA-compliant grid services [13] 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 the services 'Partition Request', 'Merge Results' and the 'Common Interface Wrapper' which are responsible for the application specific part. In addition, a Web Service deployment descriptor (WSDD) has to be written, as specified by the Axis framework [16], which GT3 is partly based on. At run-time, 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 a global 'Registry' 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 7 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) [8] 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 the work, the 'Dynamizer' registers for notifications from the slaves and waits for results. After all slaves have returned, the 'Dynamizer' 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 'Dynamizer' 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.

In the scenario described in Section 2, there is one dynamic service (bundled with a 'Dynamizer') which accepts streamed data from the patients life vest and ECG. This service acts, from the point of view 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 pathological and non-pathological data. The time intensive comparison of the stream 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 'Dy-

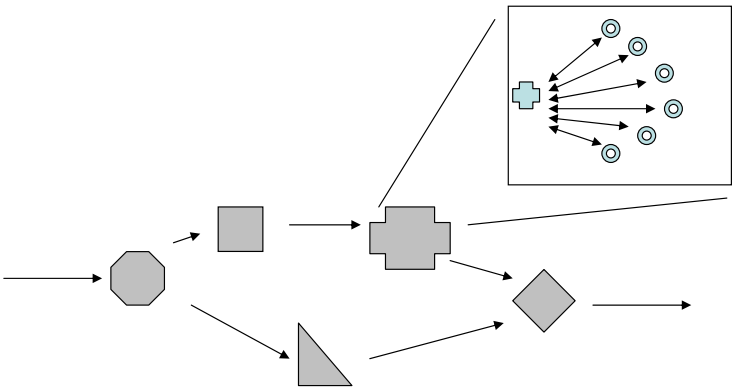

Fig. 7. Process containing a dynamically acting Grid Node

namizer' 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).

5.3 From Master/Slave to Process Execution

A dynamic service can generally be seen as a grid service that controls the execution and dataflow among a set of services whose availability, number, and distribution is only known at 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 are transparent to the process execution engine. Figure 7 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 'Dynamizer' 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.

These 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. Preliminary results on performance and usage can be found in [4].

In order to avoid a centralized process execution service that could lead to a single source 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 its 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 than centralized solutions.

6 Related Work

6.1 Data Stream Management

DSM aspects are addressed by various projects like NiagaraCQ [17], STREAM [18], and COUGAR [19]. The main focus of these projects is on query optimization and approximate query results and data provided by sensor networks. Aurora [20] 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* [21] also address DSM in distributed environments. TelegraphCQ [22] 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 [23] provides load balancing and fault tolerance. PeerCQ [24] is a system that offers a decentralized peer-to-peer approach supporting continual queries running in a network of peers. The DFuse [25] framework supports distributed data fusion. Compared to other projects in this field, our integrated hyperdatabase and grid 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.

6.2 Grid Infrastructure

The master/slave paradigm is commonly agreed as valuable asset for the development of grid applications [15]. The master-worker tool [26] 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 [27]. 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) [28] 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 [29, 30] for the divide-and-conquer distribution pattern, and [31] for branch-and-bound.

In [32], BPEL4WS, the Business Process Execution Language for Web Services [33] 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.

7 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.

Due to the distributed nature of this architecture, a special focus has to be set on failure handling. There are numerous possibilities for the overall process to fail, not at last because of the missing control over the participating nodes within a grid infrastructure: single nodes can be disconnected without prior notification, wide area connections can be interrupted or significantly slowed down. Although some well thought out transaction models are already present within OSIRIS [9] we continue to investigate mechanisms which even better suite the special nature of grid infrastructures.

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. 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
3. Brettlecker, G., Schuldt, H., Schatz, R.: Hyperdatabases for Peer–to–Peer Data Stream Management. In: Proceedings of the International Conference on Web Services (ICWS'04), San Diego, USA (2004) 358–367
4. Wurz, M., Schuldt, H.: Dynamic Parallelization of Grid-Enabled Web Services. In P.M.A. Sloot, A.G. Hoekstra, T.P.A.R.M.B., ed.: Advances in Grid Computing - EGC 2005, Amsterdam, The Netherlands (2005) 173–184
5. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns: Elements of Reusable Object-Oriented Software. Addison-Wesley (1995)
6. VivoMetrics: VivoMetrics – Continuous Ambulatory Monitoring. http://www.vivometrics.com/site/system.html (2003)
7. Schuler, C., Weber, R., Schuldt, H., Schek, H.J.: Peer-to-Peer Process Execution with OSIRIS. In: Proceedings of ICSOC 2003, Trento, Italy, Springer LNCS, Vol. 2910 (2003) 483–498
8. The Globus Alliance: The Globus Toolkit Version 3. http://www-unix.globus.org/toolkit/ (2003)

