Cordies

Koch, Gerald G. ; Koldehofe, Boris; Rothermel, Kurt

Published in:
Proceedings of the Fourth ACM International Conference on Distributed Event-Based Systems (DEBS)

DOI:
10.1145/1827418.1827424

Publication date:
2010

Cordies: Expressive event correlation in distributed systems

Gerald G. Koch Boris Koldehofe Kurt Rothermel

Universität Stuttgart
Institut für Parallele und Verteilte Systeme
Universitätsstr. 38
D-70569 Stuttgart, Germany
{firstname.lastname}@ipvs.uni-stuttgart.de

ABSTRACT

Complex Event Processing (CEP) is the method of choice for the observation of system states and situations by means of events. A number of systems have been introduced that provide CEP in selected environments. Some are restricted to centralised systems, or to systems with synchronous communication, or to a limited space of event relations that are defined in advance. Many modern systems, though, are inherently distributed and asynchronous, and require a more powerful CEP. We present *Cordies*, a distributed system for the detection of correlated events that is designed for the operation in large-scale, heterogeneous networks and adapts dynamically to changing network conditions. With its expressive language to describe event relations, it is suitable for environments where neither the event space nor the situations of interest are predefined but are constantly adapted. In addition, *Cordies* supports Quality-of-Service (QoS) for communication in distributed event correlation detection.

Categories and Subject Descriptors

C.2.4 [**Computer Systems Organisation**]: Distributed Systems

General Terms

CEP, event correlation detection

1. INTRODUCTION

Complex Event Processing (CEP) [22] has been subject to a number of research projects. With the emergence of the Internet of Things, an ongoing pervasion of ubiquitous computing, and an increasing number of sensors in common networked devices, the number of events to process and situations to detect is expected to grow further. This presents several challenges to systems that provide CEP.

One challenge is the large *number* and geographical *distribution* of event sources. For instance, modern cellphones contain sensors for position, motion etc., which information can be gathered and aggregated to detect situations with varying granularity and dependability [8]. Clearly, a centralised system does not scale for the number of events submitted by these devices. An insufficient distribution of a CEP system also leads to the aggregation or correlation of events that arised in different contexts and therefore rather blur than clarify the situation of a certain limited location. The ability of a CEP system to work in a distributed manner goes with its ability to place CEP functionality at sensible places in the network of information providers.

A second challenge is the *diversity* of events. Current CEP systems are often designed for defined environments or single applications, e.g., a stock trading system, an electrical power grid, or river observation [11]. Their scope of observation, possible and interesting relationships between events, and information providers are known beforehand. Properties of the target environment are assumed to hold categorically (e.g., that water will always flow down the river). Future systems will not be constrained to one physical location or to just one application to control the CEP system. There will rather be a large number of users defining arbitrary situations of interest, and an even larger number of event providers, publishing previously unknown events. Therefore, CEP systems need to provide an expressive language for event correlation description, and correlation detection should make no assumptions about the observed systems.

In this paper, we present *Cordies* (*Cor*relation in *di*stributed *e*vent *s*ervices). It is inherently distributed, so it can adapt to a user-defined granularity of distribution. *Cordies* is fit for the detection of arbitrary situations in future observable systems because of the expressiveness of its correlation detection language and its independence from assumptions about the observed systems. A framework for the placement of *Cordies* functionality allows for the adherence to a number of user-defined restrictions, including, but not limited to, QoS requirements.

The structure of this paper is as follows. An overview about related work follows in Section 2. Then, in Section 3, we introduce the system model, and in Section 4, the correlation detection language (CDL) and its execution in *Cordies*. In Section 5, the placement framework for correlation functionality and its control by the user are explained starting from the language's support for distribution. In Section 6, we provide evaluation results about the placement framework. Section 7 concludes the paper.

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

2. RELATED WORK

Over the last two decades, by continuous enhancement, CEP has become the method of choice for the observation of system states and situations. Great effort has been put into the development of query languages and detection algorithms. This is reflected by its application in databases [15, 14, 10, 1, 2] and distributed systems [33, 17, 6, 28, 37, 19, 26, 7]. CEP can express temporal relations [5, 12, 31, 9], expressive situations [22, 3, 25], and even language concepts of heterogeneous CEP systems [32]. See Table 1 for an overview on the features of selected systems.

Operator expressiveness. We realise that CEP has yet potential to improve. It is noticeable, for example, that many event algebras use operators in parallel that have different levels of abstraction. *Logical* operators (conjunction, disjunction, negation) test whether or not at least one instance of an event class is present (cf. Concepts 1, 2 in Table 1). *Temporal* operators (e.g. the *sequence* operator) imply a conjunction of event instances and require a temporal order on them (Concept 3). *Collection* operators imply an ordered sequence of event instances of the same class and detect an instance's repetition or relative position within the sequence (Concepts 4, 5). A fourth class, *Time* operators, instructs the system to create virtual time events with a temporal relation to received events (Concept 6).

These operators are expressive enough for the respective environments where event existence, order, and relative temporal or sequential position are important. However, although some operators even overlap in their semantics [10], they are not sufficient for the expression of arbitrary event relations. Therefore, a number of additional event operators have been added over time, e.g., causality [22, 7] (cf. Concept 7 in Table 1), concurrency (Concept 8) in distributed systems where event sequence cannot always be established [5, 28, 37, 19], and spatial operators [37, 34] (Concept 9).

These operators will not be sufficient for future systems. However, adding another event operator when a further relation needs to be described leads to operator explosion. For example, Ode [15] provides not only one but three different sequence operators which allow to detect three different relative positions of event occurrence intervals. Even this approach is not expressive enough because there are 13 different relative positions of intervals [4]. Spatial operators have to consider the same problem [37, 34]. Operators at several abstraction levels doubtlessly increase the usability of a language and the ease of the composition of expressions. Nevertheless, it would be advantageous for a user to also be able to exactly specify the relations that he or she has in mind, with the help of just a small set of operators. *Cordies* provides a correlation detection language that allows users to directly address event content and thus specify the semantics of relations on their own. This makes *Cordies* fit for the use in future environments where new event properties may appear and familiar ones can be expressed differently.

Uniform specification of conditions. Another shortcoming of temporal operators is that they specify relative positions of events, while additional constraints on absolute temporal distances cannot be expressed. Many different annotations on temporal operators with varying expressiveness have been proposed [24, 17, 28, 9, 37, 19] (cf. Concept 10 in Table 1). With these annotations, expressions become confusing in their syntax as well as in their semantics. For instance, a sequence operator annotated with a minimal and a maximal positive temporal distance between events could be substituted by a concatenation or even be left out completely without affecting the detection result.

Furthermore, once additional conditions can be tested on timestamps, users would expect an event algebra to provide equal means to specify conditions on *arbitrary* event properties (Concept 11). Active databases like Ode [15] provide this capability. The ECA (event-condition-action) model allows the definition of conditions in addition to the specification of the triggering (composite) event. However, systems with advanced languages like GEM [24], Rapide [22], Amit [3], XChange^{EQ} [7] and others [25, 2] primarily model temporal relations with the available operators and then add other conditions in different, non-uniform notations and with varying expressiveness. Apart from the trouble to keep track of the overall semantics of separately specified conditions, there is also an efficiency problem to it. In most cases, the composite event has to be detected before the conditions are tested (Concept 12). As a result, tests on non-temporal event properties are late because all constituent events have to be detected first, and it can be costly if a lot of composite events are detected that later do not match the conditions [2]. For this reason, *Cordies* handles temporal and non-temporal conditions on events in a uniform manner and applies them already on evaluable subsets of constituent events in order to detect failing event combinations early and with minimal computation overhead. To this end, *Cordies* provides a uniform description method that is equally expressive for arbitrary event relations (Concept 13).

