CREAM: An Infrastructure for Distributed, Heterogeneous Event-based Applications

M. Cilia*, C. Bornhövd**, A. P. Buchmann

Databases and Distributed Systems Group, Department of Computer Science
Darmstadt University of Technology - Darmstadt, Germany
<lastname>@informatik.tu-darmstadt.de

Abstract. Applications ranging from event-based supply chain management to enterprise application integration and pervasive computing depend on the timely detection and notification of events. We present CREAM the event-based reactive component of the DREAM middleware platform. Here we address four key issues in distributed and heterogeneous environments: event detection and notification, event composition, an active functionality service, and ontology support. We show the need for ontology support at all levels in heterogeneous environments and present a distributed active functionality service that addresses the difficult issues of event composition in widely distributed environments. We illustrate the practicality of the proposed approach through two prototypes that are based on this infrastructure: a meta-auction service and a personalized service offering in Internet-enabled vehicles.
Keywords: event-based applications; event handling; publish/subscribe; concept-based addressing; data integration; business rules.

1 Introduction

Application are moving away from tightly-coupled systems towards systems of loosely-coupled, dynamically bound components. This trend fits the event-based application paradigm which is well suited for integrating autonomous, heterogeneous components into complex systems by means of detecting and exchanging events. Since event-based systems do not require a-priori knowledge about the consumers of events they are easy to evolve and scale.

However, the exchanged events encapsulate data about a given happening of interest, which can only be properly interpreted and used when sufficient context information is known. In traditional, centralized systems this context information is typically known by the users and left implicit. It is normally lost when data and events are exchanged across component or institutional boundaries. To process events in a semantically meaningful way, explicit information about the semantics of events and data is required. Moreover, event-based systems use an event dissemination mechanism, such as a publish/subscribe mechanism, which allows for asynchronous communication. Producers and consumers must share a common understanding in order to express their mutual interests.

* also Faculty of Sciences, UNICEN, Tandil, Argentina
** IBM Almaden Research Center, USA. e-mail: cborn@us.ibm.com

The reaction to events on the application-side represents part of the business processes and, in general, is hard-coded. Since the domain knowledge is scattered and hard-wired into applications it has been difficult to adapt to new requirements quickly.

CREAM (Concept-based REActive Mechanism) is the reactive component of DREAM [1], a flexible middleware platform for developing open distributed and heterogeneous event-based applications. CREAM supports from the ground up ontologies that provide the base for correct data and event interpretation. Rather than requiring every producer or consumer to use the same homogeneous namespace (as is common in other pub/sub systems) we provide metadata and conversion functions to map from one context to another. On top of it a higher level addressing model for event dissemination is proposed. Event-triggered business rules can be explicitly defined and managed to adapt to new business requirements. For instance, modern large-scale applications, such as e-commerce, event-based supply chain management (ESCM), Internet or Intranet applications, enterprise application integration (EAI), and emerging pervasive systems, can effectively benefit from this infrastructure.

The power of CREAM is illustrated with the help of two case studies for which prototypes have been built: a meta-auction application and the personalization of car and driver portals in Internet-enabled vehicles. Examples presented throughout this paper are related to these scenarios.

The remainder of this paper is organized as follows. In Section 2 related work in the four main areas contributing to our event-based middleware platform is presented. Section 3 provides an overview of the proposed approach with additional detail presented in the corresponding subsections. An outline of the implementation of the infrastructure and two case studies built on top of it is given in Section 4. Conclusions and future work are presented in Section 5.

2 Related Work

The work presented in this paper involves the following areas: event dissemination, complex event detection, rule processing and data/event integration. Problems in individual areas have been solved for homogeneous and centralized environments but are much harder in heterogeneous, distributed environments where they have not been solved yet. Due to space limitations it is not possible to provide a more detailed discussion of related work. Therefore, the reader is pointed to [2,3] for additional discussion of related research.

2.1 Event Dissemination

In distributed environments events must be propagated to all interested consumers. For this purpose, event notification services, or notification services for short, are widely used. In CORBA an event service [4] was introduced to provide a mechanism for asynchronous interaction between CORBA objects. Here, an event channel acts as a mediator between suppliers and consumers of events. To

overcome deficiencies of this service specification, the notification service [5] was proposed as a major extension with support for quality of service specifications and basic event filtering.

The Java Message Service (JMS) [6] provides the Java technology platform with the ability to process asynchronous messages. JMS was originally developed to provide a common Java interface (API) to legacy Message Oriented Middleware (MOM) products. This API brings portability of Java code which facilitates the replacement of the underlying messaging service without affecting existing code. JMS provides two models for messaging among clients: point-to-point (using a queue) and publish/subscribe (by means of topics). JMS was incorporated as an integral part of the Enterprise Java Beans (EJB) component model in the EJB 2.0 specification by defining a new bean type, known as message-driven bean (MDB). This new bean acts as a message consumer providing asynchrony to EJB-based applications.

In the past few years, publish/subscribe mechanisms have got more attention because they offer loosely coupled exchange of asynchronous notifications, facilitating extensibility and flexibility. The channel model has evolved to a more flexible subscription mechanism, known as subject-based, where a subject is attached to each notification [7]. Subject-based addressing features a set of rules that defines a uniform name space for messages and their destinations. This approach is inflexible if changes to the subject organization are required, implying fixes in all participating applications.

In order to improve expressiveness of subscriptions the content-based approach was proposed where predicates on the content of a notification can be used for subscriptions. This approach is more flexible but requires a more complex infrastructure [8]. Many projects in this category concentrate on scalability issues in wide-area networks and on efficient algorithms and techniques for matching and routing notifications to reduce network traffic [9,10,11]. Most of these approaches use simple Boolean expressions as subscription patterns and assume homogeneous name spaces.

2.2 Detecting Composite Events in Distributed Environments

The approaches mentioned in the previous section do not consider event composition. That means that they filter event notifications trying to deliver events of interest to consumers without considering any correlation with other event occurrences. Event composition involves the occurrence of two or more events. Composite events are expressed using an event algebra, such as those defined in HiPAC [12], or Snoop [13]. Such algebras require an order function between events to apply event operators (e.g. sequence), or to consume events. To determine which of these events should be consumed or selected, different consumption modes were defined [14]. Usually, events are timestamped to provide a time-based order with the purpose of facilitating event selection. However, in open distributed environments global time is not applicable.

