INTEGRATING AND OPTIMIZING BUSINESS PROCESS EXECUTION IN P2P ENVIRONMENTS

Marco Fernandes, Marco Pereira, Joaquim Arnaldo Martins and Joaquim Sousa Pinto
Universidade de Aveiro, Campus Univ. Santiago, 3810-193 Aveiro, Portugal

Keywords: Business process reengineering, Emerging technologies, Distributed networks.

Abstract: Service oriented applications and environments and peer-to-peer networks have become widely researched topics recently. This paper addresses the benefits and issues of integrating both technologies in the scope of business process execution and presents proposals to reduce network traffic and improve its efficiency.

1 INTRODUCTION

Service oriented computing has been a popular research topic and one of the main drivers for the software industry (Bichler & Lin, 2006). The basic principle behind service orientation is that distributed, modular, autonomous and interoperable services available in the network can be used to enhance or extend application capabilities or even to perform some of its core functionalities. Several concepts based on service orientation have surfaced in recent years:

- Service-oriented architectures (SOA) – infrastructures in which business processes are implemented through distributed services (Erl, 2005) (Marks & Bell, 2006);
- Software as a service (SaaS) – a model of software licensing in which services are provided on demand (Bennett, 2000).
- Cloud computing – the availability of services and resources on the internet, which can be consumed (and meshed) in a variety of applications. Cloud computing is commonly thought as collections of services which can also be consumed for personal use (Gruman, 2009).

Properly managing and consuming a wide range of available services presents a problem of standardization of those services. Even in the case where all services are SOAP Web Services, a standard and widely adopted technology, it is required to define the methods, data structures and interactions a priori. In the simplest case, consumers may use only a few services to add extra functionality or perform a very specific task, and in this case developers can easily perform a service call or create a service proxy. However, service orientation advantages are only being partially explored in this scenario.

Service orientation allows creating complex, composite services which are logical aggregations of other services in a flow – the business process. Orchestration and choreography languages allow defining information flows and creating these composite services. A combination of SOA, business process choreography and Web Services can bring numerous advantages for businesses (Zimmermann, 2004) such as higher automation and process integration.

1.1 BPEL

The Business Process Execution Language for Web Services (WS-BPEL or simply BPEL) is the standard for business process execution. It originated from the merger of two proprietary orchestration languages (IBM's WSFL and Microsoft's XLANG) and makes use of several XML standards: WSDL 1.1 and XML Schema 1.0 (data model), and XPath 1.0 and XSLT 1.0 (data manipulation).

1.2 P2P Networks

A Peer-to-peer (P2P) application is a networked system whose architecture does not (usually) rely on dedicated servers; instead, each network node (the peers) acts as both client and server.

The most common advantages of a P2P based network are (Taylor, 2005):

Fernandes M., Pereira M., Arnaldo Martins J. and Sousa Pinto J. (2010).
INTEGRATING AND OPTIMIZING BUSINESS PROCESS EXECUTION IN P2P ENVIRONMENTS.
In *Proceedings of the 12th International Conference on Enterprise Information Systems - Information Systems Analysis and Specification*, pages 171-177
DOI: 10.5220/0002872601710177

- It can operate at the edges of the Internet, behind firewalls and NAT systems;
- It supports highly transient connections;
- It can take advantage of unused resources of connected nodes.

Most P2P applications are very file oriented: peers are usually limited to index, search and transfer files.

1.3 BPEL and P2P

If we can properly integrate BPEL and peer-to-peer networks, a great number of advantages inherent from P2P will become available for the execution of business processes:

- Services located in computers behind firewalls and NAT systems could become reachable;
- Service availability can be largely increased by replicating in several peers;
- Previously unused machines can host services to be used in an orchestration; idle times could potentially be reduced;
- Dynamic service discovery and assignment in the P2P network can increase the flexibility and fault-tolerance of the process;
- Delegating part of the orchestration to other engines can help reduce the data transferred in the network.

The scope of this work is about analyzing how each of these characteristics and behaviours could be used to improve the efficiency of a service oriented environment.

1.4 Objectives

This work focuses on how to improve the performance and robustness of the execution of business processes in a P2P environment.

