An Architecture for Semantically Enriched Data Stream Mining

Andreas Textor, Fabian Meyer, Marcus Thoss,
Jan Schaefer, Reinhold Kroeger
Distributed Systems Lab
RheinMain University of Applied Sciences
Unter den Eichen 5, D-65195 Wiesbaden, Germany
{firstname.lastname}@hs-rm.de

Michael Frey
Humboldt-Universität zu Berlin
Department of Computer Science
Rudower Chaussee 25, D-12489 Berlin, Germany
{lastname}@informatik.hu-berlin.de

***Abstract*—Data stream mining (DSM) techniques can be used to extract knowledge from continuous data streams. In this paper, we present an approach providing a modelling and execution architecture for networks of DSM operators. It can be applied in resource-constrained environments, and provides capabilities for semantic enrichment of data streams. This allows processing of streams not only based on information contained in the streams, but also on their semantic contexts. The approach consists of a DSM runtime system, a concept for semantic tagging of stream elements, the integration of semantic information stores, and a domain-specific DSM network description language. A small ambient assisted living scenario is presented as an example application.**

***Keywords*-data stream mining, complex event processing, ontologies, semantic tagging, stream query language, IT management, ambient assisted living.**

I. MOTIVATION

The number of applications where data must be processed in real-time is constantly increasing. Classic data mining approaches are applicable to cases where all data sets are statically accessible in a persistent database, and make use of mathematical and statistical methods to recognize trends, correlations between values, or clustering. If data must be evaluated on the fly, because the amount of data is too high to be stored completely, or because it is inherently continuous, so-called data stream mining (DSM) techniques need to be employed. The goal of data stream mining is the extraction of knowledge from continuous streams of data, e.g., packet streams, monitoring data from sensor networks, streams of log records in applications etc. Usually, knowledge discovery and machine learning approaches are used to achieve this. While data mining usually operates directly on the complete, stored data, DSM tends to make use of supporting application models and data models because of the transient nature of the data to be processed.

In a related field, Complex Event Processing (CEP), streams of distinct simple events are analyzed and events are correlated in order to extract more abstract (complex) events, which may then be inserted into a stream. Any logical or physical change in the observed system is counted as an event, and the events must be processed in a limited time frame, without access to the entirety of data. CEP systems often work using pattern recognition, and abstraction and modeling of events and relationships between events. The processing technology underlying a DSM architecture will often have CEP characteristics as well.

Different approaches for implementations of both DSM and CEP exist, including runtime systems for the extraction and correlation of data and specialized stream query languages. However, few approaches are suitable for the application in resource constrained environments, e.g., home routers with little memory (which becomes relevant in the context of ambient assisted living (AAL) environments). In such environments, a DSM system additionally needs to be highly reconfigurable, as the environment can change quickly and the deployment of new software versions can be non-trivial. On the other hand, AAL installations often provide only a small number of sensors and low frequencies of sensor events. The image that can thus be created from reality tends to be fragmentary, so another motivation for supplementing sensor stream data with additional knowledge is to improve the accuracy of the interpretation of scarce sensor values.

Furthermore, a problem that arises with the application of DSM systems that are integrated into existing application environments is the preservation of semantics of the processed data. Processing needs to be possible depending on semantic information, e.g.: which type of sensor is it, in which room is the sensor located, which are adjacent rooms, used by whom, when, and so on. Equivalent semantic information is necessary in other domains as well.

In this paper, we present an approach for a DSM architecture that satisfies some basic requirements - being able to run in a resource restricted environment, providing facilities for semantic enrichment and processing of data streams that can be expressed in a high-level abstract notation, and dynamic reconfiguration capabilities. The architecture consists of a modular runtime system, which can be adapted to different environments using pluggable input and output connectors, a semantic information store component and a description language for DSM networks.

It is not the goal of the DSM network to replace existing systems, such as the free CEP system Esper [1]. Although

 ISBN: 978-1-61208-242-4

the DSM network can be used stand-alone, it is also possible to combine it with Esper, e.g., for pre-filtering streams. For certain applications (such as processing events in IT management systems), the rate of data in a stream may be much higher than in others (such as processing sensor data in AAL systems), which can make this a viable option.