An approximation for modeling the time imprecision in distributed systems has been proposed [15], which is known as the *2g-precedence* model. Since an up-

per bound to the precision is assumed, this model is not appropriated for wide area networks and open distributed systems. In [16] an approach for timestamping events in large-scale, loosely coupled distributed systems is proposed. This uses accuracy intervals with reliable error bounds for timestamping events that reflect the inherent inaccuracy in time measurements.

Schwiderski [17] adopted the 2g-precedence model to deal with distributed event ordering and composite event detection. She proposed a distributed event detector based on a global event tree and introduced 2g-precedence-based sequence and concurrency operators. However, event consumption is non-deterministic in the case of concurrent or unrelated events. Additionally, the violation of the granularity condition (2g) may lead to the detection of spurious events.

Many projects on event composition in distributed environments such as [18,19,20] either do not consider the possibility of partial event ordering or are based on the 2g-precedence model. Therefore, they suffer from one or more of the following drawbacks [16]: they do not scale to open systems, they provide the possibility of spurious events, or they present ambiguous event consumption.

Systems that support composite events must also address the semantic issues associated with processing composite events. For example, how timestamps are generated and the way in which events are selected and consumed.

2.3 Active Functionality

Once simple or composite events are detected a proper reaction must be performed. Reactive mechanisms were introduced in the late '80s in the form of Event-Condition-Action rules (ECA-rules) in active databases (aDBMS) [21]. The goal of active databases was to avoid unnecessary and resource intensive polling in monitoring applications where events are detected as changes to a database and the application reacts to the occurrence of these events.

Active functionality developed for a particular DBMS became part of a large monolithic piece of software (the DBMS). Active functionality tightly coupled to a concrete DBMS hinders its adaptation to today's Internet applications, such as e-commerce, where heterogeneity and distribution play a significant role but are not directly supported by traditional (active) database systems [22]. Another weakness of tightly coupled aDBMSs is that active functionality cannot be used on its own without the full data management functionality. However, active functionality is also needed in applications that require no database functionality at all, or that require only simple persistence support. Consequently, active functionality needs to be offered as a separate service that can be combined.

The unbundling of active databases consists of separating the active part from active DBMSs and breaking it up into components providing services like event detection, rule definition, rule management, and execution of ECA rules on the one hand, and persistence, transaction management and query processing services on the other [22]. Afterwards, only necessary components can be rebundled in order to provide the required functionality. A separation of active and conventional database functionality would allow the use of active capabilities depending on given application needs without the overhead of components

that are not needed. Various projects like C^{2}offein [23], FRAMBOISE [24], and NODS [25] have followed this approach.

Unbundling in this context means to give up the "closed world" assumption that traditionally underlies a DBMS and therefore its applicability in an open distributed environment is questionable. This is because of the inherent characteristics of such environments that impose new requirements that were not considered in centralized environments, such as the lack of global time, independent failures of nodes or communication channels, message delays, etc. The consideration of these characteristics has an enormous impact on the event detector [16], which is the essential component of an aDBMS [26]. In [27] crosseffects and potential incompatibilities arising from the combination of selective features of active, real-time and distributed object systems are discussed.

2.4 Heterogeneity

The need for additional semantic metadata for the exchange of data or messages among independent applications or services has been clearly identified, not only in the context of B2B frameworks like ebXML [28], BizTalk [29], or RosettaNet [30] but also by the W3C in efforts like Semantic Web [31], or DAML+OIL [32].

In the first case, XML [33] and XML Schema [34] are used to define common vocabularies to describe data and business processes. XML tags may be explicitly defined in a XML Schema and can be used to give hints about the assumed meaning of the represented data. XML Namespaces [35] allow to contextualize XML tags in the sense of distinguishing different meanings of the same tag name.

In the context of W3C's Semantic Web initiative RDF [36] and RDF Schema [37] are used to provide additional semantic metadata to better enable computer and users to exchange and integrate data. RDF provides an infrastructure that supports the representation and exchange of structured metadata to describe Web resources, like (parts of) Web pages, or other RDF metadata. RDF allows the description of properties of and interrelationships among those resources in terms of ⟨resource, attribute, value⟩ triples. The attributes used can be declared in RDF Schemas which, similar to XML Schemas, give information about their intended meaning, and specify restrictions on their values. RDF Schemas and XML Schemas can play a role similar to ontologies as a common semantic basis for data and metadata representation.

In our framework we use the MIX model [2,38] for the representation of event content. Like XML/XML-Schema or RDF/RDF Schema MIX provides a flexible representation model for data plus additional metadata based on a common domain-specific vocabulary. However, in addition to the functionality provided by the data models discussed above, MIX directly supports data integration by making the concept of semantic context (i.e., the explicit description of implicit assumptions about the meaning of the data) and conversion functions (which allow the automatic conversion of data/events from different sources to a common context) first class citizens of the model itself. MIX should not be seen as an alternative to the models being developed in the context of the W3C but as

a complementing approach that provides features that hopefully will find their way into the other XML-based models and standards.

3 Our Approach

CREAM provides a middleware platform for distributed, heterogeneous event-based applications. Such middleware requires event handling, support for integration of heterogeneous data and events, monitoring capabilities, reaction to events and of course an event dissemination service. Our approach is based on an ontology infrastructure which is the key to achieve our goal.

Events are described with ontology concepts and are augmented with additional context information allowing in this way their correct interpretation outside the boundaries of an event source. On this basis, a concept-based notification service is proposed with the objective to provide event producers and consumers with a common level of abstraction to describe their interests.

More and more, event-based applications need to detect complex situations based on the event stream. For this purpose, a complex event detection mechanism that takes into consideration issues related to open distributed environments is proposed. ECA-rules are incorporated to avoid the definition of reactions to simple and complex events in the form of hard-wired code in the applications. Again, these high-level (business) rule definitions are based on the ontology providing the ability to define them using the most adequate domain specific language without affecting the active mechanism underneath. This active mechanism relies on a service-based architecture where elementary ECA-rule services are composed according to rule definitions. These elementary services are able to interact with external systems or services but always taking into account the data/event assumptions of the system they interact with.