Expressive policies. The authors of Snoop [10] analysed an ambiguity in CEP when more than one event message per event class is available for the correlation test. They addressed it with the help of four *parameter contexts* which were shown to be combinations of event selection policies (cf. Concept 14 in Table 1) and event consumption policies (Concept 15) [38]. While previous systems used to follow one policy implicitly, modern systems provide a selection of parameter contexts that can be applied on the system level [1], the operator level [37] or the operand level [17, 3]. Other consumption policies than the original four have been proposed [5, 9]. It is remarkable that all proposed event selection and event consumption policies again concentrate on temporal relations between events of one class. However, an event algebra should allow the specification of preferred event combinations on other than temporal relations as well. Therefore, *Cordies* allows users to specify event instance selection and event instance consumption freely as event relations just like the relations for correlation detection itself.

Distribution and placement. Without doubt, one important future application field for CEP will be large distributed heterogeneous systems with a large number of publishers and subscribers. For efficiency and reliability reasons it will be necessary to distribute CEP functionality throughout the system (cf. Concept 16 in Table 1). CEP systems that were designed for use in distributed systems have already addressed the problem of the placement of operators. Usually, the tree of operators on events is divided in sub-trees so that each sub-expression observes two [28] or more [19] different event classes for correlated events. Expressions in the CDL can be divided in many different ways that allow to consider the runtime behaviour of relations—especially their selectivity and computational overhead for evaluation.

No.	Concept	databases						distributed systems						pub/sub-based			event algebras					
		[15]	[14]	[10]	[3]	[1]	[2]	[33]	[24]	[22]	[37]	[25]	[34]	[28]	[20]	Cordies	[5]	[17]	[12]	[31]	[9]	[7]
1	logical operators	y	y	y	y	y	y	y	y	y	y	y	y	y	y	(y)	y	y	y	y	y	y
2	negation	y	y	y	y	y	y	y	y	y	y	y	-	y	-	(y)	y	y	-	y	y	y
3	temporal operators	y	y	y	y	y	y	y	y	y	y	y	y	y	y	(y)	y	y	y	y	y	y
4	iteration	y	y	y	y	y	y	-	-	y	y	-[a]	-	y	y	y	-	-	-	y	-	y
5	selection	y	(y)	-	y	-	y	-	-	-	y	-[a]	-	-	-	(y)	-	y	-	y	-	-
6	creation of time events	y	-	-	y	y	y	-	y	-	-	-	-	-	-	y	-	-	-	-	-	y
7	causality	-	-	-	-	-	(y)	-	-	y	-	(y)	-	-	-	(y)	-	-	-	-	-	y
8	concurrency	-	-	-	-	-	(y)	y	-	-	y	(y)	y	y	-	(y)	-	-	-	y	-	-
9	spatial operators	-	-	-	-	-	(y)	-	-	-	y	y	y	-	-	(y)	-	-	-	-	-	-
10	equality test	(y)	y	-	(y)	-	y	y	(y)	y	y	y	-	y	-[b]	(y)	-	-	y	-	-	y
11	arbitrary relations	y	-	-	y	-	-	-	y	y	-	-	-	-	-	y	-	-	-	-	-	y
12	early condition testing	-	y	-	?	-	y	?	-	-	?	-	-	y	-	y	-	-	?	-	-	-
13	uniform notation	-	-	-	-	-	-	-	-	-	-	-	-	-	-	y	-	-	-	-	-	-
14	selection policies	-	-	y	y	y	y	-[c]	-[c]	-[d]	y	-[e]	-	-	-	y	y	y	-[c]	-[f]	y	-
15	consumption policies	-	-	y	y	y	y	-	-[g]	-[d]	y	-[e]	-	-	-	y	y	y	-[h]	-[f]	y	-
16	support for distribution	-	-	y	-	-	-	y	y	y	y	y	y	y	y	y	-	-	-	-	-	-
17	resource-driven placement	-	-	-	-	-	-	-	-	-	-	-	-	y	y	y	-	-	-	-	-	-
18	operator-driven placement	-	-	-	-	-	-	-	-	-	-	-	-	-	-	y	-	-	-	-	-	-

[a] possibly modelled by sequence closure [b] detected by subsequent filter [c] always *chronicle*
[d] partly dispensable because of causality [e] perhaps *continuous* [f] *recent* [g] via time window [h] each event used once
Key: y: available; (y): modelled by other means; -: not available; ?: no information

Table 1: Comparison of concept support in CEP systems

The placement of correlation detection functionality has been studied for the compliance to node and network constraints (cf. Concept 17 in Table 1) in publish/subscribe systems [20] and stream-based event systems [13, 27]. *Cordies* supports its use in distributed systems with the provision of a placement framework that specifies and maintains required system properties at the level of overlay nodes and links instead of making system-wide assumptions about their behaviour. Placement decisions in *Cordies* also consider the requirements of correlation detection itself (Concept 18).

3. SYSTEM MODEL

Cordies is a distributed application that consists of an arbitrary number of overlay *nodes*. *Cordies* nodes do not directly connect to sensors or other devices that provide data; they observe events and their correlation by subscribing to *event messages* in a publish/subscribe system. Nodes also publish composite event messages that other subscribers, including other *Cordies* nodes, can subscribe to.

Event messages are modelled as sets of tuples of the form `(attributeName, type, operator, value)`. The `attributeName` is the name of an attribute that belongs to an object or to an observed system. *Time*, for instance, is an attribute of the real world rather than a property of a particular object. The `operator` is a relational operator, e.g., equality, and the `value` is of the indicated `type` which allows for the correct interpretation of the attribute value. Single values, set expressions or range expression can be used in association with appropriate operators.

Event messages are records of a section of the observed system's state. In particular, their occurrence does not imply that a change took place in the system's state. These semantics allow for the existence of duplicate event messages and the replay of event messages. For instance, sensors that are new to a system first sense and publish the system's state, not a state change. Sensors can also provide state messages as heartbeat messages.

Event messages belong to *event classes* that are distinguished by an identical set of attribute names or values (the *signature* of the message). For the detection of situations and state changes, concrete sets of event messages are tested whether they correlate, i.e., whether they are in a particular relation or not. This relation, however, is defined on

the event classes of the involved events. Simple observations that were not the result of correlation detection are called *atomic events*, while a state change or situation that is distributed and that is detected on the basis of simple observations is called a *composite event*.

There is no difference in the message model for atomic or composite event messages: both are "event messages"—sets of tuples as described above. In particular, subscribing to atomic event messages is not different from subscribing to composite ones. The subscription to a composite event message initiates its delivery to the subscriber, not its detection (for the initiation of detection, see Section 4). As in many publish/subscribe systems, publishers of event messages (including *Cordies* nodes) are required to *advertise* events messages they are going to publish. This model keeps *Cordies* decoupled from the publish/subscribe system and is similar to the approach selected by Pietzuch et al. [28].

4. EVENT CORRELATION DETECTION

In order to perform event correlation detection, a CEP system needs to provide two means: An algebra to describe relations for observed events messages (Section 4.1), and an algorithm to detect individual event messages that are in the described relation (Section 4.3). *Cordies* employs an algebra that avoids traditional event operators. It is rather based on predicates on event attributes which allows for an expressive description of arbitrary event relations. The language's benefits are detailed in Section 4.2.

4.1 Correlation Description

Cordies provides an interface for the initiation of correlation detection where an application can dynamically submit a *Correlation Description* (CD). A CD is an XML document describing details of a specific event relation. The basic elements of the CD language (CDL) are shown in Listing 1.