The paper is structured as follows: Section II describes existing approaches for DSM and CEP systems with regards to the aforementioned requirements, and Section III gives an overview of the proposed architecture, including the structure (III-A), dynamic scripting capabilities (III-B), the semantic information store (III-C), DSM network operators (III-D) and the DSM network description language (III-E). In Section IV, details about the prototypical implementation are given and in Section V an exemplary use case is described. The paper closes with a summary in Section VI.

II. Related Work

An early, popular and widely cited DSM framework was created by the Aurora [2] project. It was designed as a single-site solution focusing on centralised high performance stream processing. Subsequent projects extended the Aurora architecture for distributed processing, notably Medusa [3], also providing a hardware platform for the distributed nodes, and Borealis [4], which resulted from a joint effort of the Aurora and Medusa teams, aiming at a better integration of modelling and distribution aspects. These architectures employ query languages that are syntactically and idiomatically derived from SQL and the underlying database query paradigm, like CQL [5], the query language of STREAM [6], another early single-site stream processing engine. In PIPES [7], a processing network similar to our approach is used, which is expressed formally using a generic operator algebra. From an application perspective, PIPES also maps CQL-related query expressions onto operator networks, focusing on query optimisation.

Since the major academic DSM projects closed down years ago after funding had ceased or key members left the projects, research results have been picked up by larger commercial data mining systems from companies like IBM, Oracle and Tibco. Research architectures were commercialised, notably Aurora, which evolved into the StreamBase CEP product [8] and PIPES, which was branded RTM Analyzer [9] and is now part of the Software AG portfolio. The remaining most prominent open and freely available stream processing framework, Esper [1], is a commercial product as well. StreamBase CEP, RTM Analyzer and Esper queries adhere to the SQL paradigm. Regarding the categories established in [10], they would be classified as using "Data Stream Query Languages", whereas the internal PIPES representation and our approach use "Composition-Operator-Based Event Query Languages".

A more generalised view on event processing and applications relating to data stream mining problems has been stated by a group of IBM researchers in [11]. Especially Processing can be linked to repositories possibly providing contextual knowledge, and events can be "enriched" with additional information, a mechanism we refer to as "tagging". Still, the main goal presented is to detect the occurrence of complex events for immediate reaction, while we strive to interrelate event stream information with long-term semantic model information.

In [12], semantic enrichment of events and subsequent complex event processing using semantic rules and ontologies is described. An architectural vision is given that resembles our concept of mapping events to elements of an ontology through semantic tagging and rule processing. The resulting complex events are not fed back to the ontology, though, but meant to be delivered to the user.

Concerning application orientation related to our work, the DigiHome [13] platform must be mentioned as a CEP-based control architecture targeting AAL environments. It does not rely on an explicit semantic model, but it integrates a large number of real-world devices and abstractions from smart living environments, managing home automation tasks through event stream processing within a service-oriented architecture. The DogOnt [14] modeling language for Intelligent Domotic Environments provides an ontology which allows to formalize all the aspects of the environment and a rule language which can automatically generate proper states and functionalities for domotic devices.

III. Architecture

The architecture presented here adds modelling and scripting facilities to a set of core components and operators, all of which are discussed briefly in the following sections.

A. Data Stream Mining Network

At the heart of our architecture, shown in Figure 1, data stream processing takes place by feeding data into a processing network, which analyses and manipulates stream contents and provides analysis results in the form of output streams or appropriate signalling of results to a monitoring, reporting or result storage environment. The basic building blocks of a DSM network, and thus also the main elements of the framework presented here, are operators manipulating data streams and queues transporting data between the operators.

Our queues are instances of unidirectional FIFO transport lanes for data stream elements ("packets") with basically unlimited storage capacity and one dedicated end for input and output of packets, respectively. Operators realise a given data stream processing functionality ("operator type") that also defines the number of dedicated input and output connection points for queues. Access to the queues is operator-initiated, that is, operators actively push output data into queues and pull input data from queues linked to them. A

 ISBN: 978-1-61208-242-4

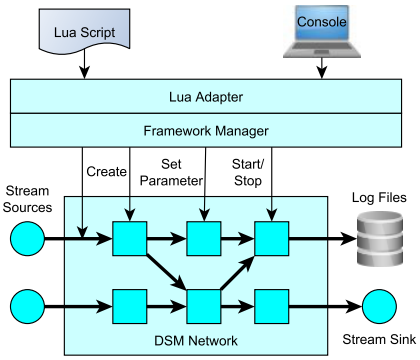

Figure 1. Core DSM Architecture

queue endpoint may be connected to one operator connection point at most.