3.1 Ontology Support

The event-based approach carries the potential for integrating autonomous, heterogeneous components into complex systems by means of exchanging events. These events encapsulate data about a given happening of interest, which can only be properly interpreted and used when sufficient context information about its intended meaning is known. In general, this context information (at least the larger part of it) is left implicit and as a consequence is lost when data/events are exchanged across institutional or system boundaries. For this reason, to exchange and process events from independent sources in a semantically meaningful way, explicit information about its semantics in the form of additional metadata is required.

Our infrastructure is founded on the use of shared concepts expressed through common ontologies [39,40,41,42]. By *concept* we understand an abstraction of characteristics common to a set of real world phenomena. By associating a specific concept with a data object we describe the correspondence between the data and the respective real world phenomena.

Depending on the application domain at hand, ontologies as they are used in our infrastructure, can be obtained by negotiation between a small set of

companies (like in the case of EDI), by a consortium responsible for providing standards for a given domain (e.g., the Unicorn standard [43] for travel data), or by formalizing existing commonly used vocabularies (e.g., for stock trading) by a service provider. It is important that ontologies, other than database schemas for example, are source-independent and need to be extensible to be useable in real-life situations.

We represent events, or to be more precise event content, using a self-describing data model called MIX [2,38]. MIX refers to concepts from a domain-specific ontology to enable the semantically correct interpretation of event content. Simple attributes of an event are represented as triplets of the form $\langle C, v, \$\rangle$, with C referring to a concept from the underlying ontology, v representing the actual data value, and $ providing a set of additional meta-attributes (also known as the *semantic context* of v) to make implicit modeling assumptions explicit. The semantic context specifies the interpretation context of a data value and is also represented as MIX data objects. For example, the fuel level of a gas tank can be represented as $\langle$*FuelLevel, 20,* $\{\langle$*VolumeUnit, "Liter"*$\rangle, \langle$*ScalingFactor, 1*$\rangle\}\rangle$.

Complex data objects are represented in MIX as $\langle C, \mathbb{A}\rangle$ pairs, where C refers to a concept from the common ontology, and $\mathbb{A}$ provides the set of simple or complex sub-objects that represent its attributes. These attributes are divided into those that are mandatory, and additional attributes that are optional. Identifying attributes, which are used, similar to key attributes in the relational model, to identify an object of a given concept have to be mandatory attributes. For example[1], a PlaceBid event can be represented with a complex semantic object as follows:

CSO = <PlaceBid, {<ParticipantId, 412, {<IdentifierCode,"eBayCode">}>,
<ItemId, 5423, {<IdentifierCode, "eBayCode">}>,
<BidAmount, 99, {<Currency, "USD">, < Scale, 1>}>,
<ParticipantType, "Gold">,
...} >

In the following, we refer to events represented in the MIX model, i.e., based on concepts from the common ontology and enhanced by additional context metadata as *semantic events*. Semantic events from different heterogeneous sources can be integrated by converting them to a common semantic context using conversion functions. Conversion functions can be specified in the underlying ontology if they are domain-specific and application-independent. Application- or service-specific conversion functions may be defined and stored in an application-specific conversion library [38].

As depicted in Fig. 1, we use ontologies at three different levels of abstraction:

- *Basic Representation Ontology*: defines concepts like Integer or String for the platform-independent representation of data. It is domain-independent and provides the basis for the higher levels of the ontology.

[1] Mandatory attributes are underlined for presentation purposes.

– *Infrastructure-specific Ontology*: contains the concepts needed for describing the infrastructure, i.e., event hierarchy, time notions, and notifications.
– *Domain-specific Ontology*: provides all the concepts needed for a particular domain. For example, PlaceBid, BidAmount, etc. for auctions and FuelLevel, VehicleStatus, GetInto, etc. for the Internet-enabled car.

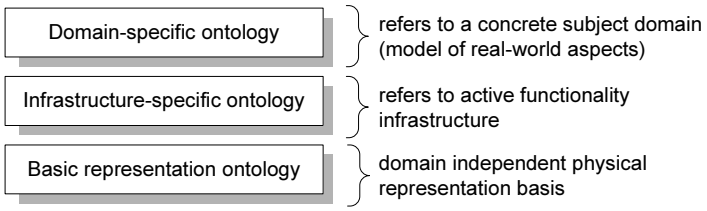

Fig. 1. Three abstraction levels of ontology concepts

Concept definitions from the last two levels are associated with a physical representation by inheriting from an appropriate concept of the basic representation ontology.

3.2 Events and Notifications

An *event* is a happening of interest. Events coming from diverse sources must be mapped to the common vocabulary. This is basically the task of *event adapters*. These components convert source-specific events into their corresponding concepts of the ontology augmented with semantic contexts. Event adapters deliver a semantic event. The association of context information with events serves as an explicit specification of the implicit assumptions made by the event source. Without this additional information the event content cannot be correctly interpreted once the event leaves the source boundaries. Based on the explicit description of the underlying context these semantic heterogeneities can be resolved by converting the data to a common context using appropriate conversion functions. As mentioned before, this common context is specified by the consumer of the semantic event. For instance, consider the placement of a bid that is generated at an American auction site. This happening is then mapped into the PlaceBid concept and the assumptions about the data involved are attached in the form of semantic context. Taking a closer look at one of its attributes, e.g. the bid amount, it is augmented with USD as currency in order to be correctly interpreted outside this particular auction site. PlaceBid consumers can specify the currency of interest, e.g. Euro, as the target context. This conversion is automatically done by the ontology support.

A *notification* is a message reporting a semantic event to interested consumers. A notification carries not only data about the event itself but also important operational data, such as detection time, event source, time-to-live, etc. Concepts related to notifications (e.g. Notification, OperationalData, DetectionTime, EventSource, TimeToLive) are specified as part of the infrastructure-specific

ontology. On the other side, concepts related to the content of semantic events should be specified in the corresponding domain-specific ontology.

3.3 Event Dissemination

A *notification service* based on the publish/subscribe paradigm is responsible for delivering events to interested consumers. Here a notification flows from an event producer to one or more consumers. Consumers place a standing request for events by subscribing. A publisher makes information available for its subscribers. A publish/subscribe mechanism provides asynchronous communication, it naturally decouples producers and consumers, makes them anonymous to each other, allows a dynamic number of publishers and subscribers, and provides location transparency without requiring a name service.

