Event Composition in Time-dependent Distributed Systems

C. Liebig, M. Cilia†, A. Buchmann

Database Research Group - Department of Computer Science
Darmstadt University of Technology - Darmstadt, Germany
{chris, cilia, buchmann}@dvs1.informatik.tu-darmstadt.de

Abstract

Many interesting application systems, ranging from workflow management and CSCW to air traffic control, are event-driven and time-dependent and must interact with heterogeneous components in the real world. Event services are used to glue together distributed components. They assume a virtual global time base to trigger actions and to order events. The notion of a global time that is provided by synchronized local clocks in distributed systems has a fundamental impact on the semantics of event-driven systems, especially the composition of events. The well studied 2g-precedence model, which assumes that the granularity of global time-base g can be derived from a priori known and bounded precision of local clocks may not be suitable for the Internet where the accuracy and external synchronization of local clocks is best effort and cannot be guaranteed because of large transmission delay variations and phases of disconnection. In this paper we introduce a mechanism based on NTP synchronized local clocks with global reference time injected by GPS time servers. We argue that timestamps of events can be related to global reference time with bounded accuracy and propose that event timestamps are modeled using accuracy intervals. We present algorithms for event composition and event consumption which make use of accuracy interval based timestamping and illustrate the problems that arise due to inaccuracy and message transmission delays.

I. Introduction

Event-based computing is an emerging paradigm for composing applications in open, heterogeneous distributed environments [4,23,20,13]. Applications like workflow management [7,19,14], CSCW [5] and monitoring applications ranging from Air Traffic Control [3,29] to Health Care Systems [12] may be constructed by leveraging event services for detection and distribution of events in a publish/subscribe manner. The use of *generic* event services requires that the semantics of event services that is presented to the application developer be not only formally specified [45,49] but also unambiguous. Failing to do so may cause mission-critical applications to malfunction or behave indeterministically, and may result in unreliable software and impose unacceptable risks.

The use of absolute and relative temporal events to trigger actions, the need to measure duration of activities, and the detection and composition of events that may originate in distributed components that are loosely coupled render distributed event-driven systems time-dependent. A well defined event service depends on three basic factors: the proper interpretation of time, the adoption of partial order of events and the consideration of transmission delays between producers and consumers of events. In order to describe and detect complex situations, advanced event services provide the notion of composite events. Typically we are interested in causal dependencies between real-world happenings or computations. Temporal order is a prerequisite for causal order. Therefore, potential causality can be detected - or excluded - when examining the order of event occurrences. However, occurrence time and global order of events can only be determined by an omniscient external observer. In practice, detection and timestamping of events is delayed from the instant of occurrence. Additionally, time as provided by a distributed time service is imprecise with respect to clock readings at different nodes and inaccurate with respect to physical time. As a consequence, timestamps are inherently inaccurate and may distort the real order of occurrence of events. The inability to provide precise and accurate timestamps has additional impact on event consumption, i.e. the selection of events that are to be composed. Consumption policies like *recent* and *chronicle* rely upon the temporal order of events when selecting the latest events (recent) or the oldest events (chronicle) out of the event stream. Furthermore, event consumption must contemplate variable transmission delays, especially in the case of multiple, independent remote publishers.

In this paper we focus on timestamping and composition of events in large scale, loosely coupled, distributed systems without centralized management, like the Internet. Unpredictable bounds and large variations on message transmission delays, possible phases of disconnection and independent failure modes are characteristic for such an environment and complicate the realization of a general purpose event service. In particular, it is not possible to determine a-priori the precision bounds for all local clocks in the system. Therefore, we

† Also ISISTAN, Faculty of Sciences, UNICEN, Tandil, Argentina.

argue that ordering of events based on a sparse time base or the 2g-precedence model does not scale up to the Internet. In our solution we make use of the Network Time Protocol (NTP).

The remainder of this paper is organized as follows. Next, an overview of related work is presented. Section III. introduces the concept of global time based upon synchronized local clocks. We give a brief overview on NTP time services and then present a mechanism for timestamping events based upon accuracy intervals. We introduce an accuracy interval order that is the basis for event composition and consumption. Section IV. shortly describes the architecture of our event service. After that we discuss the implementation of simple event composition operators and point out the potential pitfalls due to the very nature of distributed systems. Finally we address open issues and present current and future work.

II. Related Work

General-purpose event notification services have been proposed recently as part of major *middleware* initiatives [37,38,39,20,31]. However, most of them are restricted to primitive events and do not consider any consumption policies.

Composition of events was proposed together with the concept of Event-Condition-Action rules in active databases [10]. Active databases support composite events but assume the existence of a totally ordered event history, and therefore, are restricted to centralized systems. Active databases handle database events, temporal events, and user-defined events. HiPAC [11] considered ECA rules in general, and provided basic mechanisms for composite event specification. Compose [18] introduced powerful event operators. Snoop [8] introduced a formal definition of primitive and composite events based on a global history log, and four event consumption policies: recent, chronicle, continuous and cumulative. Reach [6] provided mechanisms for efficient detection and composition based on the SAMOS [16] algebra. Ode [22] proposed complex event composition but used timestamps for event identification and required a total ordering. Recent efforts have concentrated on unbundling database functionality to provide, among others, active functionality services through configurable components [17,25]. None of the previously mentioned approaches has addressed properly the problems of global time, imprecise timestamps of events, and composition delays. Instead, they all assume a total ordering of events.

In [27], Lamport presented the *happened before* relation, which defines a partial ordering of events based on the causality principle. An event *a* happened before an event *b* (depicted $a \rightarrow b$) if *a* could have influenced *b*; *a* and *b* are said to be causally dependent. If neither $a \rightarrow b$ nor $b \rightarrow a$, the events are said to be concurrent and causally independent. A system of logical clocks is introduced which assigns a natural number to each event (logical timestamp). Logical clocks are consistent with causality [41]: if $a \rightarrow b$, then *a*'s timestamp is smaller than *b*'s timestamp - the contrary is not true. In [41] the concept of vector time is presented and it is shown that vector time characterizes causality: two events are ordered by vector time iff they are causally dependent. However, neither logical clocks nor vector clocks can deal with causal relations that are established through hidden channels and also can not represent timed real world events. Thus they are not appropriate for open systems.

