GINSENG Data Processing Framework

Zbigniew Jerzak, Anja Klein, and Gregor Hackenbroich

Abstract. For many applications guided by sensor networks, such as production automation and health monitoring, an efficient data processing with performance assurance is crucial, especially for metrics such as delay and reliability. Our study of current middleware approaches showed that they do not allow a sophisticated complex event processing, neither the performance monitoring. In this chapter we present the GINSENG middleware architecture that provides a 3-tier data processing framework to exploit the benefits of basic publish/subscribe systems, traditional event stream processing and complex business rule processing. Furthermore, the GINSENG middleware architecture provides performance control mechanisms, i.e., monitoring metrics and improvement methods, both of the underlying sensor network and the middleware itself. Finally, it supports the constraints of industrial environments by allowing for the distributed middleware deployment and data processing.

1 Introduction and Motivation

The overall goal of the GINSENG project is to develop a Wireless Sensor Network (WSN) that meets application-specific performance targets and integrates existing industry resource management systems. In order to achieve this goal, the GINSENG project focuses not only on the development of the physical sensor network but also on the development of a middleware platform which gathers and processes information coming from wireless sensors and connects them to ERP systems.

The target application of the GINSENG project is pipe and oil tank monitoring in a refinery environment. Here, pressure, volume flow, and tank level sensors are applied to control the status and enable the predictive maintenance of pipelines and

Anja Klein · Zbigniew Jerzak · Gregor Hackenbroich
SAP Research Dresden, Chemnitzer Straße 48, 01187 Dresden, Germany
e-mail: {zbigniew.jerzak,anja.klein,gregor.hackenbroich}@sap.com

S. Helmer et al.: Reasoning in Event-Based Distributed Systems, SCI 347, pp. 125–150.
springerlink.com

tanks. Moreover, refinery employees may be equipped with mobile sensors to measure gas leakages and raise warnings in case of hazardous situations. In these scenarios a continuous performance control of all involved systems is crucial in order to prevent undetected emergencies. Such emergencies can be a result of simple delays concerning data transfer or data processing. Further performance-critical application areas relevant for the GINSENG project include: fire detection, energy management in manufacturing plants or health care, where status of patients is monitored at intensive care units.

Beyond the continuous performance control, these scenarios require also the analysis of temporal relationships between incoming events as well as continuous data stream processing. Moreover, all information has to be processed as close to the source as possible. This reduces the transferred data volume and minimizes the congestion probability for the limited bandwidth wireless sensor networks. Finally, the physical distribution of sensors requires a distributed middleware deployment.

To meet these constraints, the GINSENG project is developing a modular, hierarchical middleware architecture. The GINSENG middleware can be deployed in a distributed manner on three different levels (sensor node, gateway and central server) providing a flexible platform for distributed data processing – the GINSENG Data Processing Framework.

The remainder of this chapter is structured as follows. In Section 2 we present the overall architecture of the GINSENG middleware. Subsequently, in Section 3, we focus on the GINSENG Data Processing Framework: a 3-tier architecture for distributed data processing. In Section 4 we describe how the GINSENG Data Processing Framework is extended to monitor the performance of the wireless sensor network and the GINSENG middleware. Finally, in Section 5, we summarize the related work in the field of middleware technologies, publish/subscribe mechanisms, event stream and business rule processing. We conclude with a summary of this chapter and an outlook on future work in Section 6.

2 System Architecture

This section gives a brief overview of the GINSENG middleware architecture with specific focus on the event processing. The GINSENG middleware architecture is driven by two important factors: (i) it provides an event-based middleware which (ii) decouples GINSENG components from each other.

The event-based middleware of GINSENG equips the application programmer with an extensive functionality for the creation of distributed systems. Through its components the GINSENG middleware allows the programmer to collect, process and reason on the data as it moves through the system. Moreover, the GINSENG middleware supports a many-to-many interaction scheme overcoming the shortcomings (tight coupling, inflexibility) of the traditional request-reply schemes [39].

The GINSENG middleware is an event-based system which provides a strong notion of decoupling which applies to both internal middleware components and external components which communicate using the GINSENG middleware. The

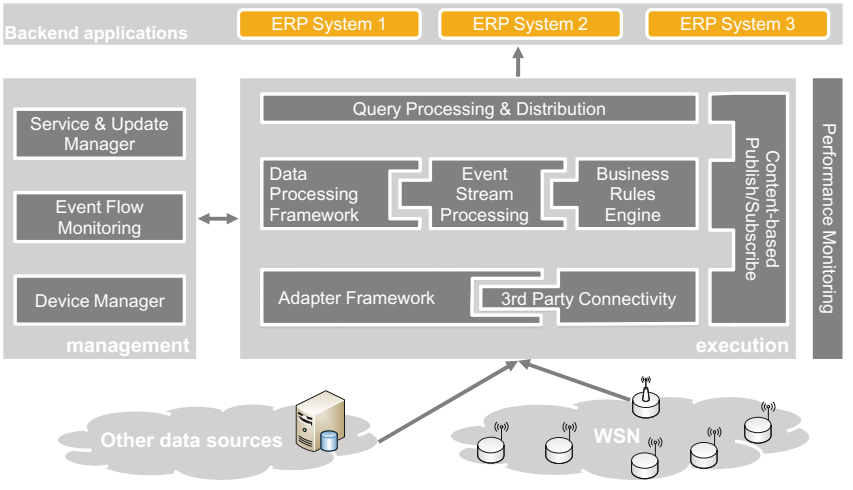

Fig. 1 The architecture of the GINSENG middleware – centralized deployment

external components include Wireless Sensor Networks (or other data sources) and Business Applications (or visualization components) – see Figure 1. Internal components include query processing and distribution, and the adapter framework.

The decoupling of GINSENG components is provided by the underlying communication scheme: the publish/subscribe system. Using the publish/subscribe scheme, components can register their interest for particular data and receive asynchronous notifications about events matching their interest. This type of communication is realized by all components within the GINSENG middleware. This approach is also aligned with the asynchronous (push-based) creation of events by wireless sensors.

Figure 1 shows the details of the GINSENG middleware architecture. The GINSENG middleware is split in two parts: (i) design-time components for all management functionality and (ii) runtime components that are relevant during the execution. The design time components provide user interfaces for all administrative tasks and thus allow the monitoring and configuration of the runtime components.

2.1 Core Components

All core components of the GINSENG middleware are connected using the *Content-based Publish/Subscribe* system. The publish/subscribe system plays a crucial role in the GINSENG middleware as it not only delivers events, but also takes part in the filtering of the information, allowing movement of the computation to within the proximity of the data sources.

The *Adapter Framework* is a pluggable infrastructure which allows the GINSENG middleware to connect to arbitrary data (event) sources. Any required 3rd party driver or service enabling the connectivity can be dynamically plugged into the adapter framework as an independent module. Such modules provide

connectivity not only to SQL-databases, the SAP Business Suite (e.g., an ERP system) or arbitrary web services, but also to smart items such as wireless sensor nodes or smart meters.

The *Data Processing Framework* acts as a foundation which allows for plugging of arbitrary Event Stream Processing (ESP) and Business Rules (BR) engines to benefit from existing well-known and well-tested complex event processing techniques and to support declarative as well as rule-based event processing.

Finally, the *Query Processing and Distribution* controls the ESP and BR engine(s). It is plays a significant role in distributed deployments (see Figure 2) of the GINSENG middleware as it allows optimization of the distributed query execution. It also provides a single, generic query language to the user or backend application to encapsulate the applied ESP engine and performance-related meta-information.

The *Performance Monitoring* is a cross-cutting component. The Performance Monitoring receives relevant meta-information (e.g., latency, packet loss, processing time) from all middleware components, provides them to the interested user and triggers activities for performance improvement.