In order to provide a higher level of abstraction to describe the interests of publishers and subscribers, *concept-based addressing* is proposed for our framework. Since semantic events are represented with concepts of the ontology, consumers can benefit from this situation and can specify their subscription patterns by also using the underlying ontology. In this way, consumers do not need to take care of proprietary representations and all participants use a common vocabulary not only for its physical and structural representation but also for expressing their interests. In addition, publishers do not require to specify additional information for event dissemination since the destination of notifications is determined by self-contained information.

The delivery of notifications to consumers is the responsibility of the delivery mechanism. Consequently, there is a correspondence between the concept-based approach and the addressing model (i.e. content-based, subject-based) of the underlying delivery mechanism. The latter is responsible for using efficiently the resources involved (network bandwidth, size of routing tables, etc.)[2] while the concept-based approach is responsible of providing a common level of abstraction for producers and consumers.

In our current implementation the concept-based addressing is built on top of a commercial delivery mechanism that uses the subject-based addressing model. Before getting into more details about the mapping between these two models, subject-based addressing needs to be introduced. Subjects define a uniform name space for messages and their destinations. A subject is associated with each notification. Subject names consist of one or more elements (usually a string) organized in a tree by means of a dot notation. Subjects are used to direct notifications to their destinations. Therefore, notifications need to have attached a specific subject which is a path on the subject tree e.g. NEWS.SPORTS.BASKETBALL. Subscribers could also use wildcards for subscription purposes (e.g. NEWS.SPORTS.*).

In our prototype the subject name space is organized in two main parts. The first one is to provide control of the destination of notifications (if needed). This

[2] It is out of the scope of this paper to discuss routing algorithms and efficient use of the brokers' resources. For a detailed discussion see [44].

control part is used to concatenate services in the service chain (more details in Section 3.6). The second part is used to capture the content of the semantic event in question. For this purpose, the subject for a semantic event is derived from its identifying attributes. This mapping is done by flattening the semantic event into a subject structure. The first position of the subject is used for its concept name and subsequent positions are used to locate the value of identifying attributes by traversing the semantic event. Notice that both parts of the subject organization are configurable. That is, the number of fields that form part of the control as well as the depth of the traversing algorithm for mapping event content into a subject can be configured. All this information and the name space organization is maintained in a repository.

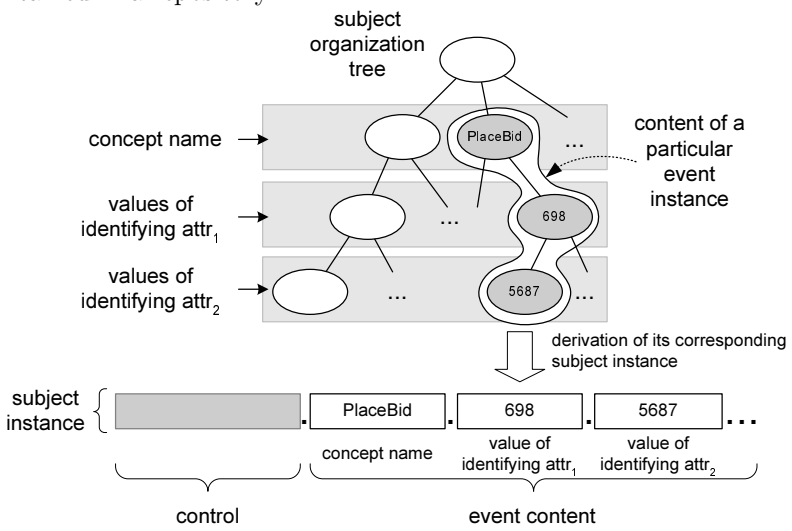

Fig. 2. Subject organization and subject instance derivation

Figure 2 shows how the content of a semantic event is mapped into the subject model and how a particular subject instance is derived. The obtained subject instance is attached to the notification that corresponds to the event in question and then the notification is passed to the underlying delivery mechanism.

3.4 Composite Event Detection

Composite events involve the occurrence of two or more events. The component events can be simple events or may be composite events themselves. Composition is described through an event algebra. Event composition in its general form depends on the ability to determine the order of occurrence of events. The determination of this order is important not only for event operators such as sequence, but also for all other operators since the consumption of events directly depends on it[3]. Logical clocks can not be used for this purpose because they

[3] Four consumption modes were defined in [14]. Recent and chronicle are of most common use. Recent selects the latest event occurrences of a given type, while chronicle selects the oldest event occurrences of a given type out of the event stream.

can not represent timed real world events. Therefore, event order is achieved by using timestamps that are attached to event occurrences.

In addition to defining an event algebra, middleware platforms that support event composition must also address the semantic issues associated with processing composite events. For example, the manner in which timestamps are generated and interpreted, and the way in which events are selected and consumed. Consequently, the adopted assumptions must be clearly exposed to the application developers and it must be possible for them to influence the service behavior by applying (predefined or user-defined) policies.

Inherent characteristics of distributed environments increase the difficulty of composite event detection and invalidate the use of approaches designed for centralized systems. Consider, for instance, the reuse of operators implemented for centralized environments where a total order of events was assumed and no transmission delays or failures were considered. Even though the intended meaning of the event operator is the same, its implementation may be invalid and demand a re-design due to the requirements of the distributed environment.

Our infrastructure was designed to be used in a variety of scenarios. This impacts the semantics of composite event detection. Therefore, the objective is not to define yet another event algebra but to provide a flexible platform for event composition that explicitly exposes to the application developer the decisions that must be taken and the policies that must be applied under particular circumstances. Three areas that received particular attention are:

- **Proper interpretation of time**. Since this infrastructure was designed to be used in a variety of scenarios different time assumptions and timestamp representations must be considered. At an abstract level the required functionality is basically the same in all cases: find a correlation among timestamps. Two main issues must be solved: i) how to represent timestamps in a flexible and open yet "understandable" way, and ii) how to correlate them. Timestamps and their related concepts are defined in the ontology. An abstract timestamp concept is defined and particular timestamp representations can be specialized for different scenarios and environments according to the adopted time model.

