Decentralized environmental simulation and feedback in robust geosensor networks

Matt Duckham[†] and Femke Reitsma[‡]
[†] Department of Geomatics, University of Melbourne, Australia
[‡] School of Geosciences, University of Edinburgh, UK

November 25, 2008

Abstract

Environmental simulation and modeling has become a basic tool for scientific exploration of complex natural systems. Geosensor networks are an emerging technology for spatiotemporal monitoring of natural systems. This paper investigates the potential for embedding environmental simulations in geosensor networks, with the aim of increasing system robustness to both sensor and model errors. The approach taken is to construct a formal, process-oriented model of a decentralized geosensor network capable of executing a spatially distributed environmental model. This formal model is used as the basis for implementing a computer simulation of a geosensor network tasked with monitoring a simple hydrological system. Three experiments are then conducted to investigate the ability of this system to tolerate different types of sensor and model errors. The results indicate that embedded environmental models can indeed contribute to improved system robustness, by detecting, distinguishing between, and even correcting for certain types of error.

Keywords

Decentralized spatial computing; Wireless sensor networks; Geosensor networks; Environmental monitoring; Spatiotemporal systems; Process-orientation

1 Introduction

Environmental modeling and simulation is a long-established technique in the environmental sciences for the investigation of complex natural systems, from hydrology (Beven, 2002; Pullar, 2003) to vegetation dynamics (Muetzelfeldt and Massheder, 2003). Geosensor networks (wireless networks of tiny, sensor-enabled, collaborating computing devices) are a relatively new, emerging technology for environmental monitoring applications, enabling fine-grained monitoring of remote, hazardous, and sensitive natural and managed environments, such as coral reefs (Kininmonth et al., 2005), vineyards (Beckwith et al., 2004), and sea bird nesting sites (Szewczyk et al., 2004a,b). The objective of this paper is to fuse techniques from environmental modeling and geosensor networks. This combination is expected to have benefits in at least three key areas:

1. Feedback: Real-time data from geosensor nodes can be validated by comparison with predicted data from environmental models, supporting error detection, calibration, and improved robustness of geosensor networks. Conversely, sensed data may also be used to calibrate and validate the environmental model.
2. Event detection: Conventional approaches to event detection in geosensor networks typically rely on the definition of simple and static thresholds. Integrating simulation into geosensor networks allows more complex event definitions, based on the dynamic characteristics of changing environments.

3. Process-orientation: The combination of simulation and geosensor networks can be used to embed in the network an explicit representation of the spatial processes expected to occur within the environment. This can help support process-oriented modeling, moving geosensor networks beyond purely snapshot-based representations of change.

In this paper we focus primarily on the first of these advantages: feedback. Feedback from sensed data to environmental models can be used as a basis for reparameterization of these models. Conversely, feedback from environmental models to sensed data can be used to detect and correct sensor errors. We propose a new theoretical approach to implementing short-term environmental models within geosensor networks, and investigate empirically how the combination of environmental modeling and geosensor networks can help improve robustness and fault-tolerance.

Geosensor networks are a highly distributed form of spatial computing, where individual sensor nodes are able not only to collect data, but also to process their data and that of their immediate neighbors. A key goal in geosensor network research is to perform as much data processing as possible *in the network* (termed *decentralized* spatial computing) for three main reasons. First, geosensor networks are resource-limited computing environments, in particular having limited battery power. Processing data is much less energy intensive than communicating data, so research has focused on reducing communication by performing in-network processing. Second, the enormous quantities of data produced by geosensor networks means that processing in the network helps avoid the potential for the communication bottlenecks and information overload associated with centralized processing architectures, thus making networks more scalable. Third, the information generated by geosensor networks is often required by the network itself in order to modify its behavior or control other systems (termed sensor/actuator networks). In such cases, removing sensed data from the network, processing it centrally, and then reinserting processed data into the network for actuation is inefficient.

In fact, geosensor networks are just one example of a suite of new emerging technologies for earth observation networks, which have collectively enjoyed rapid advances and development over recent years (Lubick, 2005). Potentially, the approach developed in this paper for combining real-time monitoring and modeling has application across the entire range of such technologies. However, the unique resource constraints of geosensor networks, identified above, present particular challenges for the development of this technology, and make decentralized tools and techniques investigated in this paper especially relevant. Consequently, in the remainder of this paper we focus specifically on the application of decentralized environmental simulation and feedback to geosensor networks.

Following a brief motivational example in section 1.1, this paper provides some background information on the key topics of geosensor networks and simulation in section 2. A formal model of decentralized in-network environmental modeling systems for geosensor networks is then set out in section 3. This model is then itself implemented in an agent simulation system, Repast (section 5). The results of this simulation are then discussed in section 6, with conclusions contained in section 7.

1.1 Motivational example

To illustrate the general goals outlined above, consider the example of environmental monitoring of coral reefs using geosensor networks (such as the geosensor network monitoring of the Great Barrier Reef, described in Kininmonth et al., 2005). High temperatures cause the symbiotic relationship between the coral animals and zooxanthellae plants to break down. In turn, this causes "bleaching" of usually colorful corals. Sustained high temperatures can lead to permanent bleaching and ultimately death and disintegration of coral reefs. Monitoring high temperatures using geosensor networks is therefore an important input to decision making in coral reef environments. It can also enable marine scientists to understand the responses of corals to temperature change at much finer spatial and temporal granularity than previously possible.

In such a situation, combining environmental simulation with geosensor network would be expected to have benefits in several key areas:

1. Feedback: Replacing sensor nodes in remote marine environments is expensive and hazardous. Lost data can never be recovered. Decentralized in-network environmental models could provide improved capabilities for automated detection of faulty or mis-calibrated sensors.
2. Event detection: The precise environmental triggers for coral reef bleaching are poorly understood, but are certainly more complex than simple thresholds. Decentralized in-network environmental models could provide an additional tool for investigating bleaching processes, enabling scientists to treat real environments as research labs, formulating and testing hypotheses through decentralized in-network environmental models.
3. Process-orientation: Natural resource managers often require answers to high-level process-oriented queries, such as "How is the coral bleaching process changing over time?" rather than low-level snapshot-oriented queries, such as "What is the current temperature across this area?" Decentralized in-network environmental models embed knowledge about spatial processes within geosensor networks, ultimately supporting the use of such process-oriented queries.

1.2 Summary

The key contributions of this paper are:

- the development of a formal model of distributed, decentralized, in-network environmental modeling, based on IO automata; and
- the application of this formal model in an empirical investigation of a simulated simplified watershed runoff model.

2 Background

2.1 Environmental modeling

Researchers in the field of environmental modeling are recognizing the potential of geosensor networks for capturing data at finer spatiotemporal scales and allowing for new types of data capture for environmental applications (Hart and Martinez, 2006). Enhancing environmental models with data captured by geosensor networks will involve either calibrating existing models to finer spatiotemporal scales or developing new models. In capturing data at new scales, we are likely to gain new insights into how environmental processes work and interact, which will be reflected in new processes represented in environmental models.

Environmental models represent environmental processes with mathematical equations or sets of rules. There are many different approaches to classifying models (Wainright and Mulligan, 2004). One useful approach is to characterize them in terms of their representation of space, including:

- *lumped models*, where values across space are lumped together and the heterogeneity of space is ignored;
- *semi-distributed models*, where we have multiple lumps reflecting different parts of the environmental system;
- *fully decentralized models*, where our equations or rules are applied to a regular set of discrete spatial units that tessellate the area of interest (e.g. raster cells); and
- *spatially distributed models*, which is not only fully decentralized, but spatial relationships that exist across spatial units are recognized in the model equations or rules. These interactions may be in one, two or three dimensions.

Geosensor networks can support fully decentralized and spatially distributed models. In this paper we examine spatially distributed models, where the rules are fully decentralized and account for spatial relationships linking neighboring model units.