2.2 Core Technology

The component infrastructure of the GINSENG middleware is developed based on the OSGi Service Platform [59] – a dynamic module system for Java. Each GINSENG middleware component is developed as two OSGi bundles[1] – one for the design-time configuration and one for the runtime execution. The design-time bundle exposes the administration interface by component-specific User Interfaces which are incorporated into the Management Console – see Figure 2. The Management Console connects to the Central Instance which is a single stop for master copies of all configuration data and the runtime component code.

The runtime system consists of one or more middleware nodes. Each node provides a middleware runtime environment, where components' runtime bundles responsible for device connectivity, system integration, data processing, data querying, performance monitoring and performance improvement are deployed and executed. The runtime bundle of a component is a piece of Java code that communicates with a real world entity or external application via the content-based publish/subscribe system.

Each middleware node may run on a regular personal computer, on an embedded system, or within a virtual machine. The middleware runtime environment is built on Java technology and therefore platform-independent. All agents deployed within a node run in an OSGi environment (Eclipse Equinox [35]) which enables dynamic remote code modifications, without requiring a reboot. This OSGi functionality enables a minimal footprint for the runtime. Only bundles required for the specific application scenario are deployed and executed. Nevertheless, it allows this footprint

[1] An OSGi bundle consists of Java classes and other resources that deliver functions of a specific application (component) to application users, as well as providing services and packages to other bundles.

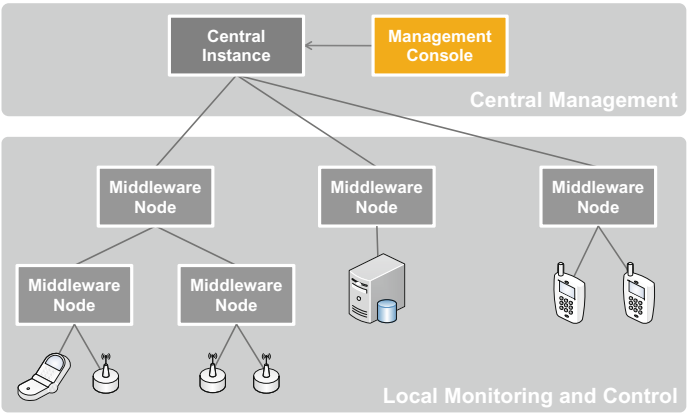

Fig. 2 The distributed deployment of the GINSENG middleware

to grow according to the changing requirements. The OSGi environment allows the deployment of additional bundles, e.g., adapters to connect new data sources, during runtime. Of course, the lightweight implementation of runtime bundles is a key to optimal middleware footprint, that developers need to keep in mind.

3 Data Processing Framework

The GINSENG data processing framework advances the current state of the art in that it combines three technologies to build a unified data processing framework. The three technologies are a content-based publish/subscribe communication system, Event Stream Processing (ESP) engine and Business Rules Engine (BRE). To the best of our knowledge it is the first approach which combines these technologies to build a unified, event-driven data processing framework – see Figure 3.

Events created by wireless sensors are transformed using the Adapter Framework of the GINSENG middleware into the internal GINSENG middleware event format, which is used by all middleware components. Events in this format are subsequently passed to the data processing framework which is responsible for stateful processing of events according to the specified rules. The result of the stateful event processing in the data processing framework (see Figure 3) is a set of complex (business) events which are consumed by the backend applications and/or management components.

The data processing framework in the GINSENG middleware consists of two main parts: the stateless and the stateful part. The input to the data processing framework consists of simple events produced by the adapter framework. Within the GINSENG middleware the content-based publish/subscribe system is considered as the part of the data processing framework. However, since the publish/subscribe system is also used to handle events leaving the data processing framework we indicate this fact by placing it as a separate component in Figure 1.

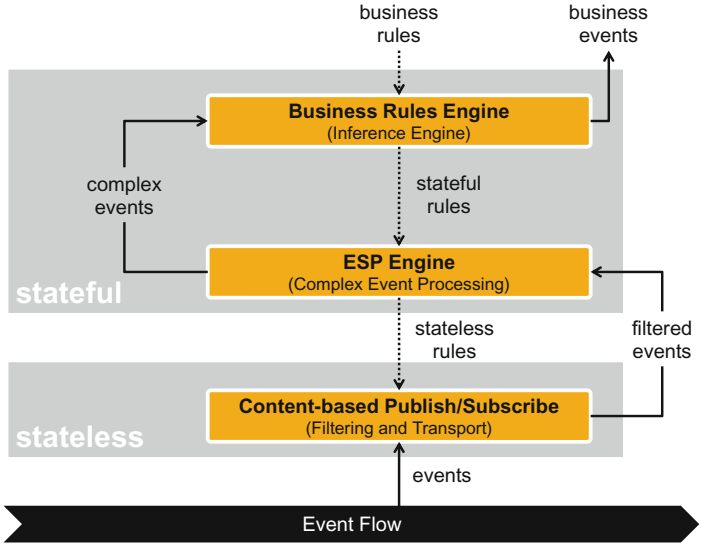

Fig. 3 Data processing framework in the GINSENG middleware

The data processing framework follows the principle of upstream evaluation and downstream replication, which is a well established concept in the literature [10]. Following this principle the GINSENG middleware routes an event in one copy as far as possible and replicates events only downstream. This means events are replicated as close as possible to the interested components (downstream replication). Filters and rules are applied, and patterns are assembled upstream, i.e., as close as possible to the sources of events (upstream evaluation).

The upstream evaluation and downstream replication principle has the following impact on the GINSENG middleware: stateful patterns and business rules entering the GINSENG middleware are evaluated as close to the source of relevant events as possible. However, since publish/subscribe systems are not well suited for handling of stateful rules, the GINSENG middleware decomposes the business rules into stateless and stateful parts, pushing the processing of the stateless rules within the proximity of the data producers.

The stateless event processing in the data processing framework is therefore handled by the content-based publish/subscribe system. Publish/subscribe systems use stateless filters to decide upon the destination of events. In case no destination matches a given event it is dropped – thus reducing the workload on the stateful parts of the data processing framework.

The stateful part of the data processing framework consists of two processing engines: the Event Stream Processing engine (based on the PIPES [33] system) and the Business Rules Processing engine (based on the JBoss Drools [27]). This GINSENG approach is aligned with the vision of ESP and BRE approaches merging into unified CEP platform [7]. The GINSENG vision is driven by the fact that

Business Rules Engines expose a more mature interface for the non-technical users while Event Stream Processing engines provide better performance for operations on multiple sources of homogeneous events.

In addition to the above, the GINSENG middleware extends this approach by asserting that the future data processing frameworks will be based on an asynchronous, data oriented communication protocol: the content-based publish/subscribe system. The three technologies: ESP, BRE and publish/subscribe share a set of similarities which further underline the applicability the GINSENG approach. All three technologies are event-based. They are inherently asynchronous and very well suited for the processing of large quantities of events. In what follows we describe in detail our effort of merging the three data driven techniques.

3.1 Business Rules Engine

Business rules engines rely on the Rete algorithm [22] to process incoming events against a set of user-defined rules – see Section 5.4 for details. In the GINSENG middleware we apply the Business Rules Management System (BRMS) JBoss Drools [27], where events are represented by Java classes. Every event in JBoss Drools can be equipped with meta-data which can state the role of the event, a timestamp (time-point), duration or an expiration time. JBoss Drools processes events as they arrive, and due to the ability to perform temporal reasoning it has a built-in mechanism for garbage collection of events that can no longer match any existing rule.

