

# A Space and Time Requirements Logic for Sensor Networks

Rachel Cardell-Oliver, Mark Reynolds, Mark Kranz  
 School of Computer Science & Software Engineering  
 The University of Western Australia  
 {rachel,mark,mkranz}@csse.uwa.edu.au

**Abstract**—User requirements for sensor network applications can be expressed as spatial and temporal constraints on the events observed by sensor network nodes. In this paper a novel logic is introduced for expressing such sensor network requirements. The bi-modal requirements logic is based on timed propositional temporal logic and a one-hop spatial modal logic. The logic is used for expressing sensor network requirements, called spatial-temporal situations. In order to detect the occurrence of spatial-temporal situations in a sensor network, we also propose implementation protocols that run on sensor nodes, filtering events and searching for required properties. The feasibility of spatial-temporal situations and situation detection for sensor network programming is illustrated by two examples: a temporal situation for recognising the occurrence of an explosion, and a spatial situation for detecting contours in a sensor node field.

## I. INTRODUCTION

Wireless sensor network technology promises systems of thousands of low power, low cost, wireless nodes that can monitor and act upon their environment, be it a natural landscape or agricultural setting, factory, warehouse, home, or hospital. However, realising systems that meet this promise is currently constrained by the lack of an effective method for programming large scale sensor networks. There is a significant gap between the goals of sensor network users, and the realisation of those goals as sensor network programs. Current methods for programming sensor networks are based on traditional distributed systems paradigms: a network is programmed to provide a general purpose data logging and data collection system, and off-line support is provided for the analysis of gathered data [11]. But this per-node approach is too low level for the convenient development of sensor network programs, and as a consequence, such sensor networks suffer from inefficiencies, and failure to function correctly.

If, instead, we consider a user’s view of a sensor network, we find goals based on temporal and spatial requirements. For example, a typical user requirement is “if chemical levels or temperatures exceed safe thresholds for at least 30 percent of available measurements in room 1.20 within a period of 15 minutes, then activate the extraction fan and sound an alarm for that room”. This type of requirement goal captures both spatial (e.g. in room 1.20) and temporal (e.g. within 15 minutes) constraints. *Situation requirements* define user goals in terms of temporal and spatial constraints on the events observed by sensor network nodes. We introduce a new logic for expressing situation requirements, that is based on

existing spatial and temporal modal logics. *Situation detection protocols* are sensor network programs for identifying when a situation has occurred. We are working towards a requirements logic and detection protocols that share a model, since then we can verify that a detection protocol satisfies users’ situation requirements.

There are many advantages to the logic-based approach for programming sensor networks that are not provided by existing imperative language solutions:

- our situation requirements logic and situation detection protocols address both temporal and spatial aspects of sensor network requirements, where existing languages address only one aspect;
- we can often take situation requirements formulae directly from users’ stated goals for sensor network behaviour and so can more easily see the correctness of requirements;
- situation detection protocols can be verified to check for desired properties;
- situation requirements can be used as a basis for synthesizing a system to meet a given specification.

The main contribution of this paper is a framework for situation programming in sensor networks. A new logic, timed real-time temporal logic (TRTL), a variant of timed propositional temporal logic (TPTL), is used for expressing *temporal* situation requirements. A combination logic using TRTL and a one-hop spatial logic is used to express *spatial* situation requirements. Situation detection *protocols* are specified in pseudo-code.

We illustrate the effectiveness of this framework with two examples from the literature: a temporal situation for recognising the occurrence of an explosion [20], [21], and a spatial situation for detecting contours in a sensor landscape [42].

## II. RELATED WORK

There have been a number of high level programming primitives proposed for sensor networks including temporal situations [20], abstract regions [42], [25], abstract channels [41], data base abstractions [27] and virtual machine programming languages [22], [41]. SENSID is a middleware system for programming temporal situations on sensor network nodes [20]. EnviroSuite [25] provides an object-oriented model that can encode spatial abstractions on events within a group of nodes. Abstract regions provides an interface for programming

spatial requirements. Kairos is a global macroprogramming model for sensor networks [16]. Kairos provides abstractions for node lists, neighbour lists and remote data access for shared variables. A rule-based language provides static scheduling of periodic tasks, and a channel abstraction for shared variable communication [41]. Each of these techniques focus on protocol implementation issues; they have limited support for defining requirements. Further, existing languages support either temporal or spatial situations but not both. Our framework addresses the problem of expressing situation requirements in a logic, and implementing protocols that satisfy both temporal and spatial aspects of users' goals for sensor network programs.

