Implementing a practical spatio-temporal composite event language

Ken Moody[1], Jean Bacon[1], David Evans[1], and Scarlet Schwiderski-Grosche[2]

[1] Computer Laboratory, University of Cambridge,
15 JJ Thomson Ave, Cambridge CB3 0FD, UK
{firstname.lastname}@cl.cam.ac.uk
[2] Microsoft Research, 7 JJ Thomson Ave, Cambridge, CB3 0FB, UK
scarlets@microsoft.com

Abstract. An earlier paper introduced *SpaTeC*, a composite event language that enables simultaneous matching of event occurrences over space and time. The driving case study is taken from a paper that describes techniques for monitoring small animals in New Zealand. The semantics of *SpaTeC* is presented in detail with the aid of the case study, but the syntax is essentially mathematical. This paper describes a programming language based on the *SpaTeC* model, illustrating it through a practical application, the analysis of GPS traces of buses serving Cambridge, UK. We describe some of the questions that Stagecoach, the bus operator, wish to have answered, and use these to motivate our extensions to *SpaTeC*. Composite event patterns are essentially those of the earlier paper, with the addition of *primitive patterns*, which enforce restrictions on the space and/or time of event occurrences. Data fields identified during pattern matching can be tested by predicates that further restrict the relevant combinations of primitive events. We show how the language can be used to answer questions posed by Stagecoach and discuss its realisation.

Key words: Composite event language, mobile systems, spatio-temporal reasoning, session types

1 Introduction

SpaTeC, a composite event language that enables simultaneous matching of event occurrences over space and time, is described in [1]. *Primitive event* occurrences carry both time and location stamps; the target of the work is to support sensor-based applications in which both clients and the objects monitored can be mobile. The paper defines operators to combine primitive events to form composite event occurrence patterns, specifying their semantics in terms of the time and location stamps of participating primitive event occurrences. A composite event notification service receives primitive (low-level) event occurrences, performs composite event matching against the current set of patterns,

and forwards details of composite (higher-level) event occurrences to the relevant subscribers. There was no implementation, and although the paper presented a detailed mobile event scenario, the data were not easily available for experiment. The present paper describes a new scenario with more challenging features for which data are available, using it to help design a *SpaTeC* programming language that builds on and extends the earlier model. We shall not refer to that paper again, but we assume the reader has access to it and can discover any relevant details from it.

Physical measurements in both time and space are subject to error. We follow the approach established in [2] to take account of local error using interval timestamps, and extend the technique to handle location stamps as well. Time values are essentially one-dimensional, and it makes sense to say that one event took place before another, though if the uncertainty intervals overlap we cannot be sure. For location stamps, the most that we can assert is that two events occurred at different places. Recent work taking account of location, such as [3–5], is discussed in the earlier paper.

Operators in *SpaTeC* establish time and location stamps for each composite event occurrence matched, and the semantics of the composite event language depends on the details. Two different semantics are described, which correspond in the time domain to *set-based semantics* and *interval-based semantics* respectively. Which is the more appropriate depends on the specific application; here we adopt interval-based semantics for timestamps, with the corresponding convex-hull semantics for composite event location stamps.

The application scenario described in [1] stems from work carried out in the Department of Biological Sciences at the University of Waikato, New Zealand[1]. King et al. develop effective techniques for predator-control operations on pastoral farmland, where ferrets and other small carnivores are monitored "both for keeping track of the distribution and numbers of native species, and for locating, guiding and auditing control operations against alien species" [6]. Monitoring is done using the so-called Scentinel®, a "smart" tracking tunnel for small mammals that records time, date, weight and a digital photograph of every animal entering it. Identifiers of tagged animals that get close to a Scentinel but do not enter are also recorded. Scentinels represent fixed nodes, radio-tagged animals represent mobile nodes in the system (GPS satellite collars allow continuous monitoring, but they are only suitable for animals weighing 15 or more kg).

Data rich events occur only in the vicinity of Scentinels, hence at fixed locations, though at random times. The present paper uses a complementary scenario, inspired by the needs of one of the Computer Laboratory's current research projects, TIME-EACM [7]. The local Cambridge bus company, Stagecoach, has fitted its buses with GPS transmitters supported by a local company, ACIS, who display anticipated arrival times at bus stops. With the agreement of both Stagecoach and ACIS the data feeds are also sent to the TIME-EACM project. Unlike Scentinel data, bus position data is transmitted at fixed times, but bus

[1] http://www.bio.waikato.ac.nz/research/research1.shtml♯zoology

locations depend on factors such as traffic congestion. A paper giving a statistical analysis of this bus probe data is forthcoming; see [8]. Stagecoach are interested in the work, and have suggested other questions that we may be able to answer for them. The original *SpaTeC* was designed for applications that require temporal and spatial reasoning. The detailed requirements stemming from the Stagecoach questions have helped us to develop a theoretical design into a practical composite event language.

