In: Proceedings of the 2^{nd} International Conference on Web Services (ICWS'2004), pages 26-34, San Diego, CA, USA, June 2004.

Scalable Peer–to–Peer Process Management — The OSIRIS Approach

Christoph Schuler [1] Roger Weber [1] Heiko Schuldt [2] Hans-J. Schek [1,2]

[1] Swiss Federal Institute of Technology (ETH), CH–8092 Zurich,

[2]University for Health Sciences, Medical Informatics, and Technology (UMIT), A-6020 Innsbruck,

E-mail: {schuler,weber,schek}@inf.ethz.ch, heiko.schuldt@umit.at

Abstract

The functionality of applications is increasingly being made available by services. General concepts and standards like SOAP, WSDL, and UDDI support the discovery and invocation of single web services. State–of–the–art process management is conceptually based on a centralized process manager. The resources of this coordinator limit the number of concurrent process executions, especially since the coordinator has to persistently store each state change for recovery purposes. In this paper, we overcome this limitation by executing processes in a peer-to-peer way exploiting all nodes of the system. By distributing the execution and navigation costs, we can achieve a higher degree of scalability allowing for a much larger throughput of processes compared to centralized solutions. This papers describes our prototype system OSIRIS*, which implements such a true peer–to–peer process execution. We further present very promising results verifying the advantages over centralized process management in terms of scalability.*

1. Introduction

In the last years, information technology has undergone several major changes. Especially the current trend towards service-orientation, i.e., dedicated services that allow to access data and/or applications, had a strong impact of information systems and middleware and has radically changed the way information processing takes place. In particular, web services that can be invoked by common web protocols (SOAP over HTTP) have led to the recent proliferation of service-oriented computing. System support for the invocation of single web services is widely available. However, one of the most important tasks when dealing with web services is to combine existing services into a coherent whole. Such applications spanning several (web) service invocations are usually realized by *processes*. The platform independent definitions like XML, SOAP, and WSDL further simplify such a composition.In many application scenarios, we can use processes for maintenance purposes to implement replication or to enforce consistency constraints over different sources. This is done by automatically triggering the execution of a process whenever the violation of a constraint has occurred. Furthermore, it is also natural for such systems to implement accesses to data and queries from users by processes that gather the information via different service calls and aggregate the retrieved data according to the user's demand. As a result, the infrastructure of such systems has to cope with large numbers of concurrent processes and has to ensure the same quality of service as traditional database applications. For all these reasons, service-orientation and process support come along with a set of new challenges:

- **Dynamics**: New services and service providers may enter or leave the system at any time, and the system must keep track of such changes in parallel to service invocations and process execution.
- **Optimal Routing**: Sophisticated routing strategies for service invocations to distribute requests among service providers at run-time using approximate knowledge about availability and load.
- **Reliability and Correctness**: Processes have to be executed reliably and with dedicated correctness guarantees even in case of failures.
- **Scalability**: The infrastructure must scale with the number of services, processes, and users. Clearly, centralized systems will soon reach their limitations. Therefore, each node of the community is equipped with a small software layer which is part of the overall infrastructure. This layer also implements process support in a peer–to–peer fashion and thereby gets rid of a monolithic process management system.

In view of these challenges, a number of groups have started new projects (e.g., Infopipes [25], ObjectGlobe [6], ISEE [20], METUFLOW [10] and MARCAs [12], or eFlow [7]) breaking out of conventional

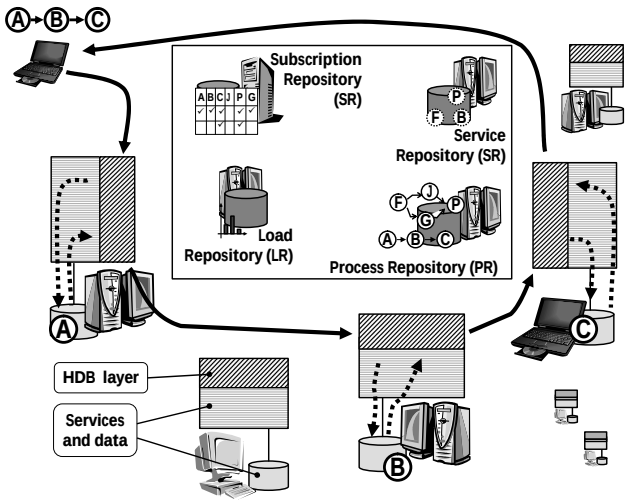


Figure 1. Peer–to–peer process execution

technology. At ETH Zurich, the hyperdatabase vision [27, 28] was established several years ago with the objective to identify a new middleware infrastructure based on well-understood concepts evolving from database technology. While database systems handle data records, a hyperdatabase system deals with services and service invocations. Services in turn may use a database system. In short, a hyperdatabase takes care of optimal routing similar to query optimization in a conventional database, and it provides process support with transactional guarantees over distributed components using existing services as a generalization of traditional database transactions [30]. Most importantly and in contrast to traditional database technology, a hyperdatabase does not follow a monolithic system architecture but is fully distributed over all participating nodes in a network. Every node is equipped with an additional thin software layer, a so-called hyperdatabase layer (HDB layer) as depicted in Figure 1. The HDB layer extends existing layers like the TCP/IP stack with process related functionalities. As such, the HDB layer abstracts from service routing much like TCP/IP abstracts from data packet routing. Moreover, while the TCP/IP protocol guarantees correct transfer of bytes, the HDB layer guarantees the correct shipment of process instances. Of course, a distributed process infrastructure requires that each service provider locally installs this additional software layer. Ideally, this layer comes together with the operating system much like the TCP/IP stack does (comparable to the .NET framework [23]).