Temporal logic has long been one of the most popular methods of presenting formal specifications for computing applications including hardware, software, communication protocols, distributed systems, databases and multi-agent systems [5], [28]. The original applications were for simple step-by-step computations and work with a propositional linear time temporal logic (PLTL) over a countable discrete model of time  $0, 1, 2, \dots$  [33]. Sensor network applications, however, typically interact with continuous-valued environments and have a large and indefinite number of nodes working in parallel. For such systems, real-time models of time are more appropriate [18], [9]. In (propositional) real-time temporal logic [36] we assume that each of a set of atomic propositions is true or false at each point of a dense linear flow of time (such as the real numbers or rationals). Fortunately, recent advances [38] show that real-time is no more difficult to reason with than discrete time.

As in many other applications of complex systems from safety critical systems [31] to multimedia specifications [8], timing or metric considerations are important in sensor network specifications. A good account of this so-called real-time logic area appears in [3] and more recent work is reported in [17], [26], [7]. The formula  $G_{=} (x.p \rightarrow F(q \wedge x \leq 1))$  of the logic TPTL [3], for example, can express that every  $p$ -state is followed by a  $q$ -state in at most one unit of time.

Combinations of temporal logic with other modal logics such as deontic and epistemic logic have been used successfully in a wide variety of applications [13], [14]. Various combinations of temporal and spatial modal logic have also been proposed and given solid theoretical foundations [15]. Recent suggestions for using spatial-temporal combinations, as is proposed in this paper, include geographical reasoning [15], distributed systems [32], and traffic monitoring [4].

Some of the main reasoning tasks we will need to undertake (eventually) are *model-checking* [12] of an implementation against a logic specification, *synthesis* [34] of an implementation from a specification, or direct *execution* of the logic [6].

### III. SITUATIONS IN SENSOR NETWORKS

A sensor network is a set of nodes located in a spatial landscape, each sensing and actuating events in its local environment over time. In this way, a sensor network collects a set of events over time and space. An event is characterised by its type, its time of occurrence, and one or more value

attributes. For example, a high temperature event occurs when any reading is above 37 degrees. A reading of 40 degrees Celsius, observed at 1300 hours, is represented by event  $e$  with  $e.type = hightemp$ ,  $e.time = 1300$ , and  $e.value = 40$ . The transmission or reception of a radio packet is also a type of event.

A *situation* is a pattern on a set of events. Situations, first defined for active databases [1], provide a natural way for sensor network users to describe their goals [20]. For example, "at least 50% of nodes in a given region detect temperature above a threshold", or "within a period of 12 hours, soil moisture decreases and no rainfall events are detected".

*Situation requirements* define patterns on sets of events; they can be defined formally using a combined temporal and spatial logic. The neighbour relation, defined by wireless broadcast messages for communication between sensor nodes, is represented by the next node operator of spatial logic  $\cdot$ . Temporal relationships between events, both packet reception and sensor readings, are specified using temporal logic clocks. *Situation detection protocols* are the programs executed on each node in the network in order to recognise and report the occurrence of a given situation. The following sections describe our combined temporal spatial logic for situation requirements, and implementations of detection protocols.

## IV. SITUATION REQUIREMENTS LOGIC

Situations generally involve a combination of spatial and temporal aspects. We first look just at the temporal aspect.

### A. Temporal Situations

In specifying temporal behaviour, there is a variety of formalisms including a wide variety of temporal logics. The more expressive logics tend to be less amenable to human or automated reasoning as they may have undecidable decision problems or be unaxiomatizable. As many typical sensor specifications involve specific durations, a reasonably expressive but manageable approach here is to use propositional metric temporal logics. Even here there is much choice. Examples include MTL [19], RTTL [30], MITL [2] and TPTL [3]. We look here at our own new variation on TPTL which we call TRTL (for timed real-time temporal logic). By allowing unrestricted variability of environmental propositions in real-time, it imposes minimal assumptions on the behaviour of interest.

We will evaluate TRTL expressions on models which have a non-negative real numbers  $\mathbb{R}_0^+$  flow of time and allow propositions from a set  $\mathbf{AP}$  of atomic propositions to vary in their truth values without restriction over time. We use a valuation  $h : \mathbf{AP} \rightarrow \wp(\mathbb{R}_0^+)$  to specify the truth value. In particular, for any  $p \in \mathbf{AP}$ ,  $h(p) \subseteq \mathbb{R}_0^+$  is just the set of time points at which  $p$  is true.