The main elements of a CD are `sources`, `computations`, `predicates`, `expressions` and `events`. The `sources` are event messages that a *Cordies* node subscribes to. As soon as event messages arrive, the node calculates necessary results by evaluating `computations` that are defined on event attributes or on the results of other `computations`. The complete set of conditions that have to hold on the composite event are expressed in `predicates` that are defined on event attributes and `computation` results. `Expressions` have to evaluate to `true` so that a composite `event` can be detected. The propositions within the expressions are `predicates`. *Cordies* provides built-in `predicates` for different comparisons on numbers and strings, along with unary predicates `any` and `none` which test for the existence or absence of a value's definition and consequently for the presence or absence of `sources`, e.g., event messages.

Element	Explanation
`cd`	root element
`source`	template for an event message
`attribute`	required attribute of the source
`anchor`	a reference to use of the result within the CD
`filter`	pub/sub filter on observed events
`collection`	container for a set of event messages that cannot be addressed individually
`computation`	template for the calculation of a result
`predicate`	template for the evaluation of a n-ary predicate
`expression`	template for the evaluation of a logical expression on predicates
`event`	template for a composite event message

Table 2: CDL element explanation

```
<cd id="..." version="...">
 <sources>
  <source access="..." class="...">
   <attribute name="..." type="..." anchor="...">
   [<filter operator="..." />
     {expression (value, interval, set) or reference
          to an attribute or computation}
    </filter> ]
   [... ]
   </attribute>
  [... ]
  </source>
 [... ]
 [<collection access="..." class="..." anchor="...">
   <attribute name="..." type="..." anchor="...">
    <filter operator="..." />
     {expression (value, interval, set) or reference
          to an attribute or computation}
    </filter>
    ...
   </attribute>
   ...
  </collection> ]
 [... ]
 </sources>

[<computations>
  <computation type="..." anchor="...">
   {recursive operations (arithmetic, string, ...),
        or method calls on values or references}
  </computation>
  ...
 </computations> ]

 [<predicates>
  <predicate name="..." anchor="...">
  [{value or reference to an attribute or computation
      } ]
  [... ]
  </predicate>
 [... ]
 </predicates> ]

 <expressions>
  <expression anchor="...">
   {recursive logic expression on references to
        predicates}
   [... ]
  </expression>
 [... ]
 </expressions>

 <events>
  <event access="..." class="..." trigger="...">
  [<attribute name="..." type="...">
    {expression (value, interval, set) or reference
         to an attribute or computation}
   </attribute> ]
  [... ]
  </event>
 [... ]
 </events>
</cd>
```

Listing 1: CD structure

The CDL's semantics are similar to the Rapide language [22]. However, there are significant differences. First, Rapide uses operators and patterns on events while the CDL applies operators to predicates on event attributes rather than events. Second, the Rapide correlation test is divided

to the event pattern and to the context test which obscures the entirety of conditions that actually apply to correlated events. Third, Rapide uses logical, pattern, structural, and temporal operators for event patterns in parallel although these are not orthogonal. In contrast, the CDL provides a uniform way to specify arbitrary semantics of relations.

We now illustrate the use of the CDL in short example. Assume that the task is to record the parameters of a physician's visit who checks the well-being of patients in a hospital. A visit involves several events: The physician would access the patient's clinical record twice (for preparation and update), see the patient, and disinfect before doing so. A medical insurance is interested in the time that the physician spent for the patient and in the quality of treatment. Therefore, a composite event is defined that spans the physician's actions. These actions can be followed by his or her interactions with the IT system and by wireless readers for a badge in the physician's pocket.

Assume there are three classes of basic events available. The *access* message that represents the access of the medical file has the signature

$(patient, \quad string, \quad =, \quad \dots)$
$(room, \quad int, \quad =, \quad \dots)$
$(physician, \quad string, \quad =, \quad \dots)$
$(accesstype, \quad string, \quad =, \quad \dots)$
$(begin, \quad time\_t, \quad =, \quad \dots)$
$(end, \quad time\_t, \quad =, \quad \dots)$

The *disinfect* message that represents the use of disinfectant has the signature

$(physician, \quad string, \quad =, \quad \dots)$
$(time, \quad time\_t, \quad =, \quad \dots)$

The *door* message that represents a passing of a door has the signature

$(physician, \quad string, \quad =, \quad \dots)$
$(room, \quad int, \quad =, \quad \dots)$
$(time, \quad time\_t, \quad =, \quad \dots)$

The CD defines the correlation detection in five steps. First, it defines the five events (`sources`) that are observed: two *access* events (S_1, S_5), one *disinfect* event (S_2), and two *door* events (S_3, S_4). Two `filters` require S_1 to have the *accesstype read*, and S_5 to have the *accesstype write*.

Second, the CD defines the computation of time intervals between events as shown below:

Ref.	*Type*	Computation	Meaning
C_1	*double*	$S_3.time - S_1.end$	time from file access to visit
C_2	*double*	$S_3.time - S_2.time$	time from disinfection to visit
C_3	*double*	$S_4.time - S_3.time$	visit duration
C_4	*double*	$S_5.begin - S_4.time$	time from visit to file access
C_5	*double*	$S_5.end - S_5.begin + S_1.end - S_1.begin$	time for overall paperwork

Third, some `predicates` check sequences or equalities:

Ref.	Predicate	Meaning
P_1	$greater(C_1, 0.0)$	file access before visit?
P_2	$greater(C_2, 0.0)$	disinfection before visit?
P_3	$equal(S_1.room, S_3.room)$	patient's room or not?
P_4	$equal(S_3.room, S_4.room)$	identical room?
P_5	$equal(S_2.physician, S_3.physician)$	identical physician?
P_6	$greater(C_3, 0.0)$	correct door event sequence?
P_7	$greater(C_4, 0.0)$	file access after visit?
P_8	$equal(S_1.patient, S_5.patient)$	file of identical patient?

Fourth, the `expression` that causes a composite event *visit* is the simple conjunction $P_1 \wedge P_2 \wedge \dots \wedge P_8$.

Last, the attributes of the composite event and the origins of the attribute values are given, e.g. by their anchors.

$(physician, \quad string, \quad =, \quad S_3.physician)$
$(patient, \quad string, \quad =, \quad S_1.patient)$
$(roundtime, \quad double, \quad =, \quad C_3)$
$(paperwork, \quad double, \quad =, \quad C_5)$

Whenever a set of two `access` events, two `door` events and the `disinfect` event meet all conditions of the `expression`, the `round` event will be published using the respective attribute values.

The benefits of the CDL that are displayed by the above example are discussed in the next section.

4.2 Benefits of Correlation Descriptions

Expressiveness. The CDL provides direct access to event attribute. They can be referenced in computations and predicates which have well-known semantics. Thus, even without introducing system-dependent operator semantics, the CDL's expressiveness exceeds the expressiveness of operators of other languages (cf. Section 2) For instance, the exact semantic of a *sequence* can be explicitly defined each time. The example CD in Section 4.1 demonstrates how sequences can be defined and how they can be extended by any number of additional predicates on whatever event attributes and computation results are available. The user has the power to redefine the semantics of sequence without changing the CEP system at all, for example, by using sequence numbers if they were provided by event messages.

Clarity. Of course, temporal operators like the *sequence* operator are desirable as short-cuts. However, many CEP languages provide no equally easy access to describe non-temporal relations like spatial relations, if at all. For instance, with spatial event operators [34], a simple statement like "within a range of 20 to 40 length units from my position" is difficult, if not impossible to express. Operators with annotated constraints make the additional implicit assumption that all events have a uniform "spacestamp" where distance can be checked on, which requires a global naming scheme even for independent event providers. In the CDL, any relation between events is described on attribute values or values computed from them at the same level of ease and with the same notation.

Independence. The CDL is fit for use in systems with a changing set of event types and volatile semantics of event attributes. The calculation of time intervals, for instance, is straightforward although some events have time points and others have time intervals for timestamps. Also, the use of arbitrary attribute names for the starting and ending time of an event with duration is possible because event properties are not accessed implicitly. This is because predicates in the CDL are defined on computation results and attribute values rather than on event messages.