In [24,47] a global time approximation is proposed, assuming that the maximum time difference between any two clocks at the same instant of time is bounded by δ. The *granularity condition* states that the granularity of the global time-base *g* should not be smaller than δ, $g > \delta$, ensuring that global clocks do not overlap. A global and total order of events can be determined if event timestamps are two or more clock ticks apart, a fact known as *2g-precedence*. If this assumption does not hold in all cases, one has to face partial ordering of events.

Schwiderski [42] 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 may lead to the detection of spurious events.

The Cambridge Event Architecture (CEA) [2] presents the *publish-register-notify* paradigm. Mediators provide the means to compose events. CEA is oriented to support multimedia, mobility, group interaction and composition of heterogeneous software components [5]. The implementation of CEA is based on a proprietary RPC system, limiting interoperability. Recently, COBEA [31] was proposed, which extends the CORBA Event Service [37] with the CEA publish-register-notify paradigm, supporting fault tolerance, composite events, server-side filtering and access control.

In EVE [19,45] an event-based middleware layer is proposed as platform for a workflow enactment system. The workflow is mapped to services and brokers. The behavior of brokers is defined by ECA-rules using composition of distributed events. Specifically, EVE requires chronicle consumption mode of events to correctly interpret workflow notifications.

In CEA, COBEA and EVE, the detection of global composite events is based on Schwiderski's approach.

[49] presents a formal refinement of Schwiderski's approach and extends the Snoop event algebra to support event composition in distributed environments.

The 2g-precedence based approaches cited above do not scale to open systems and still are ambiguous with respect to event consumption.

III. Timestamping and Global Time

We will give a short overview of the concept of global time and distinguish between internal and external clock synchronization algorithms. We then present how we leverage upon a time service like NTP for provision of a global

reference time and introduce the concept of accuracy intervals. We define abstract interfaces for local as well as global clock readings used for timestamping events.

If we are merely interested in relative ordering of events detected at the same node, a monotonically increasing counter, e.g. the local clock reading, might be sufficient. In the real world, we must differentiate between the occurrence of an event and the time it takes until detection. We have to distinguish the case where it can be assured - at the application level - that occurrence and detection of distinct events never overlap such that timestamps at detection time always reflect the order of occurrence. The more realistic scenario is however, that timestamping of local events does not yield a total order because there is uncertainty about occurrence time and detection time of events. We will therefore define a - partial - local order that recognizes this fact and a - partial - global order that additionally respects the inaccuracy which is inherent in the artificial notion of reference time.

A. Clock Synchronization

The instant of time at which an event occurs in the physical world will be called the physical time of the event. Reference time *RT* - as provided by UTC or GPS time - is a granular representation of dense physical time. Note that reference time is a conceptual artifact and inaccurate by nature. In fact GPS time servers carry an error encompassing relativistic effects as well as more significant inaccuracies due to synchronization and clock reading errors.

In order to provide a global timebase in distributed systems, a common solution is to create a virtual clock at each node using a local hardware clock. The clock synchronization problem consists of reaching some degree of mutual consistency between virtual clocks and compensating for hardware clock skew and frequency drift. Note, that perfect synchrony cannot be achieved by the very nature of our universe.

A virtual clock is represented by a function $C(t)$: $RT \rightarrow CT$, $CT \subset RT$ that maps reference time to clock time *CT*. A hardware clock typically consists of an oscillator and a counting register that is incremented by the ticks of the oscillator. The hardware clock has a certain granularity G by which the counter can be incremented. For a local hardware clock to be correct, we require a bounded drift rate:

Linear Envelope:

$$s,t \in RT : s \le t$$
$$(1-\rho)(t-s) - G \le C(t) - C(s) \le (1+\rho)(t-s) + G$$

For most modern hardware clocks the constant ρ is in the order of 10^{-4} to 10^{-6}, i.e. the clock drifts more than 0.06 milliseconds in one minute which compares to 6000 instructions on a 100 MIPS machine.

Internal clock synchronization consists of keeping virtual clocks within some maximum deviation from each other, i.e. for all correct clocks C_i, C_j it is guaranteed:

Precision: $\exists\delta : |C_i(t) - C_j(t)| \le \delta$, $t \in RT$

External clock synchronization aims at maintaining virtual clocks within some maximum deviation from a time reference external to the system, i.e for each correct clock C_i it is guaranteed:

Accuracy: $\exists\alpha : |C_i(t) - t| \le \alpha$, $t \in RT$

Internal clock synchronization algorithms [43,26,30] guarantee precision in case of known bounds on transmission delays of the network. Otherwise, internal clock synchronization is best effort [9,46] and precision δ cannot be a-priori determined for all *t*. As accuracy α always implies precision *2*α, externally synchronized clocks are also internally synchronized. At the opposite, internally synchronized clocks do not necessarily maintain accuracy with respect to external reference time. If accuracy is a requirement, internal clock synchronization algorithms can be integrated with external clock synchronization as in recent hybrid clock synchronization algorithms [15,40,46].

Timestamping based on internal clock synchronization and the application of the 2g-precedence model [42,47] for ordering and composing events does not scale to loosely coupled distributed systems like the Internet. As transmission delays vary significantly and are in general not known a-priori for all nodes of the network, it is not feasible to determine a precision δ that holds for all *t*. For the same reason such an approach is not suitable for mobile environments [44] with long phases of disconnection. In fact, the above approaches merely present viable solutions for systems interconnected by real-time networks or selected broadcast based LANs with restricted load patterns, where at design time it is possible to determine and guarantee a bound on δ for all instants *t* and all virtual clocks of the system [47].