2.2 Geosensor networks

A substantial amount of existing research in the area of wireless sensor and geosensor networks has focused on communication and routing issues. The literature contains a wide variety of different techniques for efficient, ad-hoc routing of information through the network (Karp and Kung, 2000; Perkins et al., 2001; Braginsky and Estrin, 2002). By contrast, this paper is not concerned with routing issues. The model presented in this paper uses only direct communication between immediate neighbors in the network, and hence no routing is required. Routing might be required for subsequently retrieving the information generated by a decentralized, in-network, environmental simulation (for example, to display the results of a query to our system using the sensor web, or communicate an SMS alert to a scientist or decision maker). However, such issues are beyond the scope of this paper, and we assume here that the existing routing techniques are adequate for this purpose.

This paper is, however, concerned with another issue of general interest to wireless sensor network research: modeling the *global* properties of wireless sensor networks based on the *local* rules for individual sensor nodes. The decentralized nature of wireless sensor networks presents significant problems to the designers of these systems. Researchers must design new algorithms and applications for wireless sensor networks by first defining the local rules and protocols to be obeyed by each individual sensor node. Conventionally, having defined these local rules, the next step is to implement the system (either within a real wireless sensor network or more usually within a virtual simulation of a wireless sensor network). This implementation is then used as the basis for empirical investigation of the global properties of the system, such as how scalable or efficient the protocol is.

However, in this paper we argue that an additional *analytical* stage should be adopted between design and implementation. This analytical stage can have at least three important benefits:

- it allows more rigorous verification of the global system properties that emerge from local behaviors than can be achieved with empirical methods;
- it provides a reference documentation for implementation, lessening the likelihood of impedance mismatch when implementing the design;
- it enables effective communication of the precise system design, enhancing comparison between related systems.

The formal modeling tool used to support the analytical stage in this work is IO automata, discussed further in sections 3 and 4.

3 DINEM: Decentralized in-network environmental modeling

In this section, the structure and behavior of a *decentralized, in-network environmental modeling* (DINEM) system is formally defined.

We assume individual sensor nodes monitor one or more environmental variables, such as temperatures or flows. In addition, sensor nodes are equipped with a local model of the *process* of change of one or more of the environmental variables being sensed. We further assume that each sensor node is able to determine its own geographic location. Finally, each node is able to communicate information about its location and current environmental variables to its immediate neighbors.

Given these assumptions, each sensor node can locally compute the expected value at one or more of its neighbors' locations, and communicate that information to its neighbors. These neighbors can subsequently identify whether their own observed values agree with the predictions of its observed values. This information might have a variety of applications, from simply creating an alert event where discrepancies are discovered, to more sophisticated uses discussed in later sections.

3.1 Preliminaries

We assume a set of sensor nodes $V = \{v_1, ..., v_n\}$ with the potential for direct local communication between pairs of nodes, modeled as an (undirected) graph $G = (V, E)$ where $E \subseteq V \times V$. There exists a great deal of literature addressing different algorithms and techniques for establishing such communication networks (e.g., Akyildiz et al., 2002; Braginsky and Estrin, 2002; Karp and Kung, 2000 to name but three). Our model is highly decentralized in that we only assume single hop communication between immediate neighbors. Let $nbr(v_i) \subset \mathbb{N}^+$ denote the set of indexes for the immediate neighbors of v_i in the graph, i.e., $nbr(v_i) \equiv \{j|\{v_i, v_j\} \in E\}$ and let $loc(v_i) \in \mathbb{R}^2$ denote the location of the node in geographic space (for example, a coordinate location in the plane). We assume that each sensor node knows its own location. Finally, in this simple case the sensors are monitoring a single environmental variable (or for brevity *en-var*). Without loss of generality, we assume the en-var has the set of values $\mathbb{R}$ as its domain in the sequel.

3.2 Sensor node IOA

In this paper we adopt a formal model of process-oriented communicating systems, called Input/Output (IO) automata, or IOA. IOA are used in engineering as a formal model of distributed concurrent systems. The detailed specifications of IOA models are contained within literature (Lynch and Tuttle, 1989; Lynch, 1996), so we provide only enough information to build our specific model.

Decentralized and distributed systems are notoriously complex, and as a result design of such systems is often ad hoc and purely empirical, for example relying on simulation (e.g., using cellular automata). Formal models like IOA are analytical tools that enable precise specification of system design, allowing rigorous verification of system properties as well as providing a reference documentation for subsequent empirical simulation implementation. Consequently, in this paper we adopt the position that formal models, like IOA, are an important intermediate stage in the distributed system development process that should be used in support of subsequent empirical investigation and implementation.

The IOA model is closely related to process algebraic formalisms also used in spatial information science, such as Hoare's CSP (communicating sequential processes) or Milner's CCS (calculus of communicating systems, cf. Worboys, 2005; Worboys and Duckham, 2006). These formalisms essentially provide a mechanism to model the behavior and interactions of dynamic systems. The primary use of such formalisms is to help the designer to understand how the *local* behaviors of elements of a dynamic system combine to yield desirable *global* system properties. Although IOA lack some of the expressive power and rich structure of process algebra, IOA was chosen as the formal modeling tool for this research as it provides a higher level of abstraction than process algebras (Vaandrager, 1991), and as a result can be simpler to use. IOA are also supported by a simulation language (IOA) and system (the IOA simulator), although in this paper we only use IOA for modeling and analysis purposes.

The key components of an IOA are its *signature*, *states*, and *transitions*.

3.2.1 Signature

The signature of an IOA is the set of actions an automaton can perform. There are three types of actions: input, output, and internal actions. For example, the signature for a generic feedback sensor node v_i contains six actions, one input, two output, and three internal as in Figure 1.

An individual sensor node has internal actions that enable it to sense the en-var value ($s \in \mathbb{R}$) at its location (*sense*); predict values for the en-var at each of the node's neighbors locations (*predict*); and compare its locally sensed en-var value with the expected value predicted for its own location by its neighbors (*check*). The output action *alert*(l) flags when the internal *check* action uncovers a discrepancy between the sensed information and the process model. The data l output by *alert* is the location of the sensor node that created the alert. Finally, the pair of input and output actions $comm_{j,i}$ and $comm_{i,j}$ are used to communicate information about sensed and predicted en-var values between neighboring nodes. The information transmitted from a node v_i to one of its neighboring nodes v_j is

Signature v_i:

Input:	Internal:
$comm_{j,i}(x, s)$	$sense_i(s)$
Output:	$predict_i$
$comm_{i,j}(x, s)$	$check_i$
$alert_i(l)$	

Figure 1: Signature of sensor node v_i

composed of the current en-var value at v_i's location (s), and sensor node v_i's prediction of the current value at the location of v_j (x).

3.2.2 States

Actions change the state of an automaton. The states of the sensor node IOA are defined and initialized as in Figure 2. For a sensor node v_i, s_i stores that node's most recently sensed en-var value; s_j stores the most recently sensed en-vars at the location of each neighboring node, $j \in nbr(v_i)$; x_j stores v_i's most recent predictions for the en-vars at the locations of neighboring nodes; p_j stores the most recent predictions for the en-var value at node v_i's location by its neighboring nodes. In addition, the sensor node has a number of Boolean state variables, termed "gates" (*has_input*, *can_output*, *is_alerted*), that are used to control the activation of the different actions. The locations of a node $loc(v_i)$ and its immediate neighbors $loc(v_j)$, where $j \in nbr(v_i)$ are also assumed to be known by each sensor node. This information is assumed to be static (i.e., in our model nodes are not mobile) and so for simplicity this information is omitted from the state of a node.