Detection efficiency. The organisation of *Cordies* CDs also results in a more efficient correlation detection process. A *Cordies* CD imposes no order on predicate evaluation. All `predicates` are tested as soon as they are *evaluable*, i.e., as soon as all messages are available that provide values which are required in the predicate. This makes correlation detection more efficient and, in addition, supports the distribution of the correlation algorithm which we describe below in Section 5.

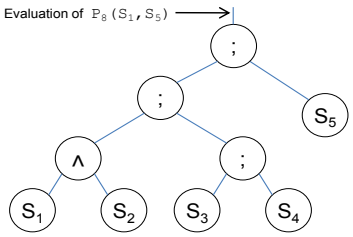

Figure 1: Late predicate evaluation in trees

On the contrary, in systems where operators are defined on events rather than predicates, a common way to perform correlation detection is arranging event templates as leaves in an operator tree and evaluate the tree in a bottom-up manner. Figure 1 shows a possible representation of the example in Section 4.1 as operator tree (with the conditions left out). In operator trees, conditions on events are tested as soon as the operator that combines the events is evaluated. In Figure 1, `predicate` P_8 that checks whether the `sources` S_1 and S_5 refer to the same patient can be evaluated only after S_2, S_3 and S_4 were detected and the detection algorithm advanced to the root of the tree. This is especially problematic if very selective conditions are evaluated rather late. P_8 is an example for a selective condition because of all possible combinations of *access* events, very few refer to the same patient.

Other detection algorithms that evaluate ECA (event-condition-action) rules [15] or use FSA (finite state automata) [28] also defer the evaluation of *all* conditions to the end of the detection and therefore encounter the problem of late evaluation of selective predicates. Note that a division of the expression [28] cannot reduce this problem because the tree structure is not changed and data from S_1 and S_5 still be compared only at the root.

4.3 Correlation detection algorithm

4.3.1 Preparation

When running correlation detection on the basis of a CD, a *Cordies* node creates a data structure that consists of a *template* for each `source`, `computation`, `predicate`, `expression` and composite `event` of the CD. The templates contain processing information (e.g., the algorithm to be executed for the `computation`) and are linked with other templates with which they are related. This structure allows for efficient correlation detection. For example, a computation is evaluated only once, even if its result is used in several predicates and again in an attribute of the composite event message (see the computation C_3 in the above example).

The `expression` leading to the correlation detection is transformed into a disjoint normal form (DNF). Each term in a DNF contains only the logical operators $\wedge$ and $\neg$. Therefore, each term can independently lead to the detection of the correlation. For each term, an *inducing set template* of source classes is defined. It contains the classes of event messages that are necessary to receive in order to evaluate all predicates which are referenced in the term. Inducing set templates are also determined for other templates in the data structure, namely, `computation`s, `predicate`s, and `expression`s.

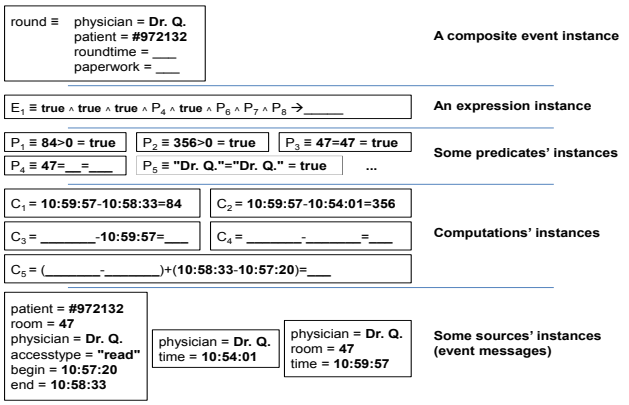

Figure 2: Exemplary correlation detection structure

When the installation of the data structure and the subscriptions to the `sources` mentioned in the CD were successful, *Cordies* advertises the publication of the composite `event` messages.

4.3.2 Execution

Each template can create an unbounded number of *instances* that represent incomplete or complete evaluations of the template. `Source` instances are event messages; `computation`, `predicate` and `expression` instances are (semi-)finished evaluations (see Figure 2).

Incoming event messages are mapped to the inducing set templates. For each mapping, a new *inducing set instance* and a new *event instance* for the complex event are created. New *predicate instances* and *computation instances* are created if the inducing set templates of the predicate template and computation template reference attributes from the incoming event message.

A received event is also mapped to the inducing set instances of already existing event instances. The state of an event instance, called its *configuration*, is defined by the event messages that are mapped to the source templates and the templates that are still unmapped. If the configuration of an existing inducing set instance allows the mapping of the received event to unmapped source templates, then a copy of the `event` instance is created in which the new event is integrated. Copies are also created of `computation`, `predicate` and `expression` instances that will be extended or evaluated using the new event.

An evaluable predicate may evaluate to `false`. Then, the complex `event` instance and all other instances that exclusively depend on this predicate are deleted, as well as the involved `source`s if they contribute to no other instance. If all predicates and expressions that belong to a term in the DNF can be evaluated and the term evaluates to `true`, a composite event is created out of the completed `event` instance and the event instance is deleted together with all other involved instances that contribute to no other instance.

4.4 Correlation detection ambiguity

A well-known problem of correlation detection is its ambiguity [10]. For instance, in the example provided in Section 4.1, several messages about the `disinfect` event S_2 might be available, so that the selection of pairs of S_2 and

S_3 messages for the evaluation of the C_3 `computation` is ambiguous. This problem has been addressed by introducing *selection policies* and *consumption policies*. Such policies state that, for instance, only the most recent or only the oldest messages may be paired and whether a message is consumed if successfully tested, or not.

The expressiveness of such policies is, however, restricted. The "oldest" or "most recent" events often cannot be determined in distributed systems with inaccurate clock synchronisation [21]. Before all, the proposed policies concentrate on temporal relations between events, ignoring other relations that might be better suited for deciding whether a specific set of event messages should be tested for correlation and whether individual messages are consumed for correlation detection in the case of success.

Therefore, the CDL provides another set of predicates managing event selection and consumption at a fine-grained level (see Listing 2). An incoming event message may have produced a set of (semi-)finished composite `event` instances. The `<self>` expression determines whether a new instance has a policy defined on it or not. If so, the instance will be compared to other instances matching the `other` expression. Then, the `better` expression which has the ternary domain `{true, false, incomparable}` is applied pairwise. If the new instance is "better", the old one will be dropped, and vice versa. The consumption policy is implemented by the `<previous>` expression. If the new instance is an extension of a less complete instance, and has not been dropped yet, then the `<previous>` expression determines whether the less complete instance be dropped, or the new one, or neither. All expressions are defined on `predicates` and the `predicates` can include `computations`. The procedure in pseudo-code is given in Algorithm 1.

```
<cd ...>
 ...
 <preferences>
  <preference>
  [<self>reference to expression</self>]
  [<previous>reference to expression</previous>]
  [<other>reference to expression</other>]
  [<better>reference to expression</better>]
  <preference>
  ...
 </preferences>
</cd>
```

Listing 2: Preferences in CDs

The CDL provides short-cut predicates for use in `preference` predicates. In the `self` condition, a predicate `complete` tests whether the instance is complete. In the consumption context, a complete instance may delete all incomplete instances that share some sources in their inducing sets. In the `test` condition, a predicate `equivalent` tests whether both compared instances have the same unmapped `sources` in their *inducing sets*.

4.5 Extensions to the CDL

Three additional capabilities of the detection algorithm are aggregation, data enrichment and the initiation of temporal events.