The heart of the JBoss Drools system is the fast ReteOO algorithm, which provides support for sliding windows and temporal operators (before, after, coincide, during, finishes, finished by, includes, meets, met by, overlaps, overlapped by, starts, started by) for temporal reasoning. Rules can be specified either using the Drools

Listing 1 Two example JBoss Drools rules

```
1  define rule1:
2      if s1.val>5
3          && s2.val<8
4          && s3.tmp=5
5          && s1.loc=s2.loc
6          && s2.loc=s3.loc
7      then alarm()
8  end

10 define rule2:
11     if s1.val>5
12         && s2.val<8
13         && s1.qos=s2.qos
14     then alarm()
15 end
```

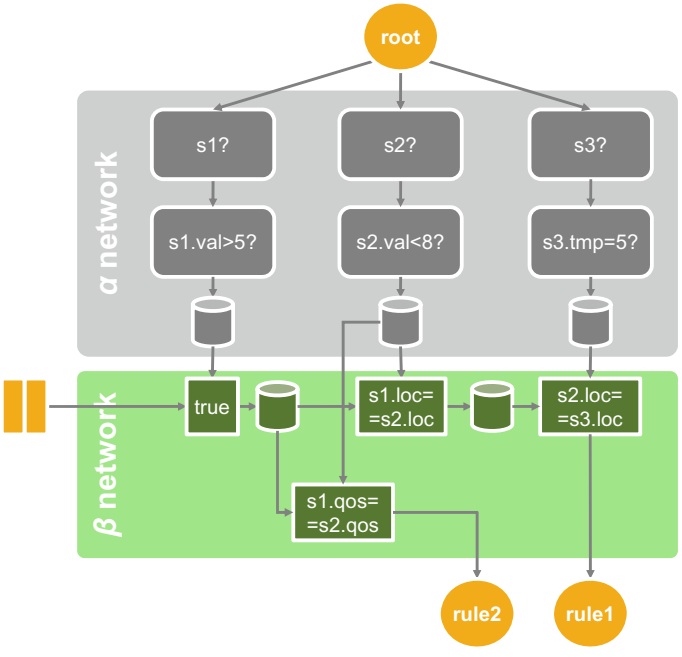

Fig. 4 The Rete network containing rules depicted in Listing 1

native procedural rule language, or via the use of a custom defined domain specific language (DSL). Within the GINSENG project we are developing a GINSENG domain specific language, tailored for use within a refinery environment, with specific focus on the oil tank and pipe monitoring. The use of the GINSENG DSL allows building of an interface between the non-technical refinery personnel and the rule engine.

The basic idea of the Rete algorithm is to create a directed acyclic graph of rule conditions, a so-called Rete network. Nodes in the graph represent rule conditions, e.g., a node can realize a selection operator that filters data based on certain constraints. Whenever a new event appears or the state of the network changes a representation of the event, a so-called working memory element (WME) is created. The WME is then propagated through the Rete network. This is performed in a forward-chaining fashion from the root to the leaf nodes of the network. During this process, every node in the network checks conditions or performs joins and only matching WMEs are passed on to child nodes. Every WME or tuple of WMEs that reaches a leaf node represents a match and results in an activation of the corresponding rule. A rule firing can influence the working memory, i.e., it can change events. If this is the case, the system again creates WMEs from these events and propagates them through the network.

Let us consider the set of rules specified in Listing 1. The rule `rule1` states that if an event of type `s1` and field `val` greater than 5 and an event of type `s2` and field

Table 1 Filters and events in predicate-based semantics

Node	Description
Root	Starts each Rete network (`root`). It has no ancestor nodes.
Type	Distinguishes between different event types (e.g. `s1?`). Type node has only one input, and acts as a filter by passing only events matching the type of the node. The number of type nodes is equal to the number of event types occurring in rules.
Alpha (α)	Performs stateless filtering similar to a selection in relational algebra (e.g. `s1.val>5?`).
Beta (β)	Combines two different types of WMEs to produce a joined result (e.g. `s1.loc==s2.loc`). Beta nodes usually perform joins, however, extensions to realize a universal quantifier, an existential quantifier, a negation and different aggregation functions are available.
Terminal	A leaf node in the Rete network (e.g. `rule2`). If an event reaches the terminal node, this represents the fulfillment of the corresponding rule. Therefore, the number of terminal nodes is equal to the total number of rules.

`val` less than 8 and an event of type `s3` and field `tmp` equal 5 all have the field `loc` set to the same value than the `alarm()` function should be called. Similarly, the rule `rule2` states that if an event of type `s1` and field `val` greater than 5 and an event of type `s2` and field `val` less than 8 have the same value of field `qos` than the `alarm()` function should be called as well.

The set of rules specified in the Listing 1 after loading into the working memory of the Rete algorithm is presented in Figure 4. The description of each of the node types which are illustrated in the Figure 4 are presented in Table 1.

3.2 BRM and Publish/Subscribe

For the evaluation of the Business Rules Engine we have used the Linear Road Benchmark [5] – see Figure 5. The Linear Road Benchmark simulates a tolling system on a fictional expressway, where every car is equipped with a transponder which every 30 seconds emits a car's position. Position reports are used to generate traffic statistics which in turn determine the toll charges. Our evaluation has indicated that the GINSENG Data Processing Framework requires additional mechanisms to lower the memory consumption of the rules processing engine. We have developed a two-stage strategy for coping with this issue. The first stage encompasses the use of the publish/subscribe layer while the second (currently in development) extends this approach to embrace event stream processing engines.

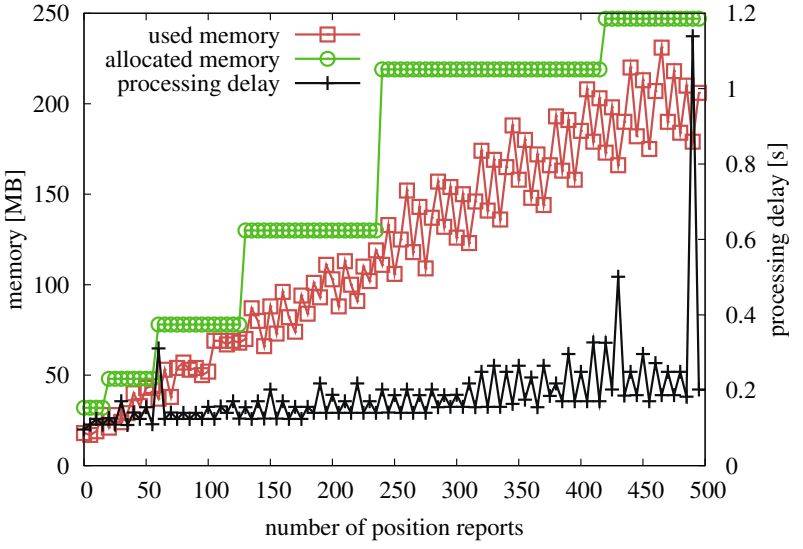

Fig. 5 Linear Road Benchmark – memory consumption using the Drools business rules engine

For the first stage – using the publish/subscribe layer – we exploit the nodes available in the α network in order to drive the processing in the publish/subscribe layer. Specifically, we extract the stateless conditions contained within α network and re-use them in the publish/subscribe layer. We extract both type and α nodes and use their conditions as topics and filters within the publish/subscribe system, respectively.

For example, let us consider the leftmost branch of the α network depicted in the Figure 4. This branch will be converted to an OSGi filter with following properties (props):

Listing 2 OSGi filter

```
1 Properties p = new Properties();
2 p.put(EventConstants.EVENT_TOPIC, "ginseng/events/s1");
3 p.put(EventConstants.EVENT_FILTER, "s1.val>5");
4 context.registerService(EventHandler.class.getName(), this, p);
```

The important aspect of the extraction process is that the filters are not removed from the α network. Instead, filter copies are extracted and following the upstream evaluation principle moved to within the proximity of the data producer. We have decided to use this approach for pre-processing of the business rules as it allows us to exploit the rule parsing and merging mechanism within the Rete network.