States v_i:
$s_i : \mathbb{R},\ s_i \leftarrow 0$
$s_j : \mathbb{R},\ s_j \leftarrow 0,\ j \in nbr(v_i)$
$x_j : \mathbb{R},\ x_j \leftarrow 0,\ j \in nbr(v_i)$
$p_j : \mathbb{R},\ p_j \leftarrow 0,\ j \in nbr(v_i)$
$has\_input_{j,i} : \{true, false\},\ has\_input_{j,i} \leftarrow false,\ j \in nbr(v_i)$
$can\_output_{i,j} : \{true, false\},\ can\_output_{i,j} \leftarrow false,\ j \in nbr(v_i)$
$is\_alerted_i : \{true, false\},\ is\_alerted_i \leftarrow false$

Figure 2: States of sensor node v_i (where $a : b, c$ indicates the state variable name a, the state variable type or domain b, and the initial state c)

3.2.3 Transitions

Finally, transitions define the changes in state caused by an action. All actions have a (possibly empty) list of "effects" that define the changes in state induced by that action. All output and internal actions also have a (possibly empty) list of "preconditions" that specify the conditions under which the action is permitted to occur. IOA are not allowed to block input, and so input actions always have no preconditions. Figure 3 lists the transitions defined in the sensor node IOA.

These transitions may be summarized as follows:

- $comm_{i,j}(x, s)$: The *comm* output action requires as a precondition that the node v_i is ready to output its prediction for neighbor v_j (i.e., $can\_output_{i,j} = true$) and has the effect of blocking the repeated output of the same information ($can\_output_{i,j} \leftarrow false$).
- $comm_{j,i}(x, s)$: The *comm* input action for v_i receives and stores the neighboring v_j node's sensed en-var and v_i's prediction of its own en-var. Like all input actions, this transition can have no preconditions.

Transitions v_i (where $j \in nbr(v_i)$):

```
$comm_{i,j}(x, s)$
    Preconditions:
        $can\_output_{i,j} = true$,
        $x = x_j$, $s = s_i$
    Effects:
        $can\_output_{i,j} \leftarrow false$

$comm_{j,i}(x, s)$
    Effects:
        $has\_input_{j,i} \leftarrow true$,
        $p_j \leftarrow x$, $s_j \leftarrow s$

$alert_i(l)$
    Preconditions:
        $is\_alerted_i = true$,
        $l = loc(v_i)$
    Effects:
        None

$sense_i(x)$
    Preconditions:
        $x \in \mathbb{R}$
    Effects:
        $s_i \leftarrow x$

$check_i$
    Preconditions:
        $has\_input_{j,i} = true$ for all $j \in nbr(v_i)$
    Effects:
        $has\_input_{j,i} \leftarrow false$ for all $j \in nbr(v_i)$,
        if $s_i \sim \kappa(\{p_k | k \in nbr(v_i)\})$
            then $is\_alerted_i \leftarrow true$
        else $is\_alerted_i \leftarrow false$

$predict_i$
    Preconditions:
        None
    Effects:
        $x_j \leftarrow \pi_{i,j}(s_i, \{s_k | k \in nbr(v_i)\}, \{loc(v_k) | k \in nbr(v_i)\})$
        $can\_output_{i,j} \leftarrow true$
```

Figure 3: Transitions of sensor node v_i

- $\mathit{sense}_i(x)$: The *sense* internal action receives data about the en-var (i.e., from a node's sensor). This ev-var value is stored as s_i. Note that a more sophisticated model might treat *sense* as in input action and explicitly model the sensor node's interaction with an "environment" automaton.

- check_i: The *check* internal action compares the observed en-var s_i with some combination of p_j, the values predicted for the en-var of s_i by its neighbors. Depending on the result of this comparison, an alert may or may not be enabled through the Boolean value $\mathit{is\_alerted}_i$. The operation κ denotes an application-specific combination of the values of p_j, (such as computing the sum, product, average, maximum, etc). The relationship $\sim$ denotes an application-specific comparison of s_i and the operation, such as equals, less than, greater than, etc. As a precondition, this transition checks that all neighbors v_j have communicated their predicted value to v_i.

- $\mathit{alert}_i(l)$: The *alert* output action outputs the location of the sensor node v_i that has detected a discrepancy between the predicted and actual en-vars. As a precondition, this transition checks whether the $\mathit{is\_alerted}_i$ flag has been set by the check_i action.

- $\mathit{predict}_i$: The *predict* internal action computes the expected en-var values for the neighbors v_j of v_i. The operation $\pi_{i,j}$ denotes an application-specific procedure for v_i to compute the predicted values of the en-var for neighbors v_j. This operation is expected to use the sensed value s_i in addition to other locally available information, including the sets of current sensed en-vars $\{s_k | k \in \mathit{nbr}(v_i)\}$ and spatial locations of neighbors $\{\mathit{loc}(v_k) | k \in \mathit{nbr}(v_i)\}$.

3.2.4 Composition

The IOA formalism provides a mechanism for combining multiple automata to form new composite automata. As a result, having defined an individual sensor node automaton v_i, it is possible to defined a composite sensor network automaton as the combination of multiple sensor node automata. Informally, the composition of a set of automata associates together actions that have the same name. So, for example, the output $\mathit{comm}_{1,2}$ of v_1 will become associated with the input $\mathit{comm}_{1,2}$ of v_2, and vice versa (output $\mathit{comm}_{2,1}$ of v_2 becomes associated with the input $\mathit{comm}_{2,1}$ of v_1). In essence, compositions are well-formed as long component automata have disjoint sets of action names, aside from pairs of connecting input/output actions. The reader is referred to Lynch (1996) for the precise formal details of IOA composition.

We can define the DINEM IOA N to be the composition Π of sensor nodes $N = \Pi_{i \in 1..n} v_i$. Such a composition is well-formed because for any set of sensor nodes $\{v_i | i \in 1..m\}$ only pairs of *comm* actions will have the same name.

3.2.5 Schedules

For a composite automaton N, an *execution* of N is defined as a sequence $s_0, a_1, s_1, a_2, ..., a_n, s_n$ of alternating states s_i and actions a_i, with the constraint that all subsequences s_i, a_{i+1}, s_{i+1} (termed *steps*) are *enabled* in N. For a step to be enabled, the initial state s_i must satisfy the preconditions (if any) specified in the transition for action a_{i+1}. A *schedule* is a subsequence of actions from an execution. For simplicity, schedules will be used in place of executions to describe the behavior of a DINEM IOA in the following text. Schedules are effectively equivalent to executions in this paper, because DINEM IOA are *deterministic*, in the sense that given an initial state s_i any enabled action a_{i+1} leads to exactly one resulting state s_{i+1} (hence the states may be omitted).

For example, for a DINEM IOA N, a schedule of the form $\mathit{sense}_i(s), \mathit{predict}_i, \mathit{comm}_{i,j}(x, s), \mathit{comm}_{j,i}(x, s), \mathit{check}_i, ...$ is allowed since sense_i and $\mathit{predict}_i$ have no preconditions, the precondition for $\mathit{comm}_{i,j}$ is enabled by $\mathit{predict}_i$, the precondition for $\mathit{comm}_{j,i}$ is enabled by $\mathit{comm}_{i,j}$, and the precondition for check_i is enabled by $\mathit{comm}_{j,i}$. Many other schedules are not allowed, such as $\mathit{sense}_i(s), \mathit{comm}_{i,j}(x, s), \mathit{comm}_{j,i}(x, s), \mathit{predict}_i, \mathit{check}_i, ...$ since a node must compute a prediction before the *comm* action (which communicates that prediction to a neighbor) becomes enabled.

Using IOA provides a powerful tool for precise definition and analysis of the structure and dynamic behavior of a decentralized feedback system. These capabilities are especially useful in moving from the formal IOA model to developing empirical simulations, explored further in the following section.

4 From formal model to implementation

Constructing a formal model using IOA can assist in designing and developing more practical simulations and implementations in three main ways. Specifically, the IOA model:

1. allows pen-and-paper exploration of the dynamic behavior of the feedback model;
2. supports formal analysis and verification of system properties; and
3. provides a basis for precise communication with the system designer, assisting in the development of design structures like UML diagrams.