Aggregation: A common operation on attribute values is aggregation over an unspecified number of events that occur between an initiating and a terminating event. Therefore, in a CD, a source can be defined to hold a number of messages of one type. This source is called a `collection` (see Listing 1). All sources in a collection must match all conditions that are defined on the `collection`. When calculations like sums, products, or the `max` and `min` operators refer to the source, the operator is applied to all individual messages. This behaviour can be used to calculate a minimal, maximal, average or other value from a numeric attribute.

Algorithm 1 Disambiguation of correlation detection

Require: Reception of a source s
1: $I \leftarrow$ { inducing set templates that contain the source type of s }
2: **for all** $i \in I$ **do**
3: $Instances \leftarrow$ { instances with an inducing set that lacks a source of the type of s }
4: **for all** $inst \in Instances$ **do**
5: create a **new instance** $inst'$ by adding s to the inducing set of $inst$
6: evaluate all `computations` that can now be evaluated
7: evaluate all `predicates` that can now be evaluated
8: **if** a `predicate` makes a successful detection impossible **then**
9: delete $inst'$
10: continue
11: $P \leftarrow$ { all `preferences` }
12: $P_{self} \leftarrow P \backslash$ { `preferences` where the `self`-expression evaluates to `false` for $inst'$ }
13: bool $best \leftarrow$ `unknown`
14: **for all** $inst'' \in$ { instances of the current inducing set } **do**
15: **if** the `better` expression evaluates to `true` **then**
16: $best \leftarrow$ `true`
17: delete $inst''$
18: **else if** the `better` expression evaluates to `false` **then**
19: $best \leftarrow$ `false`
20: **if** $best ==$`false` **then**
21: delete $inst'$
22: **else if** a `previous` expression exists **then**
23: **if** the `previous` expression evaluates to `true` **then**
24: delete $inst$
25: **else if** the `previous` expression evaluates to `false` **then**
26: delete $inst'$

Data enrichment: Event messages may lack information about the event or its context which is required for CEP [32]. In the provided example, the *disinfect* and the *door* messages might just contain the identifiers of the badge and reader that published the reading event. The message's content needs to by enriched with the physician's name and the reader's location. Both information can be accessed by querying a database with the tag's and the reader's ID as search keys. *Cordies* can additionally enrich detected messages with data that depends on the correlator node's context. This allows, for instance, to temporally block the detection of specific correlations when the device is in a certain context, e.g., in presence of a power shortage or when it is located in a special region.

Initiation of temporal events: Any source that is defined in the CD can trigger a time event for an absolute or relative time in the future which will be created by the *Cordies* node. As time events are again sources, occurring time events can also trigger other time events, if all other conditions hold.

5. DISTRIBUTION

Being distributed itself, *Cordies* provides CEP in a distributed manner. Two capabilities that support *Cordies*' distribution are discussed in the next subsections: first, the

divisibility of CDs into a number of CDs with desired properties (Section 5.1); second, their decentrally managed placement on *Cordies* nodes (Section 5.2).

5.1 CD divisibility

Resource constraints or usage policies for executing devices and the communication may require to divide CDs into sub-expressions and execute them separately. At the device scope, storing a smaller number of incomplete composite event instances consumes less memory, and dropping unsuccessful instances before performing expensive calculations can save CPU cycles. At the network scope, a placement considering the CD's *selectivity* (i.e. the ratio between received and published events) can reduce network load.

Traditional CEP algebras often generate structures that can be easily divided into parts—e.g., operator trees into sub-trees—where each part can be executed separately. However, only a considerate generation of sub-expressions allows to form sub-expressions which reduce network, CPU and memory usage. The CDL was designed to inherently support a division scheme which creates sub-expressions that can be placed following optimisation metrics on network usage and resource consumption. The proposed division scheme considers the trade-off between the selectivity and the computational overhead induced by CDs. We first describe the division algorithm and give an example below.

The algorithm computes the *inducing set templates* (Section 4.3.1) and `computation`s for each `predicate` and groups the predicates with matching sets of `source`s and `computation`s. Using a combined metric for the selectivity of `predicates` and the computational overhead of `computations`, predicate groups are separated into *layers* where selectivity decreases and computational overhead increases from the lowest to the highest layer. The idea is that predicates in lower layers will form separate CD expressions which can be executed close to the event publishers. Then, the intersection between the *inducing set templates* of predicates in adjacent layers are analysed. If the unions of intersecting and non-intersecting *inducing set templates* are disjoint, the upper layer will be split again. The final step of the algorithm determines new intermediate complex event messages and re-use relationships and aggregates predicates to CDs. The result is a division of the CD's `predicates` and `computations` into CDs that are rather distinct in selectivity, computational overhead, and subscribed event messages, which allows for an optimising placement.

We demonstrate the CD division algorithm with the example from Section 4.1. The algorithm creates five groups of `predicates` with common *inducing set template*s and `computations`:

No.	Predicates	Ind. Set Templ.	Computations
1)	P_1, P_3	$\{S_1, S_3\}$	C_1
2)	P_8	$\{S_1, S_5\}$	—
3)	P_2, P_5	$\{S_2, S_3\}$	C_2
4)	P_6, P_4	$\{S_3, S_4\}$	C_3
5)	P_7	$\{S_4, S_5\}$	C_4

We use a simple metric: the predicate selectivity is high for equality tests and medium for the other comparisons, and the computational overhead is equal for C_1 – C_4 (C_5 does not appear in predicates). This leads to three layers:

Layer:	L1	L2	L3
Sets:	2)	1), 3), 4)	5)

The analysis of *inducing set template* intersection yields an intersection of Sets 2) and 1), so that L2 will be split in 2 layers—L2a containing Set 1), L2b containing Sets 3) and 4). Layer 3 cannot be split. As a result, Set 2) should be evaluated separately and publish a new event S_6. It includes S_1 and S_5, so Sets 1) and 5) can re-use S_6. Sets 3) and 4) are aggregated into one CD providing a new event S_7 which includes S_2, S_3 and S_4. The events for Sets 1) and 5) are now completely contained in S_6 and S_7 so that both can be aggregated into one CD.

The result of the division algorithm are three new CDs with disjoint sets of `predicates`, `computations` and `sources` where CD3 publishes the required composite event:

CD:	CD1	CD2	CD3
Sources:	$\{S_1, S_5\}$	$\{S_2, S_3, S_4\}$	$\{S_6, S_7\}$

The new CDs can be placed independently so that the more selective CD1 and CD2 can be placed closer to the respective publishers. The freedom to transfer the evaluation of *arbitrary* predicates to new CDs directly exploits the CDL's way of describing correlations of events using predicates on event attributes.

5.2 CD placement

While CD division supports efficient distributed event correlation detection, it also creates dependencies between the newly created CDs, additional to the dependencies that already existed to publishers and subscribers of the CD. In the example from Section 5.1, CD1 and CD2 depend on their publishers, and subscribers depend on CD3. CD division added new re-use relations between CD1 and CD3 and between CD2 and CD3. The CD placement framework ensures that relations between CDs are duly considered when assigning CDs to *Cordies* nodes.

Table 3 shows several relations of interest that constrain the absolute or relative assignment of CDs to *Cordies* nodes. These constraints are enforced with the aid of an additional interface that *Cordies* provides: Restriction Graphs.

Arity	Relation
	Relations on CDs
1	unique (no duplicates) blockable (while a duplicate runs) immovable (no migration) time needed to process
2	Minimum/maximum distance (e.g. delay metric) Minimum bandwidth
n	Collocation (placement in the same domain)
	Relations on CDs and Cordies *nodes*
2	node's real-time capabilities node's available security methods
	Relations on CDs and executing devices
2	device's available memory (stable, RAM) device's CPU power device's network interface technology device's type of energy supply device's type of mobility device's reliability device's identity

Table 3: Exemplary Relations

5.2.1 Restriction Graphs

A *Restriction Graph* (RG) is a document that contains CDs and restrictions on the CDs' absolute or relative placement in the network of *Cordies* nodes. A RG represents the counterpart to restrictions imposed by constrained network and device resources because it defends the application's interests for the execution of CDs.