3.3 BRE and ESP

The second stage aiming at the optimization of the event processing within the GINSENG Data Processing Framework involves event stream processing engines. Our initial experiments have shown that performing operations like joins on simple events consumes less memory if done within the Event Stream Processing network than within the Business Rules Engine. Therefore, we the goal of the second stage is to migrate parts of the β network into the event stream processing system.

The migration of the β network nodes imposes several implications for the BRE. The most important issue is the need to change the corresponding parts of the Rete network. The reason for the change is the fact that execution of the operators within the β network results in the creation of new event types. This in turn, invalidates both the α and the β network constructs as new events appear at the input of the BRE. For example, let us assume that β node `s1.loc==s2.loc` visible in Figure 4 has been migrated to an ESP system. This implies that ESP system produces a new type of event which is a join between events of type `s1` and `s2`. This in turn requires the creation of new α node in the Rete network which would detect such a new compound even type.

We are currently evaluating the set of operators and corresponding event types which pose the best candidates for the migration from the BRM into the ESP system. In order to provide a general framework we will perform tests using not only the GINSENG specific data, but also the general benchmarking systems [5].

3.4 Domain Specific Languages for Rule Definitions

In order to shield users from changes in the underlying system GINSENG provides a domain specific language (DSL) which allows non-technical users to specify new and modify existing rules. The GINSENG DSL is based on an existing set of tested rules, which constitute the GINSENG knowledge base. Based on these rules a set of DSL queries is developed. As an example let us consider the following rule expressed using the Drools native procedural rule language:

```
1 If WSNMessage(TankLevel<20, GenTimestamp>152467362)
2     then System.out.println("Oil tank is almost empty.");
```

The above rule monitors the level of a tank in the refinery. It states that messages of type `WSNMessage` contain information with respect to the generation time stamp (`GenTimestamp`) and current tank level – `TankLevel`. The above rule fires, i.e., produces a message `Oil tank is almost empty`, whenever the tank level is below 20 and the generation time stamp is newer than 152467362. It can be translated into a DSL rule using the following specification:

```
1 [when] If a message indicates that=WSNMessage()
2 [when] - oil tank level is lower than
3          {EnteredLevel}=TankLevel>{EnteredLevel}
4 [when] - and it was generated no later than at
5          {EnteredTimestamp}=GenTimestamp<{EnteredTimestamp}
6 [then] then log
7          "{Message}"=System.out.println("{Message}");
```

In the above specification first a test for a correct message type (line 1) is executed. Subsequently (lines 2–3), it is tested whether the user entered tank level (`EnteredLevel`) is lower than the one contained in the message and (lines 4– 5) and whether the user entered time stamp (`EnteredTimestamp`) is newer than the one contained in the message. Finally (lines 6–7) a user entered message (`Message`) is displayed whenever the previous conditions are met. The final rule written in the GINSENG Domain Specific Language can take the following – easy to write and understand – form:

```
1 If a message indicates that
2 - oil tank level is lower than 20
3 - and it was generated no later than at 152467362
4 then log "Oil tank is almost empty."
```

The DSL definition of the rule acts like a template for the technical definitions, which allows the business user not only to understand the rule meaning, but also frees him from the underlying implementation details, simultaneously providing the ability to modify the DSL rules. In the example above users can select single constraints in DSL rules, allowing them to, e.g., test only for the tank level without performing the test for generation time.

4 Performance Control in Data Processing Framework

The first step towards a comprehensive performance monitoring is the collection and definition of available and required performance parameters. To evaluate the performance and quality of the event streams, we identified three classes of performance and data quality indicators: (i) event latency, (ii) event loss (reliability), and (iii) event content quality.

The metadata dimensions detailing these classes depend on (i) application requirements and (ii) used sensor nodes and their capabilities of metadata provisioning. To allow the comprehensive evaluation of the data quality of sensor measurement streams, we propose a set of 13 data quality (DQ) and performance dimensions derived from the DQ categories provided by [52]. We show these in Tables 2, 3, and 4. The source of the respective metadata item is either the sensor node (S) or the middleware (MW) itself. Further metadata dimensions can be calculated (C) based on other performance information, allowing easy extension of the provided list. This calculation is performed by the GINSENG middleware, when the respective meta-information is required. This list can be easily extended by deriving further dimensions.

Table 2 List of performance dimensions for event latency

Dimension	Description	Source
GenerationTimestamp	Timestamp of event message generation, e.g., timestamp of sensor measurement	S
MwArrivalTimestamp	Timestamp, indicating the event's arrival at the middleware	MW
MwLeavingTimestamp	Timestamp, indicating the event is leaving the middleware layer towards an application	MW
NetworkLatency	Time interval required for transferring this specific event message within the WSN	C
NodeLatency	Average time interval required for transferring events from the source mote of this event	C
MWLatency	Time interval required for transferring and processing this event in the middleware	C

Table 3 List of performance dimensions for event loss (reliability)

Dimension	Description	Source
PacketLossPerMote	Number of message packets lost during data transmission in the WSN per mote (calculated based on the MoteID and MessageID)	C
PacketLossAvg	Average packet loss over all sensor motes	C

Table 4 List of performance dimensions for event content quality

Dimension	Description	Source
Timeliness	Age of this event message since its generation, calculated as difference of current system time and generationTimestamp	C
Completeness	Fraction of original sensor values	C
Accuracy	Maximal systematic numeric error of a sensor measurement	MW
Confidence	Maximal statistical error of a sensor measurement	C
DataBasis	Amount of raw data underlying a data processing result or complex event	C

4.1 Performance Monitoring Infrastructure

To record and manage the above listed parameters within the WSN and the middleware, event messages have to be enriched with performance information. However, the metadata dimensions listed above would significantly increase the data volume. Thus, within the GINSENG middleware we apply the window-based approach (first proposed in [30]) for the data quality management in data streams. To allow for efficient data quality management, the event stream D, comprising a continuous stream of m events consisting of n attribute values $A_i(1 \leq i \leq n)$, is partitioned into κ consecutive, non-overlapping data quality windows $w_i(k)(1 \leq k \leq \kappa)$, each of which is identified by its starting point t_b, its end point t_e, the window size ω and the corresponding attribute A_i. In addition to the event data $e_i(j)(t_b \leq j \leq t_e)$, the window contains a set of d performance and data quality information items, each describing one performance dimension. Each window-wise performance information item is calculated as an average of the original event-wise meta-information items. For example, Figure 6 shows the window's network latency $l_{WSN,w}(k)$ the window accuracy $a_w(k)$ and the window completeness $c_w(k)$ with $\omega = 5$.

The window size ω can be defined independently for each event attribute and/or window. Small jumping windows result in high-granularity performance information at the expense of a higher data overhead. A wider window definition guarantees the important resource savings that are essential for data stream environments; this happens by risking information with lower granularity and decreased correctness due to error deviations introduced by the window-wise metadata aggregation.

4.2 Performance and Data Quality Algebra

To compute the performance and quality of event steam processing results, the traditional stream operators have to be extended as illustrated in Figure 7. For each data processing function F consisting of operators $o \in O$, a metadata function F^M has to be composed of the data quality operators $o^M \in O^M$ to compute the metadata M^Y, describing the derived knowledge $Y = F(X)$.

Table 5 lists the operators o extracted from traditional event stream processing engines and the GINSENG application scenarios, for which metadata operators o^M have to be described. The data quality algebra defines how operators influence each DQ dimension. For a more detailed description the reader is referred to [32].