4.1 Simple DINEM IOA

This section provides an example of how the IOA model can be used directly to explore the dynamic behavior of a DINEM system. Consider the very simple sensor network $N = \Pi_{v_i \in V} v_i$, composed of sensor nodes $V = \{v_1, v_2, v_3\}$ where the communication network is described by $E = \{\{v_1, v_2\}, \{v_2, v_3\}\}$. Suppose the combination operation κ in the *check* action is the sum of the predicted values (i.e., $\kappa(S) = \Sigma_{s \in S} s$) and the comparison operation $\sim$ is equivalence ($=$). Further, let us define the operation $\pi_{i,j}$ as follows:

$$\pi_{i,j}(s_i, \{s_k | k \in nbr(v_i)\}, \{loc(v_k) | k \in nbr(v_i)\}) \mapsto \begin{cases} s_i \text{ if } i < j \\ 0 \text{ otherwise} \end{cases}$$

Finally, we assume each sensor node may sense the total amount of water in its vicinity (from both overland flow and precipitation). This sensor network might model a very simple hydrological model, where rainfall flows downhill, local sensed by nodes v_1, v_2, v_3 as illustrated in Figure 4. Clearly, this model is far too naïve to provide useful hydrological inferences, as it contains only overload flow, lacking any models of processes like infiltration, interception, evaporation, or models for uncertainty. Despite being highly oversimplified from a hydrological perspective, the model is useful from a decentralized systems perspective, and can help provide insights into the dynamic behavior of the DINEM system. It is also worth noting that the process model π is designed such that the predictions of a node v can only have an impact for a node v' that is downhill from v. In general, the *model neighborhood* for a node (i.e., the neighbors that a node can make predictions for) will be a subset of the *communication neighborhood* (i.e., the neighbors that a node can communicate with).

Given the DINEM IOA N, the following schedule models a rainfall event of 3mm at v_1, and the subsequent step of communication of predicted en-vars between neighboring pairs of sensor nodes. Note that for simplicity time has been deliberately "factored out" of our automata, and we only model time in terms of a sequence of actions (i.e., the order of actions is known, but our model does not specify the length of time that passes between consecutive actions). Again, a more sophisticated model would require a quantitative rather than a purely qualitative model of time, but the simplification is appropriate for our purposes here.

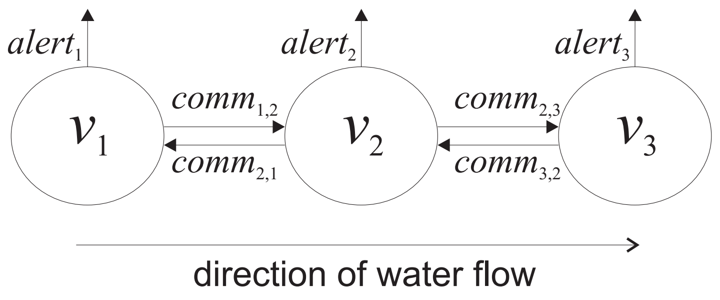


Figure 4: DINEM IOA to model simple hydrological system

$$\begin{aligned}&\mathit{sense}_1(3),\mathit{sense}_2(0),\mathit{sense}_3(0),\mathit{predict}_1,\mathit{predict}_2,\mathit{predict}_3,\\&\quad\mathit{comm}_{1,2}(3,3),\mathit{comm}_{2,3}(0,0),\\&\quad\mathit{comm}_{2,1}(0,0),\mathit{comm}_{3,2}(0,0),\end{aligned}$$

In the next steps in the execution, we show the slightly more complex situation where as expected v_2 senses a total of 3mm of water in its vicinity, but another rainfall event in v_3 causes a discrepancy between the predicted (0mm) and actual (2mm) en-vars, leading to the generation of an alert.

$$\mathit{sense}_1(0),\mathit{sense}_2(3),\mathit{sense}_3(2),\mathit{check}_1,\mathit{check}_2,\mathit{check}_3,\mathit{alert}_3(\mathit{loc}(v_3))$$

We end the schedule with one further step, computing and communicating new predicted values to neighbors in the presence of no further rainfall events, and so generating no alerts:

$$\begin{aligned}&\mathit{predict}_1,\mathit{predict}_2,\mathit{predict}_3,\\&\quad\mathit{comm}_{1,2}(0,0),\mathit{comm}_{2,3}(3,3),\\&\quad\mathit{comm}_{2,1}(0,3),\mathit{comm}_{3,2}(0,2),\\&\quad\mathit{sense}_1(0),\mathit{sense}_2(0),\mathit{sense}_3(3),\\&\quad\mathit{check}_1,\mathit{check}_2,\mathit{check}_3\end{aligned}$$

Note that in this last sub-schedule none of the nodes' *alert* actions are enabled, because the preconditions of the *alert* behavior are not met (the $\mathit{is\_alerted}_i$ state is *false* for all $i \in \{1,2,3\}$).

4.2 Formal analysis of system properties

While the pen-and-paper analysis in the previous section can be useful in gaining an intuition for the dynamic behavior of a DINEM system, IOA models also provide for more formal analysis of system properties. This section highlights some of the key results that can be obtained using IOA.

4.2.1 Fairness

There are an unlimited number of possible executions for a DINEM automaton, but we are primarily interested in executions where every action gets a "fair" chance to be performed. For example, for the simple DINEM automaton used as an example in the previous section (Figure 4), the schedule

$sense_1(x), comm_{1,2}(1,1), comm_{1,2}(1,1), comm_{1,2}(1,1), ...$ is *allowed.* However, such a schedule is "unfair" in the sense that node 1 keeps outputting the same data, but this data is never received by node 2, and no other actions are ever performed by node 1.

Fairness is formally defined in a general setting in (Lynch, 1996, p212), but in the specific context of a DINEM automaton, a schedule is fair if every enabled action is (eventually) performed. Thus, for an arbitrary DINEM automaton $N = \Pi_{v_i \in V} v_i$, where $V = \{v_1, ..., v_n\}$, any schedule of the form

$$predict_i, comm_{i,j}(x,s), comm_{j,i}(x,s), sense_i(s), check_i, [alert_i(l)], ...$$

is fair (where action $alert_i(l)$ is performed if it is enabled by the preceding *check* action). There are many other possible fair schedules, but the schedule above is an important example as removing any of the actions in the schedule above leads to an unfair schedule, and the order of actions ensures every action is enabled in its turn. As such, it therefore provides a useful prototype for implementing a DINEM system, helping to concisely analyze and explain the key steps and their correct order.

4.2.2 Composition and scalability

The composition operator in IOA is defined so that the properties of schedules from individual elements can be used to infer the properties of schedules for composite IOA. In general, when proving properties about composite automata, it is enough to show that the properties hold for the individual components (Lynch, 1996, p211). Thus, in the example above of automaton fairness above, since a trace of the form $predict_i, comm_{i,j}(x,s), comm_{j,i}(x,s), sense_i(s), check_i, [alert_i(l)], ...$ is fair in each individual sensor node, then the DINEM automaton formed by the composition of nodes is also fair. Further, since the composition involved an arbitrary set of nodes $V = \{v_1, ..., v_n\}$, we can infer that this schedule will be fair for *any* DINEM automaton, irrespective of size or connectivity.

Compositional reasoning is one example of how IOA provides one mechanism to reason about *global* system properties based only on the definition of *local* component behavior. Achieving such reasoning (inferring global properties from local rules) was one of the central theoretical challenges, identified at the beginning of this paper, facing any decentralized computing system .

4.2.3 Adversarial analysis

An important class of analysis techniques for decentralized systems can be termed *adversarial analyses.* In an adversarial analysis, a fictitious adversary is invented to compete with a decentralized algorithm. The objective is to use the adversary to uncover degenerate cases that expose the limitations of any decentralized system or algorithm.

