Distributed Heterogeneous Event Processing

Enhancing Scalability and Interoperability of CEP in an Industrial Context

Björn Schilling
Institute of Parallel and Distributed Systems
Universitätsstr. 38
Stuttgart, Germany
bjoern.schilling@ipvs.uni-stuttgart.de

Boris Koldehofe
Institute of Parallel and Distributed Systems
Universitätsstr. 38
Stuttgart, Germany
boris.koldehofe@ipvs.uni-stuttgart.de

Udo Pletat
IBM Deutschland Research & Development GmbH
Schönaicherstr. 220
Böblingen, Germany
pletat@de.ibm.com

Kurt Rothermel
Institute of Parallel and Distributed Systems
Universitätsstr. 38
Stuttgart, Germany
kurt.rothermel@ipvs.uni-stuttgart.de

ABSTRACT

Although a significant amount of research has investigated the benefits of distributed CEP in terms of scalability and extensibility, there is an ongoing reluctance in deploying distributed CEP in an industrial context. In this paper we present the DHEP system developed together with the IBM®[1] laboratory in Böblingen. It addresses some of the key problems in increasing the acceptance of distributed CEP, for example supporting interoperability between heterogeneous event processing systems. We present the concepts behind the DHEP system and show how those concepts help to achieve scalable and extensible event processing in an industrial context. Moreover, we verify in an evaluation study that the additional cost imposed by the DHEP system is moderate and 'affordable' for the benefits provided.

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Computer-Communication Networks—*Network Architecture and Design*; C.2.4 [**Computer Systems Organization**]: Computer-Communication Networks—*Distributed Systems*

General Terms

Distributed Systems

Keywords

Heterogeneous Systems and Networks, Distributed Complex Event Processing

[1]IBM, the IBM logo, ibm.com and WebSphere are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. Java and all Java-based trademarks and logos are trademarks of Sun Microsystems, Inc. in the United States, other countries, or both.

DEBS'10, July 12–15, 2010, Cambridge, UK

1. INTRODUCTION

Complex event processing systems have seen a change of perspective recently. While, orignally, powerful engines and languages were used in a central way to efficiently correlate events and detect situations, the emerging increase in event sources forced the community to investigate into scalable CEP systems by distributing the handling of events.

However, in reality, CEP is not yet fully exploited and is still establishing itself within the business world. Distributed CEP is even one step behind. The reason is that for industrial event processing, not only performance is important. For example, in an e-energy scenario also orthogonal attributes like functionality, expressiveness, reusability of present knowledge, user friendly interfaces, and flexibility are essential. These attributes are not present in current distributed approaches, which focus mostly on an efficient distributed detection of situations (cf. [3, 17, 10, 21, 13, 12]). Therefore, most users still rely on existing, mature complex event technology [1] which is capable of providing the required functionality, at the expense of distribution. Scalability is typically achieved by providing more powerful servers hosting the CEP System (cf. [3]). This, however, is a tremendous disadvantage in scenarios where the applications requirements vary or are even completely unknown upfront. When the requirements, for example the event rates, increase over time, the deployed system's capabilities will eventually reach its limits. Also, many scenarios, like measuring current power consumption within a nationwide or

even continental power grid, are inherently dispersed. A centralized CEP system handling all the sensor data would not only cause a huge processing cost, but cause a major communication overhead alongside.

We believe that in the future, integration of universal, heterogeneous CEP technology will be key technology to allow flexible CEP systems that are capable of adapting to user needs (cf. [18]). The reason for this is that many users rely on different products (because they specialize in some certain criteria) that are not capable to interact. This lies mainly in the fact that there does not exist any generally accepted definition language for complex event processing, though first steps in this direction have been made (cf. [14]). This is a major drawback for many applications, since interoperability and communication is mandatory in todays business world. Business partners are forced to interact, and with the emerge of event driven architectures this also affects CEP systems. Hence, commonly used and accepted communication and event descriptions have to be developed to enable interoperability.
In this paper we present our event processing system, DHEP, which focusses on these aspects, such that business processes can benefit from a distributed processing of events. We introduce a framework that supports the integration of various established centralized CEP systems in a distributed environment. The system has been developed and created in collaboration with the IBM Research and Development Laboratory Böblingen, where it is currently running. We will show that DHEP is a highly scalable complex event processing system, which enables interaction with business processes and context information. It therefore enables an efficient approach for distributed CEP in business contexts.

The rest of this paper is structured as follows. In Section 2 we present the challenges for a distributed heterogeneous CEP system on the basis of our driving e-energy scenario. After presenting the DHEP framework in Section 3 we will show some evaluation results in Section 3.2. Finally, we discuss related work in Section 5 before we conclude the paper and present future work.

2. CHALLENGES FOR DISTRIBUTED HETEROGENEOUS CEP

As already outlined in previous work, there exists a gap between event processing in academia and the industry (cf. [18]). When analyzing today's usage of event processing in large business applications, it can be observed that almost none of the distributed CEP approaches proposed in the scientific literature has made it into industrial applications so far. Instead, a wide range of dedicated, centralized CEP systems are used in various application domains to perform event correlation tasks. Also, it is interesting that current development in these application fields seems to favour investigating on highly efficient clustering mechanisms (cf. [3]) instead of distributing the process and moving the CEP functionality to dispersed locations where it is most efficient.

There are multiple reasons for this tendency. First, these 'distributed but centralized' approaches are customized to perform well when massive event loads can be processed in centralized data centers. Second, especially in business applications the CEP technology is strongly connected to business processes of the company. Thus, in order to perform the processing of the events, access to context information related to business processes is needed and such context information often resides in centralized databases. Third, exploiting CEP in truly distributed business processes is not yet understood to a sufficient extent.

Therefore, the DHEP approach is driven by demands and requirements from distributed business processes where event processing can play an essential role when implementing the required business functionality. In e-energy scenarios, as they are emerging nowadays, the efficiency of large power grids is enhanced by smart meters placed in households from where they send power consumption events to an energy agency. A simplified scenario is depicted in Figure 1. Here, smart meters in the consumers' households send the energy requests to substations owned by energy (transmission) providers. The substations keep track of the currently requested power within their subnet. If the requested power exceeds the provided power within the subnet, the substation requests additional energy from the providers' mains. The provider itself may request additional power either from other providers or from energy producers. On the return path, the provider sends notifications to the smart meters about the available energy.