In order to design a service oriented environment capable of properly integrate the BPEL language with a P2P network, we established a few pre-requisites.

On one hand, existing standards should be kept unmodified. Namely, and unless absolutely required, one should try to accommodate the existing BPEL and its underlying standards. Secondly, no particular assumptions should be made on the underlying network. The proposal should transparently accommodate different topologies and fallback to a not-optimized state if peers do not offer specific capabilities.

2 RELATED WORK

To delegate a process to multiple BPEL engines, the orchestration must be partitioned into smaller BPEL service sequences. Some authors have presented possible techniques to perform such partitioning while trying to improve the overall throughput. One such proposal (Montagut & Molva, 2005) consists in decentralizing the flow control and dynamically selecting roles. The presented approach considers only simple flows, without synchronization, restrictions, or error handling. A stateless model is adopted: a node, after executing an activity, transfers all state information to the next node.

Another technique, proposed by IBM researchers (Nanda et al., 2004), consists in partitioning a BPEL instruction sequence into a set of distributed processes, eventually reordered but with the same final output. The algorithm divides activities into fixed (receive, reply, and invoke) and portable (the rest). Each fixed activity is aggregated with a process service (receive/reply pair with the entry point), while portable ones can be moved. Parallelism is also automatically extracted from the flow activity. The final arrangement consists in partitions with one fixed activity and zero or more portable ones. According to the authors, this algorithm may increase its throughput 30% at normal system load and by a factor of two under high load, but it has the assumption that every node has BPEL runtime capabilities.

Khalaf et. al (Khalaf, 2008) discusses how to maintain data dependencies when partitioning a BPEL process into fragments. The proposal aims to tackle issues that arise from parallelism and shared variables. Our work is for now focused on the technology integration for simpler processes. While BPEL is mainly focused on the orchestration of SOAP Web Services, some efforts have been made to describe REST services with WSDL (Mandel, 2008) and to compose such services using BPEL (using extensions) (Pautasso, 2008). These contributions may be considered in the future.

3 DYNAMIC DISCOVERY

The BPEL language is built upon Web Services and therefore uses the Web Service Definition Language (WSDL) extensively. In fact, both the process and its partners (the service providers) are exposed as WSDL services.

A simplified skeleton of a BPEL process definition is presented in Figure 1.

```
<wsdl:definitions>?
  <!-- types, messages, portTypes, and
       parternLinkTypes -->
</wsdl:definitions>
<process>
  <import namespace="URI" location="URI"
     importType="URI" />*
  <partnerLinks>?
    <partnerLink name="NAME"
       partnerLinkType="QNAME" myRole="NAME"?
       partnerRole="NCName"? />
  </partnerLinks>
  <variables>?
    <!-- the variables -->
  </variables>
  <sequence>
    <!-- the activities -->
  </sequence>
</process>
```

Figure 1: BPEL XML process.

The process definition starts by declaring the WSDL types, messages, portTypes and parterLinkTypes involved in the activity execution. Namespaces are then imported, the partner links and its roles defined, and the variables declared. Only then the actual process activities are defined within the <sequence /> element.

When a service provider hosts a Web Service, it makes its WSDL definition available at some location. By inspecting this document, one can locate the service URLs in the <soap:address/> elements. A BPEL engine could however use a discovery service to find providers hosting those services – with identical WSDL definitions but at different <soap:address/> elements. Such a simple modification in the behaviour of a BPEL engine makes the process execution more flexible, as it is no longer tightly bound to specific providers. As a consequence, services can be replicated and dynamically chosen to increase the throughput.

3.1 Service Discovery in P2P

Traditionally, P2P applications were designed for file sharing purposes. Networks such as Gnutella, BitTorrent, and Napster are file oriented rather than resource oriented (files, services, etc.).

To make use of a P2P network for service discovery, we need an infrastructure which allows publishing and indexing WSDL service definitions and querying for peers which provide specific WSDL services.

To accommodate these requirements, we can use JXTA, an open-source project which consists in a group of open and generic protocols to connect heterogeneous devices in a P2P network.