For example, one question for an adversarial analysis would be whether the DINEM system can ever generate a false alert. In fact, an adversary using the fair schedule described above can generate a false alert by taking advantage of uninitialized states in the first steps of the automaton. The initial step of the schedule $predict_i, comm_{i,j}(x,s), comm_{j,i}(x,s), sense_i(s), check_i, [alert_i(l)], ...$ makes predictions without any sensed data (the first $predict_i$ comes *before* the first $sense_i(s)$). Thus, in the (likely) event that the default value (0) stored in the sensor buffer s_i does not match the actual environmental surface water depth, this initial iteration will detect a discrepancy and cause an erroneous alert. This "edge effect" can be counteracted by ensuring that nodes have a chance to initialize before checking for alerts (e.g., as in section 4.1 initialize the schedule with an extra *sense* action prepended to the schedule). These edge effects have direct implications for the design of a robust simulation system in the following section (cf. section 5.3.1).

Another important question for an adversarial analysis is whether the system can ever become *deadlocked.* Deadlock occurs when the composite automaton becomes "stuck" in a state where nodes are waiting for input from each other. In a DINEM automaton the potential for deadlock exists in connection with the *has_input* gate, where an automaton has communicated its values to all its neighbors, and must await similar input from all its neighbors before the *check* action becomes enabled. In the case where neighborhoods are not symmetric (e.g., node a is in the neighborhood of b, but b is not in the neighborhood of a) it would be possible to devise a situation where node b becomes deadlocked waiting for input from node a that will never be sent. A cycle of such asymmetries can be used by an adversary to deadlock an entire composite automaton. In our DINEM automaton, the communication network is defined to be undirected, and so symmetric, and as a result no such deadlocks can occur. In real sensor networks, however, such asymmetries do often occur, for example where asymmetries in node battery power allow one node to broadcast information to another node, but not vice versa. Although beyond the scope of this paper, further practical implementation would need to extend DINEM automata to be tolerant to such situations (e.g., by having adaptive neighborhoods, where nodes are removed if they drop out of communication for a certain time).

4.3 System design

Finally, in addition to formal analysis and pen-and-paper exploration of the system, IOA can assist in system design. For example, Figure 5 shows a UML class diagram derived directly from the DINEM IOA model. This diagram forms the basis for the development of the DINEM simulation system explored in the following section. The gsnImpl class in Figure 5 implements the key behaviors (i.e., actions) of a DINEM node, alert, check, predict, and sense. Communication between neighbors is achieved through definition of the interface gsnInterface and its comm method. Using the getNbrs method on the gsnImpl class, a node can retrieve the interface for neighboring geosensor nodes, enabling nodes to communicate data to neighbors at the same time as encapsulating (i.e., hiding) access to a node's local actions.

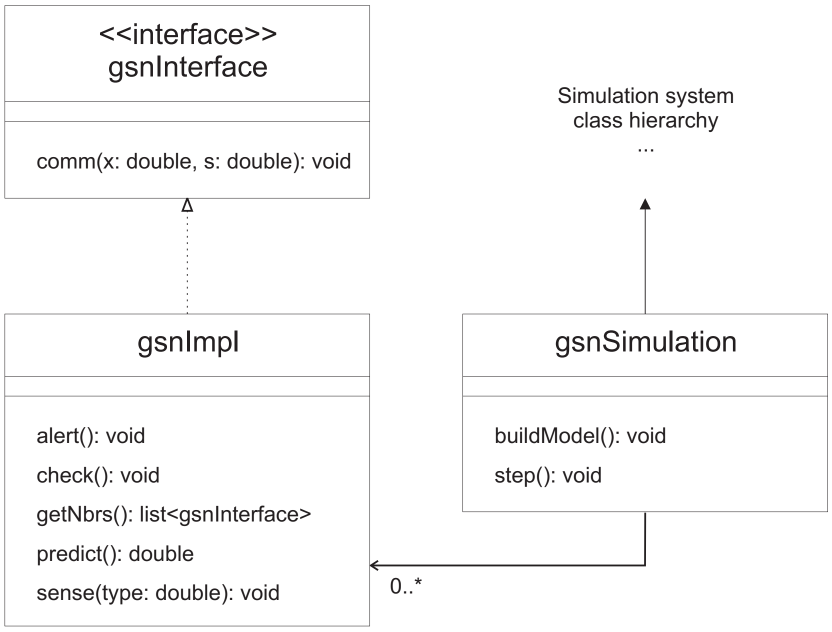


Figure 5: UML class diagram for DINEM simulation system

The gensensor node itself forms an agent within a larger simulation system, modeled as an association with a gsnSimulation class with methods to build a new simulation model and iterate over individual simulation steps.

Finally, the IOA model helps not only with the static properties of the implementation, but also with the design of dynamic interaction between DINEM components. Figure 6 shows a UML interaction diagram for the first few actions of the simple DINEM automaton given in section 4.1. The diagram illustrates the relative sequence and interaction between the three geosensor nodes, as well as the sensor simulation object (class gsnSimulation) driving environmental simulation and changes in sensed data for the nodes. Such UML diagrams are basic tools in the system design and development described in section 5, capturing key static and dynamic features of the underlying DINEM IOA model.

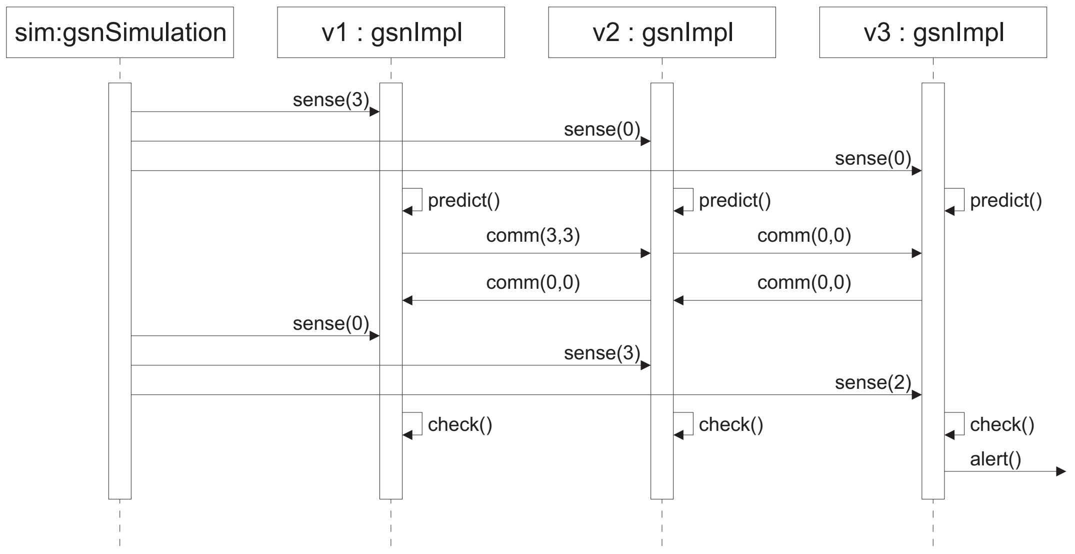


Figure 6: UML interaction diagram for three DINEM geosensor nodes and simulation engine

5 Implementation

A simple test case was developed to implement and evaluate the formal IOA model developed in sections 3 and 4. This test case involved a stripped-down, spatially distributed hydrological model on a very small subset of data. The objective was not to analyze and validate the model results against the real world, as the model is highly simplified, but rather to explore the dynamics of the geosensor network and the DINEM system.