We have implemented a prototype system called OSIRIS (short for Open Service Infrastructure for Reliable and Integrated Process Support) following these principles. While previous papers describe the hyperdatabase vision [27], the process model [30], and the general architecture [31], in the following, we concentrate on the discussion and evaluation of OSIRIS as a reliable and scalable infrastructure for the management of services and processes. The contributions of this paper are:

- It describes how to provide process execution in a true peer–to–peer manner.
- The presented solution has sophisticated failure handling mechanisms. Note that failure handling in a distributed environment is more complex compared to failure handling in a central process engine.
- We present evaluation results with OSIRIS using a first benchmark and show significant scalability improvements compared to central solutions.

In the following, we will summarize the OSIRIS architecture and discuss the main architectural decisions in Section 2. Section 3 contains the details of our peer-to-peer process execution, which are necessary to understand our evaluation results described in section 4. We will discuss related work in section 5 and conclude with open problems and future work in section 6.

2. Process Model and Architecture of OSIRIS

In this section, we summarize our earlier work on the topics of process models [30], execution guarantees [29], and the architecture of large scale peer-to-peer systems offering web services [31]. In extension to this previous work, the next two sections will describe and evaluate true peer-to-peer process execution, and will discuss the advantages of this approach compared to traditional centralized solutions in terms of scalability.

2.1. OSIRIS Process Model

A process corresponds to an ordered set of *activities*. The order of these activities defines the sequence of service calls since *activities* are directly mapped to existing services (which again can be processes). The process manager can invoke an activity if and only if all its pre-ordered activities have finished and conditions on its execution are fulfilled. Intra process parallelization is implemented with parallel branches in the process (fork-join) whereby a process (at least conceptually) always has exactly one start activity and one final activity[1]. We define the data flow with mappings from a data space in the process instance, called *whiteboard*, to the service request parameters, and back from the response values to the whiteboard after execution. Note that this implies that each branch of the process has its

1 This is required for the typical request-reply model of service infrastructures (a process is again a service).

own version of the whiteboard (no data flow between parallel branches is possible) and that these versions have to be merged when joining two or more branches. Our model also allows for loops and conditions on edges.

With respect to quality of service, OSIRIS follows the approach of transactional processes [30]. This model bases on the flexible transaction model [13] and guarantees correct process termination. Each activity is either *compensatable*, *retriable*, or *pivot*. The effects of compensatable activities can be semantically undone after the corresponding service invocation has successfully returned. An activity is pivot when it is not compensatable. Retriable activities are guaranteed to terminate correctly, even if they have to be invoked repeatedly. In this case, the last invocation succeeds while all previous invocations of this activity do not leave any effects. A process can contain several pivot activities, but it is required that there is a guaranteed successful path to a termination state after the pivot element, and that rollback of a process instance in case of a failure of the pivot element is feasible (rollback may include the invocation of compensation steps back to a state reflecting a successful execution from which forward recovery is possible, i.e., by alternative execution paths; this is required if there are several pivot elements in the process). Moreover, transactional processes can specify preferences when providing multiple correct paths. OSIRIS comes together with O'GRAPE [33], our Java–based process modeling tool that supports a designer in defining and validating processes.

2.2. Principles of the Architecture

The architecture of ORISIS is driven by the goal of implementing a true peer-to-peer process execution engine for the above process model. Thereby, it follows our hyperdatabase vision: process execution involves only those machines of the network that offer a service required by the process definition. There should be no central synchronization and navigation node to drive the process instances. As we will see, true peer-to-peer process execution is mainly a problem of efficient metadata replication.

The architecture of OSIRIS is split into two parts: firstly, each node of the network runs a hyperdatabase (HDB) layer that provides local services for routing and process navigation as envisioned in the introduction. Secondly, OSIRIS runs a number of global meta data repositories holding information about process definitions, subscription lists, service providers, load information, etc. (cf. Figure 1, middle box). These global repositories store meta information in XML documents, which are essential to run the system. However, the HDB layers that actually execute processes should not have to query meta information from the global repositories. Rather, a push mechanism replicates those parts of meta information towards the HDB layers that they require to fulfill their tasks. For instance, if a node is involved in executing process A, the definition and any changes to that definition are pushed from the process repository to the HDB layer.

From another perspective, the HDB layers perform their tasks based only on local versions of the global meta information. Often, it is even sufficient to hold only approximate versions: for instance, load information of nodes is required by the HDB layer to balance the work (i.e., process activities) among the available service providers. It is sufficient if this load information is only approximately accurate. On the other hand, changes on process definitions have to be propagated immediately to avoid version conflicts.