The rest of this paper is structured as follows: Section 2 gives the background to the GPS tracking support for the Stagecoach bus fleet in Cambridge, and discusses some of the questions of interest to the company. Section 3 presents the syntax and semantics of the *SpaTeC* programming language, motivating the extensions using Stagecoach bus scenarios. We describe our experience using *SpaTeC* to model the bus scenarios in Section 4. In Section 5 we outline some of the considerations when implementing *SpaTeC* in data intensive applications. Section 6 discusses related work. Section 7 concludes the paper.

2 GPS tracking of buses

Each bus serving Cambridge UK and surrounding areas that is operated by Stagecoach is equipped with a GPS-based location sensor. Such sensors note the geographical location of the bus and periodically (about once every 30 seconds) report it via wireless communication to a central location. Stagecoach, along with the local County Council and the firm that operates the reporting infrastructure, use these location reports to provide real-time information to passengers via displays at bus stops, web sites, and mobile phones.

Each Stagecoach bus therefore produces a series of location events, each having an associated timestamp. Both the location and the time are subject to uncertainty, the location because of error in the GPS reading and subsequent software processing, and the timestamp because location recording and data transmission are decoupled. Each bus is assigned an offset within a 30 second window at which to transmit. This means that a location report reflects the position of a particular bus some time within the previous 30 seconds. More formally, suppose that the location of bus i is recorded at time t_0 and is reported at time $t_1 = t_0 + \tau_i$. The timestamp of the location will be t_1, whereas the reading was actually taken at t_0. We are guaranteed that $\tau_i < 30$ for each bus i. Further, τ_i will be the same for each report from bus i within some session; the offsets for a given set of buses may then be updated, but this will happen only a small number of times per day. We have no easy way of determining the precise value of τ_i within a particular session. Further details are reported in [8].

Aside from providing information to passengers, Stagecoach is interested in using these location reports to answer questions like the following:

– Does wet weather, as sensed by various environmental detectors, correspond to longer journey times?

– Anecdotes suggest that journey times increase in the build-up to Christmas. Is this true?
– What is the quantitative effect on journey times of road disruptions (accidents, road works, etc.)?
– Are journey times to and from a local hospital longer on Tuesdays, when many outpatient appointments are scheduled?

Answering these questions has a component of historical analysis that amounts to data mining of the GPS traces. However, each question also defines a situation that can be detected as it is happening, so that consequent alerts can be issued. For example, the presence of rain and the lengthening of journey times can cause a "bad weather performance" alert to be issued, possibly triggering operational changes. A language based on *SpaTeC* principles must be able to describe these events.

3 The *SpaTeC* programming language

The *SpaTeC* programming language assumes an object-oriented environment. Events that occur in the application to be modelled conform to some type; variables in *SpaTeC* are typed accordingly. Each event occurrence defines an object instance in the corresponding class; attributes of this class can be used within the language.

The language has two parts. The first enables construction of *match expressions*, which are used to define *composite event patterns* that are always present in some phenomenon. The result of each match will be a set of primitive events. A match expression establishes what is, in effect, a session type having receive primitives only [9]. (If the language runtime used has direct support for session types, as can be found for Java and Moose as outlined in [10], the facilities can be used to encode results from match expressions, compare them, move them between modules of code, and so on. We discuss the use of session types further in Section 5.) The second part takes the set of composite events found by a particular match expression and restricts it using *filter predicates*. Zero or more filter predicates are attached to a match expression to form a *phrase*.

Each phrase corresponds to some situation of interest at application level. The purpose of a given phrase is to identify each *composite event occurrence* that identifies a particular instance of that situation. When a specific composite event is detected, its occurrence will depend on a specific set of primitive events. The notification service forwards details of each composite event occurrence to the relevant subscribers, for example by publishing an object instance in a class specific to this type of composite event; its attributes will reflect the associated set of primitive event occurrences.

3.1 Primitive event match expressions

Each match expression has a natural tree structure that represents the event pattern to be identified. Non-leaf nodes in the tree correspond to the operators introduced in Section 3.3. There are two categories of leaf node.

A *primitive event* is a typed data structure that represents the reification of an occurrence. The type of the event is a class (in the object-oriented programming sense) that contains all the information needed to describe a particular event occurrence of that type. In our model, every such class will contain `time` and `location` attributes.