5.1 Hydrological model data

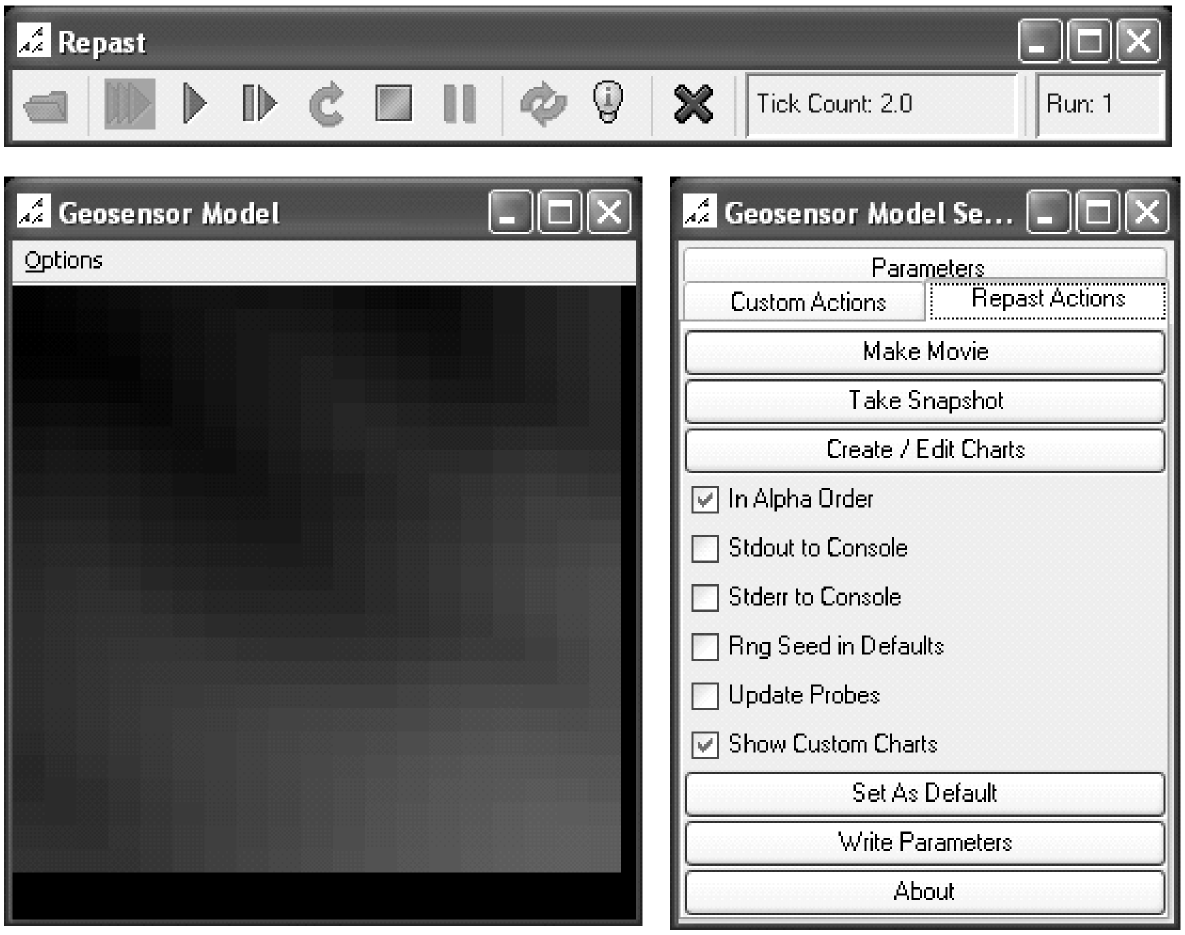
The hydrological model was constructed using a *cellular automaton* (CA). Cellular automata provide a modeling framework for spatially continuous phenomena, such as landscape processes or urban sprawl (Favis-Mortlock et al., 2000; Torrens, 2006). In its most simple form, a CA is composed of a uniformly tessellated space (1D, 2D, or 3D) whose units or cells may exist in a finite number of discrete states. The state of each cell changes based on the state of its neighborhood, where these changes can be deterministic, probabilistic, or expert rule-based (Fonstad, 2006). Despite the simplicity of construction, the dynamics of a CA model can produce complex results.

Our simplified hydrological model involves only one process, overland flow, which we model as flowing over a digital elevation model (DEM) and being initiated by precipitation. The digital elevation model is of a small sub-watershed, Upper Sheep Creek in the Reynolds Creek experimental watershed (Marks, 2001). A set of surfaces of precipitation were generated from point based data of the whole watershed. These surface layers describe hourly precipitation over a restricted temporal interval that was selected as it had a single distinct precipitation event making it easier to test and develop. Because the hydrological model has been stripped down to a simple flow routing exercise, a set of hourly layers of surface water that mirrored this simple model were generated. These surface water layers were created by running the CA model, which involved routing flow over the surface using the D8 algorithm (Endreny and Wood, 2003), and storing the distribution of surface water at each hour.

5.2 Simulator setup

Having generated example hydrological data using a CA, this data was then fed back into a simulated geosensor network, represented as a set of immobile agents with behavioral rules specifying states and transitions as described in the IOA model in the previous section. Nodes could sense precipitation, elevation, and surface water at any point in time. Nodes also had behaviors corresponding directly to actions in the IOA model (including "sense," "check," and "predict" actions, as detailed in the UML diagrams derived from the formal IOA model in section 4). The nodes of the network were uniformly distributed, with a node in each cell of the DEM. Although this simplifying assumption of uniform distribution of nodes is not realistic for actual geosensor networks, it was deemed adequate for the purposes of this simulation. Each node in the network possessed local knowledge of its own state and the state of its immediate eight (Moore's or "Queen's case") neighbors only.

Repast, an open source agent-based modeling environment that includes CA, created by Social Science Research Computing at the University of Chicago (`http://repast.sourceforge.net/`), was used for the simulation. Repast has display and scheduling classes as well as Java classes for importing and exporting GIS raster data, which were useful for the precipitation data and DEM (Figure 7). The relationship between the custom programmed classes and the Repast simulation system is summarized in the UML class diagram in the previous section, Figure 5.



Tick Count: 2.0

Run: 1

- [✓] In Alpha Order
- [ ] Stdout to Console
- [ ] Stderr to Console
- [ ] Rng Seed in Defaults
- [ ] Update Probes
- [✓] Show Custom Charts

Figure 7: Repast GUI and DEM display (lighter shading represents higher elevation)

5.3 Simulating the model

In simulating the environmental model embedded in the geosensor network, care was taken to incorporate spatial and temporal edge effects, as already mentioned previously in section 4.2.3.

5.3.1 Spatial and temporal edge effects

Due to the effect of edges on the sensed and predicted values, the spatial extent of predictions and checking those predictions against sensed values had to be constrained (Figure 8). While the whole spatial extent of the environment is sensed (e.g., for precipitation and overland flow), if the simulation were to make predictions on that full extent, the edge cells would have an incorrect prediction for overland flow as areas outside of the sensed extent would contribute to this value. Furthermore, the simulation would not correctly predict the outflow of water from a predicted cell as the topography of cells outside the boundary would not be known. Consequently, a buffer of a depth of one cell was added to solve this problem. Inside this buffer, sensor nodes only sensed their environment, but did not perform any predictions.

Checking predictions against sensed values also needs to be buffered, as sensor nodes at the edge of the prediction area are adjacent to nodes that do not make any predictions. Thus in a similar manner for predictions, we constrain the spatial extent for checking predictions against sensed values to one cell depth further than the predicted layer (Figure 8).