2.3. Replication of Meta Information

It is important to understand that the global meta data repositories are not directly involved in the execution of processes. They just maintain meta information and distribute this information to all the HDB layers when needed. In previous work, we have introduced two measures to reduce the amount of data being transferred to the nodes (see [31] for more details). Firstly, we use a publish/subscribe scheme with path predicates on the XML document for replication. The path predicate of the subscription selects those parts of the document that a node locally requires: for instance, assume that there are two processes A and B, but an HDB layer is only involved in the execution of process A, i.e., none of the services required by process B is provided by this node. Obviously, this HDB layer only subscribes for the portion of process meta data that includes the definition of A but not the one of B. The same holds true for other pieces of meta information: an HDB layer only requires load information of nodes it eventually will send requests to. The publish/subscribe scheme then ensures that changes at the central XML document are transferred if and only if they are covered by the path predicate. In the previous example, the process repository would not push updates to process definition B to that specific HDB layer.

Secondly, we use so-called freshness predicates to further reduce the number of publications. Freshness predicates are especially useful for replication of very dynamic meta data like load information: small changes of the load of a node are not significant for load balancing. To avoid such unnecessary updates, each subscription at a meta data repository further comprises a freshness predicate that defines under what circumstances a new data item has to be published. An "eager" freshness, for instance, denotes that the repository has to publish every change as soon as possible[2]. In the case of load information, a freshness predicate may denote that updates are only published if the dif-

2 We do not provide a freshness predicate with transactional guarantees, i.e., the update at the central repository is separated from the updates at the HDB layers. Otherwise, costs for updates would negatively affect the scalability of our approach.

ference between the global version and the local version are above 20%. Conceptually, the freshness predicate is a piece of code sent with the subscription that decides based on the global version and local version at the peer whether the update has to be published. As an optimization, the global repositories keep track of the current states of the subscribed peers and of the changes performed meanwhile.

3. P2P Process Execution

As described before, we deploy a true peer–to–peer system for process navigation, but gather and maintain meta data in centralized repositories (although such a repository may run on a cluster to distribute the load). To benefit from the P2P approach in terms of scalability, OSIRIS completely separates process navigation from meta data replication. This section describes in detail our peer–to–peer based process execution system. Especially, we want to address the following key problems:

- How do nodes know where to migrate a process instances to next? What information is needed for this?
- How do nodes execute processes? What minimal information is needed to reduce replication costs?
- How can the system overcome severe failures while still providing transactional guarantees?

3.1. Routing by Late Service Binding

To benefit from load balancing over different service providers, application designers do not encode the service bindings of activities at development time. Rather, they only specify the type of service to be invoked together with its parameters. The concrete service binding is selected at runtime depending on the load of machines and costs of invoking a particular service instance. For that purpose, OSIRIS requires each service provider to subscribe for tasks it can fulfill. Alternatively, an application developer can register these subscriptions on behalf of the service provider. Each subscription contains two XSLT scripts to map the generic interface of a service type to the concrete interface of the service provider as the concrete interface may have different parameters and result values. At runtime, the HDB layer selects a service provider based on costs, parameters, and conditions on its services. This is similar to work on automatic service discovery with the exception that OSIRIS requires, for performance reasons, that service discovery is precomputed and repeatedly updated for each activity in each process type. However, selection among the detected services is still performed at execution time.

Apart of service selection, process navigation and thus instance migration also requires to select a node where the next activity should be executed. In many cases, we may want to replicate a service over the entire network following the ideas of grid computing. Hence, it is not enough to select only a concrete service type. We also have to select an appropriate node to execute that service. OSIRIS uses traditional load balancing methods to select a node trying to advance process instances as fast as possible.

The tasks described above require that each HDB layer knows about subscriptions of service providers to service types (including the mappings) and the load of machines providing these services. OSIRIS maintains this information in centralized repositories (Figure 1) and uses its inherent replication feature to distribute this meta information over the network. We can deploy path predicates and freshness predicates to dramatically reduce the number of updates propagated by the repositories to its peers.

Communication. Once the HDB layer has identified the next node(s) to migrate the current process instance to, it deploys a 2PC protocol to ship the instance data. If transportation is not possible, e.g., in case of a disconnection of a mobile device or network problems, the HDB layer tries to find an alternative node. If this also fails, the instance data remains at the node until a suitable service becomes available, or, if alternative branches in the process model exist, process execution is continued with an alternative branch. Note that the 2PC protocol is rather simple as it involves exactly two nodes. It guarantees that instance data is not lost nor duplicated in cases of network, hardware or software problems. The only exception is the case that a node completely disappears. Then, all the process instances that currently are executed on that node would also disappear and, hence, would not be able to finish. Such situations are difficult to solve automatically as the difference between a normal disconnection of a mobile device and the permanent disconnection of a node is hard to detect.