Listing 3 shows the structure of a RG. An `operator` contains a CD to be run and references to all restrictions that apply to its assignment to a *Cordies* node. A `pin` is the identifier of a device (mostly hosting a publisher or a subscriber) that is in a relation with the CD. The `restrictions` are selected from predefined types and can be parametrised.

```
<restrictionGraph id="..." version="...">
 <operators>
  <operator>
   <cd ...> ... </cd>
  [<restrictionReference ref="..." />]
  [...]
  </operator>
 [...]
 </operators>
[<pins>
 ...
 </pins>]
[<restrictions>
  <restriction type="..." ...>...</restriction>
 [...]
 </restrictions>]
</restrictionGraph>
```

Listing 3: Restriction Graph structure

Upon reception of a RG, a *Cordies* node (called *initiator*) extracts the CDs and runs a placement algorithm to map CDs to *Cordies* nodes in compliance with the restrictions defined in the RG. When a placement is found, the initiator assigns the CDs to the selected nodes. The target nodes execute the CDs, subscribe to events and advertise their composite events, thus completing a directed acyclic graph (DAG) of CD re-use within the *Cordies* network.

Along with the CD, the initiator provides target nodes with a *local RG* (LRG) which consists of assigned CDs and the RG's restrictions that are defined on these CDs. Target nodes ensure the permanent compliance with the LRG autonomously. Upon detection of a violation, nodes locally attempt a re-placement of the CD to a node where the restrictions are satisfied. The permanent compliance of the evaluation of all CDs in the original RG is therefore ensured in a distributed manner rather than by the initiator, which makes the scheme scalable for large DAGs of CD re-use.

The CD assignment process includes a best-effort duplicate prevention scheme. The initiator first checks the CDs from the RG against known advertisements (*"active CDs"*). If a CD in the RG matches an advertisement, the initiator maps the CD to the advertiser. The advertiser, upon reception of the CD and the correspondent LRG, maintains restriction compliance of the CD with respect to the previous LRG combined with the new one. It also attempts re-placement for the combined LRG.

5.2.2 Placement algorithms

The *Cordies* placement framework allows to plug in user-defined placement algorithms. Popular algorithms use combinations of delay and bandwidth usage as optimisation criterion [29, 27]. However, in order to support the extended restriction semantics from the RG concept we introduced in the previous section, we have implemented own placement strategies for the initial placement of a RG and for the subsequent dynamical re-placement. Both strategies are assisted by CLIO [16], a distributed service that provides cross-layer information about device resources and network state.

Initial placement. Initial placement (Algorithm 2) assigns a RG of moderate size to nodes from a probably large network. This requires a comparison of the RG's restrictions with the resources provided by *Cordies* nodes and the communication network. The initiator accesses resource information reactively (lines 7–11). It puts individual CDs out for tender by publishing tender notices and simultaneously subscribes to corresponding bidding messages. *Cordies* nodes that can evaluate CDs subscribe to *tender notices*, using their locally available resources as filter, and receive only tender notices that match these resources. Upon reception of a tender notice, a node checks for additional network resources required in the notice and publishes a bidding message if all restrictions are provided for.

Algorithm 2 Initial placement

```
1: X ← {CD_1, ..., CD_m}
2: Levels ← create_levels(X) // heuristic 1
3: for all l ∈ Levels do
4:    if l == 0 then
5:       continue
6:    X_l ← { CDs of level l }
7:    // The following for all is executed in parallel
8:    for all cd ∈ X_l do
9:       publish_tender() // tender msg contains restrictions
         to all CDs from lower level
10:      while not timeout do
11:         receive_biddings() // fill cd's domain
12:   C_l ← { all restrictions between CDs on level l }
13:   for all r ∈ C_l do
14:      clean_domains() // AC-3 algorithm on domains of
         CDs that are involved in r
15:   D_l ← { all domains of CDs on level l }
16:   for all d ∈ D_l do
17:      order_domain(d) // heuristic 2
18: backtrack(Levels) // start backtracking on level 0
```

The initiator then decides the assignment of CDs to bidders so that the RG's restrictions are not violated (called the *RG placement problem*, RGPP). The RGPP can be mapped to the constraint satisfaction problem (CSP) [30] (see Table 4). A CSP is defined on a set X of variables where each variable x_i has a domain D_i of potential values, and a set R of constraints on the actual values. A solution to a CSP is a configuration of values that does not violate the constraints. A known strategy solve a CSP is backtracking (line 18).

We use heuristics and optimisations to address the problem that backtracking needs exponential time for the test of all possible placements. The first heuristic (lines 2–6) re-

CSP	RGPP
$X = \{x_1, \ldots, x_m\}$ set of variables	$X = \{CD_1, \ldots, CD_m\}$ set of *Cordies* CDs
$D = \{D_1, \ldots, D_m\}$ domains of the variables	$D = D_1 = \cdots = D_m = \{node_1, \ldots, node_n\}$ set of *Cordies* nodes
$C = \{C_1, \ldots, C_k\}$ set of constraints	$C = \{R_1, \ldots, R_k\}$ set of restrictions from RG

Table 4: Mapping from RGPP to CSP

duces the overhead for backtracking by ordering the CDs with decreasing strictness of the restrictions on their placement, starting with a level 0 where the *pins* are located. The idea is that for very restricted CDs the assignment space is low, so that early assignments which are decided for these CDs are unlikely to be reversed and therefore will probably not require backstepping.

The second heuristic (lines 15–17) improves the first result of the backtracking algorithm by ordering the CD's domain, i.e. the *Cordies* nodes that a CD can potentially be mapped to. The idea is that when the first solution is "good", backtracking can terminate after finding this solution without having to run to its end and compare all found solutions.

The domain ordering metric is *stability*. A stable mapping of a CD to a node is one that will probably not be revised very soon. The metric is defined as follows. Let $D_{CD} = \{d_1, \ldots, d_n\}$ be the potential *Cordies* nodes that the CD can be mapped to. For each node d_i a bidding message m_i has been received from the node. The message is a vector of n resource items that equal or exceed the resources required by a RG restriction. The corresponding vector of resource restrictions is r. We define two functions

$$f(m_i[j]) := \begin{cases} \frac{m_i[j]}{r[j]} \text{ if } r[j] \text{ was to exceed} \\ \frac{r[j]}{m_i[j]} \text{ if } m_i[j] \text{ was to exceed} \end{cases} \quad (1)$$

$$g(m_i) := min(f(m_i[1]), f(m_i[2]), \ldots, f(m_i[n])) \quad (2)$$

Two vectors are defined with

$$\vec{u}_i = \begin{pmatrix} f(m_i[1]) \\ \ldots \\ f(m_i[n]) \end{pmatrix} \text{ and } \vec{v}_i = \begin{pmatrix} f(m_i[1]) - g(m_i) \\ \ldots \\ f(m_i[n]) - g(m_i) \end{pmatrix} \quad (3)$$

Now, each candidate in a domain receives a weight

$$w_i = \sigma S_i - \tau T_i \text{ with } S_i = \frac{1}{n}\sum_{j=1}^{n} \vec{u}_i[j] \text{ and } T_i = \frac{1}{n}\sum_{j=1}^{n} \vec{v}_i[j] \quad (4)$$

S_i measures how far node d_i exceeds the required resource, T_i measures the balance of the node's resources and σ and τ are parameters. The larger the weight w_i, the more probable is a over-fulfilment of the requested resources and the more stable is the solution if d_i is selected.