B. Time Service

The Network Time Protocol defines an architecture for a time service and a protocol to distribute accurate time information in a large, unmanaged global-internet environment and is established as an Internet Standard protocol [33]. The participating nodes form a logical synchronization subnet whose levels are called *strata*. Primary servers at stratum 1 are directly connected to a time source such as a radio clock or a GPS receiver and provide accurate UTC reference time with an error ranging from some milliseconds down to a few microseconds [21] - whereas GPS time itself is accurate in the order of 30 nanoseconds [28]. Secondary servers at stratum 2 synchronize their clock with respect to stratum 1 peers plus other servers of stratum 2, servers at stratum 3 synchronize with stratum 2 peers and so on. The synchronization scheme consists of a peer selection algorithm and estimation of the offset for the local clock with respect to reference time provided by the selected peer. The peer selection algorithm chooses the best peer which is supposed to provide reliable and accurate time information. Calculating an estimation for the clock offset is based on exchanging timestamps between peers, as proposed by Cristian [9]. Additionally, statistical filters are applied to a recent sample population which significantly

reduces the error of the estimated offset. A detailed performance study of NTP can be found in [34].

C. Timestamping of Events

NTP provides a reliable error bound, the *synchronization distance*, that accounts for inaccuracies due to clock skew and offset estimation along the path to the primary reference server, plus the inaccuracy of the primary server's clock with respect to reference time. In [35] a new system call `ntp_gettime()` is introduced for reading the virtual global clock that additionally returns a reliable error bound with respect to reference time. The CORBA TimeService [36] proposes an abstract interface that supports clock readings and additionally returns an error bound, the purpose of which is to wrap existing time service implementations such as NTP or DCE TimeService. In the following we will present our abstract view on a clock reading interface for which the above approaches provide a viable implementation. Let us first introduce the notion of accuracy intervals as proposed in [32,40].

Accuracy Interval: We define the accuracy interval with reference point $t_{ref} \in RT$ and accuracy $[\alpha^-;\alpha^+]$; $\alpha^-,\alpha^+ \in RT$ *as:* $I(t_{ref}) \equiv [t_{\text{ref}} - \alpha^- ;\ t_{\text{ref}} + \alpha^+]$.
For convenience we use the shorthand notations $[t_{ref} \pm \alpha]$, $\alpha=[\alpha^-;\alpha^+]$, *lower(*$[\alpha\text{-};\alpha+]$*)*$=\alpha^-$ and *upper(*$[\alpha\text{-};\alpha+]$*)*$=\alpha^+$.

Global Time Service: The global time service provides a function *get_time()* - when called at physical time *t*, *get_time()* returns the reading of the local virtual clock *C(t)* together with a reliable error bound $synchdist_t$.
We require the global time service to be correct.

Correctness of Time Service: If *get_time()* is called at physical time *t* and returns *C(t)* with error $synchdist_t$ then:
$t \in [C(t) - synchdist_t\ ;\ C(t) + synchdist_t]$

Let $t_{occ}(e)$ be the instant of time when event *e* occurred. Actually, it takes some time *ldd* until the event is detected and is assigned a timestamp. We call *ldd* the local detection delay and denote with $t_{det}(e)$ the detection time of the event. In the following, we assume that an individual upper bound *ldd* is known for each node of the system.

Local Detection Delay:
$\exists ldd \in RT : t_{occ}(e) \in [t_{det}(e) - ldd\ ;\ t_{det}(e)]$

The effect of the delay depends largely on the signalling source. For example, the minimum delay in the detection of a local method event is caused by a timer system call. On a SUN SS10 with two CPUs at 55 Mhz the timer system call takes about 5 µsec and it takes about 0,5 µsec on a SUN Ultra II with two CPUs at 300 Mhz, whereas the granularity G of the local clock is 1 µsec in both cases.
In other words, the impact of *ldd* may be insignificant compared to the inaccuracy imposed by the clock granularity on the fast machine. However, on slow machines like the SS10 or in cases where the event is signaled by some external device, *ldd* may be significantly larger then clock granularity and additionally increases the inaccuracy of the global timestamp.

The local detection delay is taken into account by timestamping event *e* as:

Global Timestamp:
$\text{ts}(e) = [C(t_{det}) \pm \alpha]$

$\alpha = [synchdist_{t_{det}} + ldd;\ synchdist_{t_{det}}]$

The fact that the global timestamp *ts(e)* contains $t_{occ}(e)$ can easily be seen from the above definitions, because
$t_{occ}(e) \geq t_{det}(e) - ldd \geq C(t_{det}) - synchdist_{t_{det}} - ldd$
and $t_{occ}(e) \leq t_{det}(e) \leq C(t_{det}) + synchdist_{t_{det}}$.
We denote the length of the error interval α as the *inaccuracy* of the timestamp.

D. Ordering of Events

We define a partial order on accuracy intervals as follows:

Accuracy Interval Order:

$$I_j = [r_j \pm \alpha_j],\ I_k = [r_k \pm \alpha_k]$$
$$I_j < I_k \Leftrightarrow \forall s \in I_j, \forall t \in I_k : s < t$$
$$\Leftrightarrow r_j + \alpha_j^+ < r_k - \alpha_k^-$$

Accuracy interval order is merely a partial order. Obviously there exist accuracy intervals I_j, I_k such that neither $I_j<I_k$ nor $I_k<I_j$ holds. We define the order of two events to be *uncertain* if they cannot be ordered and introduce the notation $I_j \perp I_k \equiv \neg(I_j < I_k) \wedge \neg(I_k < I_j)$. As we cannot decide on the order of events in such cases, the event service should take well defined actions, as we will discuss later on. Depending on the application, the inaccuracy of timestamps can be small with respect to the temporal offset between causally dependent events. In this case, a well defined application should never generate uncertain events. However, if uncertain event orders occur, they should be resolved by application semantics. It should be noted at this point, that the worst resolution policy, i.e. ignoring the uncertainty of event order, does not perform worse then previous approaches discussed in Section II.
With our approach we can guarantee in all cases that:

- situations of uncertain event order are detected and the action taken is well defined
- events are not erroneously ordered.

