User Centric Complex Event Processing Based On Service Oriented Architectures

Feng Gao
Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland
feng.gao@deri.org

Sami Bhiri
Digital Enterprise Research Institute
National University of Ireland, Galway
Galway, Ireland
sami.bhiri@deri.org

ABSTRACT

Current Complex Event Processing (CEP) systems require considerable technical efforts for event pattern definition and event stream implementation. In this paper, we present a service-oriented framework that provides a user-centric way to define complex event patterns and implement the patterns automatically. We extend the Business Event Modeling Notations (BEMN) to allow the business users to describe their complex events with graphical notations. Member event specifications in such event patterns refer to services that deliver them. We then transform the event pattern into a stream query, subscribe to relevant services and obtain event streams required. Finally, we evaluate the query over primitive event streams to obtain results as complex events.

Keywords

BEMN, complex event processing, service oriented architecture, user-centric, stream reasoning

1. INTRODUCTION

Complex event processing (CEP) is a technique used to detect complex events in (near) real time from a set of other events called their member events. CEP systems are usually equipped with rule-based languages and engines to define and evaluate complex event patterns. To implement CEP tasks, it usually requires some programming efforts to leverage APIs provided by CEP engines and create event objects, then, with the engine’s streaming APIs, a sequence of such event objects are streamed to a rule engine, where event patterns/rules defined by a user are evaluated. Unfortunately, neither defining textual event patterns nor programming event streams are trivial for business users. As such, companies need significant technical efforts to implement a business process that requires CEP. Moreover, since event streams are hard coded in such CEP systems, they are not flexible enough to adapt to the changes of event sources.

In this paper, we propose a service-oriented framework that allows an user-centric complex event definition using graphical notations extended from Business Event Modeling Notations (BEMN) [1] and an automatic implementation of event streams on demand. Then we transform graphical event patterns into stream queries and evaluate them over the event streams created using a Data Stream Management System (DSMS). Web services serve as distributed data providers which deliver messages representing member events. We will demonstrate how SOA can be used to create event streams in a flexible and efficient way for complex event detection. The remainder of this paper is organized as follows. Section 2 briefly discusses our extensions to BEMN language. Section 3 presents the prototype system and evaluation results before we conclude in Section 4.

2. EXTENSION FOR BUSINESS EVENT MODELING NOTATIONS

The Business Event Modeling Notation (BEMN) [1] intends to provide a graphical representation for event compositions beyond conventional textual language. BEMN diagram can be integrated into BPMN process models seamlessly to facilitate complex event description in business processes.

Despite that the formal semantics of the language are defined and the execution environment is described, BEMN language did not take into account how existing data stream processing techniques can be utilized to evaluate event rules. As a result, only simplified core composition models are directly executable and the overhead of translating non-core models into core models are introduced.

Furthermore, the execution environment described in BEMN assumed that all event rules are evaluated on a single event channel, which will expose all the member events to all the subscription scopes (process engine, process instances or activities). This will have impact on the efficiency of the evaluation of event rules since it requires filtering out the primitive events from irrelevant event sources while checking conditions for composition models.

Moreover, BEMN language does not specify how an event declaration is structured, what data does an event contain and how such data are used in filters. This part is left to the programmers who program the event objects and create the event streams. Like BPMN in its early stage (1.0 version), the event pattern/process model defined using the language is only executable with the aid from technicians and little automation support is provided. Also, it is uneasy for the users to create optimal event composition models without knowing what the member events really means. Finally, BEMN does not support aggregations other than repetition.
To overcome the limitations of BEMN, we revise the language and propose extensions/modifications to its syntax and constraints to define complex event patterns. We call our extended language BEMN++. The revised language allows for an event composition model to be transformed automatically into a stream reasoning query using a Program-Structure-Tree based algorithm. Then, a stream reasoning engine will evaluate the query to execute the event composition model.

3. PROTOTYPE EVALUATION

In this section we first introduce the architecture of our proposed system by presenting the overview of the framework. Then we demonstrate the feasibility of the system with some prototypical experiments and discuss the limitations.

3.1 Overview of System Architecture

An architectural design of our system is depicted in Figure 1. First, we wrap primitive event sources as WSN services to provide notification messages. These services will register their descriptions to the event knowledge base, where background knowledge are also stored. Then, during the process of specifying complex event patterns, users will perform event service discovery tasks based on the event knowledge base using a discovery engine. Completed event pattern will be validated and forward to the pattern transformer. The transformer will produce subscription plans for the subscription manager as well as stream query for the stream reasoning engine. The subscription manager will establish subscriptions according to the subscription plan, and the sensor services will send notifications to the semantic stream adapter. Lifting schema mappings (as in SA-WSDL1) are used by the adapter to transform SOAP messages into RDF triple streams. Finally the stream reasoning engine will evaluate the query over these triple streams to obtain results as complex events.

3.2 Experiment Setup and Simulation Results

The prototype is built and tested on a 2.53GHz duo core cpu with 4 GB 1067MHz DDR3 memory. The programming language is Java and 512 MB memory is allocated to the JVM. The WSN service providers and consumers are implemented with Apache MUSE2 toolkits. In our experiments we execute a single query with a BGP mapping and a filter over multiple notification service providers and observe the processing delay. The test results are shown in Figure 2. From the experiment we can see that increasing the throughput of the system will result in the increment of both transport and message parsing delay as well as stream reasoning delay.

The main limitation of the prototype is message wrapping and parsing time introduced by SOAP based web services. On our experiment platform, a single message consumer/producer thread has the capacity of receiving/sending roughly 80/100 soap messages with a payload size less than 1 KB in a second. For higher throughput we must use multiple threads, which will consume more computational resources. A potential solution to this could be using REST services or XMPP based services.

4. CONCLUSIONS AND FUTURE WORK

In this paper we present a service-oriented framework to allow user-centric definition and implementation of complex event patterns. The user-centricity of event pattern definition is achieved by extending BEMN. The language is changed mainly in 2 aspects: (1) making the execution environment adaptive to publish/subscribe systems and (2) refining the abstract syntax for some language constructs including event declarations, filters and etc to provide more detailed information on event rules expressed in event composition models. We also improve the user-centricity of event pattern implementation by automatizing the transformation from event patterns into stream queries. In this way the event composition models can be evaluated immediately without further coding.

As a future work, we intend to develop a top-level event ontology to describe the properties and constraints for not only primitive events but also general events as member events, thus a hierarchy of events can be modeled. With semantic descriptions on event hierarch we can allow users to perform knowledge based discovery of member events.

Acknowledgments

This work is supported by the Science Foundation Ireland under Grant No. SFI/08/CE/I1380 (Lion-2).

References