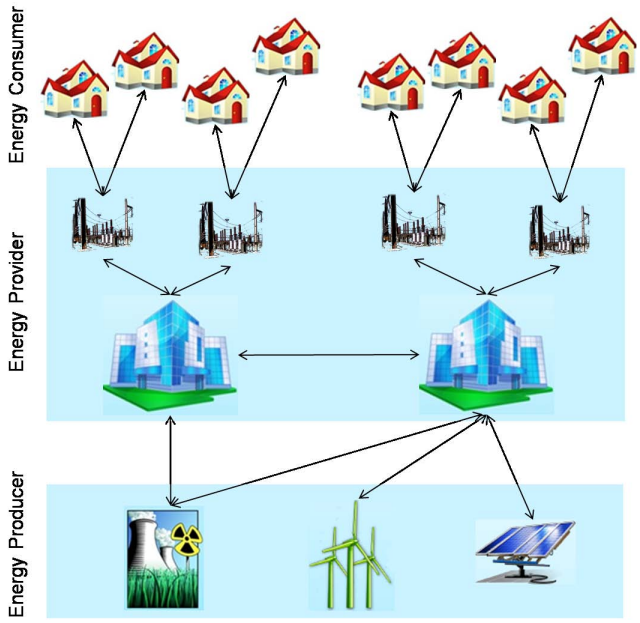


Figure 1: Simple E-Energy Scenario

The need for extensibility, the inherent distribution of energy consumers, along with the hierarchical structure of today's power management (cf. [7, 16]) calls for a distributed CEP solution which imposes specific challenges like the ones listed below.

- *Communication*: The power grid of almost every country contains a large number of substations that need to interact according to the (simplified) power consumption forecast scenario described previously. Therefore, intelligent event emission avoiding unnecessary event

traffic is a must when power consumption shall be forecast reliably in shorter time intervals like once per 15 minutes.
As can be seen in the example, different energy transmission providers, energy consumers and energy producers have to interact in a network. Hence, communication between heterogeneous CEP environments has to be supported. Events and rules have to be exchangeable between the different CEP systems running in the network.

- *Heterogeneity*: The substations within the power grid can have different complexity and tasks. While low-footprint event processing is sufficient in some places, major substations may require more powerful processing capabilities. Therefore, having different kinds of event processing engines within the power grid is important.

- *Purposeful deployment*: Furthermore, the placement and deployment of rules is essential for a distributed CEP system and has special challenges in this kind of setting. For example, domain restrictions and engine constraints may preclude the placement of rules on some network nodes.

- *Context Modeling*: Context information like the structure of the power grid or the power consumption per household is important. Hence, a powerful language for modeling context information is required. In a distributed system, necessary access to context information may be expensive from some places in the network. Therefore, it must be possible that events transport context information and that event processing rules are able to deal with context information.

By providing solutions to these key aspects, our system is a first step to deliver the benefits of distributed event processing systems to industrial event processing solutions.

3. APPROACH

The main idea behind the DHEP system architecture is to enable a distributed event processing without enforcing system users to use a new correlation technology. Hence, our goal is to interconnect existing centralized engines within a network of correlation nodes that is likely to be widely dispersed. As a consequence, we embed these centralized engines within a framework which takes care of all features necessary to build up a distributed processing system, like managing the communication between the nodes or distributing rules. Moreover, since heterogeneous engines are unable to interact without some form of event translation, we propose the concept of a meta-language (cf. [18]). Therefore, DHEP comes along with a powerful modeling language that allows the design of distributed applications. The language enables the use of ontology to design and manage events, rules and context information within the distributed system. Hence, we are able to adapt to a very wide range of possible applications and can use this semantic knowledge to enable efficient complex event processing. Finally, we provide a configuration tool for the application design. On the one hand, it is a graphical editor, where the user is modeling and managing the application. On the other hand, it is responsible for managing the placement and deployment of rules. Figure 2 gives an overview of our system. We will discuss the, the meta language, the configuration tool and the runtime environment, in more detail in the following.

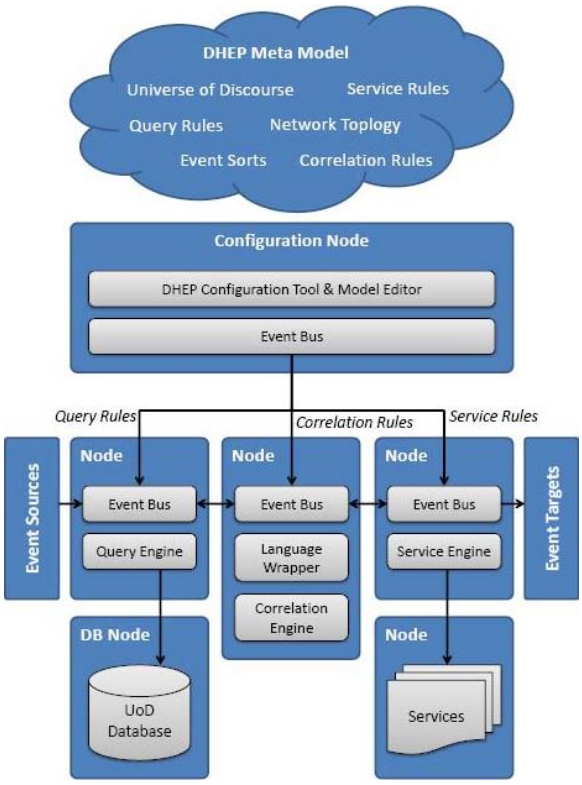


Figure 2: DHEP System Overview

The DHEP system is very flexible. By integrating more nodes into the event processing network, the load of each node can be decreased, making the whole system more robust to a higher event input. By reducing the number of nodes, resources can be saved. Therefore, the network configuration can be adapted depending on the desired overall system behaviour. Additionally, latency can be improved by several means. First, the system can benefit from placing latency-relevant rules on nodes with good network connection. Second, the knowledge about independent rule sets can be exploited: The independent rule sets may be placed on different nodes within the network, thus reducing the overall detection time by using parallel computation.