More precisely, we can guarantee that accuracy interval order is consistent with physical time order, i.e. the following important property holds:

Time Consistent Order: Given events e_j, e_k and
$ts(e_j) = I_j(t_{det}(e_j)),\ ts(e_k) = I_k(t_{det}(e_k))$ then
$I_j < I_k \Rightarrow t_{occ}(e_j) < t_{occ}(e_k)$

This proposition follows directly from the previous definitions of global timestamp and accuracy interval order, under the assumption that the time service is correct.

If the expected values of *synchdist* are sufficiently small, for example when detecting events at a stratum 1 server attached to GPS, it may be sufficient to order events based on ordering of global timestamps, as defined above. In many settings however, event detection runs at nodes of a lower stratum and reading the clock results in large *synch-*

dist values (10-50 msec and more) with respect to the granularity of the local clock. Therefore we additionally provide a mechanism for the relative ordering of events - originating from the same node - based on local clock readings.

We assume that the local clock is monotonically increasing and that clock discipline by NTP uses continuous amortization. Let e_j, e_k be events originating at the same node, then we assign the local clock readings as *local timestamps:*

Local Time Stamp: If e_j is detected at node N with local detection delay *ldd* we define: $lts(e_j) = C(t_{det}(e_j))$.

We are interested in a time consistent order for local timestamps. We know from the definition of local detection delay, that $t_{det}(e_j) < t_{det}(e_k) - ldd \Rightarrow t_{occ}(e_j) < t_{occ}(e_k)$. In other words we have to find a lower bound for the distance $t_{det}(e_k) - t_{det}(e_j)$, which can only be approximated by local clock readings. Let us assume that there are no resynchronizations between the two clock readings, then we know from the linear clock drift, that $C(t) - C(s) \le (1+\rho)(t-s) + G$. Additionally we have to consider rate adjustments by the clock discipline. For simplicity, we assume that there is a known upper bound u for a positive rate adjustment between two resynchronization points. Then we obtain:

$$C(t_{det}(e_k)) - C(t_{det}(e_j)) \le (1+\rho)(t_{det}(e_k) - t_{det}(e_j)) + G + u$$

$$\Rightarrow ldd < \frac{C(t_{det}(e_k)) - C(t_{det}(e_j)) - G - u}{(1+\rho)} \le t_{det}(e_k) - t_{det}(e_j)$$

We now can specify the condition to order local timestamps while considering the local detection delay:

Local Timestamp Order: Let $lts(e_j), lts(e_k)$ be local timestamps of events detected at the same node.

$$lts(e_j) < lts(e_k)$$

$$\Leftrightarrow ldd < \frac{C(t_{det}(e_k)) - C(t_{det}(e_j)) - G - u}{(1+\rho)}$$

We refer to Schmid and Schossmaier [40] for a detailed discussion on how to estimate duration measurements using local clock readings, where they also discuss various models of local clocks and clock discipline mechanisms.

IV. Notification Service

In this section we describe the overall architecture of our event notification service and look into the implementation details of event composition using accuracy interval based timestamping. Fig. 1. depicts the main components of the event notification service.

The architecture is similar to that of a push-style CORBA Notification Service [38]. Producer and consumer of events interact with the event channel through proxy interfaces: *ECPI* (producer) and *ECCI* (consumer). The channel itself is a conceptual artifact realized on top of multicast messaging middleware that provides a subject-based addressing scheme [39]. Producers of events register metadata for event type descriptions with the *EventTypeRepository.* Consumers as well as other producers may query the repository to find out about existing event types. If a subscriber registers interest for some type of event an appropriate *ECCI* proxy will be returned. This proxy is created by an administrative factory object and relays primitive event notifications received by the multicast messaging layer to the consumer. A producer publishes events through the call of *ECPI::signalEvent(Event e)* which also adds a local and global timestamp and the producer name to the event parameters. A consumer may connect directly to the ECCI proxy to be notified of primitive event occurrences. Composite events are detected by specialized ECCI proxies: In the first stage primitive events are captured by *InputNode*s (I), encapsulating the appropriate ECCI, and then passed on to the *CompositionNode* (C) where the operator logic is implemented and consumption takes place. Finally, if a composite event is detected, it is signaled to the consumer. As we will show later, the *CompositionNode* may raise exceptions to inform the application of ambiguities in the case when candidate events cannot be ordered.

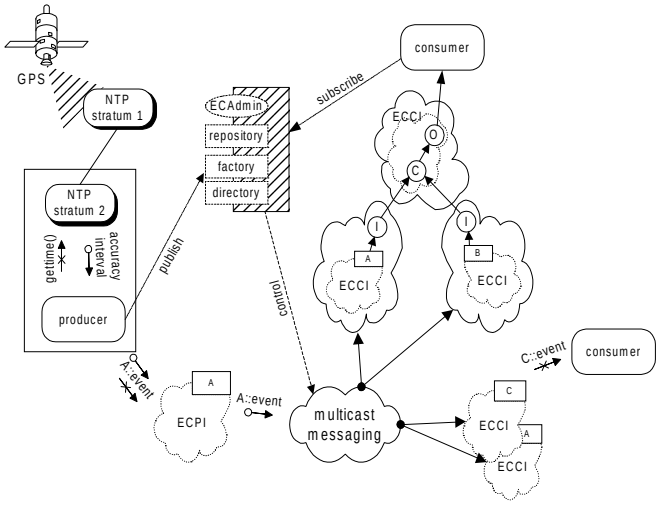

Fig. 1. Notification Service Architecture.