Syntactically, fix the set  $\mathbf{AP}$  of atomic propositions and a disjoint set  $\mathbf{CV}$  of clock variables. The formulas of TRTL are all  $\alpha$  defined by

$$\alpha ::= p|\alpha_1 \wedge \alpha_2|\neg\alpha_1|\alpha_1 U \alpha_2|\alpha_1 S \alpha_2|x.\alpha_1|x \sim c$$

where  $p \in \mathbf{AP}$ ,  $x \in \mathbf{CV}$ ,  $c$  is a rational number and  $\sim \in \{=, \neq, <, \leq, >, \geq\}$ .

We allow standard classical abbreviations such as **true**, **false**,  $\wedge$ ,  $\rightarrow$ ,  $\leftrightarrow$  and the usual temporal logic abbreviations  $F\alpha \equiv \mathbf{true}U\alpha$ ,  $F_{\neq}\alpha \equiv \alpha \vee F\alpha$ ,  $G\alpha \equiv \neg F\neg\alpha$ ,  $G_{\neq}\alpha \equiv \alpha \wedge G\alpha$ ,  $P\alpha \equiv \mathbf{true}S\alpha$ ,  $P_{\neq}\alpha \equiv \alpha \vee P\alpha$ ,  $H\alpha \equiv \neg P\neg\alpha$ ,  $H_{\neq}\alpha \equiv \alpha \wedge H\alpha$ ,  $K\alpha \equiv H\alpha \wedge \alpha \wedge G\alpha$ .

Until  $U$  and Since  $S$  are the usual linear temporal logic operators. They are the *strict* versions which are the right ones to use in dense time [38]: the non-strict versions of until and since, which are usually seen in the context of discrete time flows, depend on the current time to determine their truth values. The past operator ‘‘Since’’ is used as it allows more natural expression of common properties in requirements [24].

A clock variable  $x \in \mathbf{CV}$  holds a real number value at each time point. Normally this advances in tune with time. However, evaluation of the expression  $x.\alpha$  will reset to the  $x$  clock to zero. The propositions of the form  $x \sim c$  allows simple tests on the current value of  $x$ . In defining the semantics, to aid us in determining the current readings of clocks we record their time of being reset, or zero time, and do a subtraction. The clock zeroes are given by a map  $\gamma : \mathbf{CV} \rightarrow \mathbb{R}_0^+$ . Note that using the past-time operator  $S$  in combination with clock variable resets will allow the variables to hold negative readings.

More formally the semantics requires truth of formulas to be evaluated in models, at a certain time, with a record of clock zero times. We write  $M, \gamma, t \models \alpha$  iff the formula  $\alpha$  is true at time  $t \in \mathbb{R}_0^+$  in the structure  $M = (\mathbb{R}_0^+, <, h)$  with the current clock zeros  $\gamma : \mathbf{CV} \rightarrow \mathbb{R}_0^+$ . This is defined recursively by:

$M, \gamma, t \models p$	iff	$t \in h(p)$ , for any $p \in \mathbf{AP}$
$M, \gamma, t \models \neg\alpha$	iff	$M, \gamma, t \not\models \alpha$
$M, \gamma, t \models \alpha \wedge \beta$	iff	$M, \gamma, t \models \alpha$ and $M, \gamma, t \models \beta$
$M, \gamma, t \models \alpha U \beta$	iff	$\exists s > t. M, \gamma, s \models \beta$ and $\forall u$ , if $t < u < s$ then $M, \gamma, u \models \alpha$
$M, \gamma, t \models \alpha S \beta$	iff	$\exists s < t. s \in \mathbb{R}_0^+$ and $M, \gamma, s \models \beta$ and $\forall u$ , if $s < u < t$ then $M, \gamma, u \models \alpha$
$M, \gamma, t \models x.\alpha$	iff	$M, \gamma[x \mapsto t], t \models \alpha$
$M, \gamma, t \models x \sim c$	iff	$(t - \gamma(x)) \sim c$

By assuming a default initial arrangement with all clocks reset at time zero, we can evaluate formulas directly in a model (at time zero). Let  $\mathbf{Z} : \mathbf{CV} \rightarrow \mathbb{R}_0^+$  be the map which makes each  $\mathbf{Z}(x) = 0$ . Then we write  $M, 0 \models \alpha$  iff  $M, \mathbf{Z}, 0 \models \alpha$  (according to the definition above).