Assume that e is a primitive event type, for example a GPS bus observation. The expression e is a *primitive event match expression* that matches all event instances of type e. Further, the expression

$$e \rightarrow B$$

is a match expression that, each time it matches an event of type e, unifies B with the state carried by that event instance. One can think of B as being a session variable of type e, which in the context of a specific match describes the corresponding matching event occurrence. We shall see shortly how B is scoped and why it is useful.

3.2 Primitive patterns

Alongside primitive events are *primitive patterns*. These are match expressions that match events not by type, as with primitive events, but by space and time. They can be thought of as matching all possible event types, subject to conditions on their `time` and/or `location` attributes. Examples of patterns are "Tuesday", which contains every event with a timestamp that falls on a Tuesday[2]; and "point x", containing all events at all times having location stamps x.

The rules for matching primitive patterns are as defined in Tables 1 and 2. We also need to specify the `time` and `location` stamps of composite event occurrences matched by expressions that include patterns. First, any composite event expression consisting only of primitive patterns is a *pattern*. When a pattern matches some composite event expression that contains at least one event occurrence, the combined matched expression is given the `time` and `location` stamps of the non-pattern operand. In particular, any combination of primitive patterns can match all possible (composite) event types subject to conditions on the `time` and/or `location` attributes, and the result of matching that composite pattern will leave both `time` and `location` stamps unaltered.

We shall meet examples of the use of patterns in Section 4.

[2] We are ignoring issues such as time zones and leap years; the motivated reader can imagine a system that begins by following the ISO 8601 standard for writing down times.

3.3 General match expressions

Match expressions are combined to form *composite events* using operators. The event operators for simultaneous reasoning in space and time are listed in Table 1. When we need to reason in only one of space and time, we can use the simpler event operators listed in Table 2. Event patterns pose precise space and time requirements, but sometimes the uncertainty of measurement of the `location` and `time` attributes of primitive event occurrences will mean that we cannot be certain whether there is a match. Notions of *same location* and *same time* must always be interpreted as *within measurement error*, meaning that they are too close to be differentiated. In addition *SpaTeC* supports Boolean operations on match expressions in the obvious way.

This allows the construction of complex composite events that represent occurrences having some spatial and temporal property (such as primitive events happening in sequence, at the same place, etc.) and the restriction of these by spatial and temporal constants.

Table 1: *SpaTeC* Event Operators for Reasoning in Space **and** Time

Event Operator	Meaning	Description
$E_1 \left\{ \begin{smallmatrix} <> \\ ; \end{smallmatrix} \right\} E_2$	Location overlap, in sequence	The locations of E_1 and E_2 overlap, E_1 happens before E_2
$E_1 \left\{ \begin{smallmatrix} >< \\ ; \end{smallmatrix} \right\} E_2$	Distinct locations, in sequence	E_1 and E_2 occur at distinct locations, E_1 happens before E_2
$E_1 \left\{ \begin{smallmatrix} <> \\ \parallel \end{smallmatrix} \right\} E_2$	Location overlap, time overlap	The locations of E_1 and E_2 overlap, their timestamps also overlap
$E_1 \left\{ \begin{smallmatrix} >< \\ \parallel \end{smallmatrix} \right\} E_2$	Distinct locations, time overlap	E_1 and E_2 occur at distinct locations, and their timestamps overlap

3.4 Filter predicates

A stream of composite events is likely to be too broad to describe many phenomena of interest. We therefore introduce *filter predicates* that allow selection of the composite events emerging from a given match expression.

A filter predicate is defined recursively as the conjunction ($\wedge$) or disjunction ($\vee$) of two other filter predicates. Each predicate can be negated ($\neg$). At the lowest level, a filter predicate is a Boolean-valued expression containing elements chosen from the following:

Table 2: *SpaTeC* Event Operators for Reasoning in Space **or** Time

Event Operator	Meaning	Description
$E_1 \{<>\} E_2$	Location overlap	The locations of E_1 and E_2 overlap
$E_1 \{><\} E_2$	Distinct locations	E_1 and E_2 occur at distinct locations
$E_1 \{\|\|\} E_2$	Time overlap	The timestamps of E_1 and E_2 overlap
$E_1 \{;\} E_2$	In sequence	E_1 happens before E_2
$E_1 \{,\} E_2$	Conjunction	Both E_1 and E_2 occur
$E_1 \{\|\} E_2$	Disjunction	Either E_1 or E_2 occurs

- constants
- attributes from variables that are unified as part of the matching composite event
- simple mathematical operations (by which we mean things like arithmetic operators and exponentiation)
- functions built from the above, such as computing the distance between two points (in whatever space those coordinates lie)
- relational operators ($<, =, >, \leq, \geq$)