Fork/Join Problem. A fork is simply the migration of a process instance to different peers (or the same peer, but executing independent activities). However, to join these branches, the nodes executing the last activity of a branch must know where to meet with the other branch(es). Of course, a costly broadcast communication protocol to detect the partners of the join should be avoided. Instead, OSIRIS assigns each process instance a dedicated, reliable node for joining branches at instance creation time. The join is performed by an additional activity automatically inserted before the actual join node in the model. The service of this activity is stateful and decides when to advance the instance to the actual join node (e.g., some branches may have conditions or denote unused alternatives; see below). It also merges the whiteboards of the different branches according to the process model. Afterwards, the merged process instance is routed to a node executing the actual join activity. Note that each join in a process instance is handled by the same node. But different instances can of course have different join nodes.

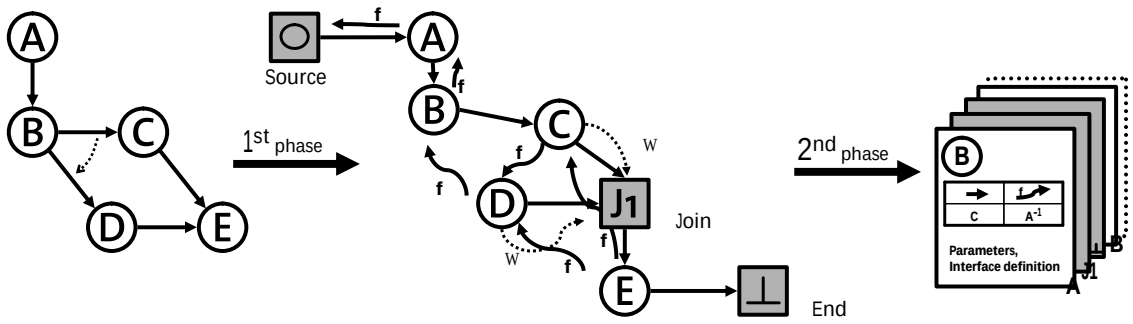


Figure 2. Decomposition of a process central description to a set of distributed execution units

3.2. Deployment and Execution of Processes

In principle, each node of the network could replicate all process definitions and execute process instances with the control flow and data flow definitions of the model. However, this would require large amounts of data to be replicated over the entire network[3]. Rather, we only want to replicate those pieces of process definitions that are locally required to drive the execution of those process instances that potentially visit a node.

Our approach within OSIRIS is to decompose a process definition at deployment time into a set of *execution units*. Each execution unit contains the information to execute the corresponding activity and to navigate the process depending on the result of the service invocation. A node only subscribes itself for those execution units of processes that invoke a locally available service. Consequently, the amount of replicated data significantly reduces. When a designer deploys a new process definition, the nodes that may execute one of its activities automatically receive the corresponding execution units.

Process Transformation. In general, we can easily extract execution units from the process definitions as they basically correspond to pairs of consecutive activities. However, there are some special cases that require further explanations. Our transformation algorithm transforms a graph-based process definition into a set of execution units using two phases (cf. Figure 2). In the first phase, the graphical notation is enriched with additional activities and edges. Each process requires a single *source* activity and end activity. Furthermore, every join is preceded by a special *join* activity (as discussed in the previous subsection; e.g., node $J1$ in Figure 2).

We further have to add edges to the graph. Consider the example process at the left hand side of Figure 2: activity B is followed by activities C and D. The arrow between the two edges from B to C and B to D denotes the preference order of the execution paths, i.e., if service C fails, the alternative path over service D is followed. The problem arises at the join: it expects two incoming edges but only exactly one of them arrives at the join at runtime. To solve this problem, assume that the service call at C was successful. In this case, we know that its alternative branch over D will not be executed. Hence, we can remove D from the process model and all subsequent nodes which are no longer reachable. This is typically known as dead path elimination performed at runtime by the process manager. OSIRIS detects such dead paths already at deployment time for all possible cases, but resolves them at execution time: if a dead path leads to a join node, we must inform this node that the corresponding path never arrives. To do so, we add a special edge from C to $J1$ that signals the join node at execution time that the path over D will not arrive (this edge Ω actually replaces the path over D). Analogously, we have to add edges for compensations (again taking care of join nodes) and execution of alternatives. Due to space restriction, we omit more detailed descriptions of these cases.

The second phase of the algorithm, as depicted in Figure 2 on the right hand side, divides the extended graph into execution units for each node of the graph. The navigation tables directly result from the edges (including the ones from dead path elimination) and preference orders.

3.3. Failure Handling

Failure handling in a distributed environment is more difficult than in a centralized one. The main reason is that there is no global knowledge about where a process instance is currently executed. In Addition, costly broadcast messages to detect the whereabouts of an instance to be avoided. In OSIRIS, we handle failures at three different levels where the first two levels also apply for centralized process management.

At the first level, the process model, we provide alternatives, compensation, restarting of activities to react on fail-

3 Note that each activity comprises two XSLT scripts for mapping whiteboard data to service request parameters and for mapping the result of the service call back into the whiteboard.

ures. Our modeling tool O'GRAPE verifies the correctness of a process based on the transactional process model. If the definition is correct, our navigation model guarantees that process execution ends in a well defined final state regardless of failures of service invocations. For compensation of activities, we further have to keep a history of where an activity has been executed. Compensation is then routed back on the same path as the process instance has advanced.