This boilerplate definition of an abstract operator is essential for the specification of processing networks, since, differently from query-based specification languages like CQL, the operator/queue structure of the network remains exposed up to the specification level. Especially the consistent view of data as streams by all operator types is different from the query approach, which requires the embedding of conversion operators from streams to relational views and back in order to apply SQL-like queries. The format of messages sent through the queues is a compact binary encoded representation of tuples of free-form name-value pairs whose structure and semantics, except for a timestamp value, are defined at application level.

For the creation of a DSM network, operators and queues are instantiated separately one at a time, and then linked as needed such that the resulting data flows and the processing through the operators realise the overall DSM application. Creation and linkage of the network elements is achieved through utilising factory and control functions of a central control entity, the “framework manager”.

Operators are active components, i.e., there is at least one thread of control per operator. Currently, there is no additional control over scheduling parameters to adapt resources to asymmetric distributions of processing load among operators, only the default OS scheduling model is used with native threads. For load shedding, there is no central component like in the Borealis architecture, instead, load shedding operators are used that must be explicitly inserted into mining network paths, similar to the concept introduced by CQL.

B. Dynamic Scripting

For some DSM applications, static configurations of operator networks are sufficient. For many problems though, changes in the queries to be conducted can be expected, i.e., in fault diagnosis scenarios, or changing numbers and content of input data streams, all of which leads to change requests targeting DSM network or operator configuration. Dynamic reconfiguration of both operators and network layout are provided by a scripting interface. It is realised as a thin layer that wraps the framework manager, operator and queue interfaces.

For the scripting language, Lua [15] was chosen, because it is considered a mature, sufficiently popular language with a compact native implementation on all mainstream platforms. Lua is object-oriented, which maps well onto the object-oriented design of the DSM framework implementation. Thus, most operations available at the C++ level can be used in the Lua scripting environment.

Technically, the functionality of the internal mining framework manager is thus externalized to an interactive console, which uses a Lua interpreter to execute scripting commands. The manipulation of network elements is expressed as a Lua statement calling methods of Lua objects immediately delegating the method call to corresponding C++ method implementations. Besides being able to alter the network dynamically and interactively using the console interpreter, the same approach can be used to execute prepared Lua script fragments. Through a script, a complete network configuration can be set up, with the added benefit of having a powerful scripting language to further express external dependencies, network variants or to enhance the compactness, expressiveness, and readability of the configuring actions by using loops and other control structures of the Lua language.

C. Semantic Information Store

The data streams in the mining framework contain information that can vary, depending on the originating source of the data (i.e., different textual or binary representations). Therefore, processing of the streams depending on their semantic context is only possible if this context is provided from the outside. For this task, the semantic information store provides access to semantic information in the form of OWL (Web Ontology Language) ontologies. The store is a separate component that is intended to run on a node with more computational resources, as the ontologies may need more memory than what is available on the data source or data stream processing nodes.

The ontologies in the store consist of two parts: A domain ontology, and, where necessary, automatically updated data values extracted by the DSM network. Domain ontologies explicitly model those aspects of the application subject to the DSM process which are not contained in the processed data streams. This allows the semantic enrichment of streams with either actual context information or with references to context information. For cases in which subsequent processing steps depend on context information, corresponding values can be retrieved from the information store and inserted into the data stream. When the original source of a datum is expected to be removed in further processing steps, then a reference to an ontology entity that describes the data source or the original data type of the date, can suffice.

 ISBN: 978-1-61208-242-4

Likewise, current values can be set as state information in the semantic data store in order to preserve them for logging, aggregation or tracking purposes before they are processed in the DSM network and do not remain in the data stream.

D. Mining Network Operators

The approach of interconnected operators is the central concept of the mining framework architecture. Hence, the user needs to be offered a set of operators which can be configured as an interacting network. Most of the operators presented here are already implemented and integrated into the mining framework. There are four classes in which operators can be divided: Analysis, Structural, Knowledge and System operators.