3.1 The DHEP Meta Language

The DHEP system is based on a meta language for defining the elements of distributed event-based applications that can be implemented according to the DHEP approach:

- an object-oriented context-model defining the information model underlying an event processing application;

- event type definitions allowing to include complex objects in events being transmitted within the system;

- event processing rules for standard event correlation, access to context model information, and invocation of web services;
- event processing nodes within a network; each event processing node can be equipped with different event processing engines;
- deployment descriptions that allow to express which event processing rule shall be executed on which engine on which node in the network.

In order to illustrate the language concepts, a simple object model is required to represent the e-energy example from Section 2. This simplified object model allows to express that power consumption can be metered at smart meters installed at a customer's house (cf. Listing 1). Customers have some contract with an energy agency which sell and bill consumed energy on behalf of energy providers owning the power plants.
The context modeling part of the DHEP language features typical object-oriented capabilities which are powerful enough to represent complex information models like the IEC 61970 standard for modeling power grids as published by the International Electrotechnical Commission (cf. [7, 16]). Standards like this gain more and more importance for the implementation of state-of-the-art industry applications.

object sort *NamedObject*
  attribute *name : String;*
  attribute *description : String;*
object sort *PowerMeter < NamedObject*
  attribute *currentConsumption : Float;*
  attribute *accumulatedConsumption : Float;*
  attribute *parent : SubStation;*
object sort *SubStation < NamedObject*

object sort *Customer < NamedObject*
  attribute *myMeter : PowerMeter;*
object sort *EnergyAgency < NamedObject*
  relation *myCustomers : Customer;*

object sort *EnergyProvider < NamedObject*
  attribute *sellThrough : EnergyAgency;*

Listing 1: Example of Object Definitions

Some typical events that need to be handled include power consumption at a certain meter or power production information provided by a power plant operator (cf. Listing 2). Event Attributes can be both standard attribute types like Integer or String and complex attribute types like defined DHEP object sorts, e.g. a *PowerMeter* defined in Listing 1.

event sort *PowerConsumption*
  field *meter : PowerMeter;*
  field *amount : Float;*
event sort *PowerForecast*
  field *meter : PowerMeter;*
  field *amount : Float;*
event sort *AggregatedPowerForecast*
  field *station : SubStation;*
  field *amount : Float;*
event sort *PowerProduction*
  field *provider : EnergyProvider;*
  field *amount : Float;*

Listing 2: Example of Event Definitions

Rules are the foundation of every situation detection. DHEP Rules (cf. Listing 3) consist of four parts: i) The event pattern (*WHEN*) specifies the events used in the rule as well as their relation. ii) The correlation condition (*IF*) is used to access and verify certain attribute values of the incoming events. iii) The restrictions that have to be met when placing the rule (*RESTRICT*). iv) The action (*EMIT*), which describes the event that will be sent to the network once the situation is detected. Also, the user specifies how the values of the result event are set.
Listing 3 shows an aggregation of two smart meter values. We attach the added *amount* values of the incoming *PowerForecast* events to the resulting *AggregatedPowerForecast* event.
Our rule language supports typical correlation operators like **SEQ**, **ALL**, **OR**, and **NOT** for detecting event patterns. Together with being able to define time constraints for detecting event patterns and thus situations, we are wellpositioned with respect to expressiveness of our rule language [2]. This holds true in particular as we also support sophisticated context model access and service invocations via dedicated rules.

rule *AggregateConsumptionForecasts*
  WHEN *SEQ(pcf1 : PowerForecast,*
    pcf2 : PowerForecast)
  IF *pcf1.meter ≠ pcf2.meter*
  RESTRICT *engine.type = amit*
  EMIT *AggregatedPowerForecast*
    (station = pcf1.meter.parent
    amount = pcf1.amount +
      pcf2.amount)

Listing 3: Example of a Simple Aggregation Rule

As can be seen in the example, deployment restrictions can be formulated (cf. Listing 3). These restrictions influence where a rule can be placed. We currently support two such restriction types. First, every rule has a rule type classification expressing on what type of engine the rule can be processed. Upon deployment this will be used to assure that the rule is only deployed to an engine capable of processing rules of the respective type. Second, the rules can be annotated with restrictions on the computational power required on the machine they shall run on, e.g., attributes like required CPU speed, memory size or network connection bandwidth.

Finally, the event processing network can be defined by a set of event processing nodes (see Listing 4), each of which can be equipped with multiple event processing engines. Furthermore, the user has the possibility to influence the placement by manually deploying rules on specific nodes. However, this is not necessary as we also provide an automatic deployment mechanism (cf. Section 3.3).

[2]Adding more sophisticated correlation operators is no big effort and not so essential to our approach.

node *a correlation node*
correlation engine *AMiT;*

node *a context model server*
query engine *queryEngine;*
deployed rule *rule R1;*
service invocation engine *serviceEngine;*

Listing 4: Example of a Node Model

3.2 The DHEP Runtime Environment

Every node in our network is running the DHEP runtime environment(RE) which is basically a middleware that encapsulates the event processing technology. It abstracts the network functionality and performs all tasks that are necessary to enable a distributed heterogeneous event processing system. Figure 3 shows the most important components of the RE. The dashed path marks the typical event flow we have to handle in the RE. In the following subsections, we describe the components as they are used in the event flow. Also, we discuss the functionality of the Rule Management and Context Enrichment components.

