Streaming Web Services and Standing Processes

Steffen Preißler, Hannes Voigt, Dirk Habich, Wolfgang Lehner
Technische Universität Dresden
Lehrstuhl für Datenbanken
dbgroup@mail.inf.tu-dresden.de

Abstract:
Today, service orientation is a well established concept in modern IT infrastructures. Web services and WS-BPEL as the two key technologies handle large structured data sets very inefficiently because they process the whole data set at once. In this demo, we present a framework to build standing business processes. Standing business processes rely on item-wise data set processing, exploit pipeline parallelism and show a significantly higher throughput than the traditional WS-BPEL approach.

1 Introduction

Today an increasing number of IT infrastructures are built as a *service-oriented architecture* (SOA). In SOA, independent systems provide their functionality as interoperable services. Systems group theses services in business processes and package them as further interoperable services. Web services and WS-BPEL are two key technologies to realize a SOA. The Web service specification [W3C02] offers standardized structures for self-description and message exchange and is, therefore, the well established standard for interoperable services. WS-BPEL [OAS07] provides workflow constructs to build fully-fledged processes with service calls as core activities.

Within the THESEUS research project [BMW07], the TEXO use case aims to built a SOA-based platform where services are tradable and business value networks can be established. As partner in TEXO, we investigate the efficient processing of large structured data sets in SOA environments. Our approach is a new type of process, which is called *standing business process*. It builds on top the workflow constructs of WS-BPEL, but exploits pipeline parallelism for large data set processing. Thereby large data sets can be considered as a stream of equally structured messages or a given set of equally structured data items. [LCF08] already discussed that the throughput of large data set processing can be increased significantly by exploiting pipeline parallelism. Traditional approaches only map each single item (or message) to a single process instance with a still step-wise execution model and single service calls. These approaches limit the processing semantic to single-item operations. However, common business process operations such as aggregations involve more than one data item.

As an example, consider the stock-ticker process in Figure 1, which handles incoming RSS-Feed messages. Whenever a message arrives, only interesting stocks are selected for

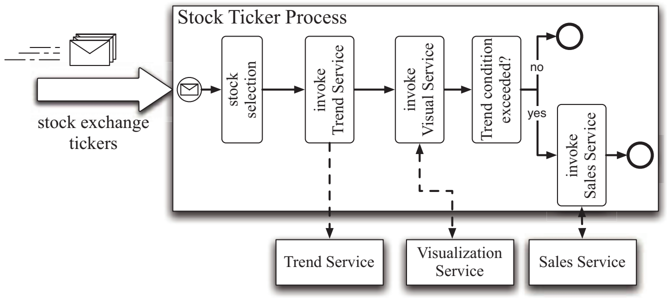

Figure 1: Sample stock ticker process

further processing. The selected stocks in a message are evaluated over a time window using a service (`Trend Service`) that monitors different stock trends and keeps a history of their values. The service returns the stock trend over the requested time window and the process sends this information to a visualization service, that displays the trend on a customer's dashboard. Afterwards the process compares the trend to a predefined value. If the trend exceeds a certain threshold, the `Sales Service` service will be triggered, otherwise nothing is done.

This type of application scenario cannot be executed efficiently with classic means. Traditionally, each ticker message triggers the explicit creation and execution of one process instance. Thereby each message is processed in its own context and a common context for ,e.g., in-house stock trend computation can not be exploited. Furthermore the start of a message's processing is delayed, until the process instance of the previous message has been executed successfully to ensure temporal integrity. This leads to a significantly lower throughput of incoming messages.

In this demo we introduce the novel notion of standing business processes that realize pipelined and context-preserving data set processing in the SOA world. To show the applicability of our concept, we present a framework to model and execute such standing processes. Our approach establishes real pipeline parallelism, increases the processing throughput and does not restrict the processing semantic.

2 Streaming in service-oriented environments

Traditional business processes in WS-BPEL follow an *instance-based execution model*. Every incoming message creates a dedicated process instance, which is executed isolated from all other instances. This type of process execution is not efficient for processing large amounts of incoming, equally structured messages that semantically belong to one context. However, in WSBPEL it is possible to express one context for a set of incoming messages with the help of a *while* loop and *correlation sets*. The *while* loop has to enclose all corresponding control flow activities for one message. Additionally, *correlation sets* route messages to specific process instances. The disadvantages of this approach are (i) the explicit modeling of a control flow loop with one iteration for every message, (ii) the still step-wise execution within the loop where only one activity is running and all others are idle and (iii) the need for a static value within the messages' body to use *correlation sets* and to map, thereby, messages with specific values to a specific process instance.