The *attributes from variables* component of filter predicates needs a little explanation. The variables are like example B in Section 3.1, and represent the data that characterise matching event instances. This allows predicates to reason over these data, selecting only composite events of interest. For example, consider the event type BUS that, in addition to the standard attributes `time` and `location`, has a vehicle identification attribute `VID` that indicates to which bus the event pertains. Let e_1 and e_2 represent events of type BUS. The expression

$$(e_1 \rightarrow B_1;\ \ e_2 \rightarrow B_2)\ [(B_1.\texttt{VID}) == (B_2.\texttt{VID})]$$

matches two observations of the same bus, the first taking place before the second. The expression

$$(e_1 \rightarrow B_1;\ \ e_2 \rightarrow B_2)\ [(B_1.\texttt{VID}) == (B_2.\texttt{VID}) \\ \wedge\ \text{distance}\ (B_1.\texttt{location},\ B_2.\texttt{location}) \geq 450]$$

matches two observations of the same bus that are at least 450 units apart.

Note that a phrase consists of a match expression and zero or more filter predicates, each of which must be satisfied. Testing a filter predicate requires access to the relevant matched variable attributes; supporting multiple predicates aids specification and offers hints to the optimiser.

4 Using *SpaTeC* to analyse the bus GPS data

In this section we look in more detail at some of the Stagecoach questions, and describe some of the basic requirements for answering them. As a first step we look at two of their interests:

– Anecdotes suggest that journey times increase in the build-up to Christmas. Is this true?
– Are journey times to and from a local hospital longer on Tuesdays, when many outpatient appointments are scheduled?

As we have said in Section 2, we are interested not so much in mining the historical record for answers to these questions as in producing events that correspond to germane situations as they happen. Of course, anecdotal evidence relates to past experience and a first step is statistical analysis of the historical record. If the anecdotal evidence is found to be justified, in general, it is then appropriate to detect patterns in real time; such a pattern may indicate for example that congestion is beginning to build up, so appropriate action can be taken, such as the deployment of extra buses.

Answering these types of question requires selecting bus observations with particular `time` or `location` attributes. The simplest case is that of a single fixed time or place, for example the stop at the local hospital.

4.1 Constant patterns and their uses

Constant patterns are provided as part of the programming environment when defining composite event expressions. To say that a pattern is constant does not mean that its value is necessarily simple; for example, the pattern *Tuesday* is constant, but the value requires careful initialisation. We begin with an example of a point location constant pattern.

To identify observations when a bus is at a particular stop, we have:

$$(e_1 \ \{<>\} \ \text{BusStopX})$$

BusStopX is a primitive pattern with the `location` stamp of BusStopX, which will otherwise match any event. The normal semantics of the $\{<>\}$ operator will cause this expression to match all events of type e_1 whose `location` attributes overlap BusStopX; see Section 3.3. Inevitably buses will not stop precisely at a single fixed location, and we select an uncertainty radius for the primitive pattern *BusStopX* to take account of this.

Suppose that we need to identify, in real time, (composite) events when the journey time from the hospital to the station is less than 6 minutes.[3] e_1 and e_2 are bus observation events. We assume that HospitalStop and StationStop identify the two bus stops.

$$((e_1 \rightarrow B_1 \ \{<>\} \ \text{HospitalStop}); \ (e_2 \rightarrow B_2 \ \{<>\} \ \text{StationStop})) \\ [(B_1.\texttt{VID} == B_2.\texttt{VID}) \ \wedge \ (B_2.\texttt{time} - B_1.\texttt{time} < 6 * 60)] \quad (1)$$

[3] Note that uncertainty in event timestamps means that this can't be detected precisely.

The composite event ($e_1 \rightarrow B_1$ {<>} HospitalStop) has the timestamp of event e_1, so we can easily check for quick journeys on a Tuesday (unlikely!) by starting with

$$(e_1 \rightarrow B_1 \ \{<>\} \ \text{HospitalStop}) \ \ || \ \ \text{Tuesday}$$

This works because of the rule for deriving composite event `location` and `time` stamps from primitive patterns (see Section 3.2). The same effect could be obtained using the expression

$$(\ e_1 \rightarrow B_1 \ \left\{ \begin{matrix} <> \\ \| \end{matrix} \right\} \ \ (\text{HospitalStop} \ \wedge \ \text{Tuesday})\ )$$

4.2 More demanding Stagecoach questions

Detailed statistical study of bus tracking data along two sections of route has allowed us to answer some more complex Stagecoach questions, see [8]. This analysis is time consuming, and by its nature cannot deliver answers in real time. Recall the other two examples given in Section 2:

- Does wet weather, as sensed by various environmental detectors, correspond to longer journey times?
- What is the quantitative effect on journey times of road disruptions (accidents, road works, etc.)?