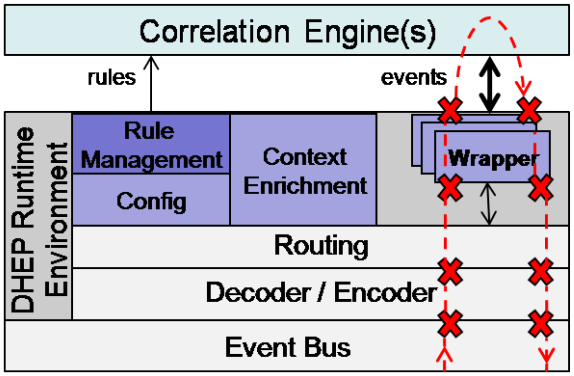


Figure 3: Eventflow in the Runtime Environment

Event Bus

The event bus constitutes the foundation of our system, as it contains all low-level elements which are needed for distributing the data, i.e. events, in our system. Since heterogeneous systems do also include heterogeneous communcation methods, the communication interface of the RE has to be designed in a very modular fashion. Thus, all communication modules implement the same common interface, which provides the *getMessage* method to receive events. Modules are built in a modular fashion making them exchangeable. This enables the integration of multiple heterogeneous message systems and communication principles. Currently, the system supports three modules (see Figure 4): a socket module, a JMS (Java® Message Service™[9]) module, and a message queueing module.

The socket module is based on the well known reactor design pattern (also known as dispatcher/notifier, cf. [19]) and uses non-blocking sockets in order to achieve high scalability. A *Selector* acts as a demultiplexer to determine which connections are ready to have operations invoked on them, without blocking the application process. The message queueing module is responsible for the communication using WebSphere® MQe. It contains a queue manager which manages the *local queue* and connections to *remote queues* [15]. The JMS module enables topic based Publish/Subscribe communication. Therefore, it requires at least one node running in a Java EE environment that hosts the topic queues [9]. For every DHEP event type there exists one topic queue.

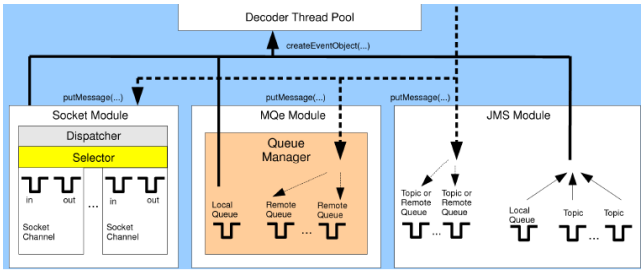


Figure 4: Event Bus & Decoder

Decoder/Encoder

As typical for a model and ontology based system, DHEP has to retrieve the semantic knowledge of every incoming event. Therefore, we deserialize every incoming event and match it against our description model, in order to encapsulate the event within an event object. This happens within the *Decoder*(see Figure 4). To simplify this process and reduce programming overhead in this step, we make use of Dynamic EMF (Eclipse Modelling Framework, cf. [20]). We save our events in a Dynamic EMF format and make use of deserialization methods provided by EMF. Accordingly, the encoder acts vice versa as a serializer which creates event messages from DHEP objects. We use a thread pool within the Decoder component to be able to decode events in parallel. This feature is configurable, as the thread pool is only beneficial if multiple CPUs are installed.

Routing

The routing components receive incoming event objects from the decoder und forwards them according the routing table. The table contains not only the information about the next hop of each event sort in the network, but also about the local engines which are meant to process the event at the node. Hence, the event bus distributes every incoming event accordingly, sending it towards one (or more) of the local engines or other nodes in the network.

Wrapper

The framework architecture has the ability to integrate different existing CEP engines even within the same network. Currently, we support two CEP engines, namely AMiT (IBM Active Middleware Technology™) and JRules (IBM WebSphere ILOG JRules™), as well as a query engine for event context enrichment and a service engine for web service invocation. The wrapper component is responsible for the integration of the different engines. It acts primarily as the adapter between the routing component and an event processing unit, i.e. engine, on the node. A wrapper is required for every different engine, because its main task is to translate from the meta-language into the language of the target engine. Due to a very modular design, a wrapper is in fact

an adapter not only to engines but to any unit that can process and act with events. A new wrapper just has to implement the provided wrapper interface.
We use the wrapper concept also for configuring our system. Therefore, we implemented a configuration wrapper, that receives all system relevant information via configuration events which are distributed via the event bus. The configuration events contain for example routing information and rule placements. The configuration wrapper takes care of the proper settings of the routing table and moves rules to their correct target engines (according to the placement algorithm).

Rule Management

The Rule Management is responsible for the distribution, migration and deployment of rules in our system. It is strongly coupled with the configuration events, since it uses these events to manage the deployment of rules. Rule management offers interfaces to move rules to other nodes as well as to deploy them on a local event processing engine which is connected via one of the wrappers. This deployment process is currently handled by the configuration tool (see Section 3.3). However, we are working on more sophisticated re-organization and optimization mechanisms performed by the rule management component in a distributed manner and tackle the constraints and optimization criteria of our heterogeneous environments.
Similar to the wrappers, the deployment of rules to engines requires a translation process, because rule descriptions made with the DHEP meta language are not understood by the various CEP engines directly. As a result, rule translators are attached to the *Rule Management*. The effort in creating translators is rather small and worthwhile, as we require about 20 lines of code per operator translation.

Event Context Enrichment

In distributed scenarios, access to context information is not always possible or might be very expensive, although it might be needed in the CEP. As a result, context enrichment and transport becomes necessary. Especially in business applications, context information is important since many business decisions can not be made solely by information contained in the events, but require additional information applying to the events. For example, events from a smart meter are very simple and small, they contain an identifier of the customer as well as some value representing the requested energy amount. However, this information is most likely not sufficient for the energy provider which requires additional data about the users contract. This information is typically located at the data center, where all business processes are running. It needs to be added to either events or the event processing system in order to use it within the complex event processing, hence allowing to make correct decisions.

In contrast to centralized approaches, distributed CEP systems did not concern the access to external information so far. The DHEP system however, is providing this feature. For accessing context information in our system, we make use of the modular, heterogeneous architecture by providing a *query engine*, which is capable of enriching events with context retrieved from external sources, such as a database. The configuration of the query engines is done via *query rules* which are also specified in the DHEP language. The transport of context is ensured since all specified objects can be serialized and deserialized within the encoder/decoder. To reduce the expected cost produced by the database access, we implemented a configurable caching mechanism.