At the second level, the HDB layer, persistent queues and the 2PC protocol guarantee that state changes of process instances and results of services are made persistent and are recoverable[4], i.e., queued transactions [4] are supported. If the HDB layer fails (hardware or software problem), it recovers the current state of its active process instances using traditional database technologies, re-instantiates service calls if necessary, and continues with process navigation of instances currently residing at this node.

At the third level, the global level, we have to deal with disconnection of nodes. A disconnection can have several reasons: network splits or failures, node crashes, disconnection of mobile devices, or permanent removal of nodes. When migrating a process instance from one node to another, a 2PC protocol between the two nodes guarantees that the instance data is safely transferred. If migration fails, the source node simply selects another destination for the migration or delays migration if the failed node was the only provider of that service. More difficult to solve is the situation when a node permanently disappears or is disconnected for a too long period. In OSIRIS, we allow application developers to specify timeouts and assign special observer nodes to watch critical peers (critical peers are non-reliable nodes that are likely to disconnect, e.g., mobile devices). If a critical peer disconnects and the timeout is reached, the process instance is migrated to an other, currently available node. When the formerly disconnected node reconnects, it is informed by the observer that its version of the process instance is no longer valid. However, this approach does not work in all cases, e.g., a non-compensatable activity cannot have a timeout since we cannot invalidate the process instance if the node reconnects and already has executed the activity. Again, O'GRAPE supports the application developer in validating the correctness of process model.

4. Performance Evaluations

In this section, we investigate the potential of P2P process execution with a set of experiments. The goal is to evaluate that, at least for a basic setup and workload, P2P execution scales better than a centralized approach and that meta data management does not hinder scalability.

4 This actually is only true if the service supports the 2PC protocol. Otherwise, the result might get lost if the host fails immediately after the delivery of the result and before it is made persistent by the HDB.

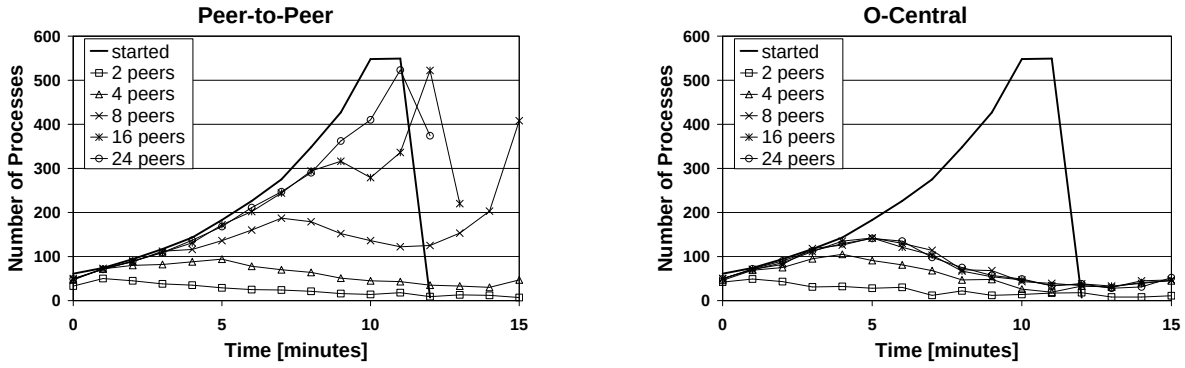
4.1. Evaluation Settings

In these preliminary experiments, we use idealized workloads and mainly investigate navigation costs and load balancing features of OSIRIS compared to an equivalent centralized approach. For the sake of comparison, we have modified OSIRIS slightly in order to allow for centralized navigation of processes. We call this modified version "O-Central". While both systems, i.e., OSIRIS and O-Central, share the same implementation, modules and configurations, they only differ in the way they execute processes. In OSIRIS, each node can instantiate and advance a process using the P2P approach described in Section 3. O-Central, on the other hand, consists of an additional peer that navigates the processes but delegates the execution of activities to the other peers. This approach represents a typical setup with coordination of distributed (web) services by a central process management system (e.g., IBM WebSphere Application Process Choreographer [34]). Using the same infrastructure and algorithms allows us to directly compare the performance figures of our P2P approach with the ones of a centralized process manager.

Our experimental setup determines the scalability characteristics of the two approaches under investigation (OSIRIS vs. O–Central). We use a linear process invoking generic activities that are available at all peers. This process corresponds to a typically short running data request process of clients. Each activity accrues costs of 2 seconds on average, and the overall execution costs of a process is 10 seconds on average (5 activities on average). Clients are simulated with a queuing model triggering a process instantiation every Δt milliseconds on average. In order to visualize the scalability characteristics, we decrease Δt every minute by 10%. Thereby, the load of the system steadily increases and, at some point, we may observe an overload situation, i.e., processes are significantly delayed due to full work queues at the peers and at the process manager. This experiment is repeated for different numbers of peers. Theoretically, the system should improve the more peers are available for activity execution, i.e., we should observe the break-down at a later point in time.