  To correlate events, the functionality of the abstract timestamp concept includes the methods before and after while the internal data representation is maintained hidden. These methods must be specialized for each particular time model. Additionally, these methods throw exceptions when decisions cannot be taken transferring the decision control to a higher level, where application semantics can be used for resolution.
- **Consideration of transmission delays**. Incoming events are maintained in a temporary data structure (the EventList) before they are used for composition. Since it is the intermediary between event producers and the event composition, it is the appropriate place to tackle the problem of transmission delays, failures at event producers, network failures, and also the order and uncertainty issues when working with event streams. Specifically, our implemented approach combines a window scheme with a heartbeat protocol. When a producer node crashes or the network is partitioned an exception

can be raised and treated by a failure handling policy. A *window mechanism* that works in tandem with the heartbeat protocol is used to separate the history of events (or event stream) into the *stable past* and the *unstable past and present* that still are subject to change. For composition purposes only events in the stable past are considered.

– **Adoption of partial order of events**. A partial order of events is adopted. With this in mind, correlation methods should include the possibility of throwing an exception (e.g. cannotDecide) in order to announce an uncertainty when comparing timestamps. That means that the underlying infrastructure is responsible for announcing an ambiguous situation to a higher level of decision, allowing the use of application semantics for the resolution. Events from the stable past are maintained (partially) ordered in the EventList according to the consumption mode criteria. EventList also implements the *event consumption interface* that is used for selecting and consuming events. With the provisions taken, it can be guaranteed in all cases that: i) situations of uncertain timestamp order are detected and the action taken is exposed and well defined, and ii) events are not erroneously ordered.

The composite event detector service is based on components and containers. Components are the event operators that are plugged into compositor containers. The container itself is the composite event detector kernel which controls the event detection process. As shown in Figure 3, the container has attached, in this case, two EventLists that play the role of event operands. Additionally, they are configured with appropriate policies according to the definition of the composite event that must be detected.

Components are responsible for the logic of the event operator. They implement the method evaluate. Components specialize the EventOperator class by re-writing the evaluate method according to the operator they represent. More details about the implementation can be found in [3].

The logic of operators detects the situation of interest. Other aspects are now under the responsibility of the EventList which implements the order in which events are accessed through the event consumption interface. The consideration

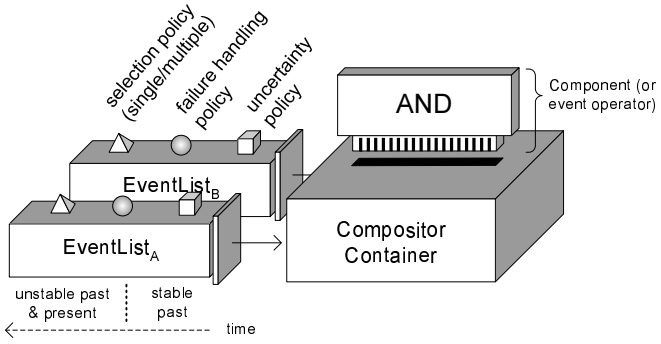

Fig. 3. Abstract view of an event compositor

of failures and transmission delays, as well as uncertainty issues are solved at the EventList by applying pre-configured policies.

Because of its uniform design, compositors can cooperate in the detection of other composite events. Compositors publish detected events in the same way primitive events are published. Thus, the output of a compositor can be used for subscription of other parties. Consequently, each compositor can be seen as an abstract tree where primitive events are injected at the leaves and compositors are located in the internal nodes. Detected events are pushed to the upper layer in the tree by using the publish mechanism. The whole composite event is detected once an event is published at the root of the tree.

3.5 ECA-Rule Definition

Rule representation is organized into three layers (see Figure 4):

- *external:* allows the tailoring of a rule's definition for each specific domain making the specification of rules convenient without the complexities imposed by a generic rule definition language. This is the layer seen by the *end-users.*
- *conceptual:* provides independence between the implementation of the underlying active mechanism and an end-user's rule definition. Concepts like Rule, Event, Condition, Action, etc. and their specialization are representatives of this layer. This is the layer of the *system developers*.
- *internal:* enables the use of a "generic" active functionality service where components or services that are involved can be implemented using different optimization criteria or different programming languages, but they all "understand" the conceptual layer while using an internal representation to process rules. This is the layer of the *service implementors*.

It must be borne in mind that domain-specific and infrastructure-specific terminology are represented here using ontologies as described previously. On this basis, developers can provide various "external" alternatives to end-users in order to define rules taking into account the domain in question, the target end-users, etc. Notice that details about event consumption, or coupling modes can be specified by the system developer hiding in this way such details from the end-user.

From the developer's point of view, all these alternatives rely on an *Ontology API* that facilitates the access and manipulation of the ontologies. In this way, all kinds of external rule definitions produce an ontology-based (conceptual) rule representation as output. As mentioned above, this conceptual rule representation provides independence between the underlying active mechanism and the end-users' rule definitions.

With the aid of ontologies as the foundation of our infrastructure, the definition of rules can benefit from the use of semantic contexts. Contexts can be associated with conditions and actions in order to evaluate them under the defined contextual information. For instance, a condition predicate that verifies distances can define "metric system" as its context. In this manner, incoming events from heterogeneous sources are first converted into the metric system

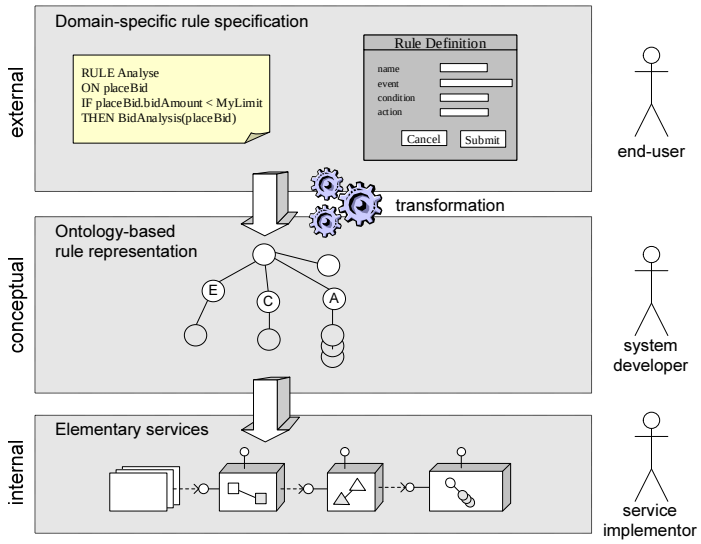

Fig. 4. Rule representation layers

(if necessary) before they are used for evaluation. Consequently, conditions and actions are always specified at a domain-specific level, and are independent of source-specific representations. This provides a very useful and powerful mechanism for interpreting events from heterogeneous sources by maintaining a high-level specification.