9. Schuldt, H., Alonso, G., Beeri, C., Schek, H.J.: Atomicity and Isolation for Transactional Processes. ACM Transactions on Database Systems **27** (2002) 63–116
10. Schek, H.J., Schuldt, H., Schuler, C., Weber, R.: Infrastructure for Information Spaces. In: Proc. of ADBIS Conf., Bratislava, Slovakia (2002) 22–36
11. Brettlecker, G., Schuldt, H., Schek, H.J.: Towards Reliable Data Stream Processing with OSIRIS-SE. In: Proc. of the German Database Conf. (BTW'05), Karlsruhe, Germany (2005)
12. Weber, R., Schuler, C., Neukomm, P., Schuldt, H., Schek, H.J.: Web Service Composition with OGrape and OSIRIS. In: Proc. of VLDB Conf., Berlin, Germany (2003)
13. Foster, I., Kesselman, C., Nick, J., Tuecke, S.: The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration (2002)
14. Sabbah, D.: Bringing Grid & Web Services Together. Presentation, IBM Software Group (2004) http://www.globus.org/wsrf/sabbah_wsrf.pdf.
15. Foster, I., Kesselman, C., eds.: The Grid 2, Blueprint for a New Computing Infrastructure. Morgan Kaufmann Publishers (2004)
16. Apache WebServices Project: AXIS. http://ws.apache.org/axis/ (2004)
17. Chen, J., DeWitt, D., Tian, F., Wang, Y.: NiagaraCQ: A Scalable Continuous Query System for Internet Databases. In: Proc. of SIGMOD Conf., Dallas, TX, USA (2000) 379–390
18. 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
19. Yao, Y., Gehrke, J.: Query Processing for Sensor Networks. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
20. Carney, D., Centintemel, U., Cherniack, M., et al.: Monitoring Streams - A New Class of Data Management Applications. In: Proc. of VLDB Conf., Hong Kong, China (2002) 215–226
21. Cherniack, M., Balakrishnan, H., Balazinska, M., Carney, D., Cetintemel, U., Xing, Y., Zdonik, S.: Scalable Distributed Stream Processing. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
22. Chandrasekaran, S., Cooper, O., Deshpande, A., et al.: TelegraphCQ: Continuous Dataflow Processing for an Uncertain World. In: Proc. of CIDR Conf., Asilomar, CA, USA (2003)
23. Shah, M., Hellerstein, J., Chandrasekaran, S., Franklin, M.: Flux: An Adaptive Partitioning Operator for Continuous Query Systems. In: Proc. of ICDE Conf., Bangalore, India (2003)
24. Gedik, B., Liu, L.: 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
25. Kumar, R., Wolenetz, M., Agarwalla, B., Shin, J., Hutto, P., Paul, A., Ramachandran, U.: DFuse: A Framework for Distributed Data Fusion. In: Proc. of SensSys Conf., Los Angeles, CA, USA (2003) 114–125
26. Goux, J.P., Kulkarni, S., Linderoth, J., Yoder, M.: An Enabling Framework for Master - Worker Applications on the Computational Grid. In: 9th IEEE Int'l Symp. on High Performance Dist. Comp., Los Alamitos, CA, IEE Computer Society Press (2000) 43–50
27. Anstreicher, K., Brixius, N., Goux, J.P., Linderoth, J.: Solving Large Quadratic Assignment Problems on Computational Grids. In: Mathematical Programming 91(3). (2002) 563–588
28. Shao, G.: Adaptive Scheduling of Master/Worker Applications on Distributed Computational Resources. PhD thesis, University of California - San Diego (2001)
29. Foster, I.: Automatic Generation of Self - Scheduling Programs. In: IEEE Transactions on Parallel and Distributed Systems 2(1). (1991) 68–78
30. van Nieuwpoort, R.V., Kielmann, T., Bal, H.E.: 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
31. Iamnitchi, A., Foster, I.: A Problem-specific Fault-tolerance Mechanism for Asynchronous Distributed Systems. In: Int'l Conference on Parallel Processing. (2000)

32. Leymann, F., Güntzel, K.: The Business Grid: Providing Transactional Business Processes via Grid Services. In: Proc. of ICSOC 2003, Trento, IT (2003) 256 – 270
33. Andrews, T., et al.: Business Process Execution Language for Web Services (BPEL4WS) 1.1. BEA, IBM, Microsoft, SP, Siebel. (2003)