4.2. Scalability of Peer–to–Peer Navigation

The graphs in Figure 3 presents the results of the scenario described above for the OSIRIS and the O-Central implementation. The x-axis denotes the time in minutes from the start of the scenario, and the y-axis denotes the number of started processes (thick line) and the number of finished processes (all other plots) in an interval of one minute. As described in the scenario, the number of started processes (thick line) increases every minute by 10% and stops at roughly 550 processes per minute. After 11 minutes, the clients stop creating new processes and the system may



(a) OSIRIS Configuration

(b) Central Configuration

Figure 3. Process Throughput in Different Configuration Settings

work off all active instances. The remaining plots (the ones with markers) denote the number of finished processes per minute for a setup with $2, 4, 8, 16$ and 24 peers.

Let us first consider Figure 3(a) which depicts the throughput of OSIRIS. As we can see, P2P execution with only a few nodes (less than 16) is not capable to handle the full workload, i.e., the system breaks down with only a small number of concurrent processes (between 50 and 200). However, adding more peers to the system successfully delays the point where the system becomes overloaded. In fact, with 24 peers, OSIRIS is able to cope with a maximum load of 550 concurrent process instances which is not the case for the centralized approach O-Central, depicted by Figure 3(b). Although the system is able to significantly improve from 2 peers to 8 peers, it does not scale any further beyond 8 peers. The reason for this behavior is due to the central process navigation: it only allows for a limited number of concurrent process executions (logging, auditing, persistence, navigation, etc.) at a time. The throughput of the central approach could be improved by using faster machines or by using a clustered approach like in IBM WebSphere [34]. However, the limitation on the number of concurrent processes always remains regardless of how many peers we add to the system. OSIRIS, on the other hand, appears to scale almost perfectly with the number of peers in the network. If the workload becomes too large, we simply can increase the number of peers to cope with the additional load. In the central solution, one would have to replace the central machine with a much faster one to cope with an increased workload (if such a machine exists).

5. Related Work

The presented OSIRIS infrastructure provides a scalable distributed process navigation platform. To achieve this, it combines a rich set of aspects. Based on the hyperdatabase vision [28], we have combined ideas from process management, peer–to–peer networks ([3] gives an overview), database technology, and GRID [14] infrastructures. All parts are glued together to a value-added, coherent whole. Processes in OSIRIS are running within a peer–to–peer community that is established by the individual service providers (in [12], service providers acting as peers are called MARCAs). However, in contrast to relatively simple file sharing applications, the execution of processes over services requires a significantly richer set of functionality. Therefore, OSIRIS implements process management concepts like state–of–the art systems (e.g., IBM WebSphere Application Process Choreographer [34], BizTalk [21]). These systems realize a centralized approach, where every call to a service provider returns to the process engine. Albeit navigation tasks can be distributed in a cluster, storage of process instances usually is done by using a single, centralized database instance (products like Oracle 10g [24] can also support clustered databases). Some prototype systems features similar architectures (e.g., Mentor–lite [16], SDM [2]). Although executing processes in a peer-to-peer way without involving a centralized component, OSIRIS provides transactional guarantees, following the model of transactional processes [30]. METUFLOW [10] and TransCoop [1] also provide transactional semantics for workflows in a distributed environment.

In the distributed Mentor–lite [16] approach, the setting of the process engine in a cluster can be changed actively using a configuration tool, while OSIRIS distributes process execution over all available service providers, and locally runs processes on provider hosts. This requires a middleware layer –the local HDB layer– on every participating component resp. service provider. A wide range of middleware solutions like .NET [23], J2EE implementations or CORBA follows similar ideas. In terms of process management, the OSIRIS middleware however provides much

more functionality, in particular meta data replication, auditing, and system workload monitoring are also part of the OSIRIS middleware.

Besides peer-to-peer process execution, dynamics both in terms of the current service providers as well as in terms of changes of their load are addressed by OSIRIS. Even dynamic changes of process models are supported. OSIRIS guarantees that each process instance is executed consistently, following the version of the process at instance creation time. For several reasons, e.g., medical treatment running processes instances have to migrate to the newest process definition, therefore systems in this field like ADEPT$_{flex}$ [26] or HematoWork [22] have to deal with process instance evolution. Ideas for migrating running process instances are currently not implemented in OSIRIS. However, these concepts are orthogonal and could be seamlessly integrated. The execution of process instances in OSIRIS follows late service binding which realizes a runtime lookup for running instances of a certain type. Other systems like, for instance, ServiceGlobe [19] implement a service discovery based on tModel types, or include service discovery into the process navigation (e.g., ISEE [20], eFlow [7] or CrossFlow [17]). OSIRIS' service discovery relies on precomputed tables, available at every node. These tables allow for a very fast decision at runtime between several semantic equivalent service types. The content of these tables can be maintained manually or using an automated semantic approach [8, 32]. Each service type is bound to a publish–and–subscribe topic that provides a channel to the actual providers. Similar ideas can also be found at other several process management systems (see, for instance, [9]). However, conventional implementations of publish–and–subscribe techniques either require a centralized publish/subscribe broker or use broadcast technologies. In contrast, the OSIRIS peer–to–peer solution needs neither of them, but uses transparent routing mechanisms similar to those of MIRS [15] or DBCache [5].