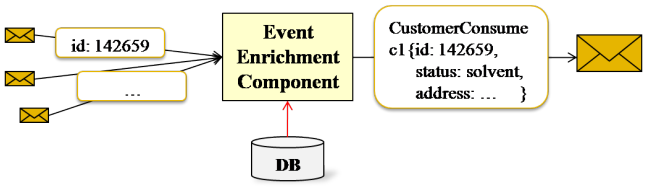


Figure 5: Enrichment of Smart Meter Events

3.3 Configuration Tool

The configuration tool is the meeting point of the DHEP meta language and the actual CEP network, where the user models, deploys and interacts with the application. From the users perspective, the tool is primarily a graphical modelling editor (see Figure 6). However, the main functionality lies in the deployment mechanisms initiated and coordinated by the tool. Therefore, the *Configuration Tool* communicates with the Rule Management component of the framework, using configuration events.

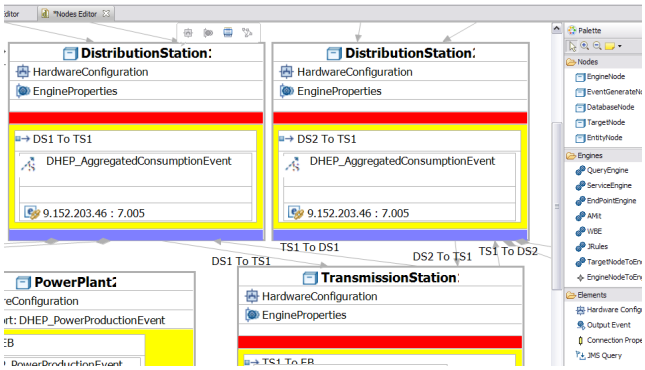


Figure 6: Excerpt of the Graphical Editor

3.3.1 *Placement in Heterogeneous Networks*

Basically, our system is designed to be capable of integrating any placement and optimization algorithm, as the deployment is handled separatly via the configuration tool and events. Also, the Rule Management component described in Section 3.2 provides interfaces to integrate distributed placement algorithms.
However, the placement of rules in a heterogeneous network is not trivial as many constraints have to be fulfilled. As we have presented in Section 3.1, our rules may have restrictions: A rule may not be placed on some nodes, because its restrictions do not match the nodes attributes. Furthermore, every rule causes a resource usage on the node where it is placed. This might result in an invalid placement of a rule not because of a constraint violation, but an exceeding resource usage. As a result of these two properties the deployment algorithms have to ensure, that a placement is valid *iff* both the constraints and the resource requirements of the placed rule are met by the node.

A difficulty arises in heterogeneous systems, as the resource usage of rules is different on the nodes. This is caused by differing resources on the nodes, as well as differing engines, which might require different costs. Basically, every placed rule produces a load which is based on the following parameters: the rule itself (when processing an event), the event rate, the event stream. The latter is important, as depending on the incoming events, the rule might trigger a situation, i.e. a result event, or not.

Figure 3 shows a typical event flow within our system. As can be seen, 7 components are passed, after an event has been received, each contributing to the resource usage: i) Decoder, ii) Routing, iii) Translation, iv) CEP, v) (Back-)Translation, vi) Routing, vii) Encoder.
Hence we can calculate the total resource usage of every event for rule e_r on node n as the sum of the resource consumption of every used component c_i. We use the required processing latency of each component as the *cost* in our function.

$$C(e_r, n) = \sum cost(c_i) \tag{1}$$

This function is in fact a lower bound calculation, as the components v) to vii) are only used if the CEP engine produces a result, and therefore require no resource consumption at all in many cases. However, with this lower bound calculation, we can give a pessimistic forecast, of how many events can be handled by a node. Since Equation 1 determines the required CPU time of one event, we can calculate a rule's maximum resource consumption C_{max} (Equation 2) for a given event rate λ and give a lower bound for the maximum event throughput λ_{max} (Equation 3):

$$C_{max}(r, n) = \lambda * C(e_r, n) \tag{2}$$

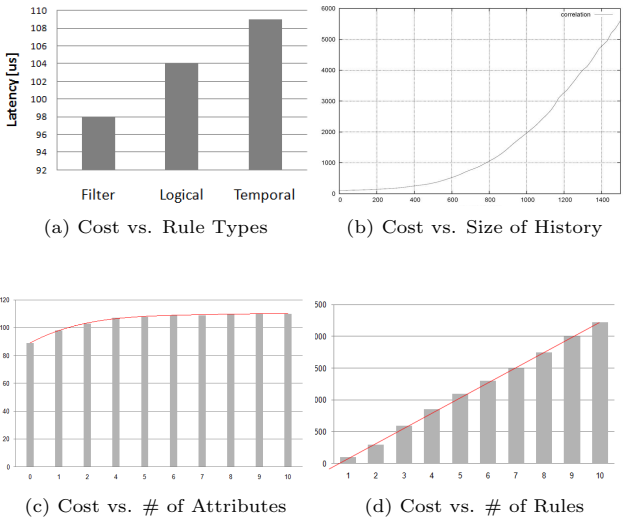
$$\lambda_{max} = \frac{1}{C(e_r, n)} \tag{3}$$

3.3.2 *Correlation Cost Analysis*

As a result from our placement conditions and cost approximations, knowledge about the resource requirements of rules is mandatory. In DHEP, we face two major system parts producing this cost: The framework (i.e. the runtime environment), and the correlation engines. The framework cost is not dependent on the rule itself, but on the event rate coming along with the rule, we will tackle this in the evaluation of our framework (Section 4). However, this is not true for the correlation cost. Here, rule types and rule characteristics have a major influence on the cost.

To understand the engines behavior and resource consumption, benchmarks on the processing cost are necessary which are collected exemplary for the IBM AMiT^{TM}engine (see Figure 7). For the benchmarks, the processing time (latency) between the input of an event to the engine's interface and the eventual output of a result event was measured.