Generation Timestamp	...	210	211	212	213	214	215	216	217	218	219	220	221	222	223	224	225	226	227	228	229
Pressure	...	180	178	177	175	176	181	189	201	204	190	194	192	189	183	215	210	211	199	187	184
Network Latency	...					12,92					13,34					12,1					12,12
Completeness	...					0,9					0,8					0,9					0,99
Accuracy	...					3					3,9					2,78					2,86

Fig. 6 Window-based approach for an event stream sample

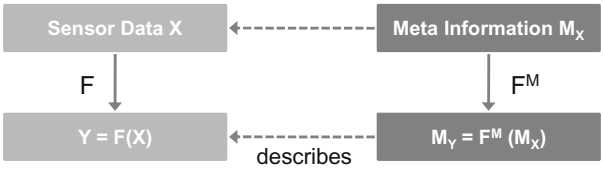

Fig. 7 Generic metadata algebra

Table 5 Data quality algebra operators

Type	Operators
Numeric Operators	Unary operators (e.g., square root); Binary operators (e.g., addition)
Signal Analysis	Sampling; Interpolation; Frequency analysis; Frequency filtering
Relational Algebra	Projection; Selection; Aggregation; Join
Rule-based Operators	Set operators (e.g., union); Boolean operators; Threshold comparison

4.3 Performance Improvement

In [31], we present the quality-driven optimization of stream processing that improves the resulting quality of data and service. We identify the targeted quality-driven process optimization as multi-objective, non-linear, continuous optimization problem with side conditions. We define the optimization objectives (for each data quality dimension) and optimization parameters (that configure the required stream processing operators). In the following, we briefly describe the developed generic optimization framework and discuss major evaluation results.

4.3.1 Optimization Framework

The optimization framework is illustrated in Figure 8. The data quality-driven optimization is executed continuously to tune the data stream processing during system runtime. As soon as an optimal parameter set is found and deployed, it has to be checked against the currently processed data stream. The online tuning allows the seamless adaptation to varying stream rates, measurement values and data quality requirements.

First, the system evaluates by means of static information like maximal sensor stream rate or sensor precision, if the user-defined quality requirements can be accomplished or if conflicts exclude a realization of all sub-objectives. In the latter case, the conflict is reported to the user. To check the satisfiability of DQ requirements, no access to streaming data is needed.

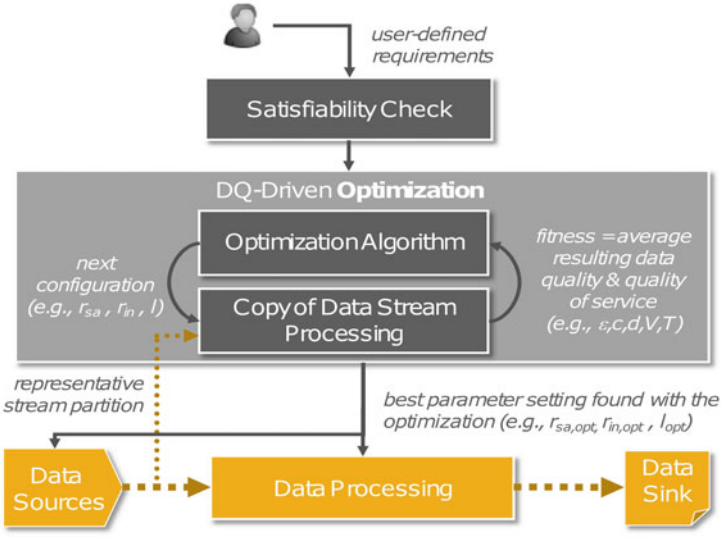

Fig. 8 Optimization framework

As the optimization must not interfere with the ongoing data stream processing, it is separated into an independent system component. To execute the optimization in parallel with the traditional data stream processing, the processing path with all its operators is copied into the optimization component. To execute the optimization, the framework applies a heuristic optimization algorithm (see Section 4.3.2 for comparison of different algorithms). In each algorithm iteration, solution individuals, defining the stream operators' configuration, have to be applied, evaluated and compared. The solution individuals are ranked according to the achieved fitness, that summarizes all sub-objectives of the optimization problem, and the best solutions are used to start the next iteration. Step by step, the resulting fitness and, thus, the configuration of the stream operators is improved.

Each solution is evaluated by directing a representative data stream partition through the copied processing path in the optimization component. This stream partition for optimization constitutes a data stream window of n data tuples. It may either be selected in batch-mode at the beginning of each optimization run and used in each iteration without changes. On the other hand, the partition can be updated for each algorithm iteration with current tuples from the original stream, to reflect the dynamic progression of the data stream and allow the continuous optimization. After the stream partition is completely processed, the fitness of the tested configuration is calculated.

As soon as the fitness accomplishes the user requirements, the optimization problem is solved. The new parameter setting is applied to the original processing path. The sampling operators are updated with the optimized sampling rates $r_{sa,opt}$. The frequency analyses and aggregations are updated with the determined group sizes

l_{opt}. Finally, the data quality initialization at the sensor nodes is re-configured with the new window sizes ω_{opt}.

The logical distinction between optimization and processing enables also their physical separation, for example on a distinct server node. Thus, the optimization task has no negative impact on the performance of the traditional data stream processing.

4.3.2 Evaluation Results

The evaluation of the optimization framework showed that the processing time rises nearly linearly for small to medium partition lengths of 100 to 1000 stream tuples. Only for very large stream partitions does the iteration duration exhibit an exponential character. Further, we can prove linear scalability for increasing processing complexity. The time required for one algorithm iteration rises linearly with increasing numbers of sensors as data sources and operators in the processing path.

Besides the evaluation of the generic framework, we compared different optimization algorithms. The Monte-Carlo-Search (MC) performs a random search over the problem domain and serves as reference value defining the lower performance bound [38]. The single-objective optimization using the heuristic Evolution Strategy [36] is executed with randomly chosen weights (SO-R) as well as optimal weights (SO-O), which determine a well-balanced objective compromise. Finally, the multi-objective optimization (MO) approximates the Pareto front of all optimal compromises.

We evaluated the overall time performance of single- and multi-objective optimization with respect to the achieved quality improvement for 16 sensor data sources and 10 randomly inserted aggregations. The quality improvement is expressed as percentage value $(q_{before} - q_{after})/q_{before}$. The Monte-Carlo-Search performs worst followed by the randomly initiated single-objective optimization, requiring 5.2 and 2.9 seconds, respectively, for a DQ improvement of 10%. The single-objective optimization executed with well-balanced weights performs best (1.6s for 10%). However, the definition of these weights requires multiple optimization runs and has to be adapted as soon as stream characteristics or user requirements change. The multi-objective optimization (MO) is a little slower (1.9s) due to the complex computation of the Pareto front. However, the result comprises the complete set of all optimal compromises and no pre-processing to determine optimal weights is necessary.

The evaluation showed that the designed quality-driven optimization provides good scalability with regard to the applied stream partition length as well as increasing complexity of the data stream processing. Data quality and quality of service could be improved within a few seconds. Further, we deduce that the single-objective optimization in batch mode is the best choice for constant user requirements and steady data streams. If streaming data values present high fluctuations or user requirements are often adjusted, the multi-objective optimization constitutes the better option.

4.3.3 Open Issues

The research work presented above provides a good starting point toward performance control in the GINSENG middleware. We need to extend this quality-driven optimization framework towards a performance-driven optimization. The optimization objectives have to be extended to cover the performance AND data quality indicators defined above. Further, the applied optimization algorithms, e.g., the evolution strategy, has to be adapted to embrace these novel objectives and the related optimization parameters. After that, the optimization framework can be applied without modifications to enable the performance maximization of event stream processing in the GINSENG middleware.