3.6 Service-based ECA-rule Processing

In CREAM, traditional ECA-rule processing is decomposed into its elementary parts (aka elementary services). These autonomous services are responsible for composite event detection, condition evaluation, and action execution. Elementary services expose two kinds of generic and very simple interfaces: i) a *service interface* with a single method that receives an event notification as an argument; ii) a *configuration interface* that is used for administration purposes, such as registration, activation, deactivation, deletion, etc. This service interface provides flexibility, enabling a simple interaction among services. ECA-rule processing is then realized as a composition of these elementary services according to the rule definition. Elementary services involved in its processing can interact with external services or systems (e.g. workflow engines, databases, Web Services) through *plug-ins* in order to complete their task. Plug-ins are also responsible for maintaining the target context of the system they interact with making possible the automatic conversion of data to the target system.

From an abstract point of view, this service composition takes the form of a chain of services, where semantic event instances flow through the composed services to carry out the corresponding rule processing. Interactions among elementary services involved in the processing of a rule can be carried out by using

traditional request/reply protocols. However, in our prototype interactions are based on the notification service described before providing several advantages as natural decoupling of services, asynchronous communication, location transparency to mention a few. Figure 5 shows the interaction among elementary services, where boxes denote services and lollipops their interfaces.

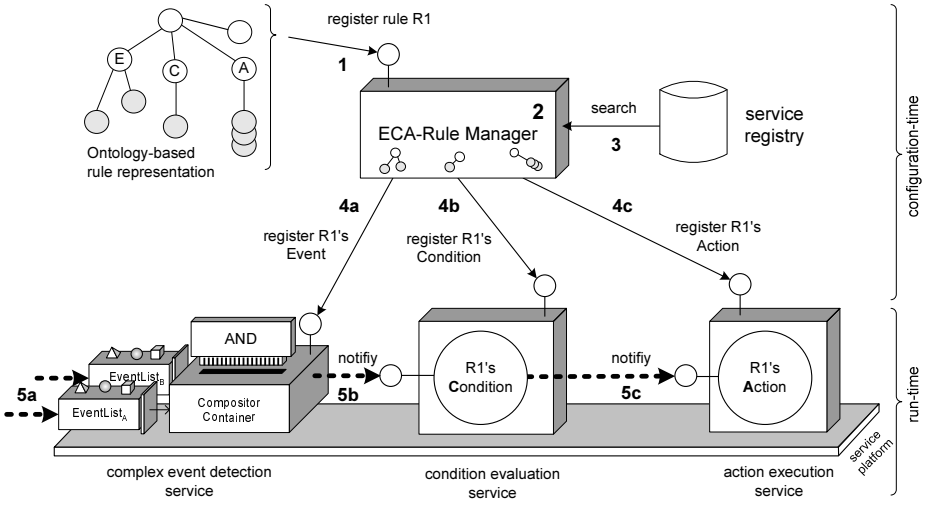

Fig. 5. Interaction among elementary services (ECA-rule processing chain)

Service composition is the responsibility of the *ECA-rule manager*. It exposes operations to register, remove, activate, and deactivate ECA-rules. The most complex of these operations is the registration of a rule, which involves the composition of elementary services. This composition consists of four steps: i) decomposing the rule, ii) finding, iii) contacting, and iv) configuring elementary services. The ECA-rule manager decomposes the rule definition passed for registration, and based on its parts it finds adequate elementary services in the service registry. The ECA-rule manager is responsible for building a chain of elementary services that will process the rule in question. Next, elementary services are contacted for configuration. The configuration of an elementary service itself comprises three steps: a) the subscription to the output of the preceding elementary service (this is achieved by using the control part of the subject as it was mentioned in Section 3.3), b) the configuration of the task under the responsibility of this service (e.g. a condition evaluation service is configured with the condition of the rule that must be evaluated) and c) the configuration of the publisher.

Interactions among elementary services rely on the notification service. Coupling modes (which specify the transactional relationship between elementary services involved) can be delegated to a notification service that supports them. For instance, the notification service implemented in the X^2TS Project [45] inte-

grates notifications and transactions allowing the specification of coupling modes to be made on a per subscription basis [46].

Consider the registration of rule R1 which includes a composite event (Figure 5). Input for the registration is a rule definition represented using the ontology. The ECA-rule manager (1) breaks the incoming rule into elementary parts (2) and searches for proper services according to the parts obtained (3). Afterwards, the manager registers the rule parts of R1 with the services obtained (4a, 4b and 4c). As a consequence the composite event detector configures policies and consumption mode according to the rule definition to detect R1's event. Then the condition evaluation service instantiates a condition object which is responsible for subscribing to the event in question, for its evaluation and if satisfied for republishing the event. The action execution service instantiates an action object that is in charge of subscribing to the event passed by the ECA-rule manager and for the action execution. This completes the *service composition phase*.

At run-time, semantic events feed the EventLists (5a) and when R1's composite event is detected the compositor container publishes this happening (5b). In this particular case, the condition object is notified. If R1's condition is satisfied, the event is republished (5c) notifying in this case the action object in question.

Notice that more than one rule may be defined and they could share the same or similar event definitions. So, conflict resolution policies may be needed according to the execution model adopted (concurrent execution, sequential execution or based on a conflict resolution policy. See [3] for details.).

4 Case Studies and Implementation

This section presents two case studies where the proposed infrastructure was used. A short description of the implementation is provided.

4.1 Meta-Auctions

A meta-auction broker [47] provides a unified view of different auction sites and services for category browsing, item search, auction participation, and auction tracking. To enable the brokering between different participating auction sites, the precise understanding of the terms used by each site is needed and is made explicit through a domain-specific common vocabulary. Notifications about events, such as the placement of a highest bid, and their timely delivery to the user represent valuable information. Propagation of events leads to an efficient non-polling realization of an auction tracking service. Events related to the auction process are disseminated using the concept-based notification service. This way, bidders and sellers use a semantic level of subscription which is common to all of them.

