Performance Measurement for CEP Systems

Alexander Wahl and Bernhard Hollunder
Department of Computer Science
Furtwangen University of Applied Science
Robert-Gerwing-Platz 1, D-78120 Furtwangen, Germany
alexander.wahl@hs-furtwangen.de, bernhard.hollunder@hs-furtwangen.de

***Abstract*—Today, Complex Event Processing (CEP) is often used in combination with service-oriented architectures. Several CEP products from different vendors are available, each of them with its own characteristics and behaviors. In this paper we introduce a concept that is able to compare different CEP products in an automated manner. We achieve that by using web service technology. We show how to build a testing environment that includes i) an event emitting component with stable interface, ii) an interchangeable CEP component based on this interface and iii) a measurement and evaluation component. The presented concept is capable to perform different test scenarios fully automated. In this paper three exemplary tests are performed on three selected CEP products.**

***Keywords*-complex event processing; web services; testing architecture; performance testing.**

I. INTRODUCTION

Processing of events is a prevalent necessity in todays information systems. Collected events of any kind are detected automatically and systematically analyzed, combined, rated and processed using Complex Event Processing (CEP) [1] systems. The application of CEP in information technology ranges from sensor networks to operational applications, like business activity monitoring (BAM), algorithmic trading systems and service oriented architectures (SOA).

But, what is CEP? According to the Event Processing Glossary, CEP is "Computing that performs operations on complex events, including reading, creating, transforming, abstracting, or discarding them" [2]. In more detail, the two main components of a CEP system are a set of CEP rules and a CEP engine. A CEP rule is a prescribed method for processing events that reflects a certain condition. An event thereby is a kind of indicator to something that happened, like e.g., a financial trade, a key stroke or a method call within an application. By correlations of these events a CEP rule evaluates if a certain condition is satisfied. If a condition is satisfied, the rule fires.

One substantial feature of CEP systems is the realtime processing of events. And in general, the faster the processing of events is performed by the system the better. A result that is detected more rapidly may be a valuable benefit. Today, numerous CEP implementations of different vendors are available. They are widely varying in the supported platforms, the scope of operations, the performance of processing, rule languages, etc.

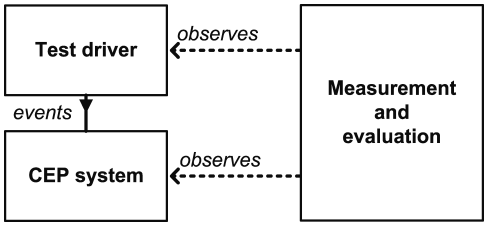

Figure 1. Main components of the concept

In this work, we will introduce a concept to measure the performance of different CEP systems in an automated manner. We thereby focus on i) interchangeability of CEP engines, ii) reproducibility and comparability of the test scenarios, and iii) the automation of measurement without user interaction. Our solution consists of three components, which are shown in Figure 1: i) a test driver component that generates reproducible sets of events, ii) the encapsulated, interchangeable CEP system component, and iii) a measurement evaluation component to calculate measurement results. Based on this solution, three exemplary test kinds are performed: latency test, pollution test and load test.

This paper is structured as follows: After a brief introduction in this section, Section II gives an overview on related work. In Section III, the requirements of the concept will be defined, followed by the description of the concept itself in Section IV. Test candidates and test scenarios are described in Section V. A detailed description of the test setup is provided by Section VI. The test results are shown in Section VII. In the final section, we will conclude the paper.

II. RELATED WORK

In the context of event processing systems, there are some frequently stressed benchmarks: Linear Road benchmark [3], NEXMark [4], BiCEP [5] and SPECjms2007 [6]. But, there is no general accepted benchmark for CEP systems.

White et al. [7] described in their work a performance study for the WebLogic event server (now Oracle CEP). In that work, solely latency testing is performed on a single CEP system.

 ISBN: 978-1-61208-215-8

Kounev and Sach [6] provide an overview of techniques for benchmarking and performance modeling of event-based systems. They introduce a benchmark, SPECjms2007, to measure the performance of message-oriented middleware.

In 2007, Bizarro introduced BiCEP [5], a project to benchmark CEP systems. His main goal was to identify core CEP requirements and to develop a set of synthetic benchmarks. In the following years Mendes, Bizarro and Marques built FINCoS, a framework that provides a flexible and neutral approach for testing CEP systems [8]. FINCoS introduces particular adapters to achieve a neutral event representation for various CEP systems. In this work, we wrap the whole CEP system using Web service (WS) technology. In FINCoS, a controller is defined acting as an interface between the framework and the user. Our concept proposes to perform tests automatically and without user interaction.