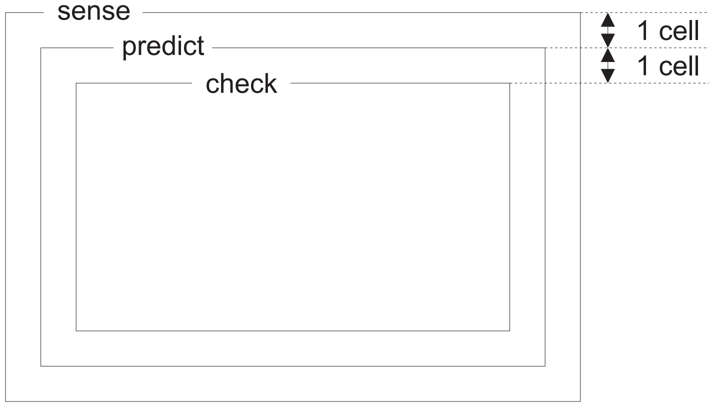


Figure 8: Constraints on the spatial extent of prediction and checking

We also need to account for the effect of the temporal edge, namely the start time of the simulation. In the first step of the model (t_1), the schedule of transitions for each sensor node is to first sense the environment and then predict the surface water. At the second step (t_2) and all subsequent steps (t_n), the schedule is to sense the environment, check the sensed values against those predicted from t_{n-1} and then make predictions for the next time step.

t_1 : sense, predict

t_2–t_n : sense, check, predict

5.3.2 Simulation steps

The geosensor network and embedded overland flow model were simulated, where at each hourly time step the precipitation layer was updated which initiated overland flow over the DEM. Barring the first time step, at each time step the sensed values of surface water are checked against predicted values of surface water. If there is a discrepancy between these values then sensor node transitions to an alerted state. This can be displayed spatially, as shown in Figure 9. Here white nodes represent sensor nodes that are not in the alerted state and where the sensed and predicted values concur. Grey nodes represent alerted sensor nodes where the sensed and predicted values are different.

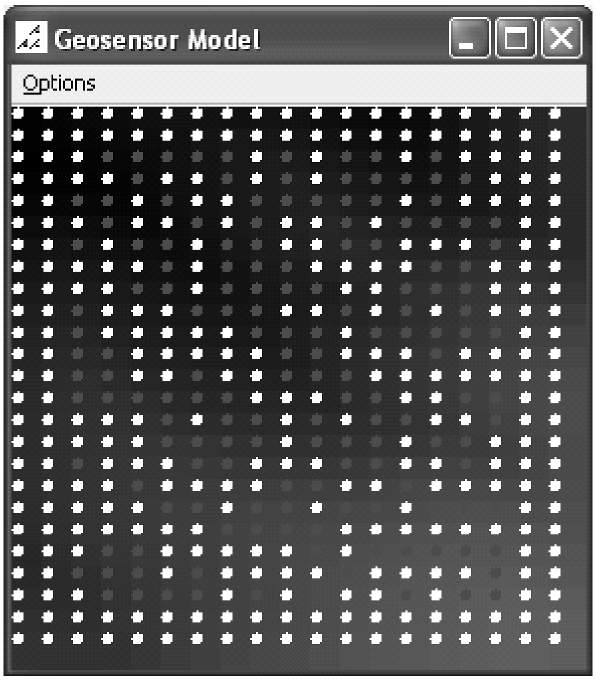


Figure 9: Sample output displaying alerted nodes in dark gray

5.4 Simulating error

Error is an endemic feature of geosensor networks and of environmental models. In order to produce low-cost, scalable geosensor networks, individual sensors need to be cheap, and so are typically error-prone and poorly calibrated. Environmental models too involve simplifying assumptions and may be incorrectly parameterized. Consequently, robust DINEM systems need to accommodate errors, compensating for low quality data and model by using the volume of data generated and the ability to perform decentralized real-time processing on that data. Four different sensor error scenarios were tested: 1) error-free sensors; 2) biased sensors (to simulate poor calibration); 3) noisy sensors, with random noise added (to simulate low cost, inaccurate sensors); and 4) biased and noisy sensors (to simulate poorly calibrated and inaccurate sensors). Two different model error scenarios were tested: 1) error-free model; 2) biased model (to simulate model mis-parameterization). Our model is deterministic, so we do not consider noisy predictions, with random error introduced. Sensor noise is assumed to vary randomly across different sensors and different timesteps. By contrast, sensor bias is assumed to be additive and vary randomly across different sensors, but be constant across different timesteps (i.e., a sensor's bias does not drift or change over time). Model bias is assumed to be constant across all sensors and all timesteps (all sensors run the same model).

These error scenarios meant a total of eight types of simulation were tested. At each step, every sensor node checks for discrepancies between its locally sensed data and the data predicted for its location by that node's neighbors, as already described in the previous section. Nodes transition into an alerted state when a discrepancy is detected, as outlined in Table 1.

	no sensor error	sensor bias	sensor noise	sensor bias & noise
no prediction error	No alert	Alert	Alert	Alert
prediction bias	Alert	Alert	Alert	Alert

Table 1: Sources of alert states

6 Results and discussion

Three experiments were conducted to investigate the robustness of the DINEM system. The first experiment compares the global system traces for the eight different error scenarios tabulated in table 1. The second experiment extends the node behavior to detect one particular source of error, sensor bias, and locally recalibrate sensors accordingly. The third experiment further extends node behavior to distinguish between two different sources of error, sensor bias and model bias, and again react to the error detected to increase the overall accuracy of the DINEM system.

6.1 Experiment #1: Global system characteristics

Table 2 presents examples of typical aggregate traces from the global model, comparing the eight different error scenarios tested. Each trace shows the *total* surface water (y axis) varying over time (x axis). The dotted line on each trace shows the actual surface water in the system (generated using the CA model, constant across all scenarios). After an initial "spike" as a rainfall event inputs water to the system, the water slowly dissipates, moving out of the simulated geographical area as overland flow. The solid line shows the sensed total surface water in the system, aggregated from the sensor values from all nodes. The dashed line shows the predicted total surface water in the system, aggregated from the predictions of all sensor nodes.

6.1.1 Discussion

The different scenarios summarized in Table 2 exhibit a range of different characteristics. Where no errors exist, actual, sensed, and predicted values are all in agreement as expected. Where sensor bias exists, the total sensed values are consistently higher than the actual values (across multiple simulations). Although sensor biases are randomly chosen, such that the set of all sensors has a mean bias of zero, surface water is a *ratio* number, so even biased sensors will not sense less than zero water. Hence, across the simulation globally, there is a tendency to overestimate the total amount of water in the system. The predicted values track closely the sensed values, elevated above the actual values as a consequence the aggregate sensor bias.

Where sensor noise exists, the sensed values exhibit random noise in the traces, but again exhibit a slightly elevated total amount of sensed surface water due to the lack of negative sensed values. Similarly, the predicted traces show the effects of random noise, often with a slight delay in the effects evident, since the predictions are based on the noisy sensor data. Where model bias exists, the total predicted values are always higher than the actual and sensed values, since the (global) prediction bias used in these examples was positive.

The different traces clearly show the range of behaviors of the system under different error scenarios. In order to increase system robustness, it will be necessary not only to detect errors, but also to distinguish between and correct errors of different types, investigated in the following two experiments. Unfortunately, in such a robust DINEM system we would not usually have access to the *actual* values as a baseline against which to compare. This "ground truth" data is available in our simulated world, but in reality we would only expect to be able to gather observations of the world (sensed data) and use those observations to model the future state of the world (predicted data). Hence, while the graphs in Table 2 show the actual data traces, we cannot assume that sensor nodes have any knowledge of these actual values. In the following experiments, we can use the actual data trace to validate the