In order to use pipelined parallelism in combination with business process types like our stock ticker process, the process engine requires the *pipes and filters execution model*. In *pipes and filters* every activity of a process is executed as a single thread and each edge between two activities contains a queue, which buffers data that belong to specific messages. Additionally the semantic and the functionality of each control flow operator are adapted to work with input and output queues and to realize one common process context for all messages.

Service invocation and execution within a standing process needs adaptation, too. The *pipes and filters execution model* implies that services are called item-wise. Consequently, with the traditional service invocation pattern, data items lose their context on service side, since one service instance is created for every item. To preserve the context of the items on service side, a standing process pushes the message queue embracing *one* service invocation down to the service instance. In addition to that, the service execution is enhanced to process these items in a stream-based fashion. By this means the standing process adds streaming semantic to the service call. This enables the service to return already processed data items while still receiving request items. This stream-based service execution eliminates the overhead of single message creation compared to traditional item-wise service invocations and preserves the context of the items at the same time. [PVHL09] discusses this approach in more detail.

The visual modeling of standing processes with our framework heavily corresponds to the modeling of standard processes (see Figure 2). Similar to standard processes, the user chooses from a set of operators provided by the framework and orchestrates them to a workflow definition. What differs in the visual representation are the queue symbols between two connected activities. These symbols represent the already mentioned message queues, so that the user can configure them at design time (e.g., maximum queue size). When executing a standing process, our framework visualizes a running process instance by displaying the modeled graph and augmenting its graphical components with information about the current workload for every queue, the average execution time for every operator as well as path counters when utilizing switch operators.

3 Demo Details

Fundamentally, the demonstration will consist of two parts. In the first part, we demonstrate the execution of standing processes with our developed standing process engine. For this, we prepared a set of predefined processes. As an example, Figure 2 shows a screenshot of the stock-ticker process discussed in the introduction.

These demo part shows the applicability of our standing process concept and the imple-

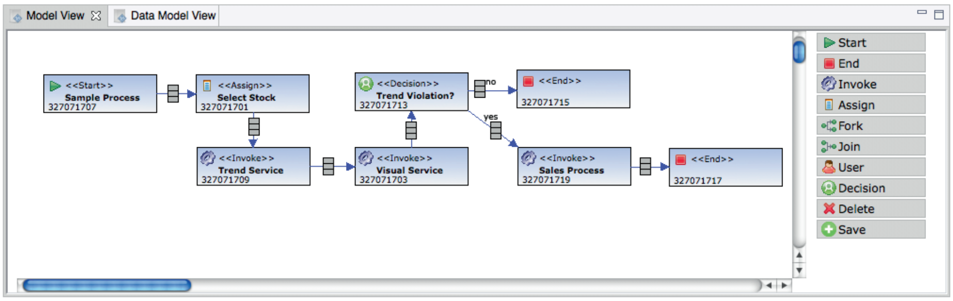

Figure 2: Screenshot of stock-ticker process

mented framework within several scenarios. In addition to the prepared processes, visitors of our demo desk will also have the possibility to experience the orchestration of new standing process definitions. This gives the visitors an understanding of the whole modeling approach and the benefits of standing processes.

In the second part, we present the implementation and usage of stream-based services. For this, we prepared stream-based services, traditional services, and an execution front-end to experimentally show the benefit of our approach. Furthermore, we describe our extension on the Web Service interface description, which allows us to identify and use stream-based services with our standing process framework. We welcome visitors of our demonstration desk to create new stream-based Web services and show the easy usage of our service framework. In this case, visitors of the demo will get an in-depth understanding of our developed concept.

4 Acknowledgements

The project was funded by means of the German Federal Ministry of Economy and Technology under the promotional reference ”'01MQ07012”'. The authors take the responsibility for the contents.

References

[BMW07] Bundesministerium für Wirtschaft und Technologie BMWi. THESEUS Programme, 2007. http://theseus-programm.de/.

[LCF08] Melissa Lemos, Marco A. Casanova, and Antonio L. Furtado. Process pipeline scheduling. *J. Syst. Softw.*, 81(3):307–327, 2008.

[OAS07] OASIS. Web Services Business Process Execution Language 2.0 (WS-BPEL), 2007. http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsbpel.

[PVHL09] Steffen Preissler, Hannes Voigt, Dirk Habich, and Wolfgang Lehner. Stream-based Web Service Invocation. In *BTW*, 2009.

[W3C02] World Wide Web Consortium W3C. Web Service specifications, 2002. http://www.w3.org/2002/ws/.