We have been able to resolve these questions to some extent using historical data, for example looking at the correlation between days when it has rained and longer journey times. However, these can also be detected in real time by writing appropriate *SpaTeC* phrases. For example, suppose that an event of type r occurs when it rains, its `time` and `location` attributes describing the time and place of the rainfall. (r can also be regarded as an event pattern that represents all the points where it is raining.) Starting from *SpaTeC* phrase 1, we can construct the phrase

$$\left(\left((e_1 \rightarrow B_1 \ \{<>\} \ \text{HospitalStop}) \ \left\{ \begin{matrix} <> \\ \| \end{matrix} \right\} \ r \right) ; \right.$$
$$\left. \left((e_2 \rightarrow B_2 \ \{<>\} \ \text{StationStop}) \ \left\{ \begin{matrix} <> \\ \| \end{matrix} \right\} \ r \right) \right)$$
$$[(B_1.\texttt{VID} == B_2.\texttt{VID}) \ \wedge \ (B_2.\texttt{time} - B_1.\texttt{time} > 20 * 60)]$$

that detects bus journeys between the hospital and the station, when it is raining at the beginning and end of the journey and the trip takes longer than 20 minutes (we shall also need to ensure that the bus is visiting the station for the first time since it left the hospital).

In addition to the bus tracking data the TIME-EACM project monitors traffic on the nearest main road in real time, and this data can provide early warning of congestion and consequent delay to buses. Equally, the bus GPS data in itself

provides evidence of congestion; for example, buses do not make good progress during the morning and evening rush hours, particularly during the school term. We hope to be able to calibrate bus movement behaviours from the historical tracking data, and so develop *SpaTeC* patterns that, if detected, indicate congestion on the roads. A first shot might be to produce an event when the same bus reports 4 successive location observations that overlap pairwise. Even this relatively simple example requires some tedious predicates.

5 *SpaTeC* - Moving Towards Deployment

SpaTeC phrases define sets of composite events, and it is the job of the implementation to detect these based on primitive event occurrences. Recall that each phrase consists of an obligatory match expression and optional predicates, which are tested against each matching set of event occurrences. A possible scenario is to implement complex event detection as a service above a publish/subscribe architecture such as Rebeca [11] or Hermes [12]. Clients of the system are publishers, subscribers or both. Event types are defined and managed by the system. Potential publishers advertise their ability to publish event instances. Subscribers subscribe to event types with a subscription filter that represents their particular interests. Notice that publishers and subscribers are mutually anonymous. The system is provided as a distributed broker network. A broker may be publisher-hosting, subscriber-hosting, both or neither; that is, some brokers may merely act as routers. A common optimisation of publish/subscribe is content-based routing (CBR), that allows communication paths to be shared when a published message is transmitted through the network to its subscribers. CBR is typically set up in the event brokers when events are advertised and subscribed to.

Following [13], each composite event detector (CED) subscribes to events and advertises and publishes higher-level composite events. A composite event match expression tree may be split into subtrees that are placed in the distributed system to optimise communication and detection latency.

Each phrase in a *SpaTeC* program is handled independently, meaning that there is no explicit inter-phrase state nor are there references between them. Therefore, a given program may be implemented as a central service or may be distributed as an optimisation.

5.1 Centralised implementation

We first outline how a centralised implementation works.

1. The text of the phrase is processed by a front-end to check syntax and extract the match expression and predicates. At this stage the match expression is checked to ensure that unified variables are not re-used and are therefore conflict-free. This is done as it would be for any programming language, so it is not discussed further in this paper.

2. Event detectors are built from the phrase's match expression. This is done using finite automata based on those used by Hermes [14, chapter 7]. The automata use initial, ordinary, and generative states but have no need for generative time states, since *SpaTeC* does not deal with time events in the same way. The operators in Tables 1 and 2 that compare location stamps are handled in the same way that Hermes deals with timestamps, with comparisons taking account of the inherent uncertainty.

   The only complication is that when primitive events are matched as part of composite event detection and their instance is unified with a variable, the attributes of that instance must be saved for future evaluation by filter predicates. A naïve approach is to keep a shared dictionary of unified variables, associating with each its class (derived from the type of the corresponding primitive event) and its attributes. The automata contain *unification states* that are inserted as required following matching transitions. These states are responsible for updating the shared dictionary.
3. The filter predicates operate on the output of the automata created in step 2, using the shared dictionary to retrieve any event instance attributes that are needed. This is done through any suitably efficient rule system, meaning that a Prolog or Datalog implementation is feasible.

Following these three steps, event processing begins with CED service nodes evaluating their assigned match expressions.