Note that TRTL generalises TPTL (under the interval semantics [7]) in that its formulas can be evaluated even with arbitrary behaviour by the atomic propositions. That is, there is no assumption of an underlying discrete state sequence with so-called finite variability. However, state sequences can be used if required.

## B. Spatial Situations

Let us now move to consider the other, spatial, dimension of situations. There are many quite different options here. The purpose-built modal logics for spatial representation and reasoning include compass logic [43] and variations (eg RCC8, BRCC-8) on encoding the Region Connection calculus of [35]. These can also be further encoded into more traditional logics such as Lewis’s modal logic  $S4_u$  [15]. There are also possible ways of using epistemic logic [14] to model sensor nodes as agents which reflect on the ‘‘knowledge’’ of themselves and other nodes.

The simple approach we use here is just adequate to capture the one-hop neighbour relation. We introduce first a purely spatial modal logic without a temporal dimension. We treat the various nodes as possible worlds in a traditional Kripke structure. Give the modality  $\diamond$  this one-hop relation as its accessibility relation and also use a universal modality  $\blacksquare$  to give access to all existing nodes. For example,

$$\blacksquare(\diamond p \rightarrow \square q)$$

denotes that every node which has at least one immediate neighbour having the property  $p$  (= danger say), also has all of its immediate neighbours having property  $q$  (= alerted say).

## C. Combined Situations

The theoretical foundations for the process of combining temporal, spatial and other modal logics is a well-established one [15], [37], [39]. Powerful techniques exist but also some open problems. However, there has been little investigation of practical applications of temporal-spatial logics in contexts such as sensor network design and verification.

To combine TRTL and our simple one-hop spatial logic we use straightforward cartesian product semantics [15]. This allows us to specify quite a reasonable range of properties about our sensor network evolving over time. As an example consider the combined temporal-spatial formula

$$G_{\neq}\blacksquare(\diamond x.danger \rightarrow F(\square alerted \wedge (x \leq 2)))$$

This says that at all times, at all nodes, if at least one immediate neighbour of a node senses danger then, within 2 seconds, all of that node’s immediate neighbours will be alerted.

## V. IMPLEMENTING PROTOCOLS FOR SITUATION DETECTION

Situation detection protocols are programs executed on sensor network nodes to identify the occurrence of situations. A simple set of programming primitives is sufficient for this purpose: broadcasting or receiving a radio message, reading a sensor, writing to an actuator, setting timers, and saving data to a log.

In our situation requirements logic, events are represented by atomic propositions. In an implemented sensor network, nodes sense their immediate environment, and identify events in terms of conditions on sensed values. For example, a network node senses the current temperature of its immediate

environment at regular intervals. If a particular temperature reading exceeds a threshold, a high temperature event is said to have occurred at that time. In our logic events may occur at any real-valued time. In implementations, physical constraints of the environment and sensor, such as sensing periods, enforce temporal constraints on event occurrences, and so limit the number of events that may be detected in a given time interval.

In order to detect situation patterns using sensor network nodes, we must first define a context in which to search within the potentially huge set of events observed by each sensor network node. Examples of sensor network events include threshold conditions on sensor readings, receiving radio packets, and timeout events. The *type context* of a situation is given by the set of event types relevant to it. The *temporal context* of a situation, also called its *lifespan*, is initiated by a particular type of event from the type context, and terminated a fixed interval of time after initiation, or by another event.

Given type and temporal contexts for a situation, a detection protocol first collects a set of candidate events from observations of a node over time. The set of candidate events is then tested in order to determine whether a situation has occurred or not.

Sensor network nodes detect a situation by performing the following tasks:

- 1) Await the start event for the situation;
- 2) Set a timer to terminate the collection of situation events;
- 3) Collect, filter and log situation events offered by the environment until the situation timer expires;
- 4) Whenever relevant, check whether the events collected so far satisfy the situation,
- 5) Notify applications whenever a situation is detected.

Candidate events need to be logged because, in general, detecting temporal situations requires backtracking through a set of previously observed events to check whether a value constraints are satisfied.

## VI. SITUATION EXAMPLES

In this section we show how our programming framework of situation requirements logic and situation detection protocols can be used model two benchmark situation detection problems from the literature: explosions [21], [20] and contours [42].

### A. Explosion Detection

Consider the problem of detecting an explosion and raising an alarm [21], [20]. A sensor network scenario for explosion detection is a collection of identical nodes located in a landscape, each independently observing and reporting explosions in its local environment. Each node is equipped with three sensors: light, sound and temperature, and one actuator: an alarm. A *flash* event is defined to have occurred whenever the light sensor returns an above threshold value. Similarly, an above threshold sound reading is a *noise* event, and above threshold temperature reading is a *heat* event.