The auction process itself is defined using state charts. Because they are event-driven, they can be easily implemented with ECA-rules. In this way, different sets of rules can describe different types of auction processes (ascending, reverse, dutch, etc.) [48]. To track an item of interest during an auction process,

e.g. to ascertain that another bidder has placed a highest bid, or that the deadline of an auction is approaching, an agent can be used. Here bidders benefit from an active functionality service to program their own agents. In contrast to current agent bidders that are owned, controlled, and implemented by the auction house, these agents can react to happenings of the auction process according to the bidders' strategy.

4.2 Internet-enabled Car

Automotive systems will no longer be limited to information located on-board, but can benefit from a remote network and service infrastructure. Consider the scenario where vehicles, persons and devices have a web presence (or portal). Within this scenario new possibilities emerge, e.g. the adjustment of instruments according to personal preferences, favorite news channels, sports, music or access to one's e-mail and calendar through the portals. Through the portals this can be made independent of a particular car and could be applied to other vehicles (rental cars, company cars, etc). But not only instruments can be adjusted, services can be personalized too. For instance, services such as, "find the route to the next gas station", or "book an appointment to change oil" can take into account car manufacturer's, company's, and/or driver's preferences.

The content of portals is kept up-to-date by means of events. Events are disseminated to interested consumers (e.g. other portals) through of the concept-based publish/subscribe mechanism. For instance, vehicle manufacturers are interested in subscribing to vehicle failures obtaining in this way an "on-line" statistic which can provide valuable data that could be fed back into the design.

Portal managers are enhanced with the active functionality service in order to provide the possibility to specify reactions according to happenings of interest. These reactions can also take into consideration user preferences. For instance, when the driver gets into the car, it is a workday and between 8:00 am and 9:00 am the vehicle can react by loading the best route to the office, and by reading out her company news, her e-mail and by checking her calendar.

Based on CREAM, our prototype [49] shows the reaction of vehicles to different situations according to a set of user-defined rules. External services are implemented using Web Services technologies.

4.3 Implementation

A prototype of CREAM has been developed. Java is used to specify and implement ontology concepts and their relationships. Ontology support and the necessary ontology concepts of the infrastructure are completely implemented. Event adapters are manually configured. The concept-based notification service was implemented on top of TIB/Rendezvous (for historical reasons). The active functionality service and its elementary services were developed using Java and run on top of HP's Core Service Framework (CSF). Event adapters (for Java applications and for XML) and plug-ins (for workflow engines and for Web Services) were also implemented.

For the meta-auction scenario event adapters were built to integrate data and events from different auction sites. An auction service on the basis of ECA-rules was also defined. For the Internet-enabled car scenario an adapted version of the CoolTown Web Presence Manager was used to manage portals and it was extended to collaborate with CREAM. Complex rule reactions which involve several services are carried out using a workflow engine. The domain-specific ontologies for both scenarios were defined.

5 Conclusions

Event-based applications require a middleware layer that includes event handling, support for integration of heterogeneous data, monitoring capabilities, reaction to events, and notification mechanisms.

Approaches found today in the literature focus on specific issues (i.e. event dissemination, active functionality) providing isolated solutions. CREAM presents a uniform and integrated approach based on ontologies. Our ontology-based infrastructure applies homogeneously the ontology approach not only to integrate events from different sources but also to support a higher level subscription abstraction. Therefore, consumers do not deal with proprietary representations. Moreover, a conceptual representation of business rules makes a high-level and domain-specific rule definition language possible providing independence between specification of rules and the active functionality mechanism.

ECA-rule processing in CREAM is decomposed into elementary services. These services provide a very simple and generic interface, where parameters of methods are represented using the common ontology. Therefore, the flow of work through services can be easily configured – inclusion or conscious exclusion of services like condition evaluation, event filtering or complex event detection is made easy. Services interact through notifications. For this purpose, a notification service, based on a publish/subscribe mechanism using concept-based addressing, is employed. The use of this mechanism is appropriate for loosely-coupled distributed systems. Because of this conceptual foundation, our architecture promotes flexibility, extensibility and integration for large-scale, event-based distributed applications.

We are currently moving our implementation from a proprietary to an open platform for the service-based architecture. The notification service is being migrated to JMS. We are also studying how to integrate Web Services with the conversion function mechanism supported by our ontology.

References

1. Buchmann, A., Bornhövd, C., Cilia, M., Fiege, L., Gärtner, F., Liebig, C., Meixner, M., Mühl, G.: DREAM: Distributed Reliable Event-based Applcation Management. In: Web Dynamics (to appear). Springer (2003)
2. Bornhövd, C.: Semantic Metadata for the Integration of Heterogeneous Internet Data (in German). Ph.D. Thesis, Department of Computer Science, Darmstadt University of Technology, ISBN: 8265-8390-6, Shaker-Verlag, Germany (2000)