5 Related Work

This section provides an overview of the state-of-the art concerning middleware developments. Further, we discuss publish/subscribe systems and event- and rule-based processing engines applicable to our processing framework.

5.1 Middleware Technology

In this section we analyze a representative set of existing middleware approaches with regard to performance control and data processing supported. The project on Wireless Accessible Sensor Populations (WASP) addresses the energy-efficient and secure integration of Wireless Sensor Networks (WSN) into applications like traffic and herd control and integrates data cleansing techniques for data quality improvement [58]. WASP fulfills business applications requirements related to communication (e.g. on demand, event based sensor data acquisition, WSN discovery), but does not support time synchronization. With respect to sensor data processing, WASP is restricted to a basic operator set and does not support complex event processing, nor performance or quality-guided stream processing.

The middleware developed within the project on PROduct lifecycle Management and Information tracking using Smart Embedded systems (PROMISE) allows managers, designers, and operators to track, manage and control product information at any phase of its lifecycle (design, manufacturing, maintenance, recycling), at any time and anywhere in the world [40]. However, PROMISE does not provide any tools or components for performance or quality management and monitoring.

The Collaborative Business Items (CoBIs) project [14] provides a service-oriented approach to support business processes that involve physical entities (goods, tools, etc.) in large-scale enterprise environments in a transparent way. The CoBIs project primarily focuses on the design of a service-oriented middleware for deploying, running, and querying services on sensor nodes and does not support data and event processing as proposed by the GINSENG approach.

The SAP Auto-ID Infrastructure (AII) enables the integration of all automated communication and sensing devices (e.g., RFID, Bluetooth and bar-code devices)

as well as intelligent programmable language controls [45]. SAP AII provides functionalities for the data transfer from sensor motes to backend applications, but does not include any data processing and performance monitoring technology.

Beyond this small excerpt of existing middleware systems, our comprehensive analysis showed that middleware approaches for connecting field devices with backend systems only support basic pre-processing of sensor data and do not provide any quality and/or performance control mechanism required by the industrial applications, where unreliable or deferred data may risk a system breakdown or even personal injuries. The GINSENG middleware addresses the performance assurances (reliability, timeliness and precision) and control related issues. Moreover, it provides a) an abstraction to conceal the heterogeneity of underlying sensor motes, b) a complex data stream and event processing engine, c) a robust monitoring and management of application logic.

5.2 Publish/Subscribe Communication

The GINSENG middleware uses the OSGi-based publish/subscribe paradigm as its underlying communication infrastructure. In what follows we give a brief overview of the existing approaches towards the design and implementation of publish/subscribe systems.

Publish/subscribe is the first communication paradigm to unify three important decoupling properties: space, time and synchronization decoupling [21] which allows for a flexible communication between content producers (publishers) and content consumers (subscribers). The decoupling properties ensure that the communication is anonymous (space decoupling), asynchronous (synchronization decoupling) and communicating parties do not need to be active at the same time (time decoupling).

The above properties position pub/sub paradigm as a very attractive interaction scheme for building loosely coupled, event driven applications. Moreover, due to the decoupling properties publish/subscribe can act as a enabling technology for higher level dynamic and distributed event-driven services, like Complex Event Processing or Business Rules Processing. It is strictly for this reason that we design the GINSENG middleware to rely on the publish/subscribe paradigm as its basic communication primitive.

The basic interaction scheme of the publish/subscribe system is based on the well known observer pattern [24]. Data consumers express their interest using subscriptions. The first publish/subscribe systems started appearing over two decades ago. One of the first publish/subscribe systems was Information Bus [37] which is similar in concept to the generative communication model of tuple spaces [9]. The Information Bus implements a topic-based publish/subscribe paradigm, i.e., filters in subscription events took a form of fixed topics. The Information Bus and other topic-based publish/subscribe systems [49, 6] allow content consumers to dynamically specify topics which segment the information into channels. Publishers can

specify to which channel the produced information belong, thus allowing interested subscribers to receive it.

The increasing need for heterogeneity and expressiveness among publish/subscribe systems has lead to the development of the content-based [41] and type-based [20] systems. Type-based systems provide type safety and encapsulation by using the hierarchy of the class structure to filter events. Content-based systems provide a much greater degree of flexibility by relying on arbitrary filter expressions (often in form of conjunctions of predicate functions [10, 29]) in order to select events of interest to the subscribers.

Parallel to the academic development of the publish/subscribe communication systems, the industry has started the adoption and standardization of the publish/-subscribe communication. One of the first widely recognized and available specifications was the Java Message Service [26] (JMS) which has been implemented in multiple commercial, e.g., SonicMQ [51], and open source products, e.g., HornetQ [28]. Other publish/subscribe specifications are WS-Eventing [17] and WS-Notification [12, 25]. WS-Eventing and WS-Notification standards are not limited to topic-based subscriptions (like JMS), allowing the definition of arbitrary filters in form of XPath [18] queries in case of WS-Eventing or user defined queries (including, e.g., XPath) in case of WS-Notification.

The OSGi publish/subscribe system has been proposed by the OSGi Alliance in the Service Platform Specification [53]. The publish/subscribe communication scheme implemented in the GINSENG middleware is defined within the Event Admin service specification [54] and provides a topic-based publish/subscribe system with additional ability to increase the filter selectivity (content-based publish/subscribe) by using a LDAP-style filter specification [50].

5.3 Event Stream Processing

The author in [34] states that CEP systems process incoming raw events from multiple data sources in real-time using algorithms and rules to determine correlations, trends and patterns expressed in outgoing complex events. This goal is achieved by handling the correlation of temporal as well as other events which occur simultaneously and in high volumes [7]. While the Event Stream Processing (ESP) part of the CEP technology is relatively new to the market with first commercial offerings appearing in 2004 [46] the Business Rules part of the CEP technology is already familiar to most organizations [44].

The event stream processing systems can perform filtering, correlation, transformation and aggregation operations on multiple event streams. There exist a number of academic prototypes which have also been partially transformed into commercial offerings. Examples include Aurora/Borealis [1, 2] (commercialized by Coral8 [15]), TelegraphCQ [11] (commercialized by Truviso [56]) and PIPES [33] (commercialized by RTM Realtime Monitoring GmbH [42]). The processing of data within ESP engines is driven by queries which can take be either declarative,

e.g., CQL: The Continuous Query Language [4], or procedural, e.g., SQuAl [2]. Declarative queries resemble the standard SQL syntax with event stream specific extensions (e.g. support for windows), while procedural approaches allow for composition of queries out of well known building blocks (operators).

In recent years due to the distributed nature of data sources and the increasing availability of the on-demand resources [16] the distributed ESP approaches have been the focus of academic research. Recent developments include systems like Borealis [1], System S [3] and NextCEP [47]. The focus of the distributed ESP research lies on the scalability (via load and query distribution across multiple system nodes) and fault tolerance (in most cases via state-machine replication [13, 48]) issues.

In what follows we give a brief descriptions of two open source ESP systems which are used within the GINSENG middleware. The first system we use is PIPES [33]. PIPES is a flexible and extensible infrastructure providing fundamental building blocks to implement a data stream management system. It constructs directed acyclic query graphs based on a publish/subscribe mechanism which is integrated into the graph nodes. PIPES is based on XXL [8] – a Java library that contains a rich infrastructure for implementing advanced query processing functionality.

The second ESP system which is used in the GINSENG middleware is Esper [19]. Esper implements an Event Query Language (EQL) which allows for registering of queries in the engine. A listener class is called by the engine when the EQL condition is matched as events flow in. The EQL enables the expression of complex matching conditions that include temporal windows, joining of different event streams, as well as filtering, aggregation, and sorting. Esper statements can also be combined together with "followed by" conditions thus deriving complex events from more simple events. Events can be represented as Java classes, XML documents or `java.util.Map`, which promotes reuse of existing systems acting as messages publishers. Esper also includes a historical data access layer to connect to databases, combining historical data and real time data in one single query.