Analysis operators offer functionality for arithmetic and statistical evaluation of numerical values of bypassing data stream elements. They produce new data streams or enrich existing data streams by new elements that contain the result of the analysis process. The operators can either be configured to analyse elements in sliding time windows or sliding element windows. Supported operators include arithmetics and standard deviation, element counter and average for statistics. Further operators like minimum, maximum etc. could be added easily.

Structural operators do not change the state of data stream elements but affect their routing in the network. Mostly, they observe the attributes of bypassing elements and perform a configured routing action. They can decide to either forward or drop elements based on configured filtering conditions (`filter`), or to split incoming data stream elements to the operator's set of output streams using configurable classification characteristics (`split`). Corresponding to the split operator, there is an operator which handles a set of input streams and merges incoming elements to a single output stream (`join`) and the extension of a single stream with additional elements (`ejoin`).

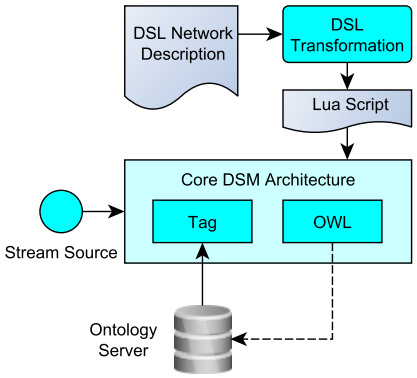

Figure 2. Overall DSM Architecture

Knowledge operators work on semantically enriched data streams. In particular, operations used in different knowledge operations are applied to a stream. This includes the application of OCL (`ocl`) and SWRL rules (`swrl`) to a semantically enriched data stream, like shown in Figure 2.

In order to add semantic tags to a stream in the first place, the tag operator (`tag`) utilises the semantic information store described in Section III-C as an external knowledge base. A SPARQL query is configured for the operator which is executed on the knowledge base for every bypassing element. The query may contain wildcards which are bound to the elements attribute state and returns a result set of key-value-pairs which are added to the element, the so-called "semantic tags". These tags contain information which is normally not present in the data stream but can only be retrieved from the knowledge base. Semantically enriched streams are streams where elements are mapped one-to-one to elements of a knowledge representation. Correspondingly, the untag operator (`untag`) removes semantic enrichments from streams again.

An example for a more complex operator is the decision tree operator (`dtree`). As opposed to the tag operator, it does not have an external knowledge base which already contains information, but builds a classifier using the knowledge contained in the bypassing stream elements. The algorithm used for the implementation is the VFDT (Very Fast Decision Tree learner) described in [16].

System operators offer functionality for loading streams into the DSM system (`in`), output of streams into files (`out`) and on screen (`print`) and logging (`log`). The different possible sources of streams are specified using internationalized resource identifiers (IRI).

E. Mining Network Description Language

The mining network description language (mDSL) is a metamodel-based domain specific language (DSL) for the definition of queries for the DSM system. With it, a domain expert specifies how a stream is structured and what type of sources and sinks for streams exist, what type of operators are executed on a stream, and how they are interconnected. In particular, mappings of model elements of knowledge representations to elements of a stream as well as operations on semantically enriched streams are defined. At the mDSL specification level, streams and operations on a stream are always typed.

Unlike in the DSM network realisation, elements of a stream in mDSL have a unique name and a data type. Streams have unique names as well and consist of one or more elements which are specified within a data structure. At present, integer, floating point numbers, booleans and character strings are supported. In addition, variables can be specified which consist of a unique name, data type and value; they are used within operators. Operators of the mDSL, covering all operator types introduced in the previous section, can be interpreted as functions on streams. Code written in the mDSL can be modularized into packages and re-used through a generic resource import mechanism

 ISBN: 978-1-61208-242-4

that can also be used for loading files provided by different knowledge representations.

mDSL is based on a metamodelling approach. Concepts of the mDSL refer to UML classes and relationships among them. As a concrete syntax of the mDSL, a textual representation was chosen. Semantics are defined using annotations in the mDSL metamodel and in the model-to-text (M2T) transformation.

When application and system design are defined through metamodels, model transformations can be established that generate executable applications from formal software models, thus supporting a model-driven software development (MDSD) process. Specifically, M2T transformations create a textual representation of the input model. Figure 2 depicts the transformation process applied to mDSL -based models. Descriptions of the mining network written in the mDSL are transformed to Lua using an M2T transformation.