In a further publication, Mendes et al. [9] use their framework to perform different performance tests on three CEP engines - Esper and two developer versions of not further specified commercial products. They run "micro-benchmarks" while they vary query parameters like window size, windows expiration type, predicate selectivity, and data values. So, they focus on the impact of variations of CEP rules. Beside they perform some kind of load test. The results thereby showed a similar behavior in memory consumption like Esper did in our tests. In summary, Mendes et al. did run performance tests, but their focus is different to ours.

In 2010, Fraunhofer IAO published a comprehensive survey of event processing tools available in the market [10]. It provides extensive information on supported platforms, licensing and features. This study lists several event processing tools, but performance considerations are out of scope.

III. Requirements on the concept

Comparing sets of implementations from different vendors are common tasks. Before a test scenario may be defined it is essential to specify the requirements towards it.

A first requirement is to ensure, that equal sets of events are emitted on every repeted test execution. Ideally, the component that creates events stays the same in all scenarios in order to eliminate any influence on the measurement itself. Next, test scenarios and measurements must be reproducible.

The aim is to create test scenarios that differ solely in the CEP engine used. The solution therefore requires to support the exchange of the CEP engine. However, some CEP engines are restricted to certain platforms. Therefore, it is desirable to keep event generating component and the CEP component separated (following the principle of loose coupling). At best the two components are able to run on different nodes.

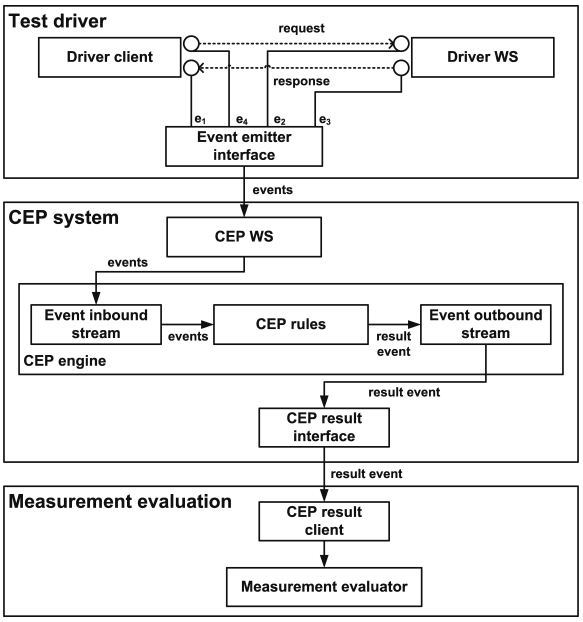

Figure 2. Detailed overview on the components of the concept

The concept requires to perform different test cases. For this work it has to run three exemplary tests, namely latency tests, pollution tests and load tests.

A major requirement is that the test cases need to run automated without user interaction, e.g., by batch script or cron job.

Next, the obtained measurements of the tests need to be analyzed and meaningful statistics have to be generated and presented to the user. In order to ensure comparability of results and statistics the corresponding component is not to be modified at all.

IV. Concept

The requirements suggest the usage of three main components (see Figure 1). One is an interchangeable component wrapping the CEP system. The two others are the test driver component and the measurement evaluation component. Between test driver component and CEP system component, as well as between CEP system component and measurement evaluation component, stable interfaces are introduced. Figure 2 shows the components in more detail.

A. Test driver component

In Figure 2, the uppermost component is the test driver. The responsibility of this component is to produce and emit events. This component reflects the application scenario. We use a WS and a corresponding WS client to reflect a SOA environment as application scenario. In Figure 2, WS and WS client are represented by Driver WS and Driver client. The communication between Driver WS and Driver client follows a request-response pattern based on messages (further named test driver messages) using the Simple Object

 ISBN: 978-1-61208-215-8

Access Protocol (SOAP) [11]. The event emitter, in brief, is linked to the communication path of driver WS and client. It analyzes the interchanged SOAP messages and emits events based on the gathered information. The event emitter itself is implemented as a WS client - as we will see in Section IV-B.

A requirement of the event emitter component is to generate reproducible sets of events. A set of events thereby has two aspects. The first aspect concerns the chronological order, which means that sequence of events and the timespans between these events stay exactly the same at any time a specific set of events is applied. The second aspect is that, using different types of events, the sequence of emitted event types also stays the same.