JXTA peers are known between each other through advertisements: nodes publish information about themselves and the resources they hold using Peer, Peer Group, Module Class, Module Specification, and Module Implementation advertisements. WSDL definitions from service providers in a P2P network can also be published using advertisements; in the JXTA-SOAP (Amoretti, 2008) project, they are encapsulated in Module Specification advertisements. This project provides an add-on to the base framework, allowing Web Service calls to be made using the P2P network rather than regular HTTP requests. That is accomplished by creating proxies at the peers, which serialize and de-serialize SOAP requests and responses into JXTA messages.

For service discovery to properly function under JXTA-SOAP, such advertisements must include the service WSDL, optionally with additional information such as a service's name, creator, version, and description. The discovery mechanism is outside the scope of JXTA-SOAP, and therefore the service lookup is actually implemented by an application, which may query any of these properties.

There are a few valid options for choosing which values should be in used in the service description (publishing) and in the queries sent to the network (discovery). Probably the most error-resilient method would be to query services by its WSDL hash (without the <soap:address/> element). This could however be inconvenient both at the provider side (as more parsing and computing operations would be needed) and the consumer/application end (as hashes would have to be stored somewhere).

A simpler option consists in using the targetNamespace attribute of the <wsdl:definitions/> element in the service WSDL, which can easily published by providers. On the consumer end, discovering services using an URI rather than a hash string is much friendlier.

There are a few disadvantages in the approach used by the JXTA-SOAP API. Since Web Services must be created and published with Axis , an Apache SOAP engine written in Java, one is obliged to only use Java based services (unlike the JXTA framework, whose API is available in a variety of

programming languages). It is very limitative, since it makes all existing non-Axis Web Services useless unless an Axis proxy is made for each of them with an identical interface.

3.2 Service Invocation in P2P

By using JXTA-SOAP, Web Service invocation is accomplished by transmitting SOAP messages using JXTA pipes. Thus, the relatively verbose XML documents (Elfwing & Paulsson, 2002), which are already serialized both at the consumer and provider, pass through an additional serialization layer. The P2P network is therefore introducing extra overhead to service invocation. While this may be absolutely necessary when service providers and consumers cannot directly exchange messages with each other, such as when at least one part is behind a firewall or NAT system, in many of the cases that does not hold true. It seems therefore apparent that applications using this service enabled P2P network could improve its efficiency if they knew whether a service provided by a peer is within direct reach.

Outside a P2P network, a computer's services are inaccessible to other machines basically in two situations:

- Local services – in this relatively common case, services are published in a private HTTP server blocking external access;
- Intranet and/or firewalled services – in this scenario, two computers cannot connect each other (although a service could be used by nodes inside the intranet).

In order to design an efficient service oriented P2P network, we can therefore work two distinct scenarios. If a service is considered to be critical to the proper functioning of an application and it is not publically available at the internet, a JXTA-SOAP proxy should always be created to guarantee its widespread availability (we are, of course, excluding those Web Services already created on Axis with this framework). If not, creating a P2P proxy is a matter of convenience – when setting up a services network one is aware that availability could be compromised depending on the network topology and security policies.

There is one final issue to be addressed: with JXTA-SOAP created services or proxies, service advertisement is accomplished transparently. Also, one can add a flag to these advertisements to indicate such services are being encapsulated by this framework. All other services, however, have no built in mechanism to make them known and discoverable by the other nodes. To overcome such limitation, and to avoid having to build a proxy for every non Axis Web Service, we designed an extension module which will be responsible for handling these advertisements. The behaviour of this module consists in reading a configuration file with the location of the WSDL definitions of the services it should handle and publishing the advertisements (with the URL and the WSDL of the Web Service) on the P2P network. It will provide no serialization or execution methods.

Figure 2 depicts the two distinct service invocation methods in a P2P network. LAN A has two leaf nodes and a rendezvous node (allowing connections to other networks), which is publically addressable from any computer. A node in LAN B wishes to invoke two services available on the first intranet: the first (1) is hosted by the rendezvous peer in a public HTTP server while the second (2) is in a leaf node.