Nodes log flash, noise and heat events, and the time at which each occurs. The occurrence of an explosion can be deduced

if all three events occur within 6 seconds, say, in the sequence flash, noise, heat. Furthermore, noise must be no more than 2 seconds after flash, but heat and noise should be at least 3 seconds apart. Between the first occurrence of flash, and 6 seconds after any subsequent flashes 0, 1 or more explosions may have occurred. Whenever an explosion is detected then the node must actuate its alarm. Usually, the alarm is actuated at the time an explosion is detected, that is when the heat event of the explosion occurs. Explosion detection is a complex temporal situation, beyond the scope of most existing event detection systems [40], [21], [20].

We now show how TRTL can be used to specify explosion situation requirements. There are many different ways to do this using either past-time or future operators. Here we present a few of these options.

Let  $f, n, h$  be atomic propositions indicating the occurrence of flash, noise, heat events respectively. The start of an explosion pattern is represented by proposition  $e$  and the end of an explosion pattern by proposition  $a$  indicating an alarm event. Let  $x, y$ , and  $z$  be clocks. First we want a declarative definition of an explosion, so we want to say that proposition  $e$  is true exactly at the same time as any flash which is part of the specified arrangement of some flash, noise, heat sequence.

$$G_=(e \leftrightarrow (x.f \wedge F(y.n \wedge x \leq 2 \wedge F(h \wedge x \leq 6 \wedge y \geq 3))))$$

This formula is read: always,  $e$  is true when and only when, a flash event occurs and clock  $x$  is reset, and in the future a noise event occurs, clock  $y$  is reset, and clock  $x$  is at most 2, and further in the future, a heat event occurs at which time clock  $x$  is at most 6 and clock  $y$  is at least 3. Note that it is quite possible for instants of noise and heat to be part of several explosion sequences.

To take us closer to an implementation we might want to specify a detection event occurring at a time when it is temporally possible for an explosion to be detected and the alarm sounded, that is at the end of an explosion pattern. This specification suggests a way to test for explosions by searching back through past events whenever a heat event is observed.

$$G_=(a \leftrightarrow (z.h \wedge P(y.n \wedge z \leq -3 \wedge P(f \wedge z \geq -6 \wedge y \geq -2))))$$

This formula is read: always, alarm proposition  $a$  is true when and only when, a heat event occurs and clock  $z$  is reset, and in the past a noise event occurred, clock  $y$  was reset, and clock  $z$  was no more than -3, and further in the past, a flash event occurred at which time clock  $z$  was at least -6 and clock  $y$  is at least -2.

Considerations of proof theory for TRTL are beyond the scope of this paper but straightforward semantic reasoning gives us the sanity check that the following is a valid consequence of the two specifications, capturing the temporal relations that hold between the alarm and explosion start propositions: if  $e$  is true then  $a$  is also true at most 6 seconds in the future and if  $a$  is true then  $e$  must have been true at most 6 seconds previously.

$$G_=(w.e \rightarrow F(a \wedge (w \leq 6)))$$

$$G_=(u.a \rightarrow P(e \wedge (u \geq -6)))$$

```

isactive=false;
loop {
  await event.type IN
    < flash, noise, heat >;

  // initiate lifespan(s) and log events
  if (event.type==flash) {
    log(event);
    isactive = true;
    stoptime = now + 6 seconds
  }

  if (event.type==noise) and (isactive) {
    log(event);
  }

  if (event.type==heat) and (isactive) {
    log(event);
    // actuate the alarm if
    // an explosion has occurred
    if (isexplosion(log,now)) {
      signal(explosion,now);
    }
  }

  // terminate lifespan
  if (stoptime<=now) {
    isactive = false;
  }
}

boolean isexplosion(log,now) {
  exists fe, ne, he IN log.
    fe.type==flash and
    ne.type==noise and
    he.type==heat and
    he.time==now and
    he.time-fe.time<=6 and
    he.time-ne.time>=3 and
    ne.time-fe.time<=2
}

```

Fig. 1. Pseudo Code for Explosion Detection

Figure 1 shows pseudo-code for a situation detection protocol that satisfies the explosion situation requirements. Situation detection on any node is triggered by an initiation event, a flash, and terminates 6 seconds afterwards. Since the termination timer is reset whenever a flash event occurs, the situation detection protocol may run indefinitely. This protocol will detect overlapping explosions. It will sound the alarm whenever at least one explosion pattern has occurred. It is also possible that there is more than one satisfying set of events for each reported situation.