5.4 Business Rules Engines

Business Rules Management Systems (BRMS) provide the ability to easily express the rules in a simple and understandable way by using abstractions, such as flowcharts, decision trees and decision tables as well as scoring models and textual if-then rules [7]. Therefore, the benefit of BRMS lies in the fact that a change in the rules can be easily reflected in the system by a non-technical user. BRMS, in contrast to CEP solutions, are a mature offering with good market penetration.

Business rules engines (BRE), which constitute the core of the BRMS offerings, are in most cases designed to accept discrete events which typically have a complex payload (multiple elements) [7]. There exist currently a number of commercial offerings for business rules platforms, including, but not limited to, Tibco Business Events [55], UC4 Automation Engine [57] and ruleCore CEP Server [43]. There exist also non-commercial or open source systems, examples being JBoss Drools [27]

and Jess [23]. The common denominator for most of the business rules engines is the use of the Rete algorithm [22] to process incoming events against the stored rules. The use of the Rete algorithm allows reuse of the common parts of rules and thus reducing the number of operations which are necessary to match incoming events.

6 Summary

In this chapter, we presented the GINSENG middleware which closes the gap between (wireless) sensors networks and arbitrary backend applications, such as the monitoring tool for a manufacturing site, the health care system at an intensive care unit in a hospital or a zoological warehouse to track the routes of endangered animals.

After a short description of the overall GINSENG architecture, the main part of this chapter focused on the distributed event stream processing. With the presented 3-tier data processing framework composed of a low-level publish/subscribe system, an exchangeable event stream processing engine and the high-level business rule processing engine, we enabled the seamless integration of raw sensor events into the complex business rule evaluation.

As GINSENG targets performance-critical application scenarios where long data transfer delays, and missing or incorrect events may be hazardous, we proposed mechanisms for the on-the-fly performance monitoring. Besides metrics for the basic performance measurement, a performance algebra to compute the accuracy of event processing results and methods for the eventual performance improvement were illustrated. Finally, we gave an overview of related work concerning middleware approaches, as well as publish/subscribe, event and rule-based processing engines applicable to our processing framework.

In further work, we will investigate and refine the integration between the event processing engine and the business rules system. Moreover, the data quality driven optimization of the event stream processing will be extended to cover all dimensions listed in Tables 2, 3, and 4 as well as the GINSENG domain specific query language. Finally, we will evaluate the GINSENG middleware against existing middleware, event stream and business rules approaches to demonstrate the advantages of the 3-tier processing architecture over stand-alone processing engines.

Acknowledgments

The research leading to these results has received funding from the European Community's Seventh Framework Program (FP7/2007-2013) under grant agreement No. 224282. We would also like to thank Sebastian Weng for his support with the execution of the Linear Road benchmark.

References

1. Abadi, D.J., Ahmad, Y., Balazinska, M., Cetintemel, U., Cherniack, M., Hwang, J.-H., Lindner, W., Maskey, A.S., Rasin, A., Ryvkina, E., Tatbul, N., Xing, Y., Zdonik, S.: The design of the borealis stream processing engine. In: CIDR 2005: Second Biennial Conference on Innovative Data Systems Research, pp. 277–289 (2005)
2. Abadi, D.J., Carney, D., Çetintemel, U., Cherniack, M., Convey, C., Lee, S., Stonebraker, M., Tatbul, N., Zdonik, S.B.: Aurora: a new model and architecture for data stream management. VLDB J. 12(2), 120–139 (2003)
3. Amini, L., Jain, N., Sehgal, A., Silber, J., Verscheure, O.: Adaptive control of extreme-scale stream processing systems. In: ICDCS 2006: 26th IEEE International Conference on Distributed Computing Systems, Lisboa, Portugal, July 2008, p. 71. IEEE Computer Society, Los Alamitos (2006)
4. Arasu, A., Babu, S., Widom, J.: The cql continuous query language: semantic foundations and query execution. The VLDB Journal 15(2), 121–142 (2006)
5. Arasu, A., Cherniack, M., Galvez, E.F., Maier, D., Maskey, A., Ryvkina, E., Stonebraker, M., Tibbetts, R.: Linear road: A stream data management benchmark. In: Nascimento, M.A., Özsu, M.T., Kossmann, D., Miller, R.J., Blakeley, J.A., Schiefer, K.B. (eds.) VLDB, pp. 480–491. Morgan Kaufmann, San Francisco (2004)
6. Baldoni, R., Beraldi, R., Quéma, V., Querzoni, L., Piergiovanni, S.T.: Tera: topic-based event routing for peer-to-peer architectures. In: Jacobsen, H.-A., Mühl, G., Jaeger, M.A. (eds.) DEBS 2008: Proceedings of the 2007 Inaugural International Conference on Distributed Event-Based Systems, Toronto, Ontario, Canada, June 2007. ACM International Conference Proceeding Series, vol. 233, pp. 2–13. ACM, New York (2007)
7. Brett, C., Gualtieri, M.: Must you choose between business rules and complex event processing platforms? Forrester Research (January 2009)
8. Cammert, M., Heinz, C., Krämer, J., Schneider, M., Seeger, B.: A status report on xxl - a software infrastructure for efficient query processing. IEEE Data Eng. Bull. 26(2), 12–18 (2003)
9. Carriero, N., Gelernter, D.: Linda in context. Commun. ACM 32(4), 444–458 (1989)
10. Carzaniga, A., Rosenblum, D.S., Wolf, A.L.: Design and evaluation of a wide-area event notification service. ACM Trans. Comput. Syst. 19(3), 332–383 (2001)
11. Chandrasekaran, S., Cooper, O., Deshpande, A., Franklin, M.J., Hellerstein, J.M., Hong, W., Krishnamurthy, S., Madden, S., Raman, V., Reiss, F., Shah, M.A.: Telegraphcq: Continuous dataflow processing for an uncertain world. In: CIDR 2003: First Biennial Conference on Innovative Data Systems Research (2003)
12. Chappell, D., Liu, L.: Web Services Brokered Notification. 1.3 (2006), `http://docs.oasis-open.org/wsn/wsn-ws_brokered_notification-1.3-spec-os.htm`
13. Clement, A., Kapritsos, M., Lee, S., Wang, Y., Alvisi, L., Dahlin, M., Riche, T.: UpRight cluster services. In: Proceedings of the 22 nd ACM Symposium on Operating Systems Principles (SOSP), pp. 277–290 (2009)
14. CoBIs. Collaborative Business Items, `http://www.cobis-online.de/`
15. Coral8 Inc. Complex event processing with coral8, `http://download.microsoft.com/.../complex_event_processing_with_coral8_final.pdf`
16. Creeger, M.: Cloud computing: An overview. Queue 7(5), 3–4 (2009)
17. Davis, D., Malhotra, A., Warr, K., Chou, W.: Web service eventing, w3c working draft (2009), `http://www.w3.org/tr/2009/wd-ws-eventing-20090317/`