Events are reliably delivered to subscribers by the underlying messaging middleware and it is also guaranteed that events are sent by a producer in the detection order and that this order is preserved by the channel.

A publish/subscribe event service per definition must support many-to-many communication. As a consequence the semantic of group membership impacts the *CompositionNode* subscribers, because we need to know which producers might have sent events that must be considered for composition. We provide two different group membership semantics: atomic membership and weak membership. When using atomic membership, a producer registers with the *DirectoryService* and must not start sending events before all consumers, which are subscribed to the respective type of events, have been notified of the new group member. We leverage on the event service itself to reliably broadcast dedicated control events, such as a group membership change event. When subscribing for some type of event a consumer may also request a list of currently active publishers. In the case of weak membership we delegate to the dynamic discovery protocol provided by the multicast messaging middleware. In that case a publisher can register without blocking at the *DirectoryService*. It is then possible

that some events of the joined publisher arrive late and invalidate former event compositions. Atomic membership prohibits such errors.

As will be discussed in the next section, we introduce a windowing scheme combined with heartbeat events to cope with node failures of consumers and network failures like poor response times or partitioning of the network.

V. Composition and Consumption

To illustrate the impact of timestamp inaccuracy and varying transmission delays on event composition and consumption we will look at the simple composite event expression *A&B*, which depicts the situation that an event of type A *and* an event of type B occurred. Although the logic of the operator does not seem to impose any ordering constraints, consumption of events must be considered. Assume there is one producer P_A for type A events and there are two producers $P1_B$, $P2_B$ for type B events which signal to an *A&B CompositionNode*, as shown below:

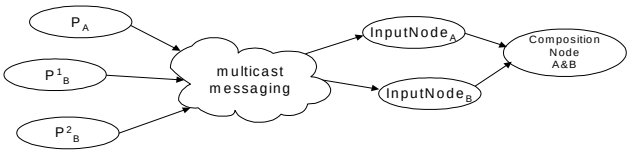

Fig. 2. Scenario.

There can be multiple A events and multiple B events, even from different nodes, that are candidates to make up the composite event. In *chronicle* consumption mode we want to combine the oldest As and Bs. In *recent* consumption mode we are looking for the latest events, i.e. lately occurred events will rule out older ones. In the following, we will assume that the *CompositionNode* contains a partially ordered list for each operand. Let *POList*<A> be a data structure that holds type A events and *POList*<B> the one to hold type B events. The method *POList<>.oldest()* returns the set of oldest events which are those events that are not preceded by any other in the *POList<>*:

$\forall T::e \in POList$<E>.oldest() :

$\neg$ ($\exists$ $e' \in POList$<E>.oldest() : $ts(e') < ts(e)$)

Note that *oldest()* may benefit from the fact that there is only one producer for type A events and there is no need to relate to reference time, as it would be when implementing the sequence operator. The optimization then would be to use the local timestamp order instead of the global timestamp order.

A. Window Mechanism

We mentioned in the beginning, that we have to consider the impact of individual transmission delays. The time diagram shown in Fig. 3. illustrates the problems that may arise. With the arrival of $b^1{}_0$ at time t_1 we detect a tentative composite $a_0 \& b^1{}_0$ event. However, we must consider the possibility that there is another A event on its way, which occurred at approximately the same time as a_0, i.e. $a_0 \perp a_1$. When a_1 arrives at t_2 we now can be sure that a_0 is the oldest A event and must be considered for composition. In the case of B events we have to additionally consider the fact that there are two producers, i.e. when receiving $b^1{}_0$ there could be events both at $P1_B$ and $P2_B$ that have not yet been delivered but would be element of *POList<B>.oldest()*. In general, we require *POList<B>.oldest()* to be *stable* before constructing a composite event. We are using a window mechanism with so called sync-points to separate the history of events as seen by the *CompositionNode* - reflected in the operand POList<> data structures - into the *stable past* and the *unstable past and present* that still are subject to change.

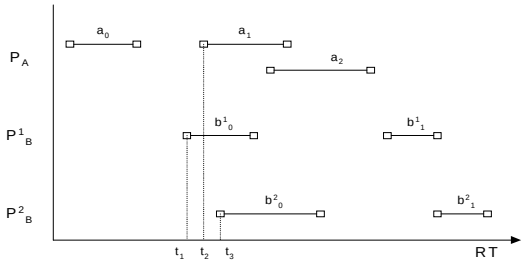

Fig. 3. Time diagram (global timestamps).

We define the local sync-point $lts_{sync}(P_A)$ with respect to a producer P_A to denote the fact that there are no more events *a* detected at P_A that have not been signaled to the *CompositionNode* and $lower(lts(a)) < lts_{sync}(P_A)$. The local sync-point moves on with each event detection and is determined by approximating a local clock value that is at least *ldd* below the local timestamp of the latest event. In a similar way we define the global sync-point $ts_{sync}(P_A)$ of a producer P_A such that there are no more events *a* at P_A that have not been signaled to the *CompositionNode* and $lower(ls(a)) < ts_{sync}(P_A)$. Whereas the local sync point refers to local clock time the global sync-point relates to reference time. Obviously, the global sync-point with respect to a producer P_A is equivalent to the lower end of the global timestamp of the latest detected event. In fact, with each event received by the consumer the respective sync-point windows move along[1]. For example in Fig. 3. the global sync-point for $P1_B$ is $t_1 = lower(b^1{}_0)$ when $b^1{}_0$ is received and moves to $lower(b^1{}_1)$. We call *POList<B>.oldest()* to be stable, if there are no more pending events b such that b would also belong to *POList<B>.oldest()*. If all global sync-points are at the right of the oldest timestamp in *POList<B>.oldest()* then there can be no pending event that intersects with all timestamps in *POList<B>.oldest()*. Without proof we present the formal predicate for stability.