Detecting explosions requires an implementation that can backtrack through previously observed events. For example, consider the sequence of (timed) observations: (flash,0),

(noise,1), (noise,2), (heat,3), (heat,4). Of the four possible combinations of flash, noise, and heat from this sequence, only (flash,0),(noise,2),(heat,4) satisfies the explosion situation requirements. The other possible combinations, such as (flash,0),(noise,1),(heat,4), do not satisfy the requirements.

In the pseudo-code of Figure 1, all flash, noise and heat events within an active lifespan are logged. There are, however, a number of additional checks that could be made before logging an event. For example, events need only be stored in the log until they are 6 seconds old, and noise events need not be logged unless there is already a flash event within the previous 2 seconds, and a similar condition for heat events. Such conditions can be derived from the situation requirements formulae, and so used in the synthesis of situation detection protocols.

The protocol checks for a possible explosion whenever a heat event occurs whilst the explosion lifespan is active. If an explosion has occurred then the node actuates its alarm by signalling that the situation has been detected.

### B. Contour Detection

Spatial situations require a group of nodes to cooperate in order to detect an event pattern of interest. A typical example of a spatial situation is contour detection [42]: the process of discovering isolines in a field of sensor readings across a landscape. In particular, we are interested in identifying the set of nodes for which the detecting node has a high range sensor reading, and at least one of its neighbours has a low range reading. The set of all these falling gradient pairs identifies contours in the sensor field.

The sensor network scenario for contour detection is a set of nodes placed across a landscape, each equipped with a sensor, say temperature, for reading its local environment, as well as a radio for transmitting and receiving messages to its immediate neighbours. Again we use thresholds to define events: the low event is said to occur whenever the sensor reading is below a given threshold, and the high event for readings above another threshold. Nodes broadcast messages to their neighbours periodically to communicate their own state. When a node detects that it is on a contour, it signals this event, for example, by transmitting a contour message.

The situation requirement for contour detection can be specified in our simple temporal-spatial bi-modal logic: the combination of TRTL and the one-hop spatial logic. Let *high*, *low* and *contour* be atomic propositions representing the occurrence of a high or low sensed event and contour detection, respectively. The requirement states that, eventually, any node knows that it is on a contour if and only if that node has a high reading and at least one of its immediate neighbours has a low reading.

$$F\blacksquare((high \wedge \diamond low) \leftrightarrow contour)$$

In order to implement contour detection, nodes need to exchange state information and indicate the discovery of a contour. Nodes communicate by broadcasting messages that are received by their immediate neighbours. A node recognises that at least one of its neighbours has a low reading when it

```

isactive=false;
loop {
  await event.type IN
    < sensehigh, senselow, lowmsg >;

  // initiate lifespan(s)
  if (event.type==sensehigh) {
    isactive = true;
    stoptime = now + 60 seconds
    mySense = high;
  }

  //notify neighbours if sensor is low
  if (event.type==senselow) {
    broadcast(lowmsg,myID);
  }

  //detect falling contours
  if (event.type==lowmsg) and (isactive) {
    if (mySense==high) {
      hisID = event.value;
      signal(countour,
        myID, hisID, now);
    }
  }

  // terminate lifespan
  if (stoptime<=now) {
    isactive = false;
  }
}

```

Fig. 2. Pseudo Code for Contour Detection

receives a *low* message. A node indicates the occurrence of a falling contour by signalling that a contour has been detected.

Figure 2 shows pseudo-code for a detection protocol to run on each node in a sensor network. The type context for this situation has three types of event: *sensehigh* and *senselow* capture sensor readings above or below critical thresholds and *lowmsg* denotes the reception of a broadcast radio message from a neighbouring node. Nodes may receive such messages from more than one neighbour during the 60 seconds (say) after detecting a high sensor reading. Each node knows its unique identifier, *myID*, and uses a local variable, *mySense*, to remember its most recent sensor reading.

## VII. CONCLUSION

In this paper, we have demonstrated the feasibility of sensor network programming using situations. We have provided a new framework for high level programming consisting of,

- situation requirements logic: a new temporal logic, timed real-time temporal logic (TRTL), a variant of timed propositional temporal logic (TPTL), for expressing *temporal* situation requirements;
- a combination of TRTL and a one-hop spatial logic to express *spatial* situation requirements for sensor networks;

- executable situation detection protocols;
- preliminary verification of properties of situation requirements.