Figure 7(a) shows the latency of different basic rule types: *Filter rules*, *logical rules* and *temporal rules*(like a sequence). The results show that the cost increases with the rule's complexity. The second benchmark shows the impact of specific event streams on the processing time of rules (see Figure 7(b)). This is affecting sequential rules, as they are producing new rule instances. For example, an event stream A^nB produces an instance for every incoming event A, while waiting for the matching event B that triggers the rule pattern $A;B \rightarrow C$. The benchmark shows, that the increase is approximately quadratic. The reason is that every event has to the go through all n previously created rule instances until a new instance is created. This effect causes quadratic runtime for the event stream A^nB.
Figure 7(c) shows the impact of event attributes. As can be seen, the rule's latency increases logarithmic with the number of accessed event attributes. Finally, Figure 7(d) shows the impact of multiple rules that are affected by an event. As can be seen, the latency increases linearly, which shows that the engine processes every rule after the other. However, the cost increase is not an accumulation of the individual rule costs, but the engines scheduling mechanism creates an overhead for every additional rule.



(a) Cost vs. Rule Types (b) Cost vs. Size of History

(c) Cost vs. # of Attributes (d) Cost vs. # of Rules

Figure 7: Correlation Rule Cost Evaluations

4. EVALUATION

The evaluation of the DHEP framework was driven by two guiding questions:
(1) What is the systems's behavior and overhead?
(2) When is the framework applicable?

4.1 Framework Cost Analysis

In order to understand our system's behavior, as well as calculate the benefits of a distributed deployment, we first evaluated how the different components that are used in the standard event flow contribute to the total processing cost of events. The flow was described in Section 2 and shown in Figure 3. In the flow, each event is typically processed in 7 intervals, after it has been received: i) Decoder, ii) Routing, iii) Translation, iv) CEP, v) (Back-)Translation, vi) Routing, vii) Encoder
As can be seen, the intervals 1-3 correspond the incoming events, while intervals 5-7 correspond to outgoing events. Furthermore, interval 4 is independent of our framework and has to be measured seperately. An exemplary benchmark for one engine is presented in Section 3.3.2.

We deployed a small evaluation network: an event source, the processing node, and an event sink. The processing node was a Windows PC running a single core CPU@2.0GHz. IBM's AMiT^{TM}engine was connected via a wrapper to the DHEP Runtime Environment. The event source was simulating a smart meter sending powerrequest events continously. In order to receive consistent, repeatable results, only one filter rule was deployed on the engine, filtering for an id, and thus always resolving to true for every event sent.

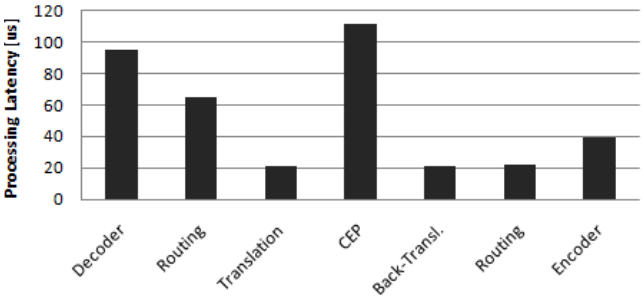


Figure 8: Processing Latency per Component

For the evaluation, we had a very low event rate of 1 event/sec to ensure that there is only one event within the framework at every time. Figure 8 depicts the first results we gained. Several results are remarkable: First, the correlation (i.e. the filtering) has the highest processing cost. Second, the wrapper concept proves to be reasonable since translation of events is pretty fast compared to other tasks. Third, deserialization requires a lot of processing cost. In fact, the cost for retrieving and generating a DHEP object for an incoming event is almost as high as the filtering process of the engine. We conclude from this fact, that the EMF's serialization methods we use are not as efficient as we hoped and leave room for optimizations.

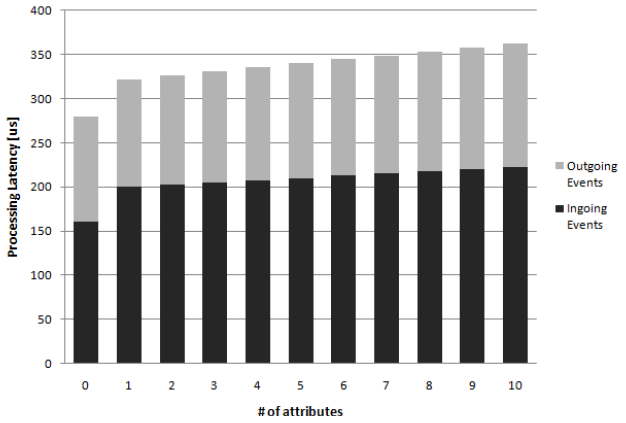


Figure 9: Framework Cost vs. Event Size

As a second evaluation, we varied the events produced from our event source in their size. We did this by adding additional attribute value pairs in the same settings. The results of our second framework cost evaluation is shown in Figure 9. As can be seen, with an increasing size of the events, here the number of event attributes, the framework cost increases too. The reason for this can be found in the additional overhead that is caused during the (de)serialization and translation of the event within the wrapper component. However, it can be seen that the cost increase is rather small and linear: About 1.5% increase for each additional attribute of the ingoing and the outgoing events. The only exception is the low cost of an attribute-less event, which is based on the fact that both (de)serialization and translation do not access the attribute list of the event at all. However, an event without any attribute is rather unusal in complex event processing scenarios. Therefore this effect only has a minimum influence.

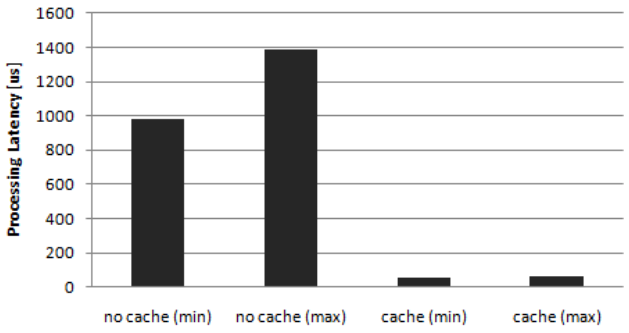


Figure 10: Event Context Enrichment Cost

4.2 Event Enrichment via Query Engine

In our next evaluation we tackled the access of context and enrichment of events with the retrieved information. This has been done with the query engine we briefly introduced in section 3.2. For the measurements, we used our basic network setup, and added a mySQL database to the processing node. We then created a rule, that should retrieve some integer value from the database and add it to the input event. The results of our processing cost measurements are depicted in Figure 10. As can be seen context access produces a massive overhead to the system. The costs for accessing a database and querying for a value and adding the value to an event are about ten times higher than filtering for an attribute.