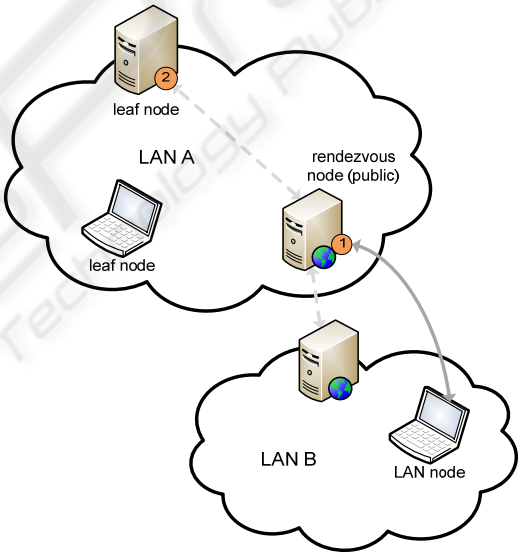

Figure 2: P2P service invocation.

After the initial discovery process, the consumer node finds service 1 to be located at a public URL location and can therefore invoke it using HTTP as the transport protocol.

Now let us consider Service 2 to be publically available to computers in LAN A. In that case, both our advertising module and a proxy could be in use, allowing for both HTTP calls and P2P calls. The consumer peer, unaware of whether the provider is in the same LAN, may try to call the "public" Web Service and, in case of error or network timeout, invoke it using the P2P proxy. If our extension module was not installed at the provider, only the later operation would be available and therefore

performed initially.

3.3 Service Replication

One extra optimization layer can be set up on top of this service network. It is apparent that services could be replicated on a P2P network to increase availability and eventually responsiveness. Such concept does not differ much from file replication, which is implemented by several file-sharing P2P applications. The requirements and dependencies make it however a lesser trivial issue to address. Many services may need more complex dependencies (such as installed programs or libraries) or even have specific hardware requirements.

Describing and managing software (and hardware) dependencies is a difficult task, and several issues and possible conflicts must be taken into account. For now, let us consider the simpler cases: self contained executables or folders with no installation or environment modifications (CLASSPATH, registry, etc.) required. For such components, one could think of replication as yet another service available at some peers (a service "push"), which could be published and discovered as any other. The input parameters of such service are the required resources (executables, WSDL, and dependencies).

It then becomes a matter of deciding if and when to replicate a given service. We propose that a simple metric is applied: if the provider implements a service queue, a replication request could be triggered so that it is broadcast to the network whenever a waiting threshold is reached. If, on the other hand, the provider simultaneously responds to incoming requests (threaded work), an internal mechanism would have to trigger the request when the number of simultaneous threads passes the threshold.

4 OPTIMIZING ORCHESTRATION

While the previously proposed modifications in the P2P layer can be seen as independent to any specific service environment, one must think in terms of a business process management and execution application to fully take advantage of them.

As discussed in the previous section, a P2P aware BPEL engine can take advantage of the available distributed (and eventually replicated) services to dynamically discover and invoke them.

The advantages are not limited to dynamic discovery. Traditionally, BPEL execution is a centralized process, in which service calls are dispatched to partner links and state is centrally managed. However, distributing the orchestration process by the service providers has several advantages, especially in high load scenarios and/or when there is a large amount of data being transferred between service providers and consumers. A careful partitioning process can reduce the number of messages and amount of data transferred and increase throughput.

4.1 Process Delegation

Previous work assumes all partner nodes have BPEL capabilities, which may not be convenient in most enterprises. We can however fall back to an always working solution.

Lets us consider our initial (starting point) engine is capable of dynamically discovering services. Before the execution starts, the runtime can find not only the service providers but also which nodes offer BPEL execution – since BPEL is seen itself as a Web Service, our advertising module could as easily publish this service in the network. If no other engine is found, process management will proceed as usual – in a centralized fashion. If, however, one or more engines are found, the BPEL process definition can be partitioned and parts of the process delegated to those peers. If any of those engines are P2P aware, this procedure could eventually be further partitioned.

Without the "BPEL in every peer" assumption, the partitioning mechanism proposed in related work is no longer valid. Nevertheless, some principles remain true: when there is parallel execution (a flow activity), an entire branch can still be partitioned if the first invoke activity exists at a BPEL-capable peer.