Using a metamodel-based approach also facilitated the integration of knowledge representations. Here, the ontology definition metamodel (ODM) [17], a metamodel-based representation of OWL specified by the OMG was used. Using ODM, our concept aims at linking model-driven software development with the use of ontologies. Shared and disjoint concepts among OWL and UML are identified and the knowledge representations are integrated by references in classes and relationships among classes in the mDSL metamodel. Thus, an mDSL-based operator class for mapping elements of an OWL ontology to elements of a stream can reference an OWL class represented as an UML class in the ODM metamodel. This metamodelling approach is very flexible since it allows to add further metamodel-based knowledge representations to the language. A limiting factor are the methods and concepts of a knowledge representation, since they are shared among all knowledge representations covered by the language.

IV. Implementation

For the architecture presented, a prototypical implementation has been created, and some of its aspects are highlighted in the following sections.

A. Data Stream Mining Network

The framework with all of its core components and operators is natively implemented using C++ and the Boost [18] libraries. For operators, a base class provides both the basic implementation of thread handling and queue connectors as a basis for new, concrete operator implementations. Every operator is implemented within a shared library of its own, such that it can be dynamically loaded and the operator instances can be created at run-time.

The operator thread implementation basically relies on operating system threads, which are encapsulated by the `thread` abstraction of Boost libraries. Of course, the concurrency introduced by active operators incurs race conditions during queue access. Valid access semantics are guaranteed by a combination of measures: First, at any given point in time, a message is owned by exactly one queue or operator, which avoids concurrency issues for access to message contents. A lock-free queue implementation is used in combination with memory barriers to ensure both correctness and minimal locking.

The encoding of message contents is handled by a BSON [19] implementation taken from the MongoDB [20] project. BSON provides a compact binary encoding of structured data. It also offers dynamic protocol buffer handling based on Boost memory object abstractions like smart pointers, which ensures that message memory is freed when a message is no longer referenced by an operator or by a queue.

The native implementation and the compact message format were chosen with resource-constrained environments in mind. The virtual image size of an engine running a network of 30 operators is around 50 MiB, and the message size is almost devoid of overhead beyond the binary representation of field names and values. For the deployment of DSM networks on small appliances like those found in home automation environments, minimisation of the overhead of both memory footprint and computing is essential because in living environments the limits of acceptance for energy consumption and noise and heat emission of computing devices are critical. Performance optimisations concerning throughput and latency have not been carried out aggressively yet, the same applies to memory pooling strategies for messages of similar size. The processing rate for a simple, linear three-operator setup with a numeric threshold filtering operator is in the range of 200k messages per second on a 2 GHz dual core PC system, which is still half a magnitude below the Java-based Esper engine.

B. Knowledge Operators

The assembly for most operators that have been presented in Section III-D is rather simple, but especially the knowledge operators have a complex structure for the integration of external knowledge bases. Hence, the tag operator is examined in this section.

The external knowledge base used by the tag operator is an ontology offered by the semantic information store presented in Section III-C. Since the exported interface of the store is an OSGi service, the interprocess communication proves difficult. Therefore, the semantic information store was extended by a socket-based communication channel that takes a SPARQL query string as input, performs the query on the knowledge base, and writes the resulting tuples of IRIs to the socket. The query is constructed using a template, containing wildcards for stream element specific data, which is configured for each tag operator instance. When an element is taken from the operator's input stream, its attribute values are bound to the wildcards and the query is sent to the semantic information store. The resulting data

 ISBN: 978-1-61208-242-4

tuples are added to the elements attributes and the element is put into the operators output stream, so that subsequent operators can use the semantically enriched data for further knowledge-based operations.

V. APPLICATION

As described earlier, one of the major motivations for the DSM is the application in AAL environments. In this section, it is shown how our DSM architecture can be applied to a small but representative example application from an AAL setting.