The expense of event enrichment can be reduced by enabling the caching mechanism as it is described in Section 2. As can be seen in the results, this can be a great benefit for applications, where the same attribute is retrieved for a large number of events. Furthermore, when caching comes into consideration, other factors like cache size and freshness of data shold have a major influence on the cache behavior. Also, a type specific caching can further improve the caching mechanisms. However, these parameters are mostly application dependent and require further investigation.

4.3 Advantages of a Distributed Deployment

Based on the previous evaluations (cf. Section 3.3.2 and 4.1), we are able to understand how different attributes are influencing the processing latency of our system. As has been shown, the systems overall processing cost is caused by both the framework and the used CEP engine. Furthermore, we can state, that the framework costs are almost constant, whereas the cost for processing an event in a CEP engine are dependent on the used engine and the deployed rule set. In our last evaluations, we want to show the scalability of DHEP: by adding nodes to the network we are able to process an increasing event load.

We tested our framework with a simple deployment of our e-energy scenario which is depicted in Figure 11: Smart meters calculate the current energy consumption of a household and send periodically consumption events to the energy provider. The energy provider first checks, whether the power consumption exceeds a predefined threshold. This is done with a simple *filter* rule. Afterwards, the exceeding consumption events are aggregated (cf. rule definition in Section 3.1).
We chose this simple scenario because of three reasons: i) The deployed rules are the most basic rules one can get for a CEP engine. Therefore, we expect the correlation process to be very fast. This favors the CEP engine, and stresses our framework, because a high event throughput can be reached by the correlation process. ii) By setting the filter values we can adapt the event rates that are used within aggregation rule. iii) With a low number of rules, the centralized deployment is in clear favor, because there is not much space for splitting and distributing the rule set.

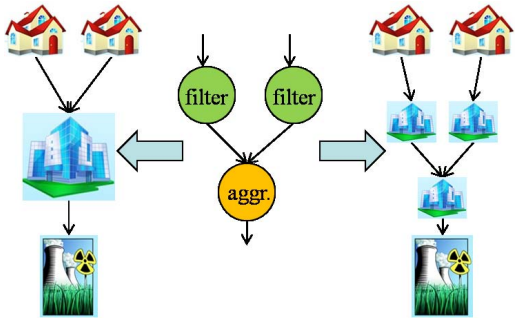


Figure 11: The two different deployment scenarios

We deployed our rule set on a centralized CEP engine, and compared the results with a decentralized deployment of the rule set, as it is depicted in Figure 11. The results of the evaluation are shown in Figure 12. As can be seen, we deployed the scenario with three different filter settings, where we changed the restrictiveness of the filters. Moreover, we used the evaluation to verify our approximation functions presented in Section 3.3.1 (cf. the lighter bars in Figure 12). We approximated the values for the various component costs from our evaluation results presented before.
The results show three main conclusions: First, the more restrictive the filters are, the more events can be processed in both settings. This is self-evident, as less events are reaching the aggregation rule, hence producing less cost. Second, our lower bound approximations are pretty close to the actual values we got from our evaluations. Third, we benefit from a distributed deployment of rules although we deployed a scenario that is likely to favor a centralized one. Furthermore, the benefit increases, as the filters get more restrictive. The reason is, that the aggregation rule slows the system and is the bottleneck in a deployment wher no events are filtered out (0% Filter). In more realistic scenarios, where the events are filtered out, the aggregation receives less events. Hence, the benefit in distributing rules increases.

As can be seen, although our system produces additional costs, the advantage in distributing rules over multiple nodes becomes apparent even in very simple scenarios. The advantage gets even more obvious, when bigger and more realistic business scenarios are deployed or when additional sources

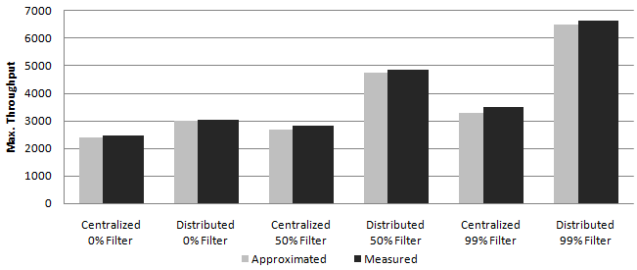


Figure 12: Maximum Event Throughput

have to be processed by the CEP system. The increase of CEP costs might exceed the available processing power and network bandwith of centralized CEP systems. With our framework, we can easily extend our setup by adding additional processing nodes to reduce both processing cost and network bandwith usage of the CEP system.

5. RELATED WORK

Since we combine in DHEP the expressiveness and power of mature CEP systems running on large servers, typically used nowadays, with the ideas of distributed CEP systems, a look in both directions is necessary.