Finally, we employ the AC-3 algorithm [23] to reduce the search space of backtracking by eliminating nodes that cannot lead to an overall solution (lines 12–14).

Monitored CD execution. Cordies runs in a dynamic environment where resources change and nodes disappear. The CD execution itself and its compliance to the LRG are therefore continuously monitored. The execution of a CD is monitored by the initiator of its RG. The initiator checks periodically whether the corresponding advertisements exist and whether they are recent. If a CD is missing, the initiator assigns the CD again to a node selected by a placement algorithm. The loss of the initiator itself is detected by the application, which then charges another initiator with the RG. Deploying a CD or a RG a second time is allowed because of duplicate detection (see Section 5.2.1).

The node running a CD ensures the compliance of the CD execution with the RG autonomously. If a restriction cannot be satisfied, the node has to consider a new placement for the CD. This placement differs from the placement of a RG: 1) there is just one CD to place; 2) a quick solution is desirable; 3) the target node will be selected from the neighbourhood rather than the whole network in order to keep effects on the overall operation of *Cordies* low. Therefore we opted against a backtracking-based algorithm and implemented a placement algorithm that is based on incomplete, proactively collected resource information.

Resource data about potential target nodes is collected using the gossip-based *Resource Collection Protocol* (RCP) that we developed (Algorithm 3). Nodes running RCP periodically send local device and network resource data to their neighbours. Upon reception of a RCP message, a node stores the data (lines 2–5, 8) and rates its benefit following three metrics: First, a comparison of the resources with the best known resources; second, a rating of the volatility of the resource over time; and third, a rating of the balance in the quality of different resources of the same node (lines 6, 7). The combined rating results in a value in the interval $(0, 1]$ and determines the probability with which the RCP message is forwarded to neighbours (lines 9, 10). While information about the best resources is thus distributed network-wide, the probability for the distribution of mediocre information decreases exponentially with each hop.

When a new local placement is inevitable, the executing node selects a target node from its local view which complies best with the restrictions of the violating CD. Missing information about device resources and network behaviour is accessed using the cross-layer information service.

Algorithm 3 RCP message forwarding algorithm

```
1:  B = ∅ // benefit values
2:  while RCP message m about node n received via node f do
3:      R ← { all resources from m}
4:      for all r ∈ R do
5:          replace previous RCP information about n with new one
6:          compute benefit b(r) ∈ (0, 1] from r
7:          B = B ∪ {w_r(b(r))} // add weighted benefit from r
8:      store path resources to n via f
9:      b = compute_benefit(B) // b ∈ (0, 1], overall benefit of n
10:     forward m to other neighbours with probability b
```

6. EVALUATION

We implemented parts of *Cordies* concepts for the SpoVNet project [36]. This section addresses the initial RG placement algorithm which we evaluated for correctness and to measure its success rate.

The initial placement algorithm relies on two subsystems: an event notification service for the bidding procedure, and a resource observation system. Motivated by the RG, we employ a latency-aware publish/subscribe service that considers latency requirements for individual subscribers [35]. There would be little sense in placing CDs on nodes with a certain communication latency if the publish/subscribe service did not actually restructure to provide the required latency on the application layer.

Resource information is provided by the distributed cross-layer information provider CLIO [16]. CLIO accepts one-time and continuous queries for local and remote resources by a local interface. All *Cordies* nodes run continuous queries on their local resources and adapt their subscriptions to the bidding process as soon as resources change. Nodes that participate in a bidding process use one-time queries to access link resource information to neighbours that are specified by the initiator.

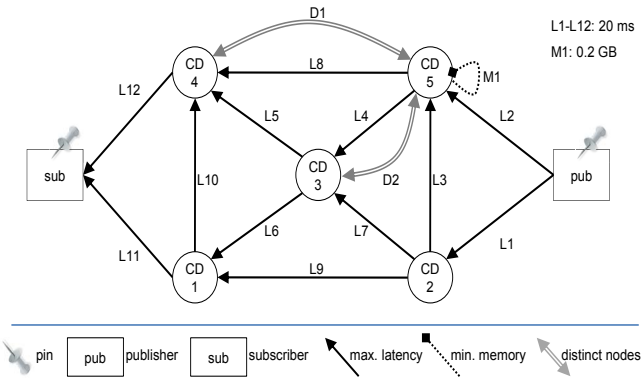

Figure 3: RGs for evaluation

Node	Resources	Assigned CDs			
	RAM	Var. A	Var. B	Var. C	Var. D
1[a]	0.8 − 1.4 GB	1, 2, 5	1, 2, 3, 4	4	
2[b]	0.3 − 0.4 GB	3		2, 3	
3	0.4 GB		5		1
4	0.4 GB			1, 5	2, 3, 4
5	0.2 GB				
6	0.4 GB	4			5
7	0.1 − 0.2 GB				

[a] pinned subscriber [b] pinned publisher

Average latencies:

Pairs	1-2	1-{3–5}	1-6	{1,2}-7	2-{3–6}
Latency (ms)	9	6	4	24	5
Pairs	{3–5}-7	3-{4,5}	{3–5}-6	4-5	6-7
Latency (ms)	21	1	2	1	22

Table 5: Placement results

We installed the placement framework on 7 test-bed nodes and executed the initial placement of a complex RG (Figure 3). The CD contains latency restrictions, `distinct` restrictions that require to place the affected CDs on different nodes, and unary restrictions on available memory. Pins (stationery nodes) are included in the RG to represent subscribers and publishers.

In 75% of all runs, the algorithm found a valid solution without any backstepping. In other cases, up to 14 backsteps were necessary, which is still little in comparison to the 16.807 different assignments that the backtracking algorithm could possibly test. Table 5 shows four different variants of assignments that were found by the algorithm because the available resources on test-bed nodes were volatile. Variant D occurred only once and represents an abnormal case in which the cross layer information subsystem at nodes 1 and 2 yielded no results and therefore both nodes did not participate in the bidding procedure.

7. CONCLUSION

In this paper, we have shown the benefits of an expressive algebra to specify arbitrary event relations on events with previously unknown signatures so that situation detection can be initiated dynamically. We have proposed the CDL language which allows to define any operator semantics on the level of event attributes. Unlike other CEP languages, the CDL does not focus on temporal relations with rudimentary support for other relations. Instead, any relations that can be defined on event attributes and even on context attributes of the event or the correlation node can be described at the same level of ease. The language also extends previously known approaches regarding the control of ambiguity in the detection of correlated events.

Furthermore, we have discussed the necessity to distribute correlation detection functionality. We verified that the CDL supports a distribution that is governed by efficiency and resource consumption considerations instead of the structure of the expression. We have presented a way to manage the initial and continuous placement of sets of correlation descriptions so that not only node and network constraints are satisfied but also requirements of the correlation detection.

In the future, we plan to further investigate the benefits of the CDL and the placement framework. For instance, we have previously shown how users can gain confidence in correlation detection results although event attributes are imprecise and uncertain [18]. Probability theory blends seamlessly into the *Cordies* correlation detection algorithm because the handling of probability distributions for `computation` results and of probability measures for `predicates` and `expressions` are well understood. We plan to integrate the framework for continuous placement with the latency-aware publish/subscribe system [35] so that the influence of event latency on the quality of correlation detection can be expressed in probabilistic statements as well.

Acknowledgements

This work was partially funded by the SpoVNet project of Baden-Württemberg Stiftung gGmbH. We are grateful for the comments by our reviewers and Prof. Adrian Paschke.

8. REFERENCES

[1] R. Adaikkalavan and S. Chakravarthy. SnoopIB: interval-based event specification and detection for active databases. *Data Knowl. Eng.*, 59(1):139–165, 2006.

[2] R. Adaikkalavan and S. Chakravarthy. Event specification and processing for advanced applications: Generalization and formalization. In *DEXA*, pages 369–379, 2007.