We plan to address a number of open questions in our future work including: the extension of spatial logic beyond single-hop relations; a proof system for TRTL; synthesis (semi-automatic) of situation detection protocols from requirements; extensions to make spatial-temporal requirements more readable for non-experts; and investigation of robust situation detection for different sensor network topologies and communication properties.

## REFERENCES

- [1] Asaf Adi and Opher Etzion. Amit — the situation manager. *The VLDB Journal*, 13(2):177–203, 2004.
- [2] Rajeev Alur, T. Feder, and Thomas A. Henzinger. The benefits of relaxing punctuality. *J. ACM*, 43(1):116–146, 1996.
- [3] R. Alur and T. A. Henzinger. Real-time logics: Complexity and expressiveness. *Inf. and Contr.*, 104:35–77, 1993.
- [4] Stefania Bandini, Davide Bogni, Sara Manzoni, and Alessandro Mosca. St-modal logic to correlate traffic alarms on italian highways: Project overview and example installations. In Moonis Ali and Floriana Esposito, editors, *IEA/AIE*, volume 3533 of *Lecture Notes in Computer Science*, pages 819–828. Springer, 2005.
- [5] H. Barringer, M. Fisher, D. Gabbay, and G. Gough, editors. *Advances in Temporal Logic*. Kluwer Academic Publishers, 2000.
- [6] H. Barringer, M. Fisher, D. Gabbay, R. Owens, and M. Reynolds, eds. *The Imperative Future*. Research Studies Press, 1996.
- [7] P. Bouyer, F. Chevalier, and N. Markey. On the expressiveness of TPTL and MTL. In R. Ramanujam and Sandeep Sen, editors, *Proc. FSTTCS 2005*, pages 432–443. Springer LNCS, 2005.
- [8] H. Bowman, H. Cameron, P. King, and S.J. Thompson. Mexitl: Multimedia in executable interval temporal logic. *Formal Methods in System Design*, 22:5–38, January 2003.
- [9] J. P. Burgess and Y. Gurevich. The decision problem for linear temporal logic. *Notre Dame J. Formal Logic*, 26(2):115–128, 1985.
- [10] Rachel Cardell-Oliver. *Why Flooding is Unreliable in Multi-hop, Wireless Networks* Technical Report UWA-CSSE-04-001, February 2004.
- [11] Rachel Cardell-Oliver, Keith Smettem, Mark Kranz, and Kevin Mayer. A reactive soil moisture sensor network: Design and field evaluation. *International Journal of Distributed Sensor Networks*, pages 149–162, 2005.
- [12] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.
- [13] F. Dignum and R. Kuiper. Specifying deadlines with continuous time using deontic and temporal logic. *International Journal of Electronic Commerce*, 3(2):67–86, 1998.
- [14] R. Fagin, J. Halpern, Y. Moses, and M. Vardi. *Reasoning about Knowledge*. The MIT Press, 1995.
- [15] D. M. Gabbay, A. Kurucz, F. Wolter, and M. Zakharyashev. *Many-Dimensional Modal Logics*. Elsevier, 2003.
- [16] R. Gummedi, O. Gnawali and R. Govidan. Macro-programming Wireless Sensor Networks using *Kairos*. In *DCOSS 05 (Proc. Conf. Distributed Computing in Sensor Systems)*, volume 3560 of *Lecture Notes in Computer Science*, pages 126–140. Springer-Verlag, Berlin, 2005.
- [17] Y. Hirschfeld and A. Rabinovich. Logics for real time: Decidability and complexity. *Fundamenta Informaticae*, 62:1–28, 2004.
- [18] Y. Kesten, Z. Manna, and A. Pnueli. Temporal verification of simulation and refinement. In *A decade of concurrency: reflections and perspectives: REX school, Noordwijkerhout, 1993*, pages 273–346. Springer-Verlag, 1994.
- [19] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Syst.*, 2(4):255–299, 1990.
- [20] Mark Kranz. *SENSID: A situation detector for sensor networks*, Honours Thesis, School of Computer Science and Software Engineering, University of Western Australia, 2005.
- [21] Li, S., Lin, Y., Son, S. H., Stankovic, J. A., and Wei, Y. Event Detection Using Data Service Middleware. in *Distributed Sensor Networks Special Issue on Wireless Sensor Networks of Telecommunications Systems*, 26(June):351–368, 2004.