Furthermore, information about the services themselves could be used to try to infer the best tasks to be delegated. Process delegation can greatly reduce the amount of data being transferred by eliminating the round trips in the invocation calls. We are therefore interested in those services whose transmitted messages/variables are predictably large, particularly in the response message. While there is no standard way to know a priori which those services are, a few assumptions could be made.

The return type of a service, for instance, can provide hints on the extent or size of the response

message. It is safe to assume that the efficiency gain will likely be much smaller when delegating the process to a service returning an integer than the gain when doing so on a service returning an array of bytes. We suggest the enforcement of a simple rule: perform no process delegation if the next service returns messages with simple types (numeric, Boolean, and strings or complex types based on these types).

4.2 Limitations

Inner process delegation does present some difficulties when trying to achieve some common features such as process monitoring. While keeping track of this progress is simple in a centralized scenario, doing so in a decentralized orchestration environment is not as trivial. While this is a non-critical issue and one which only occurs for those engines enhanced to support BPEL delegation, one should be aware of this limitation.

5 CASE STUDY

Let us consider a digital newsstand website which allows registered users to view a range of newspapers as they were published. The website receives PDF files from publishers, which are converted into an image format (JPEG) to be shown in a viewer, and whose texts are extracted for searching purposes. As part of the submission process, several services are invoked:

- Image conversion/resizing
- Automatic image whitespace cropping
- PDF text extraction
- Optical character recognition (OCR)
- Storage (whose response is the new system identifier)

Figure 3 depicts a functional diagram of how this process is implemented. The input to this process is a PDF file and a XML document with the metadata. The process starts with two parallel branches. In the first one, the text from the PDF file is extracted. In the second, the PDF is converted to an array of PNG files, whose white space is then cropped. The resulting images are then used to make an OCR (whose service input must be in TIFF files) and to convert to the final, screen resolution, JPEG images. The final activity consists in sending all non-intermediary files to a storage service.

Assuming each of the blocks in the diagram represent a service in a different peer (the worst case scenario), there is a large amount of data being passed back and forth through the wire. With centralized orchestration, one expects the total amount of data to be:

$$T = 3S_{PDF} + 5S_{PNG} + 2S_{TIF} + 2S_{OCR} + 2S_{TXT} + 2S_{JPG} + S_{XML} + S_{ID}$$

where S_x represents the message size of the transmission of X.

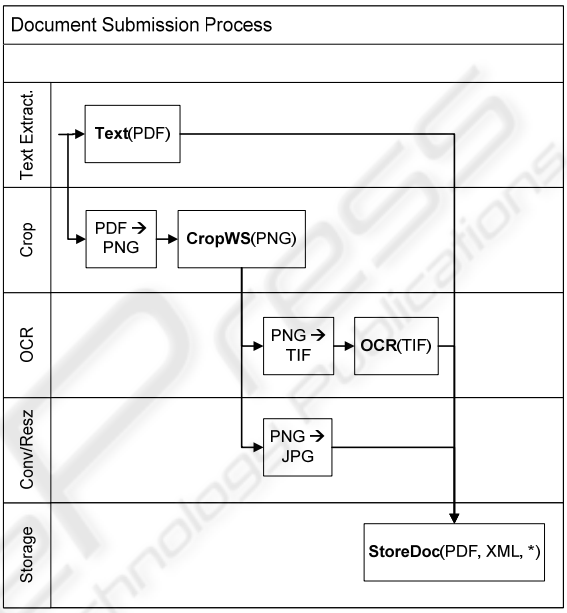

Figure 3: Cross functional diagram of a document submission process.

The simplest improvement one can do in branched processes is to delegate an entire branch of activities. Let us suppose the image conversion service is available at a BPEL-capable node. In that case, a BPEL process can be made with the activities in the "OCR" band from the diagram. By doing so, the PNG to TIFF conversion call is replaced with a process start call and, since the TIFF files don't have to be returned to the original caller, those response messages no longer have to be transmitted through the wire. In this particular digital newsstand application, the intermediate TIFFs generated are about 3MB each, and so this modification would reduce close to 120 MB of traffic in a 40 page newspaper.