Stability: Given *POList<E>* and the known set of producers for E events, PR(E):

is_stable($POList$<E>.oldest()) $\Leftrightarrow$

$$min_{e \in POList\text{<E>.oldest()}}(upper(ts(e))) < min_{\forall P_E \in PR(E)}(ts_{sync}(P_E))$$

By definition we consider the empty set not to be stable.

1. Special attention is needed, when the *synchdist* error significantly increases

B. Composition

Now that we can determine if the candidate sets are stable, we can present the algorithms for conjunction using the *chronicle* policy. The activity diagram below shows the execution flow when processing incoming events. First the sync-points are updated with respect to the sender of the event.

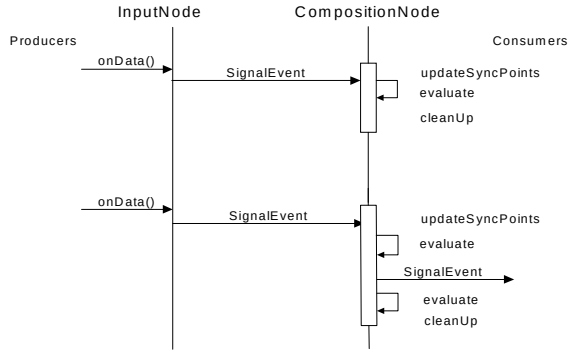

Fig. 4. Activity diagram.

Then we evaluate the operand lists and check if there are stable events that can be composed. At the end we clean up the operand lists. Below we sketch the algorithms implemented in the *CompositionNode*:

```
SignalEvent(Event e):{
  switch typeof(e)
        case heartbeat: break;
        case A: POList<A>.add(e);
              update_sync_points(e);
              while( evaluate() );
              cleanup();
              break;
        case B: // analogous to above
}
evaluate: returns boolean {
// AND-chronicle
  Set<A> oldest_a; Set<B> oldest_b;
  if (not_empty(POList<A> and not_empty(POList<B>))
        oldest_a=POList<A>.oldest();
        if (is_stable(oldest_a))
              if (sizeof(oldest_a) > 1)
                    // (exception multiple a)
              oldest_b=POList<B>.oldest();
              if (is_stable(oldest_b))
                    if (sizeof(oldest_b) > 1)
                          // exception (multiple b)
                    compose(oldest_a, oldest_b);
                    return (TRUE); // A & B
              else
                    // expect sync-point to increase
                    return(FALSE);
        else
              // expect sync-point to increase
              return(FALSE);
  return(FALSE);
}
```

C. Heartbeat

In the case that *oldest_a* or *oldest_b* is not stable yet, we must wait for the global sync-points to be increased. This will either be in case of following A or B events, which again trigger the evaluation algorithm, or in case heartbeat events are signaled. We require producers to signal events with a minimum frequency. If the event stream is less frequent or no more events occur at some producer node, the producer will generate an artificial heartbeat event for the sake of increasing the sync-point window. When a producer crashes or the network is partitioned for long periods then the *CompositionNode* could be blocked - possibly indefinitely. This problem is dealt with by using timeouts in the *InputNode* which in turn raise an exception at the consumer.

D. Accepting Uncertainty

Because the accuracy interval order is only a partial order of events, the situation may arise that we cannot uniquely identify an *oldest* event. As can be seen from the definition of the *oldest()* method, the result may be a set of events, with uncertain temporal order. In the above example of Fig. 3., *oldest_b* contains $b^1{}_0$ and $b^2{}_0$. This situation is considered to be exceptional in a sense that the event service cannot guarantee the proposed semantic of *chronicle* consumption. Therefore we explicitly raise an exception. Alternatively we could present the operand candidate sets *oldest_a* and *oldest_b* to the application and let the user decide.

In the following we will illustrate the effect of uncertainty on order dependent operators. As an example we use the simple sequence operator *A;B*. We implement the *evaluate()* method as follows:

```
evaluate: returns boolean {
// SEQUENCE-chronicle
  Set<A> oldest_a; Set<B> oldest_b;
  if (not_empty(POList<A> and not_empty(POList<B>))
        oldest_a=POList<A>.oldest();
        if (is_stable(oldest_a))
              if (sizeof(oldest_a) > 1)
                    // exception (multiple a)
              oldest_b = POList<B>.oldestFollowing(oldest_a);
              if (is_stable(oldest_b))
                    if (sizeof(oldest_b) > 1)
                          // exception (multiple b)
                    else
                          compose(oldest_a, oldest_b)
                          return (TRUE); // A ; B
              else
                    // expect sync-point to increase
                    return(FALSE);
        else
              // expect sync-point to increase
              return(FALSE);
  else
        return(FALSE);
}
```

The method *POList<>.oldestFollowing(Set<>)* returns the set of oldest events which are those events that are following the oldest event in Set<> and are not preceded by any other in the *POList<>*:

$$\forall T::e \in POList\text{<E>.oldestFollowing(Set<F>)}:$$

$$f_{min} \in \text{Set<F>}\ ,\ lower(f_{min}) = min_{f \in \text{Set<F>}}(lower(f))$$

$$f_{min} < e \quad \vee \quad f_{min} \perp e$$

$$\neg\ (\exists\ e' \in POList\text{<E>.oldestFollowing(Set<F>)} : ts(e') < ts(e))$$

Note that the above *evaluate()* algorithm presents the most strict implementation of the sequence operator. In fact,

there could be pairs of events $a \in oldest\_a$ and $b \in oldest\_b$ for which $a<b$ holds. However, the notification service may not silently decide upon which events to compose. We suggest that the user may specify a callback to implement application specific selection policies. On the other hand we can say, that if we do not explicitly recognize such situations, then there is the possibility for erroneously signaling a complex situation that actually did not occur.