5.2 Distributed implementation

In reality a centralised implementation is unlikely to be appealing for the usual reasons, including poor scalability, inflexibility in the context of a heterogeneous communication infrastructure, and unreliability. In addition, it is likely to be inappropriate for multi-organisation systems. A distributed implementation of a *SpaTeC* program follows the pattern of a centralised one. Phrases are parsed and the match expressions and filter predicates extracted. This is likely to remain centralised as the program is presented to the system at only one place, and processing is likely to be done only once. In step 2, the automata are distributed throughout the network in the same manner as are CED subtrees. A shared dictionary is no longer an option for maintaining event instance attributes, but a relevant subset may be included with the composite events produced by the automata at each node. Filter predicates are similarly distributed to the CED service(s) that can, based on the decomposition derived above, subscribe to each required event type. In other words, if the allocation of match expressions means that events of type e are never sent to a particular node, that node need not be sent predicates involving attributes of variables of type e.

In Section 3 we noted that each phrase establishes a session type associated with its match expression. Channels of that session type can be used to communicate attributes of variables to a central site, where the predicates associated

with the phrase can be tested. We are planning to build a prototype implementation of *SpaTeC* above Java, with session types provided by SJ [15]. SJ is itself implemented above Polyglot [16, 17]. Polyglot has been around for some time, and should help to provide a stable framework.

In practice, for phrases that associate primitive events from widely distributed locations, some of the predicates may test attributes of variables whose locations of occurrence are clustered. It may make sense to move the evaluation of such predicates nearer to the relevant locations. Session types associated with match expressions should support such optimisations, but we shall need to experiment.

6 Related Work

The event-based paradigm emerged during the 1990s when asynchronous notification was seen to be crucial for a wide range of applications. [18] investigates the use of events for building distributed applications based on an object-oriented distributed programming environment. Other applications include sensor-based systems for environmental monitoring and, in general, applications with low latency requirements. For example, in 1999 [19] discusses the need for event-based middleware in Air Traffic Control.

The detection of meaningful patterns of events (so-called composite events) became a subject of research, for example [13]. But early composite event languages supported only temporal reasoning, based on the temporal properties of events; composite event detection was driven by the times of occurrence of constituent events.

In distributed systems, times of occurrence are fundamentally uncertain, because of the way earth-time is measured, the way computers' clocks are set and drift, and because clocks are synchronized intermittently. In [20] event operators *sequence* and *concurrency* are used to determine whether one event happened before another or whether they may have happened "at the same time", that is, their order cannot be established due to the closeness of the timestamps. In 1999, Liebig and others proposed source-specific interval timestamps to capture this uncertainty [2]. A *sequence* is detected if the uncertainty interval of one event precedes that of another. If the uncertainty intervals overlap it cannot be determined which one occurred first. It may be important for the application to be made aware of this, in cases where physical causality is an issue, such as in real-time monitoring for the control and audit of physical systems.

Spatial reasoning, that is, monitoring the spatial properties of events, was not supported in traditional composite event detection. When the nodes in distributed systems are stationary the location of occurrence of node-related events is implicit, and spatial reasoning is "hard-wired" into the system.

Mobile environments were explored in [21–24, 4, 25, 26] where the focus lies on the adaptation of the publish/subscribe paradigm to different aspects of mobility, such as client mobility or the lack of a system-wide service infrastructure. In

general, location is being considered in the design of a publish/subscribe architecture for mobile systems, but not in the underlying event detection capabilities. In [27], Fiege and others consider mobility in publish/subscribe middleware. The Rebeca [11] content-based publish/subscribe middleware is extended in [24] to accommodate mobile clients, achieving location transparent access to the middleware without loss of quality of service. Chen and others define the notion of spatial event and enable a spatial subscription model [3]. In [4], Cugola and Munoz de Cote develop a distributed publish/subscribe middleware where spatial restrictions can be issued by publishers and subscribers. [5] points out that high-level spatial events are necessary to observe physical world events and mentions that traditional composite event systems are not sufficient for this purpose. However, general spatial reasoning with composite events is not supported.

In [28], Römer and Mattern acknowledge the suitability of event-based mechanisms for monitoring physical phenomena and their spatio-temporal properties in sensor networks, and investigate composite event detection for detecting the real-world states associated with such phenomena. In later work they propose spacetime, a four-dimensional vector space where space is represented in three and time in one dimension, to record the time and location of occurrence of an event [29].

Limiting the visibility of events is another important requirement in sensor networks where the lack of an infrastructure and the dynamically changing network topology demand efficient and adaptive event handling. The scoping concept, as introduced in [30], meets this requirement. It represents a fundamental structuring mechanism for event-based systems where components are bundled recursively into higher-level components, so-called scopes, yielding a hierarchical structure. Event notifications are only delivered to subscribers within the same scope. In [31], Jacobi et al. apply the scoping concept to sensor networks.