This procedure could be repeated and, in the optimal scenario where all peers can run BPEL processes, the partitioning algorithm could be identical to those used in the related work. However, some delegation could prove to be counter-productive: consider there were services just before the storage stage dedicated to provide unique

identifiers, produce checksums, or calculate hashes based on the metadata of the new document. Delegating the orchestration of one those services and the storage to those providers would actually increase network usage: instead of invoking the first service, receiving the id/checksum and sending all to the StoreDoc, PDF and image files would have to go to the first service and from there to the StoreDoc provider. Therefore, instead of

$$T_{final} = 2S_{XML} + 2S_{ID} + S_{PDF} + S_{JPG}$$

we would have

$$T_{final} = 2S_{XML} + S_{ID} + 2S_{PDF} + 2S_{JPG}$$

which represents one less SID but one more SPDF and SJPG. By using our proposed criterion, and since the id/checksum service has a predictably small (numeric) response message, no delegation would take place.

A final optimization could consist in trying to merge activities in peers providing multiple consecutive services. Although this could greatly reduce network traffic, it would be difficult to analyze the improvements of this strategy if factors such as throughput were to be weighed. The case of the last service called (storage) is however a particular one – if the P2P network were to be used also as the storage medium, this service could be directly executed by the caller peer.

6 CONCLUSIONS

In this paper, the advantages of integrating peer-to-peer networks, service orientation and process execution orchestration were discussed.

Based on existing frameworks and on the previous related work from other authors, we made some architectural analyses and presented proposals to improve the overall efficiency which covered the P2P network framework (dynamic discovery, invocation, and replication) and the way BPEL engines function (dynamic discovery, partitioning, and delegation). The proposed enhancements can easily fall back when the network does not support such optimizations.

ACKNOWLEDGEMENTS

This work was funded in part by FCT – Portuguese Foundation for Science and Technology – grant number SFRH/BD/23976/2005.

REFERENCES

Amoretti, M. e. a., 2008. Enabling Peer-to-Peer Web Service Architectures with JXTA-SOAP. In *IADIS e-Society*. Carvoeiro, 2008.

Bennett, K. e. a., 2000. Service-based software: the future for flexible software. In *Proc. Seventh Asia-Pacific Software Engineering Conference, APSEC'00*., 2000.

Bichler, M. & Lin, K.-J., 2006. Service-oriented computing. *Computer*, pp.99-101.

Elfwing, R. & Paulsson, U., 2002. *Performance of SOAP in Web Service environment compared to CORBA [master thesis]*. Sweden: Blekinge Institute of Technology.

Erl, T., 2005. *Service-oriented Architecture: Concepts, Technology, and Design*. Prentice Hall.

Gruman, G.a.K.E., 2009. *What cloud computing really means*. [Online] Available at: http://www.infoworld.com/article/08/04/07/15FE-cloud-computing-reality_1.html [Accessed March 2009].

Khalaf, R. K. O. L. F., 2008. Maintaining data dependencies across BPEL process fragments. *International Journal of Cooperative Information Systems*, 17(3), pp.259-82.

Mandel, L., 2008. *Describe REST Web services with WSDL 2.0*. [Online] Available at: http://www.ibm.com/developerworks/webservices/library/ws-restwsdl/ [Accessed March 2009].

Marks, E. A. & Bell, M., 2006. *Service-Oriented Architecture (SOA): A Planning and Implementation Guide for Business and Technology*. New York: John Wiley & Sons, Inc.

Montagut, F. & Molva, R., 2005. Enabling pervasive execution of workflows. In *International Conference on Collaborative Computing Networking, Applications and Worksharing*. San Jose, CA, EUA, 2005.

Nanda, M.G., Chandra, S. & Sarkar, V., 2004. Decentralizing execution of composite Web Services. In *Proceedings of the 19th annual ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications*. Vancouver, 2004. ACM.

Pautasso, C., 2008. BPEL for REST. In *7th International Conference on Business Process Management (BPM08)*. Milan, Italy, 2008. SpringerLink.

Taylor, I., 2005. *From P2P to Web Services and Grids: peers in a client/server world*. London: Springer-Verlag.

Zimmermann, O. e. a., 2004. Service-Oriented Architecture and Business Process Choreography in an order management scenario: rationale, concepts, lessons learned. In *Object-Oriented Programming, Systems, Languages and Applications Conference*. Vancouver, 2004. ACM.