As we mentioned before, the test driver component generates sets of events. In our application scenario four events, e_1, e_2, e_3 and e_4, are emitted each time test driver messages are exchanged. Suppose that these four events form a set. How does the CEP engine determine which events relate to each other? We, therefore, introduce an ID for each exchange of test driver messages - and thereby also for each set of events. This ID becomes a parameter of the emitted events. With that the CEP engine can determine which events relate to a certain set and afterwards processes these events.

B. CEP engine component

The CEP engine component is formed by four main parts: a CEP engine, CEP rules, CEP WS and CEP result interface. The CEP engine processes incoming events based on CEP rules. The CEP rules and the CEP engine are the core of the CEP system component. CEP rules reflect conditions that are to be satisfied by the system, e.g., the occurence of subsequent events within a certain timespan. Each CEP rule has at least one event inboud stream and fires a result event via an output stream once the condition is satisfied. For the result event a complex event is typically used, which means that it is a more abstract event. A complex event thereby contains a set of incoming events. In our case, a complex event $e_5(e_1, e_2, e_3, e_4)$ is generated.

But, what if more than one driver message exchange is performed? Or in other words: What if more than one set of event is emitted? How can the CEP engine correlate the events of a set? Therefore, the ID parameters of the events are needed. The CEP system is advised to correlates only event with the same ID.

The evaluation of CEP formulas essentially requires events to be passed to the CEP engines input stream. This input stream is exposed to the event emitter component. In more detail, CEP WS defines a unique interface for all CEP systems that accepts events of any structure and routes it to the event inbound stream. The second interface of the CEP system deals with the result events of the CEP system. Next to the event outbound stream is the CEP result interface. A component that is interested in the result events can subscribe itself here. The mechanism behind follows the observer pattern: The CEP result client subscribes at the CEP result interface. Every time a result event occurs, the CEP result client is notified and is provided with the CEP result event. Again, by this mechanism a stable interface, similar to the CEP WS, is created between the CEP system component and the measurement evaluation component. CEP WS and CEP result interface together encapsulate the CEP system and ensure interchangeability.

Interchangeability of the CEP engine component is achieved by defining two stable interfaces: i) Event emitter and CEP WS, and ii) CEP result interface and CEP result client. By the definition of such stable interfaces, and by using WS technology, we achieve the following: i) the CEP component can be designed considering these interfaces, ii) interoperability of the components is achieved by using WS, and iii) no adjustments in code is needed if a CEP component is exchanged. To clarify the latter: Consider the transition from Test driver component to CEP system component. Using different CEP systems each of them can define, for example, its own type of event inbound stream. Therefore adapters between event emitter and event inbound stream are necessary. Once the CEP system is exchanged the event emitter has to use the corresponding adapter, which required to modify the event emitter. By introducing a stable interface the event emitter can use this interface and does not need any changes at all. There is still an adapter needed, but by that means it can be hidden within the CEP system. The adapter then just requires to implement the stable interface. The same applies at the transition between CEP system component and measurement and evaluation component.

C. Measurement evaluation component

The aim of the measurement evaluation component is to perform the calculation necessary to analyze the performance of the individual CEP engine components. Its first component is the CEP result client - a client for the event outbound WS of the CEP engine component. By that client, the component receives the result events that include all the information needed for the performance measurement. Based on these information from the result events, the measurement evaluator determines the performance of a CEP engine component.

D. Automated measurement

As described before, the proposed concept provides three main components. The first component represents the application scenario and acts as the test driver. In our case it consist of an application server including a deployed Driver WS. Automated startup of an application server is a commonly performed task. A typical WS client can be seen as an application, which again can be started easily in an automated manner, e.g., by batch script.

 ISBN: 978-1-61208-215-8

We encapsulated the complete CEP system using WS technology. Thereby, an automated interchange of CEP systems is equivalent to deploying and undeploying the CEP WS of the corresponding CEP system. Again, for an application server this is a commonly performed task and can be automated.

Another option is the usage of several dedicated machines respectively virtual machines (VMs): One for Test driver component and measurement evaluation component, and one per CEP system. Automation of measurement in that case is reduced to start and stop the machines.

V. Test candidates and test scenarios

A decision on the usage of a CEP Engine of a specific vendor rises some interesting questions. For example: If events are emitted to the CEP engine, how long does it take to process the events and to generate a result event? What happens if events that are irrelevant for the CEP rules are emitted? What is the impact on the result calculation time? How does the CEP engine behave on certain event loads? How complex are the rule sets that are necessary to describe a condition?