18. DeRose, J.C.S.: Xml path language, xpath (1999), http://www.w3.org/tr/xpath
19. EsperTech. Esper reference documentation (1999), http://esper.codehaus.org/esper-3.3.0/doc/reference/en/pdf/esper_reference.pdf
20. Eugster, P.: Type-based publish/subscribe: Concepts and experiences. ACM Transactions on Programming Languages and Systems 29(1), 1–50 (2007)
21. Eugster, P.T., Felber, P.A., Guerraoui, R., Kermarrec, A.-M.: The many faces of publish/-subscribe. ACM Comput. Surv. 35(2), 114–131 (2003)
22. Forgy, C.: Rete: A fast algorithm for the many patterns/many objects match problem. Artif. Intell. 19(1), 17–37 (1982)
23. Friedman-Hill, E.: Jess, http://www.jessrules.com/
24. Gamma, E., Helm, R., Johnson, R., Vlissides, J.: Design Patterns. Elements of Reusable Object-Orineted Software. Addison-Wesley Professional, Reading (1995)
25. Graham, S., Hull, D., Murray, B.: Web Services Brokered Notification. 1.3. Web Services Base Notification. 1.3 (2006), http://docs.oasis-open.org/wsn/wsn-ws_base_notification-1.3-spec-os.htm
26. Hapner, M., Burridge, R., Sharma, R., Fialli, J., Stout, K.: Java message service (April 2002), http://java.sun.com/products/jms/
27. JBOSS. Drools, http://labs.jboss.com/drools
28. JBoss. Hornetq, http://www.jboss.org/hornetq
29. Jerzak, Z., Fetzer, C.: Bloom filter based routing for content-based publish/subscribe. In: DEBS 2008: Proceedings of the second international conference on Distributed event-based systems, Rome, Italy, July 2008, pp. 71–81. ACM, New York (2008)
30. Klein, A.: Incorporating quality aspects in sensor data streams. In: Proceedings of the 1st ACM Ph.D. Workshop in CIKM (PIKM), pp. 77–84 (2007)
31. Klein, A., Lehner, W.: How to optimize the quality of sensor data streams. In: ICCGI 2009: Proceedings of the 2009 Fourth International Multi-Conference on Computing in the Global Information Technology, pp. 13–19. IEEE Computer Society, Los Alamitos (2009)
32. Klein, A., Lehner, W.: Representing data quality in sensor data streaming environments. J. Data and Information Quality 1(2), 1–28 (2009)
33. Kraemer, J., Seeger, B.: Pipes - a public infrastructure for processing and exploring streams. In: Weikum, G., Koenig, A.C., Deßloch, S. (eds.) Proceedings of the 9th ACM SIGMOD International Conference on Management of Data, pp. 925–926. ACM, New York (2004)
34. Leavitt, N.: Complex-event processing poised for growth. Computer 42(4), 17–20 (2009)
35. McAffer, J., VanderLei, P., Archer, S.: OSGi and Equinox: Creating Highly Modular Java Systems. Addison-Wesley Professional, Reading (2010)
36. Michalewicz, Z.: Genetic Algorithms Plus Data Structures Equals Evolution Programs. Springer, Heidelberg (1994)
37. Oki, B.M., Pflügl, M., Siegel, A., Skeen, D.: The information bus – an architecture for extensible distributed systems. In: Liskov, B. (ed.) Proceedings of the 14th Symposium on the Operating Systems Principles, pp. 58–68. ACM Press, New York (1993)
38. Patel, N.R., Smith, R.L., Zabinsky, Z.B.: Pure adaptive search in monte carlo optimization. Mathematical Programing 43(3), 317–328 (1989)
39. Pietzuch, P.R.: Hermes: A Scalable Event-Based Middleware. PhD thesis, Computer Laboratory, Queens' College. University of Cambridge (February 2004)

40. PROMISE. PROduct lifecycle Management and Information tracking using Smart Embedded system, http://www.promise.no/
41. Rosenblum, D.S., Wolf, A.L.: A design framework for internet-scale event observation and notification. SIGSOFT Softw. Eng. Notes 22(6), 344–360 (1997)
42. RTM Realtime Monitoring GmbH, http://www.realtime-monitoring.de/
43. ruleCore. Cep server, http://rulecore.com/
44. Rymer, J.R., Gualtieri, M., Brown, M., Salzinger, C.: The forrester wave: Business rules platforms, q2 2008 (April 2008)
45. SAP AG. SAP Auto-ID Infrastructure, http://www.sap.com/platform/netweaver/autoidinfrastructure.epx
46. Schulte, W., Blechar, M., Jones, T., Sholler, D., Thompson, J., Malinverno, P., Gassman, B.: The growing impact of commercial complex-event processing products. Gartner Research (October 2009)
47. Schultz-Moller, N.P., Migliavacca, M., Pietzuch, P.: Distributed complex event processing with query rewriting. In: DEBS 2009: Proceedings of the 2009 International Conference on Distributed Event-Based Systems, pp. 1–12 (2009)
48. Singh, A., Fonseca, P., Kuznetsov, P., Rodrigues, R., Maniatis, P.: Zeno: eventually consistent byzantine-fault tolerance. In: NSDI 2009: Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation, pp. 169–184. USENIX Association, Berkeley (2009)
49. Skeen, M.D., Bowles, M.: Apparatus and method for providing decoupling of data exchange details for providing high performance communication between software processes. U.S. Patent No. 5,557,798 (July 1989)
50. Smith, M., Howes, T.: Lightweight directory access protocol (ldap): String representation of search filters. Request for Comments: 4515 (2006)
51. SonicMQ, http://web.progress.com/en/sonic/
52. Strong, D.M., Lee, Y.W., Wang, R.Y.: Data quality in context. Communications of the ACM 40(5), 103–110 (1997)
53. The OSGi Alliance. Osgi service platform - core specification (2009), http://www.osgi.org/
54. The OSGi Alliance. Osgi service platform - service compendium (2009), http://www.osgi.org/
55. TIBCO. Businessevents, http://www.tibco.com/software/complex-event-processing/businessevents
56. Truviso. Web analytics software, http://www.truviso.com/
57. UC4. Automation engine, http://www.uc4.com/products-solutions/automation-engine.html
58. WASP. Wireless Accessible Sensor Populations., http://www.wasp-project.org/
59. Wütherich, G., Hartmann, N., Kolb, B., Lübken, M.: Die OSGi Service Platform: Eine Einführung mit Eclipse Equinox. dpunkt, Heidelberg (2008)

Glossary

GINSENG	The goal of the EU-project GINSENG is the development of a performance-controlled wireless sensor network.
WSN	A **W**ireless **S**ensor **N**etwork is a network of sensor nodes that communicate wirelessly.
Middleware	The middleware is a computer software that connects software components or applications.
Publish/-Subscribe	Publish/Subscribe (or pub/sub) is a messaging paradigm where senders (publishers) broadcast messages that are only received by Subscribers who defined their interest in advance.
BR	**B**usiness **R**ules are application- or domain-specific rules defining the selection of alternative execution paths in complex business processes.
BRP	The **B**usiness **R**ule **P**rocessing evaluates incoming business and/or event data against business rules to guide business processes.
BRM	The **B**usiness **R**ule **M**anagement includes the definition, management and processing of business rules.
ESP	The **E**vent **S**tream **P**rocessing embraces all techniques and methods for the real-time processing of continuous or discrete event data.
CEP	The **C**omplex **E**vent **P**rocessing combines and evaluates incoming raw events against rules or patterns to create outgoing complex events.
DSL	A **D**omain **S**pecific **L**anguage is a programming or specification language dedicated to a particular problem domain, a particular problem representation technique, and/or a particular solution technique.
DQ	The **D**ata **Q**uality defines the appropriateness of a given data item for a specific task, expressed e.g., as accuracy or completeness.
Performance	The performance of a system describes its non-functional ability to solve a specific task, expressed e.g., as latency or reliability.
Performance Control	The performance control includes the monitoring of the current system performance as well as methods for the performance improvement.
MO	The **M**ulti-objective **O**ptimization targets for the Pareto front of optimal compromises between all involved sub-objectives.
SO	The **S**ingle-objective **O**ptimization summarizes all sub-objectives in one objective function, which is optimized afterwards.
MC	The **M**onte-**C**arlo-Search performs a random search over all possible solutions to find the optimal one.