7 Conclusions

We have defined the *SpaTeC* composite event language that enables simultaneous matching of event occurrences over space and time. We have built on the model for *SpaTeC* presented in [1]. As a source of examples throughout the paper we have used a real application scenario for which data are available; the analysis of the GPS traces of Cambridge buses made available by Stagecoach to the TIME-EACM project.

We have described some questions that Stagecoach wish to have answered, and used them to motivate our extensions to *SpaTeC*. Composite event patterns are essentially those of the earlier paper, with the addition of *primitive patterns*, which enforce restrictions on the space and/or time of event occurrences. Data fields identified during pattern matching can be tested by predicates that further restrict the relevant combinations of primitive events. We have shown how the language can be used to answer some of the questions posed by Stagecoach.

We have described an implementation scenario for *SpaTeC* composite event detection (CED) above a publish/subscribe system. Following [13] we ensure a clean separation between CED services and the event propagation infrastructure; a CED service is a subscriber to low-level events and a publisher of higher-level events. As in [13], processing of composite events arising from a given *SpaTeC* phrase may be distributed to optimise communication and detection latency.

A major concern in this area of work is the measurement uncertainty both for space and time. We have generalised the foundational work of [2], that embodies the fundamental uncertainty in the measurement of earth-time, to a similar uncertainty in location stamps. The `time` and `location` stamps of the composite events detected are an open interval and a convex hull respectively, see [1]. The `time` stamp of a composite event can therefore be represented naturally; describing a general convex hull is not straightforward, and the `location` stamp representation adopted will depend on the semantics of the application.

Future work is to implement *SpaTeC* as a composite event programming language and to carry out experimental evaluation.

Acknowledgements

The paper draws on a number of research projects over the years. We acknowledge the support of the UK Engineering and Physical Sciences Research Council (EPSRC) through grants GR/T28164 (EDSAC21) and EP/C547632 (TIME-EACM). Stagecoach in Cambridge has greatly assisted TIME-EACM by making GPS data feeds from their buses available in real time, and by suggesting some of the scenarios developed in the paper.

We acknowledge the many fruitful interactions with Alejandro Buchmann and the Databases and Distributed Systems Research Group at the Technische Universität, Darmstadt. We are grateful for helpful contributions to the paper from several of our colleagues at the Computer Laboratory, in particular David Eyers and Alan Mycroft. The latter emphasised the relevance of session types. Interaction with Simon Gay by e-mail identified SJ as a potential tool for implementation.

References

1. Schwiderski-Grosche, S., Moody, K.: The SpaTeC composite event language for spatio-temporal reasoning in mobile systems. In: DEBS '09: 3rd ACM Intl. Conf. on Distributed Event-Based Systems, New York, NY, USA, ACM (2009) 1–12
2. Liebig, C., Cilia, M., Buchmann, A.: Event composition in time-dependent distributed systems. In: 4th Intl. Conf. on Cooperative Information Systems (CoopIS '99). (September 1999)
3. Chen, X., Chen, Y., Rao, F.: An efficient spatial publish/subscribe system for intelligent location-based services. In: 2nd Intl. Workshop on Distributed Event-Based Systems (DEBS'03). (2003) 1–6