A. Test candidates

There are several CEP engines available from different vendors. In this work we choose three CEP engines for testing: i) Microsoft SQL Server StreamInsight 1.1 [12], ii) EsperTech Esper 4.3 [13], and iii) JBoss Drools Fusion 5.3.0 [14]. A first difference arises while comparing the rule sets, as we will show in the following paragraphs. We will use an example that fits to the application scenario described before (WS and client). The task is to collect four related events of the driver message exchange. Once all four events are detected a complex event, the CEP result event, is created.

1) StreamInsight: Microsoft StreamInsight is based on the Microsoft SQL Server and has a high-throughput stream processing architecture. It is equipped with several adapters for input and output event streams. LINQ is used as query language to specify the CEP rules. The corresponding CEP rule is as follows:

```
var filtered = from e in inputstream
  group e by e.ProcessId into con
  from item in con.HoppingWindow(
    TimeSpan.FromSeconds(180),
    TimeSpan.FromSeconds(3),
    WindowInputPolicy.ClipToWindow,
    HoppingWindowOutputPolicy.ClipToWindowEnd)
    select new TimeAggregation()
    {
      Id = con.Key,
      Value = (int)item.Count(),
      TimeStart = item.Min(t=>t.Timestamp),
      TimeEnd = item.Max(t=>t.Timestamp),
    };
var filteredFromValue = from e in filtered
                        where e.value>=4
                        select e;
```

2) Esper: EsperTech Esper is an open-source CEP engine available under GNU General Public License (GPL) license. It is available for Java as Esper and for .NET as NEsper. It offers the Event Processing Language (EPL) to specify CEP formulas. The CEP rule is defined by

```
String expression =
  ''insert into CEP.LatencyEvent'' +
  ''select one.timestamp,'' +
         ''two.timestamp,''+
         ''three.timestamp,''+
         ''four.timestamp''+
  ''from CEP.inputStream1.win::time(180) as one,'' +
       ''CEP.inputStream2.win::time(180) as two,'' +
       ''CEP.inputStream3.win::time(180) as three,'' +
       ''CEP.inputStream4.win::time(180) as four'' +
  ''where four.id = two.id and '' +
        ''three.if = one.id'';
statement =
  epService.getEPAdministrator().createEPL(expression);
```

3) Drools: Fusion is a module for JBoss Drools to enable CEP. Like Esper, it is open-source software. It can run Java and .NET and supports a variety of input adapters. The Drools Rule Language (DRL) is used to specify CEP formulas. The CEP rule is

```
rule''cep-formula''
when
  $pay:InternalPayload($ide:identifier,
                       $step:step,
                       step>=4)
       from entry-point StoreOne
  $minTime:Number()
  fromaccumulate(InternalPayload(identifier==$ide,
                                 step==1,
                                 $id:identifier,
                                 $time:timestamp)
                 from entry-point StoreOne,
                                  init(long tm=0;),
                                  action(tm=$time;),
                                  result(tm))
then
  System.out.println(''step is: ''+$ide+
                     '' min ''+$minTime'');
End
```

B. Test scenarios

1) Latency test: The following setup applies to our latency test setup: driver WS and its client both emit two events - one for inbound SOAP message and one for outbound SOAP message. All four messages include the timestamp of their creation, and are passed to the CEP component. Within the CEP engine, the four events are correlated and combined to a complex event, the result event. Once the result event occurs at the CEP result client, the measurement evaluator calculates the overall latency and estimates the CEP component latency.

2) Pollution test: In a pollution test setup, the CEP engine is confronted with events that are relevant for a CEP rule and events that are irrelevant. In the pollution test scenario a constant rate of events is emitted to the CEP component. During the test the rate of irrelevant events is increased step by step. The aim of the CEP component is to filter out the relevant events. The observed parameters were changes in latency, CPU load and memory usage.

 ISBN: 978-1-61208-215-8

3) Load test: A load test in brief applies different event rates to the CEP component. The test starts with one driver WS and client pair. During the test the amount of driver WS and client pairs is continuously increased. The observed parameters were latency, CPU load and memory usage.