- [22] P. Levis and D. Culler. Maté: a tiny virtual machine for sensor networks. In *Proc. of the 10th Int. Conference on Architectural support for programming languages and operating systems*, pages 85–95. ACM, 2002.
- [23] P. Levis, D. Gay, V. Handziski, J-H. Hauer, B. Greenstein, M. Turon, J. Hui, K. Klues, C. Sharp, R. Szewczyk, J. Polastre, P. Buonadonna, L. Nachman, G. Tolle, D. Culler and A. Wolisz *T2: A Second Generation OS For Embedded Sensor Networks*, Technical Report TKN-05-007, Telecommunication Networks Group, Technische Universität Berlin, November 2005.
- [24] O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. In R. Parikh, editor, *Logics of Programs (Proc. Conf. Brooklyn USA 1985)*, volume 193 of *Lecture Notes in Computer Science*, pages 196–218. Springer-Verlag, Berlin, 1985.
- [25] Liqian Luo, Tarek F. Abdelzaher, Tian He, and John A. Stankovic. Envirosuite: An environmentally immersive programming framework for sensor networks. *ACM Transaction on Embedded Computing Systems*, 2006. to appear.
- [26] C. Lutz, D. Walther, and F. Wolter. Quantitative temporal logics: PSPACE and below. In *TIME*, pages 138–146. IEEE Computer Society, 2005.
- [27] Samuel R. Madden, Michael J. Franklin, Joseph M. Hellerstein, and Wei Hong. TinyDB: an acquisitional query processing system for sensor networks. *ACM Transactions on Database Systems*, 30(1):122–173, 2005.
- [28] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Springer-Verlag, New York, 1992.
- [29] Sule Nair and Rachel Cardell-Oliver. Formal specification and analysis of performance variation in sensor network diffusion protocols. In *MSWiM '04: Proceedings of the 7th ACM international symposium on Modeling, analysis and simulation of wireless and mobile systems*, pages 170–173. ACM Press, 2004.
- [30] J. S. Ostroff. *Temporal Logic for Real-Time Systems*. Advanced Software Development Series. John Wiley and Sons, 1989.
- [31] J. Ostroff. Composition and refinement of discrete real-time systems. Technical report, DCS, York University, Ontario, Canada, 1998. CS-1998-10.
- [32] Dirk Pattinson and Bernhard Reus. A complete temporal and spatial logic for distributed systems. In Bernhard Gramlich, editor, *FroCos*, volume 3717 of *Lecture Notes in Computer Science*, pages 122–137. Springer, 2005.
- [33] A. Pnueli. The temporal logic of programs. In *Proceedings of the Eighteenth Symposium on Foundations of Computer Science*, pages 46–57, 1977. Providence, RI.
- [34] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *16th ACM Symp. on Principles of Programming Languages, Austin*, pages 179–190, 1989.
- [35] D. Randell, Z. Cui, and A. Cohn. A spatial logic based on regions and connection. In Nebel, Rich, and Swartout, eds., *KR'92. Princ. of Knowledge Representation and Reasoning: Proc. of the 3rd Int. Conf.*, pages 165–176. Morgan Kaufmann, California, 1992.
- [36] M. Reynolds. An axiomatization for U and S over the reals without the IRR rule. *Studia Logica*, 51:165–193, 1992.
- [37] M. Reynolds. A decidable logic of parallelism. *Notre Dame Journal of Formal Logic*, 38:419–436, 1997.
- [38] M. Reynolds. The complexity of the temporal logic with until over general linear time. *Journal of Computer and System Sciences*, 66:393–426, 2003.
- [39] M. Reynolds and M. Zakharyashev. On the products of linear modal logics. *JLC*, 11(6):909–931, 2001.
- [40] Romer, K., and Mattern, F. Event-Based Systems for Detecting Real-World States with Sensor Networks: A Critical Analysis. In *DEST Workshop on Signal Processing in Sensor Networks at ISSNIP*, pages 389–395. Melbourne, Australia, 2004.
- [41] Shondip Sen and Rachel Cardell-Oliver. A Rule-Based Language for Programming Wireless Sensor Actuator Networks using Frequency and Communication. In *EmNetS-III. The Third IEEE Workshop on Embedded Networked Sensors*, Harvard, 2006.
- [42] Matt Welsh and Geoff Mainland. Programming Sensor Networks Using Abstract Regions. In *Proceedings of the First USENIX/ACM Symposium on Networked Systems Design and Implementation*, NSDI '04, March 2004.
- [43] Y. Venema. Expressiveness and completeness of an interval tense logic. *NDJFL*, 31:529–547, 1990.