The scenario presented here consists of two light sensors, one outside and one inside a house, a sensor to detect a person in the house and an actor to control window blinds. Furthermore, the network contains a decision tree operator to classify sensor values. The goal of the given scenario is to decide, on the basis of the given sensor values, whether or not to open or close the window blinds, and then perform the particular action. In a simple example such as this, a set of rules or a purpose-built operator could be easily employed to make a decision. In more realistic situations, though, more sensors and additional information would have to be taken into account (e.g., more persons, a notion of time, whether the TV is turned on etc.).

```
package example {
  stream sensors {
    int id;
    float value;
  }
  stream actors { int action; }
  stream queryresult { bool return_value; }
  sensors indoors, outside, person, aggregated;
  actors blinds;
  queryresult result;

  indoors = in("http://wwwvs.cs.hs-rm.de/dsm/sensors/
    indoors/#1", "localhost", 9597);
  outside = in("http://wwwvs.cs.hs-rm.de/dsm/sensors/
    outside/#1", "localhost", 9595);
  person = in("http://wwwvs.cs.hs-rm.de/dsm/sensors/
    person/#1", "localhost", 9596);
  aggregated = join(indoors, outside, person);

  result = sparql({SELECT ?"blinds"  WHERE { ?"room"
    <http://wwwvs.cs.hs-rm.de/dsm/contains> ?"blinds" .
    ?"room" <http://wwwvs.cs.hs-rm.de/dsm/contains>
    "indoorslightsensor"}}, result.return_value,
  aggregated);

  blinds = dtree("PERSON" {"IN", "OUT"}, "LIGHT_OUTSIDE"
    {"YES", "NO"}, "LIGHT_INDOORS" {"YES", "NO"}, 0.0001,
    0.025, 0.98, 450, "BLINDS" {"YES", "NO"}, result);
}
```

Listing 1. Example application

Each of the sensors provides a separate data stream in the DSM. The streams from the three sensors are merged into one stream, which in turn is passed through an instance of the tag ("sparql") operator that enriches the stream with the information about which blinds should be controlled. This is information that is not present in the stream - depending on the room, in which the light sensor is situated, different window blinds could be selected. The enriched stream is then passed to the decision tree operator, which decides to open or close the window blinds, and finally to the actuator.

Listing 1 shows the DSM network description language code for the scenario described above.

VI. SUMMARY AND FUTURE WORK

In this paper, we presented an approach for a data stream mining architecture that takes into account the requirements for being applied in resource-constrained environments and for providing facilities for semantic enrichment and processing of data streams. The approach includes a model for DSM networks, a modular runtime system, a semantic information store component and a description language for DSM networks.

The DSM network is application agnostic, as the framework and the operators operate on data streams which are not further specified, i.e., contain untyped information, except for the internal typing necessary to encode binary data representations. This design decision allows the construction of highly specialised and thus resource saving operators. Nevertheless, a set of standard operators was described and implemented, covering both structural operators such as join and merge, numerical operators such as sum and average and classification operators such as the VFDT operator.

Using the semantic information store, stream data can be enriched with semantic information. The tag operator can be used in the DSM network in places where additional information from the knowledge base external to the actual DSM elements is required for further processing. Semantic information specific to the data source or data stream can be added to the stream, such that subsequent operators (e.g., classification operators) can incorporate the additional information.

The DSM description language is a metamodel-based domain specific language that can be used to specify a network. Programs in the language are compiled into a sequence of commands to set up the DSM network incrementally. While, for performance reasons, the elements of data streams in the runtime system are not explicitly typed, the specification language does support type information about the streams and can use it to ensure consistency of the network.

Next steps in the project include further performance improvements and the implementation of more DSM network operators: While the description language already supports the specification of operators for the query languages OCL and SWRL, this has yet to be implemented in the runtime system. An operator that feeds back semantic tag information from result streams to an ontology will be realised to complete the semantic transfer cycle. It is also planned to apply the approach to a scenario dealing with a larger AAL infrastructure.

 ISBN: 978-1-61208-242-4

REFERENCES

[1] EsperTech, "Esper Complex Event Processing System," http://esper.codehaus.org/. Last access 2012-07-10.