VI. Conclusions and Future Work

Previous work on event composition in distributed environments either does not consider the possibility of partial event ordering or is based on the 2g-precedence model. Therefore, existing approaches suffer from one or more of the following drawbacks: lack of applicability to large scale open systems, possibility of spurious events and ambiguous event consumption.

In this paper we present a new approach for timestamping events in a large-scale, loosely coupled distributed system. We use accuracy intervals with reliable error bounds for timestamping of events reflecting the inherent inaccuracy in time measurements. We leverage existing time service implementations, like the Network Time Protocol, that provide reference time injected by GPS time servers and additionally return reliable error bounds.

We propose a window mechanism to deal with varying transmission delays when composing events from different event sources. Most important, when detecting composite events we explicitly consider the fact that events can only be partially ordered. We introduce an accuracy interval order that guarantees the property of *time consistent order*: events are not erroneously ordered and situations of uncertain event order are always detected and signaled to the application. Thereby, event consumption modes like *recent* and *chronicle* can be unambiguously defined. In our ongoing research we examine different strategies to handle uncertainty of event order. Possible approaches could be to provide policies as service configuration options or to introduce up-calls to the application level to let the user decide and make event composition programmable.

As many applications like CSCW need more powerful temporal relations between composite events [48], we suggest to think of composite events having a start and endpoint thus associating an interval with the composite event instead of using the timestamp of the terminating event. Then we can provide composition operators that allow for interval relations [1].

Applications with demands for high accuracy time stamping and timer signal handling, like real-time systems, are supposed to make use of special low-cost hardware equipment that directly integrates GPS time signals and may achieve down to 1 µsec accuracy [21] and guarantees precision of down to 2 µsec. The foundations of the proposed interval based approach are in general applicable to such a high accuracy and high precision time environment. Our approach also fits well into mobile environments, provided that the mobile devices are equipped with GPS receivers.

We have implemented a prototype on top of a CORBA platform with multicast capabilities to experiment with accuracy interval based event composition. Currently we are incorporating event composition based on interval relations and are making extensions for up-call support.

VII. Acknowledgement

We wish to thank Jean Bacon and Ken Moody for many fruitful discussions during their recent visit. Thanks are also due to Ulf Meyer who implemented portions of the first prototype.

VIII. References

[1] J.F. Allen. Maintaining Knowledge about Temporal Intervals. CACM, Vol. 26, No. 11, November 1983.

[2] J. Bacon and K. Moody and J. Bates. Active Systems. Technical Report. Computer Laboratory, University of Cambridge, December 1998.

[3] F. Barabas and A. Poddany and J.-P. Florent and G. Klawitter. Java Shared Objects for Flexible Distributed Applications - Prototype of a Flight Data Management System. DIFODAM project, Eurocontrol, Brussels, http://www.eurocontrol.fr/projects/difodam/.

[4] D. Barret and L. Clarke and P. Tarr and A. Wise. A Framework for Event-based Software Integration, ACM Transactions on Software Engineering and Methodology, Vol. 5, No. 4, 1996.

[5] J. Bates and J. Bacon and K. Moody and M. Spiteri. Using Events for the Scalable Federation of Heterogeneous Components. In Proceedings of the SIGOPS European Workshop on Support for Composing Distributed Applications, September 1998.

[6] A. Buchmann and J. Zimmermann and J. Blakeley and D. Wells. Building an Integrated Active OODBMS: Requirements, Architecture, and Design Decisions. In Proceedings of ICDE '95, pp. 117-128, March 1995.

[7] F. Casati and S. Ceri and B. Pernici and G. Pozzi. Deriving Active Rules for Workflow Management. In Proceedings of DEXA'96, pp 94-115, September 1996.

[8] S. Chakravarthy and V. Krishnaprasad and E. Anwar and S. Kim. Composite Events for Active Databases: Semantics, Contexts and Detection. In Proceedings of the International Conference on Very Large data Bases (VLDB '94), pp. 606-617, 1994.

[9] F. Cristian. Probabilistic Clock Synchronization. Distributed Computing (3), Springer, 1989.

[10] U. Dayal and A. Buchmann and D. McCarthy. Rules are Objects too: a knowledge model for an active, object-oriented database system. In Proceedings of the 2nd Intl. Workshop on Object-Oriented Database Systems, Lecture Notes in Computer Science 334, Springer, 1988.

[11] U. Dayal et al. The HiPAC Project: Combining Active Databases and Timing Constraints, ACM SIGMOD Record, Vol. 17, No. 1, pp. 51-70, March 1988.

[12] U. Dayal and M. Hsu and R. Ladin. Organizing Long-Running Activities with Triggers and Transactions. In Proceedings of the 1990 ACM SIGMOD International Conference on Management of Data (SIGMOD'90), pp. 204-214, May 1990.

[13] DCOM, Microsoft Corp., http://www.microsoft.com/com/dcom.asp/

[14] J. Eder and H. Groiss and H. Nekvasil. A Workflow System Based on Active Databases. In Proceedings of Connectivity '94: Workflow Management - Challenges - Paradigms and Products (CONN'94), pp. 249-265, 1994.

[15] C. Fetzer and F. Cristian. Integrating External and Internal Clock Synchronization. Real-Time Systems, Vol. 12, No. 2., 1997, Kluwer Academic Publishers, Boston

[16] S. Gatziu and K. Dittrich. Events in an Active Object-Oriented Database System. In Proceedings of Rules in Database Systems (RIDS '93), pp. 23-39, August 1993.

[17] S. Gatziu and A. Koschel and G. v. Buetzingsloewen and H. Fritschi.

Unbundling Active Functionality, SIGMOD Record. Vol.27, No. 1, pp. 35-40, March 1998.

[18] N. Gehani and H. Jagadish and O Shumeli. Event Specification in an Active Object-Oriented database. In Proceedings of International Conference on Management of Data (SIGMOD'92), June 1992.