While initial research focused on expressive languages and efficient centralized situation detection mechanisms (cf. [1, 6, 8]), academia has shifted its main focus on distributed CEP. Multiple approaches have been designed recently (cf. [10, 12, 13, 17, 11]). Especially with the increasing popularity of Publish/Subscribe systems (cf. [22, 4]), event processing has become a highly distributed technology. CEP scales well in these systems, since it can benefit from the Publish/-Subscribe paradigm by subscribing to specific happenings of interest, which can extended to support complex events. A CEP engine attached to the Publish/Subscribe system takes care of the detection of complex events. Furthermore, the possibility to detect complex events on any node in the system opens a lot of optimization possibility, such as moving functionality closer to the source in order to reduce the overall network usage.
While we can learn from these approaches in terms of efficient network usage and rule migration, they lack the special industrial requirements we have defined in Section 2. They consider homogeneous engines for their migration and deployment techniques. Furthermore, they do not integrate context information of applications. Finally, distribution techniques do not seem to be mature enough to be deployed in business applications. They take only static node resources into account, but ignore important aspects for business application, such as dynamic load, additional restrictions for the processing of rules, security aspects, and the proximity to context data sources that are necessary for rule processing.
Attempts to interconnect heterogeneous components within a common distributed platform can be found in the context of service oriented architectures in form of the SCA or JBI specifications (cf. Service Component Architecture [2], Java Business Integration [23]). Similar as in DHEP, different services can interact and communicate with each other based on a common communication bus. Adapters

are used to translate between the internal languages of the services and the communication language. However, the focus in these systems is to interconnect existing functionality encapsulated in service instead of distributing application parts among various configurable nodes.
First steps towards increasing the performance of CEP systems by combining heterogeneous correlation technology has been made by Chakravarthy et al.[5]. They manually combine a classical event processing engine and a stream processing engine in order to achieve an efficient situation detection.
Most closely to our goal in maintaining the scalability of mature correlation technology is the work of Biger et al.[3]. They integrate several centralized CEP systems into a distributed correlation network without modifying the system itself. In addition to connecting input and output of individual powerful correlation nodes and deciding which correlation tasks to perform at which node, the configuration and deployment of each correlation machine becomes a main issue. However, the approach has several characteristics contradicting our goal. First, the network setup as well as the CEP configuration has to be tailored towards the target application beforehand. Second, the intention of stratification is to achieve a maximum throughput by splitting a specific set of rules and pipelining it through multiples stratas. It is not intended for inherently distributed applications like the e-energy scenario.

6. CONCLUSION

In this paper we presented the complex event processing system DHEP, which has been designed and created to close the gap between current CEP systems and business requirements. The concepts behind DHEP focused on providing a very modular architecture, that comprises various different event processing engines, and enables communication among them within a distributed system. Moreover, DHEP comes along with a powerful object oriented definition language, that enables efficient, tool-aided designing of big industrial CEP applications. The evaluation results show that, although the functionality provided by DHEP imposes additional cost, the system scales well by exploiting distributed detection of situations.
Our future work will include experiments with larger scenarios, where we will integrate multiple heterogeneous correlation engines, such that DHEP users can benefit from using many diverse systems. Furthermore, we will design and practically evaluate a variety of placement algorithms in large scale scenarios.

7. REFERENCES

[1] A. Adi and O. Etzion. Amit - the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.

[2] M. Beisiegel et al. Sca service component architecture - assembly model specification, version 1.0. Technical report, Open Service Oriented Architecture collaboration (OSOA), 2007.

[3] A. Biger, O. Etzion, and Y. Rabinovich. Stratified implementation of event processing network. In *Distributed event-based systems (DEBS)*, 2008.

[4] J. A. Briones, B. Koldehofe, and K. Rothermel. Spine: Adaptive publish/subscribe for wireless mesh networks. *Studia Informatika Universalis (Hrsg)*, 7(3):320–353, 2009.

[5] S. Chakravarthy and R. Adaikkalavan. Events and streams: harnessing and unleashing their synergy! In *Proc. of the second international conference on Distributed Event-Based Systems (DEBS)*, 2008.

[6] S. Chakravarthy and D. Mishra. Snoop: an expressive event specification language for active databases. *Data Knowl. Eng.*, 14(1):1–26, 1994.

[7] CIM. Current cim model. http://cimug.ucaiug.org, March 2009.

[8] S. Gatziu and K. Dittrich. Samos: an active object-oriented database system. *Data Engineering*, 15:23–26, December 1992.

[9] M. Hapner et al. Java message service. Technical report, Sun Microsystems Inc., 2002.

[10] G. Jakobson. The technology and practice of integrated multiagent event correlation systems. *Integration of Knowledge Intensive Multi-Agent Systems*, pages 568–573, 2003.

[11] G. G. Koch, B. Koldehofe, and K. Rothermel. Cordies: Expressive event correlation in distributed systems. In *Proc. of the 4th Int. Conf. on Distributed Event-Based Systems (DEBS)*, 2010.

[12] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Proc. of the Int. Conf. on Middleware*. Springer-Verlag New York, Inc., 2005.

[13] Y. Liu, I. Gorton, and V. K. Lee. The architecture of an event correlation service for adaptive middleware-based applications. *J. Syst. Softw.*, 81(12):2134–2145, 2008.

[14] D. Luckham and R. Schulte. Event processing glossary, July 2008.

[15] B. Mann. Providing a backbone for connectivity with soa messaging. Whitepaper, IBM Websphere, 2009.

[16] A. McMorran. An introduction to iec 61970-301 & 61968-11: The common information model. cimphony.org/cimphony/cim-intro.pdf, 2007.

[17] P. R. Pietzuch, B. Shand, and J. Bacon. A framework for event composition in distributed systems. In *Proc. of the 4th Int. Conf. on Middleware*, 2003.

[18] B. Schilling, U. Pletat, and K. Rothermel. Event correlation in heterogeneous environments. *it - Information Technology*, 5:270–275, 2009.

[19] D. C. Schmidt. Reactor - an object behavioral pattern for demultiplexing and dispatching handles for synchronous events. In *Proceedings of the First Pattern Languages of Programs conference*, 1994.

[20] D. Steinberg, F. Budinsky, M. Paternostro, and E. Merks. *EMF: Eclipse Modeling Framework*. Addison-Wesley Professional, 2008.

[21] W. Tai, D. OSullivan, and J. Keeney. Distributed fault correlation scheme using a semantic publish/subscribe system. *Network Operations and Management Symposium (NOMS), IEEE*, pages 835–838, 2008.

[22] M. A. Tariq et al. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Int. Conf. on Parallel Computing (Euro-Par)*, 2010.

[23] R. Ten-Hove and P. W. (editors). Java business integration (jbi) 1.0 final release. Sun Microsystems, USA, 2005.