6. Conclusions and Future Work

Service-oriented computing not only necessitates the possibility to combine existing services to applications at a higher level of abstraction (processes), these applications also impose certain requirements on the infrastructure for process support. In addition to the dynamics of service providers, scalability in terms of the number of services and processes the infrastructure can support is an important issue. State–of–the–art process management systems feature highly optimized algorithms to provide a certain degree of throughput and performance. For that, they however require specialized, powerful hardware. Nevertheless, the performance and throughput of a centralized approach to process management will still be limited when the number of concurrent process instances further increases. A true peer–to–peer based approach, in contrast, where the process execution is distributed among the peers of the entire network is able to significantly outperform a centralized approach in terms of scalability.

In this paper, we have presented the OSIRIS system as an implementation of a hyperdatabase that provides process support in a true peer-to-peer way. Peer-to-peer process execution is achieved by collecting metadata on the available processes, providers, and the load of the latter using global repositories and by applying sophisticated replication algorithms to distribute this metadata. Local hyperdatabase layers associated with each service provider manage replicas of global metadata that is sufficient to locally drive the execution of a process. However, in order to provide a reliable infrastructure for process execution even without central control, OSIRIS applies sophisticated failure handling strategies. Another important aspect is to support correct concurrent process executions. To do this –also in a truly distributed way– we are currently investigating distributed concurrency control protocols [18]. These protocols will enable process synchronization without introducing a centralized component through which each process instance has to be routed. Moreover, we have presented first experiments underlining the scalability features of OSIRIS. In a next step, we want to more carefully investigate under what circumstances the peer-to-peer approach is more beneficial than a centralized approach. For that purpose, we are currently developing a benchmark suite for various application scenarios (including failure handling and concurrency) which should reveal the strengths and weaknesses of different system architectures.

Acknowledgment: We want to thank Frank Leymann for supporting the work on OSIRIS and IBM for providing us with equipment enabling further studies on the benefits of the peer-to-peer approach of OSIRIS.

References

[1] K. Aberer, J. Klingemann, T. Tesch, J. Wäsch, and E. Neuhold et al. Transaction Support for Cooperative Work: An Overview of the TRANSCOOP Project. In *Proc. of the Workshop on Extending Data Management for Cooperative Work*, Darmstadt, Germany, 1997.

[2] A. Altintas, S. Bhagwanani, D. Buttler, S. Chandra, Z. Cheng, M. A. Coleman, T. Critchlow, A. Gupta, W. Han, L. Liu, B. Ludäscher, C. Pu, R. Moore, A. Shoshani, and M. Vouk. A Modeling and Execution Environment for Distributed Scientific Workflows. In *Proc. of the Int. Conf. on Scientific and Statistical Database Management (SSDBM)*, Cambridge, USA, 2003.

[3] M. Bawa, B. F. Cooper, A. Crespo, N. Daswani, P. Ganesan, H. Garcia-Molina, S. Kamvar, S. Marti, M. Schlosser, Q. Sun, P. Vinograd, and B. Yang. Peer-to-peer research at Stanford. *SIGMOD Rec.*, 32(3):23–28, 2003.

[4] P. Bernstein and E. Newcomer. *Principles of Transaction Processing.* Morgan Kaufmann Publishers, 1997.

[5] C. Bornhövd, M. Altinel, S. Krishnamurthy, C. Mohan, H. Pirahesh, and B. Reinwald. DBCache: Middle-tier Database Caching for Highly Scalable e-Business Architectures. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, San Diego, California, USA.

[6] R. Braumandl, M. Keidl, A. Kemper, D. Kossmann, A. Kreutz, S. Seltzsam, and K. Stocker. ObjectGlobe: Ubiquitous query processing on the Internet. *VLDB Journal: Very Large Data Bases*, 10(1), 2001.

[7] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M. Shan. Adaptive and Dynamic Service Composition in eFlow. In *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*, Stockholm, 2000.

[8] F. Casati and M.-C. Shan. Semantic Analysis of Business Process Executions. In *Proc. of the Int. Conf. on Extending Database Technology*, Prague, Czech Republic, 2002.

[9] U. Dayal, M. Hsu, and R. Ladin. Business Process Coordination: State of the Art, Trends, and Open Issues. In *Proc. of the Int. Conference on Very Large Databases*, Roma, Italy, 2001.

[10] A. Doğaç, E. Gokkoca, S. Arpinar, P. Koksal, I. Cingil, B. Arpinar, N. Tatbul, P. Karagoz, U. Halici, and M. Atinel. *Design and Implementation of a Distributed Workflow Management System: METUFlow*, p. 60–90. In: [11]. Istanbul, Turkey, 1997.

[11] A. Doğaç, L. Kalinichenko, T. Özsu, and A. Sheth, eds. *Workflow Management Systems and Interoperability*, volume 164 of *NATO ASI Series F: Computer and System Sciences*. Springer-Verlag, 1998. Proceedings of the NATO Advanced Study Institute (ASI) on Workflow Management Systems. Istanbul, Turkey, August 1997.