[19] A. Geppert and D. Tombros. Event-based Distributed Workflow Execution with EVE. In Proceedings of Middleware '98 (IFIP Intl. Conf. on Distributed Systems Platforms and Open Distributed Processing), September 1998.

[20] R.E. Gruber and B. Krishnamurthy and E. Panagos.High-level Constructs in the READY Notification System. ACM SIGOPS European Workshop on Support for Composing Distributed Applications, September 1998.

[21] W.A. Halang and M. Wannemacher. High Accuracy Concurrent Event Processing in hard Real-Time Systems. Real-Time Systems, Vol. 12, No. 1, 1997, Kluwer Academic Publishers, Boston.

[22] H. Jagadish and O. Shmueli. Composite Events in a Distributed Object-Oriented Database. In M. Tamer Özsu, U. Dayal and P. Valduriez (editors), Distributed Object Management, Morgan Kaufmann, San Mateo, California, 1994.

[23] JavaBeans, Sun Microsystems, http://java.sun.com/beans/

[24] H. Kopetz. Sparse Time versus Dense Time in Distributed Real-Time Systems. In Proceedings of the 12th Intl. Conf. on Distributed Computing Systems (ICDCS), Yokohama, Japan, 1992.

[25] A. Koschel and R. Kramer et.al. Configurable Active Functionality for CORBA. In 11th ECOOP'97 Workshop: CORBA Implementation, Use and Evaluation, June 1997.

[26] L. Lamport and M. Melliar-Smith. Synchronizing Clocks in the Presence of Faults. Journal of the ACM, Vol. 32, No. 1, January 1985.

[27] L. Lamport. Time, clocks, and the ordering of events in a distributed system. CACM Vol. 21 No. 7, pp. 558-565, July 1978.

[28] W. Lewandowski and J. Azoubub and W.J. Klepczynski. GPS: Primary Tool for Time Transfer. Proc. of the IEEE, Vol. 87, No. 1, January 1999.

[29] C. Liebig and B. Boesling and A. Buchmann. A Notification Service for Next-Generation IT Systems in Air Traffic Control, GI-Workshop "Multicast - Protokolle und Anwendungen", pp. 55-68, Braunschweig, Germany, May 1999.

[30] J. Lundelius and N. Lynch. An Upper and Lower Bound for Clock Synchronization. Information and Control, Vol. 62, No. 2-3, 1984.

[31] C. Ma and J. Bacon. COBEA: A CORBA-Based Event Architecture. In Proceedings of the USENIX Conference on Object-Oriented Technologies and Systems, pp. 117-131, June 1998.

[32] K. Marzullo and S. Owicki. Maintaining the Time in a Distributed System. ACM Symp. on Principles of Distr. Computing 1983, in ACM SIGOPS, 1985.

[33] D.L. Mills. Network Time Protocol Version 3. Network Working Group Report RFC-1305, University of Delaware, March 1992.

[34] D.L. Mills. On the Accuracy and Stability of Clocks Synchronized by the Network Time Protocol in the Internet System. ACM Computer Communication Review, Vol. 20, No. 1, 1990.

[35] D.L. Mills. Unix Kernel Modifications for Precision Time Synchronization. Electrical Engineering Department Report 94-10-1, University of Delaware, October 1994.

[36] Object Management Group (OMG), CORBA Services: Common ObjectServices, Time Service. Technical Report formal/97-12-21, ftp://www.omg.org/pub/docs/formal/97-12-21.pdf, Famingham, MA, July, 1997.

[37] Object Management Group (OMG). Event Service Specification. Technical Report formal/97-12-11, ftp://www.omg.org/pub/docs/formal/97-12-11.pdf.

[38] Object Management Group (OMG). Notification Service Specification. Technical Report telecom/98-06-15, ftp://www.omg.org/pub/docs/telecom/98-06-15.pdf.

[39] B. Oki and M. Pfluegl and A. Siegel and D. Skeen. The Information Bus - An Architecture for Extensible Distributed Systems. In Proceedings of SIGOPS 93, 1993.

[40] U. Schmid and K. Schossmaier. Interval-based Clock Synchronization. Real-Time Systems, Vol. 12, No. 2., 1997, Kluwer Academic Publishers, Boston.

[41] R. Schwarz and F. Mattern. Detecting Causal Relationships in Distributed Computations: In Search of the Holy Grail. Distributed Computing, Vol. 7, No. 3, 1994.

[42] S. Schwiderski. Monitoring the Behavior of Distributed Systems, PhD Thesis, Selwyn College, Computer Lab, University of Cambridge, June 1996.

[43] T.K. Srikanth and S. Toueg. Optimal Clock Synchronization. Journal of the ACM, Vol. 34, No. 3, July 1987.

[44] B. Sterzbach. GPS-based Clock Synchronization in a Mobile, Distributed Real-Time System. Real-Time Systems, Vol. 12, No. 1, 1997, Kluwer Academic Publishers, Boston.

[45] D. Tombros and A. Geppert and K. Dittrich. Semantics of Reactive Components in Event-Driven Workflow Execution, In Proceedings of the 9th International Conference on Advanced Information Systems Engineering, June 1997.

[46] P. Verissimo and L. Rodrigues and A. Casimiro. CesiumSpray: a Precise and Accurate Global Clock Service for large-scale Systems. Real-Time Systems, Vol. 12, No. 3., 1997, Kluwer Academic Publishers, Boston.

[47] P. Verissimo. Real-Time Communication. In Sape Mullender (Editor), Distributed Systems, Addison-Wesley, 1993.

[48] T. Wahl and K. Rothermel. Representing Time in Multimedia-Systems. IEEE Conf. on Multimedia Computing Systems, Boston, 1994.

[49] S. Yang and S. Chakravarthy. Formal Semantics of Composite Events for Distributed Environments. In Proceedings of the International Conference on Data Engineering (ICDE 99), pp. 400-407, Sydney, Asutralia, March 1999.