VI. TEST SETUP

The testing was performed using different virtual machines (VM). As virtualization product Oracle VirtualBox [15] was used. As an alternative several dedicated machines can be used. The VMs were interconnected using a virtual network. The operating systems of VM1 and VM2 was Microsoft Windows 7 with Service Pack 1. VM1 includes the event emitter component and the measurement component. For the implementation of the event emitter component WS technology was used. Communication between event emitter and CEP WS was based on SOAP messages.

The event emitter were implemented as message inspectors. Each time a SOAP message passes the message inspector its content is analyzed and an event is emitted. Each event thereby includes a timestamp and an unique ID, as described before. In our case emitting such an event means that an event is embedded in a SOAP message and is sent to the CEP WS of the CEP system component.

The measurement component is part of VM1, which reduces the number of VMs needed. In consistence with the driver WS and its client the measurement component also uses WS technology. The measurement and evaluation results are provided in database tables.

Within the second VM (VM2) the CEP system component is implemented. Several instances of VM2 exist, one for each tested CEP system. The individual instances of VM2 are completely interchangeable and can potentially even run in parallel - with slight modifications at the event emitter and separate IP addresses for each instance.

The technology used to implement the event inbound WS depends on the CEP engine used. In our work either WCF technology or JAX-WS is used. For interoperability reasons the binding type 'basicHttpBinding' (following the WS-I Basic Profile [16]) was used for all WS.

VII. TEST RESULTS

As described before, we performed three exemplary tests using three different CEP systems. For completeness the results are displayed in this section.

A. Latency test

Comparing the latency all of the three CEP systems showed similar result. As the main source for fluctuation the driver message exchange was identified. The CEP system latency showed no significant differences.

CPU usage of B and C was similar. A, in contrast, has a significantly higher CPU consumption. In terms of memory usage A and B showed a modest behavior compared to C.

Table I
POLLUTION TEST RESULTS

CEP system	Pollution	Latency	CPU	Memory
A	10events	2.5ms	86%	27M
	25events	1.5ms	81%	28.5M
	50events	1.7ms	83%	28.4M
	100events	2.3ms	89%	29.6M
	200events	1.9ms	92%	33.9M
B	10events	0.1ms	24%	85.1M
	25events	1ms	25%	88.5M
	50events	1ms	23%	89.3M
	100events	1ms	24%	97.2M
	200events	3ms	24%	93.4M
C	10events	0.1ms	41%	122M
	25events	1ms	38%	160M
	50events	1ms	50%	164M
	100events	1ms	73%	165M
	200events	1ms	79%	165M

Table II
LOAD TEST RESULTS

CEP system	Threads	Events/sec	CPU	Memory
A	1	8	2%	18.1M
	5	40	12%	18.2M
	10	80	16%	18.1M
	20	128	30%	18.4M
	30	128	20%	18.4M
B	1	8	1%	24M
	5	40	4%	30M
	10	80	7%	33.4M
	20	128	10%	42M
	30	128	10%	43.2M
C	1	8	1%	65M
	5	40	4%	67M
	10	80	5%	113M
	20	128	9%	120M
	30	128	10%	126M

B. Pollution test

The pollution test showed interesting differences between the CEP systems (see Table I). Regarding the latency change A showed an overall pretty constant latency. However, in comparison with B and C the overall latency was significantly higher. At the beginning B and C were almost equal - with slight advantages for C. But at higher pollution rate the latency of B highly increased, in contrast to C with a constant latency.

Concerning the average CPU usage, A showed a slight increase with increasing pollution rate. Its overall CPU usage was higher that with B and C. For B the average CPU usage did not change at any pollution rate. With C an increase in pollution rate resulted in a significant increase of average CPU usage. A had a low memory usage at any pollution

 ISBN: 978-1-61208-215-8

rate. With B memory usage, again, increased with pollution rate. C has a constant memory usage for all pollution rates, but at high level.

C. Load test

Table II shows the results of the load test. We could not identify significant differences to the results of the latency test applying load to the test candidates. We will therefore reinvestigate on load tests in more detail in future work.

For all candidates, the applied load and the CPU usage correlated. The same applies to the memory usage with B and C. For A the memory consumption was almost constant.

VIII. Conclusion

Today, CEP is commonly used and several products from different vendors are available in the market. This paper described a platform to compare different products under controlled and equal conditions. The test thereby were performed in an automated manner, which means that no user interaction was necessary.

We described that a complete CEP system can be encapsulated using WS technology. Thereby interchangeability of CEP systems can be achieved, which further simplifies for automation of performance tests. The paper also reproduced characteristics concerning performance and resource consuption already described elsewehere.