3. Cilia, M.: An Active Functionality Service for Open Distributed Heterogeneous Environments. Ph.D. Thesis, Department of Computer Science, Darmstadt University of Technology, ISBN:3-8322-0790-2, Shaker-Verlag, Germany (2002)
4. Object Management Group: Event Service Specification. Technical Report formal/97-12-11, Object Management Group (OMG) (1997)
5. Object Management Group: CORBA Notification Service Specification. Technical Report telecom/98-06-15, Object Management Group (OMG) (1998)
6. Hapner, M., Burridge, R., Sharma, R.: Java Message Service. Specification Version 1.0.2, Sun Microsystems, JavaSoftware (1999)
7. Oki, B., Pfluegl, M., Siegel, A., Skeen, D.: The Information Bus – An Architecture for Extensible Distributed Systems. In: Proceedings of SIGOPS, USA (1993) 58–68
8. Carzaniga, A., Rosenblum, D.R., Wolf, A.L.: Challenges for Distributed Event Services: Scalability vs. Expressiveness. In: Proc. of EDO. (1999)
9. Opyrchal, L., Astley, M., Auerbach, J., Banavar, G., Strom, R., Sturman, D.: Exploiting IP Multicast in Content-based Publish-Subscribe Systems. In: Proceedings of Middleware. Volume 1795 of LNCS., Springer (2000) 185–207
10. Mühl, G., Fiege, L., Buchmann, A.: Filter Similarities in Content-Based Pub/Sub Systems. In: Proc of ARCS. Volume 2299 of LNCS., Springer (2002) 224–238
11. Fabret, F., Llirbat, F., Pereira, J., Jacobsen, A., Ross, K., Shasha, D.: Filtering Algorithms and Implementation for Very Fast Publish/Subscribe. In: Proceedings of ACM SIGMOD. (2001) 115–126
12. Dayal, U., et al.: The HiPAC Project: Combining Active Databases and Timing Constraints. ACM SIGMOD Record **17** (1988)
13. Chakravarthy, S., Mishra, D.: Snoop: An Expressive Event Specification Language for Active Databases. Data and Knowledge Engineering **14** (1994) 1–26
14. Charkravarthy, S., Krishnaprasad, V., Anwar, E., Kim, S.: Composite Events for Active Databases: Semantics, Contexts and Detection. In: Proc. of VLDB. (1994) 606–617
15. Kopetz, H.: Sparse Time versus Dense Time in Distributed Real-Time Systems. In: Proc. ICDCS, Yakohama, Japan (1992) 460–467
16. Liebig, C., Cilia, M., Buchmann, A.: Event Composition in Time-dependent Distributed Systems. In: Proceedings of CoopIS. (1999) 70–78
17. Schwiderski, S.: Monitoring the Behaviour of Distributed Systems. PhD thesis, Selwyn College, Computer Lab, University of Cambridge, United Kingdom (1996)
18. Ma, C., Bacon, J.: COBEA: A CORBA-based Event Architecture. In: Proceedings of COOTS'98, New Mexico, USA, USENIX (1998) 117–131
19. Geppert, A., Tombros, D.: Event-based Distributed Workflow Execution with EVE. In: Proceedings of Middleware, The Lake District (1998)
20. Yang, S., Chakravarthy, S.: Formal Semantics of Composite Events for Distributed Environments. In: Proceedings of ICDE, Sydney, Australia (1999) 400–407
21. Paton, N., ed.: Active Rules in Database Systems. Springer (1999)
22. Gatziu, S., Koschel, A., v. Buetzingsloewen, G., Fritschi, H.: Unbundling Active Functionality. ACM SIGMOD Record **27** (1998) 35–40
23. Koschel, A., Lockemann, P.: Distributed Events in Active Database Systems - Letting the Genie out of the Bottle. Data & Knowledge Engineering **25** (1998) 29–53
24. Fritschi, H., Gatziu, S., Dittrich, K.: FRAMBOISE - an Approach to Framework-based Active Data Management System Construction. In: Proc. of CIKM. (1998)
25. Collet, C.: The NODS Project: Networked Open Database Services. In et.al., K.D., ed.: Object and Databases 2000. Number 1944 in LNCS, Springer (2000) 153–169

26. Buchmann, A.: Architecture of Active Database Systems. In: Active Rules in Database Systems. Springer (1999) 29–48
27. Buchmann, A., Liebig, C.: Distributed, Object-Oriented, Active, Real-Time DBMSs: We Want It All – Do We Need Them (At) All? Annual Reviews in Control **25** (2001)
28. Eisenberg, B., Nickull, D.: ebXML Technical Architecture Specification v1.04. Technical report (2001) `http://www.ebxml.org`.
29. Microsoft Corp.: BizTalk Framework 2.0: Document and Message Specification. Microsoft Technical Specification (2000)
30. RosettaNet: RosettaNet Implementation Framework: Core Specification v2.00.01. RosettaNet Technical Specification (2002)
31. Berners-Lee, T., Hendler, J., Lassila, O.: The semantic web. In: Scientific American. (2001)
32. Conolly, D., van Harmelen, F., Horrocks, I., at al.: Daml+oil (march 2001) reference desciption. W3C Note, W3C (2001)
33. Bray, T., Paoli, J., Sperberg-McQueen, C.: Extensible markup language (xml) 1.0. W3C Recommendation, W3C (1998)
34. Fallside, D.: XML Schema Part 0: Primer. W3c recommendation, W3C (2001)
35. Bray, T., Hollander, D., Layman, A.: Namespaces in XML. W3C Recommendation, W3C (1999) `http://www.w3.org/TR/REC-xml-names`.
36. Lassila, O., Swick, R.: Resource Description Framework (RDF) Model and Syntax Specification. W3c recommendation, W3C (1999)
37. Brickley, D., Guha, R.: RDF Vocabulary Description Language 1.0: RDF Schema. W3c working draft, W3C (2002) `http://www.w3.org/TR/rdf-schema`.
38. Bornhövd, C., Buchmann, A.: A Prototype for Metadata-Based Integration of Internet Sources. In: Proc. of CAiSE. Volume 1626 of LNCS. (1999) 439–445
39. Gruber, T.R.: Towards Principles for the Design of Ontologies Used for Knowledge Sharing. Int. Journal of Human-Computer Studies (IJHCS) **43** (1995) 907–928
40. Guarino, N.: Understanding, Building and using Ontologies. Int. Journal of Human-Computer Studies (IJHCS) **46** (1997) 293–310
41. Mena, E., Kashyap, V., Illarramendi, A., Sheth, A.: Domain specific ontologies for semantic information brokering on the global information infrastructure. In: Intl. Conf. on Formal Ontology in Information Systems, Trento, Italy (1998)
42. Heflin, J., Volz, R., Dale, J.: Requirements for a web ontology language. W3C Working Draft, W3C (2002) `http://www.w3.org/TR/webont-req/`.
43. UNICORN Maintenance Authority: UNICORN Application Standard. Technical Report TTIP03 V4.0, Travel Technology Initiative Ldt. (1994)
44. Mühl, G.: Large-Scale Content-Based Publish/Subscribe Systems. PhD thesis, Darmstadt University of Technology, Germany (2002)
45. Liebig, C., Malva, M., Buchmann, A.: X^2TS: Unbundling Active Object Systems (Short Paper). In: Proceedings of Middleware. Volume 1795 of LNCS. (2000)
46. Liebig, C., Tai, S.: Middleware Mediated Transactions. In: Proc. of DOA'00. (2001)
47. Bornhövd, C., Cilia, M., Liebig, C., Buchmann, A.: An Infrastructure for Meta-Auctions. In: Proceedings of WECWIS, IEEE Computer Society (2000) 21–30
48. Cilia, M., Buchmann, A.: An Active Functionality Service for E-Business Applications. ACM SIGMOD Record **31** (2002) 24–30
49. Cilia, M., Hasselmeyer, P., Buchmann, A.: Profiling and Internet Connectivity in Automotive Environments. In: Proc. of VLDB. (2002) 1071–1074