[12] A. Doğaç, Y. Tambag, A. Tumer, M. Ezbiderli, N. Tatbul, N. Hamali, C. Icdem, and C. Beeri. A Workflow System through Cooperating Agents for Control and Document Flow over the Internet. In *Cooperative Information Systems, 7th International Conference, CoopIS 2000*, volume 1901 of *Lecture Notes in Computer Science*, p. 138–143, Eilat, Israel, 2000.

[13] A. Elmagarmid, Y. Leu, W. Litwin, and M. Rusinkiewicz. A multidatabase transaction model for InterBase. In *Proc. of the Int. Conference on Very Large Databases*, Brisbane, Australia, 1990.

[14] I. Foster and C. Kesselman, eds. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2^{nd} edition, 2004.

[15] R. Giladi, C. Glezer, N. Melamoud, P. Ein-Dor, and O. Etzion. The metaknowledge-based intelligent routing system (MIRS). *Data Knowl. Eng.*, 34(2):189–217, 2000.

[16] M. Gillmann, G. Weikum, and W. Wonner. Workflow Management with Service Quality Guarantees. In *Proc. of the ACM SIGMOD Int. Conf. on Management of Data*, Madison, Wisconsin.

[17] P. Grefen, K. Aberer, H. Ludwig, and Y. Hoffner. CrossFlow: Cross–Organizational Workflow Management for Service Outsourcing in Dynamic Virtual Enterprises. *IEEE Data Engineering Bulletin*, 24, 2001.

[18] K. Haller and H. Schuldt. Consistent Process Execution in Peer-to-Peer Information Systems. In *Proc. of the Conf. on Advanced Information Systems Engineering (CAiSE)*, Klagenfurt/Velden, Austria, 2003.

[19] M. Keidl, S. Seltzsam, K. Stocker, and A. Kemper. ServiceGlobe: Distributing E-Services Across the Internet.

[20] J. Meng, S. Su, H. Lam, and A. Helal. Achieving Dynamic Inter-organizational Workflow Management by Integrating Business Processes, Events, and Rules. In *Proc. of the Annual Hawaii Int. Conf. on System Sciences (HICSS)*, Big Island, Hawaii, USA, 2002.

[21] B. Metha, M. Levy, G. Meredith, T. Andrews, B. Beckman, J. Klein, and A. Mital. BizTalk Server 2000 Business Process Orchestration. In *IEEE Data Engineering Bulletin*, volume 24, 2001.

[22] R. Müller and E. Rahm. Rule-Based Dynamic Modification of Workflows in a Medical Domain. In *btw*, Freiburg, Germany, 1999.

[23] Microsoft .NET. `http://www.microsoft.com/net/`.

[24] Oracle Database 10g: The Database for the Grid. `http://otn.oracle.com/products/database/oracle10g/`.

[25] C. Pu, K. Schwan, and J. Walpole. Infosphere project: system support for information flow applications. *ACM SIGMOD Record*, (1), 2001.

[26] M. Reichert and P. Dadam. $\text{ADEPT}_{\text{flex}}$ — Supporting Dynamic Changes of Workflows without Losing Control. *Journal of Intelligent Information Systems*, 10(2), 1998.

[27] H.-J. Schek, H. Schuldt, C. Schuler, and R. Weber. Infrastructure for Information Spaces. In *Proc. of the East European Conf. on Advances in Databases and Information Systems (ADBIS)*, volume 2435 of *Lecture Notes in Computer Science*, Bratislava, Slovakia, 2002.

[28] H.-J. Schek, H. Schuldt, and R. Weber. Hyperdatabases – Infrastructure for the Information Space. In *Proc. of the Working Conf. on Visual Database Systems (VDB)*, Brisbane, Australia, 2002.

[29] H. Schuldt. Process Locking: A Protocol based on Ordered Shared Locks for the Execution of Transactional Processes. In *Proc. of the ACM Symposium on Principles of Database Systems (PODS)*, Santa Barbara, California, USA, 2001.

[30] H. Schuldt, G. Alonso, C. Beeri, and H.-J. Schek. Atomicity and Isolation for Transactional Processes. *ACM TODS*, 2002.

[31] C. Schuler, R. Weber, H. Schuldt, and H. Schek. Peer-to-Peer Process Execution with OSIRIS. In *Proc. of the Int. Conf. on Service Oriented Computing (ICSOC)*, Trento, Italy, 2003.

[32] K. Sivashanmugam, K. Verma, A. Sheth, and J. Miller. Adding Semantics to Web Services Standards. Las Vegas, Nevada, USA, 2002.

[33] R. Weber, C. Schuler, H. Schuldt, H.-J. Schek, and P. Neukomm. WebService Composition with O'GRAPE and OSIRIS. In *Proc. of the Int. Conference on Very Large Databases*, Berlin, Germany, 2003.

[34] IBM WebSphere Application Process Choreographer. `http://www-106.ibm.com/developerworks/websphere/zones/was/wpc.html`.