4. Cugola, G., de Cote, J.M.: On introducing location awareness in publish-subscribe middleware. In: Distributed Computing Systems Workshop, IEEE (2005)
5. Bauer, M., Rothermel, K.: An architecture for observing physical world events. In: ICPADS '05: 11th Intl. Conf. on Parallel and Distributed Systems (ICPADS'05), IEEE Computer Society (2005) 377–383
6. King, C., McDonald, R., Martin, R., Tempero, G., Holmes, S.: Long-term automated monitoring of the distribution of small carnivores. Wildlife Research (34) (2007) 140–148
7. Bacon, J., Beresford, A., Evans, D., Ingram, D., Trigoni, N., Guitton, A., Skordylis, A.: Time: An open platform for capturing, processing and delivering transport-related data. In: IEEE Consumer Communications and Networking Conference, IEEE (January 2008) 687–691
8. Bejan, A., Gibbens, R., Evans, D., Beresford, A., Bacon, J., Friday, A.: Statistical modelling and analysis of sparse bus probe data in urban areas. In: 13th IEEE Intelligent Transportation Systems Conference, Madeira, Portugal, IEEE (September 2010)
9. Takeuchi, K., Honda, K., Kubo, M.: An interaction-based language and its typing system. In: PARLE'94 Parallel Architectures and Languages Europe, Springer LNCS 817 (July 1994) 398–413
10. Honda, K., Yoshida, N., Carbone, M.: Multiparty asynchronous session types. SIGPLAN Not. **43**(1) (2008) 273–284
11. Mühl, G., Ulbrich, A., Herrmann, K., Weis, T.: Disseminating information to mobile clients using publish-subscribe. IEEE Internet Computing **8**(3) (2004) 46–53
12. Pietzuch, P.R., Bacon, J.M.: Hermes: a distributed event-based middleware architecture. In: 1st Intl. Workshop on Distributed Event-Based Systems (DEBS'02), Vienna, Austria (July 2002) 611–618
13. Pietzuch, P.R., Shand, B., Bacon, J.: A framework for event composition in distributed systems. In Endler, M., Schmidt, D., eds.: 4th ACM/IFIP/USENIX Intl. Conf. on Middleware (Middleware '03), Rio de Janeiro, Brazil, Springer (June 2003) 62–82 Best paper award.
14. Pietzuch, P.R.: Hermes: A scalable event-based middleware. Technical Report UCAM-CL-TR-590, University of Cambridge Computer Laboratory (2004)
15. Hu, R., Yoshida, N., Honda, K.: Session-based distributed programming in java. ECOOP, Springer LNCS **5142** (2008) 516–541
16. Nystrom, N., Clarkson, M.R., Myers, A.C.: Polyglot: An extensible compiler framework for java. In: 12th International Conference on Compiler Construction, Springer-Verlag (2003) 138–152
17. Polyglot: `http://www.cs.cornell.edu/projects/polyglot/`
18. Bacon, J., Bates, J., Hayton, R., Moody, K.: Using events to build distributed applications. In: SDNE '95: 2nd Intl. Workshop on Services in Distributed and Networked Environments, Washington, DC, USA, IEEE Computer Society (1995) 148
19. Liebig, C., Boesling, B., Buchmann, A.: A notification service for Next-Generation IT systems in air traffic control. In: GI-Workshop: Multicast-Protokolle und Anwendungen, Braunschweig, Germany (May 1999)
20. Schwiderski, S.: Monitoring the behaviour of distributed systems. PhD thesis, University of Cambridge (1996)
21. Meier, R., Cahill, V.: STEAM: event-based middleware for wireless ad hoc networks. In: Intl. Workshop on Distributed Event-Based Systems (DEBS'02). (2002) 639–644

22. Huang, Y., Garcia-Molina, H.: Publish/subscribe in a mobile environment. Wireless Networks **10**(6) (2004) 643–652
23. Caporuscio, M., Carzaniga, A., Wolf, A.: Design and evaluation of a support service for mobile, wireless publish/subscribe applications. IEEE Transactions on Software Engineering **29** (2003) 1059–1071
24. Zeidler, A., Fiege, L.: Mobility support with Rebeca. In: 23rd Intl. Conf. on Distributed Computing Systems (ICDCS'03), Washington, DC, USA, IEEE Computer Society (2003)
25. Yoneki, E., Bacon, J.: Unified semantics for event correlation over time and space in hybrid network environments. In: IFIP Intl. Conf. on Cooperative Information Systems (CoopIS'05), Cyprus (November 2005)
26. Frey, D., Roman, G.C.: Context-aware publish subscribe in mobile ad hoc networks. In: COORDINATION. (2007) 37–55
27. Fiege, L., Gärtner, F.C., Kasten, O., Zeidler, A.: Supporting mobility in Content-Based publish/subscribe middleware. In Endler, M., Schmidt, D.C., eds.: ACM/IFIP/USENIX Intl. Middleware Conference (Middleware 2003). Volume 2672 of LNCS., Springer-Verlag (June 2003) 103–122
28. Römer, K., Mattern, F.: Event-based systems for detecting real-world states with sensor networks: a critical analysis. In: DEST Workshop on Signal Processing in Wireless Sensor Networks at ISSNIP, Melbourne, Australia (December 2004) 389–395
29. Römer, K., Mattern, F.: Towards a unified view on space and time in sensor networks. In: Elsevier Computer Communications. Volume 28(13). (August 2005) 1484–1497
30. Fiege, L., Mezini, M., Mühl, G., Buchmann, A.P.: Engineering event-based systems with scopes. In Magnusson, B., ed.: Proceedings of the European Conference on Object-Oriented Programming (ECOOP02). Volume 2374 of LNCS., Springer-Verlag (June 2002) 309–333
31. Jacobi, D., Guerrero, P.E., Petrov, I., Buchmann, A.: Structuring sensor networks with scopes. In: 3rd IEEE European Conference on Smart Sensing and Context (EuroSSC), Zurich, Switzerland, IEEE Communications Society (October 2008)