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

[4] J. F. Allen. Maintaining knowledge about temporal intervals. *Commun. ACM*, 26(11):832–843, 1983.

[5] J. Bailey and S. Mikulás. Expressiveness issues and decision problems for active database event queries. In *ICDT '01: Proc. 8th Int. Conf. on Database Theory*, pages 68–82. Springer-Verlag, 2001.

[6] C. Bornhövd and A. P. Buchmann. CREAM: An infrastructure for distributed, heterogeneous event-based applications, 2003.

[7] F. Bry and M. Eckert. Rule-based composite event queries: The language xchangeeq and its semantics. *Lecture Notes in Computer Science*, 4524:16–30, 2007.

[8] A. T. Campbell, S. B. Eisenman, N. D. Lane, E. Miluzzo, R. A. Peterson, H. Lu, X. Zheng, M. Musolesi, K. Fodor, and G.-S. Ahn. The rise of people-centric sensing. *IEEE Internet Computing*, 12(4):12–21, 2008.

[9] J. Carlson and B. Lisper. An event detection algebra for reactive systems. In *EMSOFT '04: Proc. 4th ACM*

Int. conf. on Embedded software, pages 147–154. ACM, 2004.

[10] S. Chakravarthy and D. Mishra. Snoop: An expressive event specification language for active databases. *Data Knowledge Engineering*, 14(1):1–26, 1994.

[11] C.-H. Chen-Ritzo, C. Harrison, J. Paraszczak, and F. Parr. Instrumenting the planet. *IBM J. Res. Dev.*, 53(3):1:1–1:16, 2009.

[12] S. Courtenage. Specifying and detecting composite events in content-based publish/subscribe systems. *Proc. 22nd Int. Conf. on Distributed Computing Systems Workshops*, pages 602–607, 2002.

[13] M. J. Franklin, S. R. Jeffery, S. Krishnamurthy, and F. Reiss. Design considerations for high fan-in systems: The HiFi approach. In *CIDR*, pages 290–304, 2005.

[14] S. Gatziu and K. Dittrich. Detecting composite events in active database systems using petri nets. *Proc. 4th Int. Workshop on Research Issues in Data Engineering*, pages 2–9, 1994.

[15] N. H. Gehani, H. V. Jagadish, and O. Shmueli. Event specification in an active object-oriented database. In *SIGMOD '92: Proc. ACM Int. Conf. on Management of Data*, pages 81–90. ACM, 1992.

[16] D. Haage, R. Holz, H. Niedermayer, and P. Laskov. CLIO – a cross-layer information service for overlay network optimization. In *Kommunikation in Verteilten Systemen (KiVS) 2009*, 2009.

[17] A. Hinze and A. Voisard. A parameterized algebra for event notification services. In *TIME '02: Proc. 9th Int. Symposium on Temporal Representation and Reasoning*, page 61. IEEE Computer Society, 2002.

[18] G. G. Koch, B. Koldehofe, and K. Rothermel. Higher confidence in event correlation using uncertainty restrictions. In *Proc. 28th IEEE Int. Conf. on Distributed Computing Systems Workshops (ICDCSW '08)*. IEEE Computer Society, 2008.

[19] G. Li and H.-A. Jacobsen. Composite subscriptions in content-based publish/subscribe systems. In *Middleware 2005*, number 3970 in Lecture Notes in Computer Science, pages 249–269. Springer, 2005.

[20] G. Li, V. Muthusamy, and H.-A. Jacobsen. Adaptive content-based routing in general overlay topologies. In *Middleware '08: Proc. 9th ACM/IFIP/USENIX Int. Conf. on Middleware*, pages 1–21. Springer, 2008.

[21] C. Liebig, M. Cilia, and A. Buchmann. Event composition in time-dependent distributed systems. In *COOPIS '99: Proc. 4th IECIS Int. Conf. on Cooperative Information Systems*, page 70. IEEE Computer Society, 1999.

[22] D. C. Luckham. *The Power of Events: An Introduction to Complex Event Processing in Distributed Enterprise Systems*. Addison-Wesley Longman Publishing Co., Inc., 2001.

[23] A. Mackworth. Consistency in networks of relations. *Artificial Intelligence*, 8(1):99–118, 1977. Reprinted in Readings in Artificial Intelligence, B. L. Webber and N. J. Nilsson (eds.), Tioga Publ. Col., pp. 69-78, 1981.

[24] M. Mansouri-Samani and M. Sloman. GEM: A generalized event monitoring language for distributed systems. *IEE/IOP/BCS Distributed Systems Engineering Journal*, 4:96– 108, 1997.

[25] A. Nagargadde, S. Varadarajan, and K. Ramamritham. Semantic characterization of real world events. In *DASFAA*, pages 675–687. Springer, 2005.

[26] A. Nagargadde, S. Varadarajan, and K. Ramamritham. Representation and processing of information related to real world events. *Know.-Based Syst.*, 20(1):1–16, 2007.

[27] P. Pietzuch, J. Ledlie, J. Shneidman, M. Roussopoulos, M. Welsh, and M. Seltzer. Network-aware operator placement for stream-processing systems. In *ICDE '06: Proc. 22nd Int. Conf. on Data Engineering*, page 49. IEEE Computer Society, 2006.

[28] P. R. Pietzuch, B. Shand, and J. Bacon. Composite event detection as a generic middleware extension. *Network, IEEE*, 18:44–55, 2004.

[29] S. Rizou, F. Dürr, and K. Rothermel. Solving the Multi-operator Placement Problem in Large-Scale Operator Networks. Technical Report 2009/03, University of Stuttgart, Collaborative Research Center 627, 2009.

[30] S. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Prentice-Hall, Englewood Cliffs, NJ, 2nd edition edition, 2003.

[31] C. Sánchez, S. Sankaranarayanan, H. Sipma, T. Zhang, D. Dill, and Z. Manna. Event correlation: Language and semantics. *Lecture Notes in Computer Science*, 2855:323–339, 2003.

[32] B. Schilling, B. Koldehofe, U. Pletat, and K. Rothermel. Distributed heterogeneous event processing: Enhancing scalability and interoperability of cep in an industrial context. In *DEBS '10: Proc. 4th Int. Conf. on Distributed Event-based Systems*. ACM, 2010.

[33] S. Schwiderski. *Monitoring the Behaviour of Distributed Systems*. PhD thesis, Selwyn College, University of Cambridge, 1996.

[34] S. Schwiderski-Grosche and K. Moody. The SpaTeC composite event language for spatio-temporal reasoning in mobile systems. In *DEBS '09: Proc. 3rd ACM Int. Conf. on Distributed Event-Based Systems*, pages 1–12. ACM, 2009.

[35] M. A. Tariq, G. G. Koch, B. Koldehofe, and K. Rothermel. Dynamic publish/subscribe to meet subscriber-defined delay and bandwidth constraints. In *Euro-Par'10: Proc. 16th Int. Euro-Par Conf.* Springer, 2010.

[36] O. Waldhorst, C. Blankenhorn, D. Haage, R. Holz, G. Koch, B. Koldehofe, F. Lampi, C. Mayer, and S. Mies. Spontaneous Virtual Networks: On the Road towards the Internet's Next Generation. *it — Information Technology Special Issue on Next Generation Internet*, 50(6), Dec. 2008.

[37] E. Yoneki and J. Bacon. Unified semantics for event correlation over time and space in hybrid network environments. In *On the Move to Meaningful Internet Systems 2005: CoopIS, DOA, and ODBASE*, volume 3760, pages 366—384. Springer Verlag, 2005.

[38] D. Zimmer and R. Unland. On the semantics of complex events in active database management systems. *Int. Conf. on Data Engineering*, 0:392, 1999.