[2] D. Carney, U. Çetintemel, M. Cherniack, C. Convey, S. Lee, G. Seidman, M. Stonebraker, N. Tatbul, and S. Zdonik, "Monitoring Streams: A New Class of Data Management Applications," in *Proceedings of the 28th international conference on Very Large Data Bases*, ser. VLDB '02. VLDB Endowment, 2002, pp. 215–226. [Online]. Available: http://dl.acm.org/citation.cfm?id=1287369.1287389

[3] S. Z. Sbz, S. Zdonik, M. Stonebraker, M. Cherniack, U. C. Etintemel, M. Balazinska, and H. Balakrishnan, "The Aurora and Medusa Projects," *IEEE Data Engineering Bulletin*, vol. 26, 2003.

[4] D. J. Abadi, Y. Ahmad, M. Balazinska, M. Cherniack, J. hyon Hwang, W. Lindner, A. S. Maskey, E. Rasin, E. Ryvkina, N. Tatbul, Y. Xing, and S. Zdonik, "The Design of the Borealis Stream Processing Engine," in *In CIDR*, 2005, pp. 277–289.

[5] A. Arasu, S. Babu, and J. Widom, "The CQL Continuous Query Language: Semantic Foundations and Query Execution," *The VLDB Journal*, vol. 15, pp. 121–142, June 2006. [Online]. Available: http://dx.doi.org/10.1007/s00778-004-0147-z

[6] A. Arasu, B. Babcock, S. Babu, J. Cieslewicz, M. Datar, K. Ito, R. Motwani, U. Srivastava, and J. Widom, "STREAM: The Stanford Data Stream Management System," Stanford InfoLab, Technical Report 2004-20, 2004. [Online]. Available: http://ilpubs.stanford.edu:8090/641/

[7] J. Krämer, "Continuous Queries over Data Streams - Semantics and Implementation," Ph.D. dissertation, Philipps-Universität Marburg, 2007.

[8] StreamBase Systems, Inc., "StreamBase CEP," http://www.streambase.com. Last access 2012-07-10.

[9] RTM Realtime Monitoring GmbH, "RTM Analyzer," http://www.realtime-monitoring.de. Last access 2012-07-10.

[10] M. Eckert, "Complex Event Processing with XChange EQ: Language Design , Formal Semantics, and Incremental Evaluation for Querying Events," Ph.D. dissertation, Ludwig-Maximilans-Unversität München, 2008.

[11] C. Moxey, M. Edwards, O. Etzion, M. Ibrahim, S. Iyer, H. Lalanne, M. Monze, M. Peters, Y. Rabinovich, G. Sharon, and K. Stewart, "A Conceptual Model for Event Processing Systems, IBM Form Number REDP-4642-00," IBM Redguide publication, 2010. [Online]. Available: http://www.redbooks.ibm.com/abstracts/REDP4642.htm

[12] K. Teymourian and A. Paschke, "Enabling knowledge-based complex event processing," in *Proceedings of the 2010 EDBT/ICDT Workshops*, ser. EDBT '10. New York, NY, USA: ACM, 2010, pp. 37:1–37:7. [Online]. Available: http://doi.acm.org/10.1145/1754239.1754281

[13] D. Romero, G. Hermosillo, A. Taherkordi, R. Nzekwa, R. Rouvoy, and F. Eliassen, "The DigiHome Service-Oriented Platform," *Software: Practice and Experience*, 2011. [Online]. Available: http://hal.inria.fr/inria-00563678

[14] D. Bonino and F. Corno, "DogOnt - Ontology Modeling for Intelligent Domotic Environments"," in *Proceedings of the 7th International Semantic Web Conference*, ser. LNCS. Springer, 2008, pp. 790–803.

[15] R. Ierusalimschy, *Programming in Lua, Second Edition*. Lua.Org, 2006.

[16] P. Domingos and G. Hulten, "Mining High-speed Data Streams," in *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '00. New York, NY, USA: ACM, 2000, pp. 71–80. [Online]. Available: http://doi.acm.org/10.1145/347090.347107

[17] Object Management Group, "Ontology Definition Metamodel," http://www.omg.org/spec/ODM/1.0/PDF. Last access 2012-07-10.

[18] B. Dawes, D. Abrahams, and R. Rivera, "Boost C++ Libraries," http://www.boost.org. Last access 2012-07-10.

[19] D. Merriman, "BSON - Binary JSON," http://bsonspec.org.

[20] 10gen, Inc., "mongoDB," http://www.mongodb.org. Last access 2012-07-10.

 ISBN: 978-1-61208-242-4