In summary, CEP systems can be encapsulated as a whole by means of WS technology. Thereby, interchangeability of complete CEP systems can be achieved. With interchangeability of CEP systems fully automated performance testing is available.

Acknowledgment

We would like to thank the anonymous reviewers for giving us helpful comments.

This work has been partly supported by the German Ministry of Education and Research (BMBF) under research contract 17N0709.

References

[1] D. Luckham and B. Frasca, “Complex event processing in distributed systems,” Stanford University, USA, Tech. Rep. CSL-TR-98-754, Mar. 1998. [Online]. Available: http://citeseer.nj.nec.com/luckham98complex.html

[2] D. C. Luckham and R. Schulte, “Event Processing Glossary – Version 2.0,” 2011. [Online]. Available: http://www.complexevents.com/2011/08/23/event-processing-glossary-version-2-0/, last accessed: May 13, 2012.

[3] A. Arasu, M. Cherniack, E. Galvez, D. Maier, A. S. Maskey, E. Ryvkina, M. Stonebraker, and R. Tibbetts, “Linear road: a stream data management benchmark,” in *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, ser. VLDB ’04. VLDB Endowment, 2004, pp. 480–491. [Online]. Available: http://dl.acm.org/citation.cfm?id=1316689.1316732, last accessed: May 13, 2012.

[4] J. Li, J. Maier, V. Papadimos, P. Tucker, and K. Tufte, “NEXMark Benchmark.” [Online]. Available: http://datalab.cs.pdx.edu/niagara/NEXMark/, last accessed: May 13, 2012.

[5] P. Bizarro, “BiCEP - Benchmarking Complex Event Processing Systems,” in *Event Processing*, ser. Dagstuhl Seminar Proceedings, Mani Chandy, Opher Etzion, and Rainer von Ammon, Eds. Dagstuhl and Germany: Internationales Begegnungs- und Forschungszentrum fuer Informatik (IBFI), Schloss Dagstuhl, Germany, 2007. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2007/1143, last accessed: May 13, 2012.

[6] S. Kounev and K. Sachs, “Benchmarking and Performance Modeling of Event-Based Systems Modellierung und Bewertung von Ereignis-basierten Systemen: It - Information Technology,” *it - Information Technology*, vol. 51, no. 5, pp. 262–269, 2009.

[7] S. White, A. Alves, and D. Rorke, “WebLogic event server: a lightweight, modular application server for event processing,” in *Proceedings of the second international conference on Distributed event-based systems*, ser. DEBS ’08. New York and NY and USA: ACM, 2008, pp. 193–200.

[8] M. R. N. Mendes, P. Bizarro, and P. Marques, “A framework for performance evaluation of complex event processing systems,” in *Proceedings of the second international conference on Distributed event-based systems*, ser. DEBS ’08. New York and NY and USA: ACM, 2008, pp. 313–316.

[9] M. Mendes, P. Bizarro, and P. Marques, “A Performance Study of Event Processing Systems,” in *Performance Evaluation and Benchmarking*, ser. Lecture Notes in Computer Science, R. Nambiar and M. Poess, Eds. Springer Berlin / Heidelberg, 2009, vol. 5895, pp. 221–236.

[10] K. Vidackovic, T. Renner, and S. Rex, *Marktuebersicht Real-Time-Monitoring-Software Event-Processing-Tools im Ueberblick*. Stuttgart: Fraunhofer-Verlag, 2010.

[11] W3C, “SOAP Version 1.2,” 2007. [Online]. Available: http://www.w3.org/TR/soap/, last accessed: May 13, 2012.

[12] Microsoft, “Microsoft StreamInsight.” [Online]. Available: http://www.microsoft.com/sqlserver/en/us/solutions-technologies/business-intelligence/complex-event-processing.aspx, last accessed: May 13, 2012.

[13] EsperTech, “Esper - Complex Event Processing.” [Online]. Available: http://esper.codehaus.org/, last accessed: May 13, 2012.

[14] JBoss, “Drools Fusion.” [Online]. Available: http://www.jboss.org/drools/drools-fusion.html, last accessed: May 13, 2012.

[15] Oracle, “VirtualBox.” [Online]. Available: https://www.virtualbox.org/, last accessed: May 13, 2012.

[16] Web Service Interoperability Organization, “Basic Profile Version 1.2.” [Online]. Available: http://www.ws-i.org/Profiles/BasicProfile-1.2.html, last accessed: May 13, 2012.

 ISBN: 978-1-61208-215-8