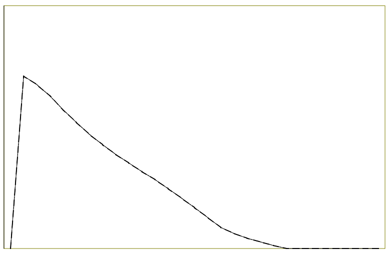
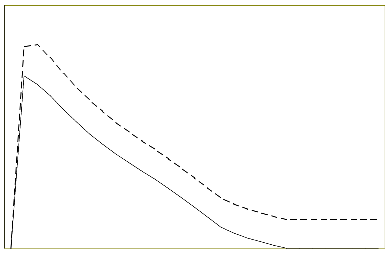
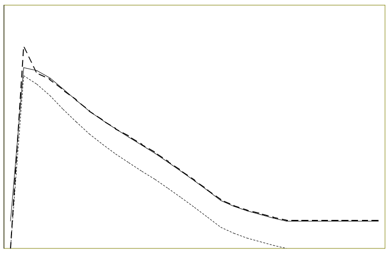
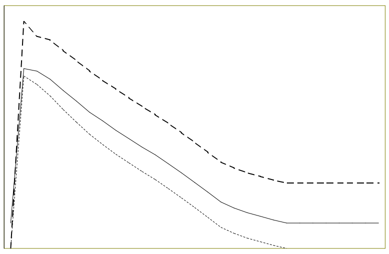
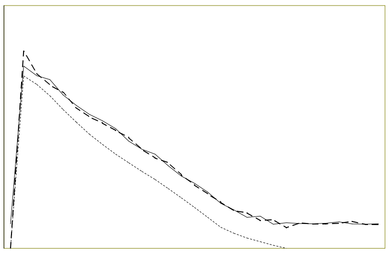
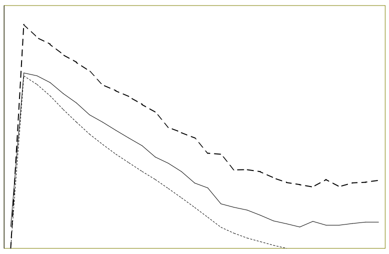
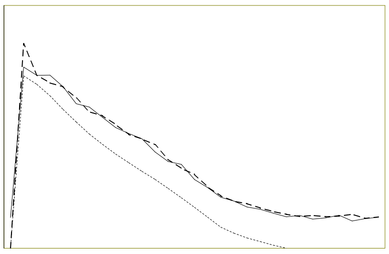
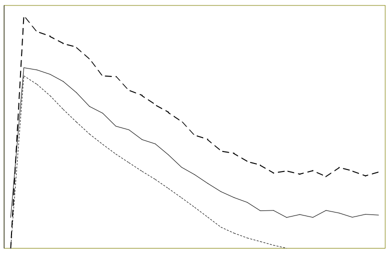
	no prediction error	prediction bias
no sensor error		
sensor bias		
sensor noise		
sensor bias & noise		

Table 2: Aggregate plots of sensed (solid black line), predicted (thick dashed line), and actual (dotted line) values of total surface water (y axis) over time (x axis)

performance of the simulated geosensor network, but the network itself does not have access to this information.

6.2 Experiment #2: Correcting for sensor bias

One of the key goals of combining environmental models and geosensor networks was to use real-time feedback to improve the robustness of the system, recalibrating sensors or validating models. As discussed above, sensor nodes are only able to use discrepancies between sensed and predicted data to detect and correct for errors.

In experiment #2, the sensor node behaviors were modified to detect and correct for one specific type of error: sensor bias. To do this, each sensor node compared the distribution of discrepancies between the sensed and predicted values of its neighbors with its own discrepancy between its sensed and predicted values. Nodes that exhibited a significant difference in discrepancy values from their neighbors might reasonably conclude that this discrepancy is due to sensor bias.

Figures 10–12 show the signature, states, and transitions for a simple modification of the DINEM sensor node automaton (presented in section 3) that implements the changed behavior. The signature of the augmented node (which we refer to as a *sensor-bias actuator node*) is slightly modified with the additional communication of a d value (discrepancy) between neighboring nodes (Figure 10). The states of the augmented node are slightly extended (Figure 11) to additionally store the discrepancy for a node (d_i) and its immediate neighbors (d_j). Finally, the transitions are also slightly modified (Figure 12) to store the discrepancy between its sensed and predicted values in the *check* action; to communicate that discrepancy to its neighbors via the *comm* action pair; and critically to recalibrate based on the heuristic of the difference between a node's own discrepancy between its sensed and predicted values and the average of its neighbors' discrepancies.

Signature v_i:

Input:	Internal:
$comm_{j,i}(x, s, d)$	$sense_i(s)$
Output:	$predict_i$
$comm_{i,j}(x, s, d)$	$check_i$
$alert_i(l)$	

Figure 10: Signature of sensor bias actuator node v_i (cf. Figure 1)

States v_i:
$d_i : \mathbb{R}$, $d_i \leftarrow 0$
$d_j : \mathbb{R}$, $d_j \leftarrow 0$
...

Figure 11: States of sensor bias actuator node v_i (cf. Figure 2)

In practice, this simple automaton failed to work adequately because neighboring nodes simultaneously recalibrating their sensors lead to an unstable system. This type of problem illustrates that while the IOA analysis is an important design tool, empirical simulation is still important to identify some emergent global system properties that are difficult to predict even using IOA analysis. In order to decrease the chance of two neighboring nodes recalibrating simultaneously, nodes were reprogrammed to only recalibrate sensors for a randomly chosen 10% of the time that they detect a discrepancy, which does stablize the system. Figure 13 compares the performance of simulations of the basic (solid line) and sensor bias actuator nodes (solid line with squares) nodes, averaged over 10 simulation runs, when presented with an error-scenario that included no model error or sensor noise, but did include sensor bias. The actual values (unavaliable to the sensors) are shown as a dashed line.

```
**Transitions** $v_i$ (where $j \in nbr(v_i)$):
    $comm_{i,j}(x, s, d)$
        Preconditions:
            $can\_output_{i,j} = true$,
            $x = x_j$, $s = s_i$, $d = d_i$
        Effects:
            $can\_output_{i,j} \leftarrow false$

    $sense_i(x)$
        Preconditions:
            $x \in \mathbb{R}$
        Effects:
            $s_i \leftarrow x + \sum_{j \in nbr(v_i)} d_j / |nbr(v_i)|$

    $comm_{j,i}(x, s, d)$
        Effects:
            $has\_input_{j,i} \leftarrow true$,
            $p_j \leftarrow x$, $s_j \leftarrow s$, $d_j \leftarrow d$

    $check_i$
        Preconditions:
            $has\_input_{j,i} = true$ for all $j \in nbr(v_i)$
        Effects:
            $has\_input_{j,i} \leftarrow false$ for all $j \in nbr(v_i)$,
            if $s_i \sim \kappa(\{x_k | k \in nbr(v_i)\})$
                then $is\_alerted_i \leftarrow true$
            else $is\_alerted_i \leftarrow false$
            $d_i \leftarrow\sim \kappa(\{x_k | k \in nbr(v_i)\} - s_i$
    ...
```

Figure 12: Transitions for sensor bias actuator node v_i (cf. Figure 3)

6.2.1 Discussion

Although the bias correction procedure required the addition of an empirical heuristic, it is relatively successful at reducing total error in the global system sensed data. On average, the sensor bias actuator nodes were able to decrease the discrepancy between sensed and actual values by 30%. A *t*-test for the means of the global overall error for the basic and augmented nodes indicates this improvement is significant at the 99% confidence level.

However, focusing purely on the global system error does tend to obscure the detailed system behavior. Figure 14 shows the individual node traces for nine neighboring nodes in the simulation (nodes 157, 158, 159, 182, 183, 184, 207, 208, 209). The sensed values for basic (solid line) and sensor bias actuator (solid line with crosses) nodes are superimposed over two otherwise identical simulations (i.e., two simulations where the nodes were switched from basic to sensor bias actuator mode, but possess the same randomly chosen sensor biases for both simulations). The arrows on the figure give the model neighborhood as defined by the D8 model (based on relative elevation, showing where water flows downhill). The movement of water through the system can be quite clearly seen (e.g., the flow of water from the cells occupied by nodes 159, 182, 183, 184 into 158). The figure also shows that individual nodes only make relatively small corrections to their sensed data (visible where the two traces on one graph diverge, such as one node 157 or 159). On some nodes the traces are identical (i.e., no corrections have been made, as on node 158). Further work is needed to investigate the detailed individual behavior of basic and sensor bias actuator nodes as a basis for improving our initial recalibration heuristic.

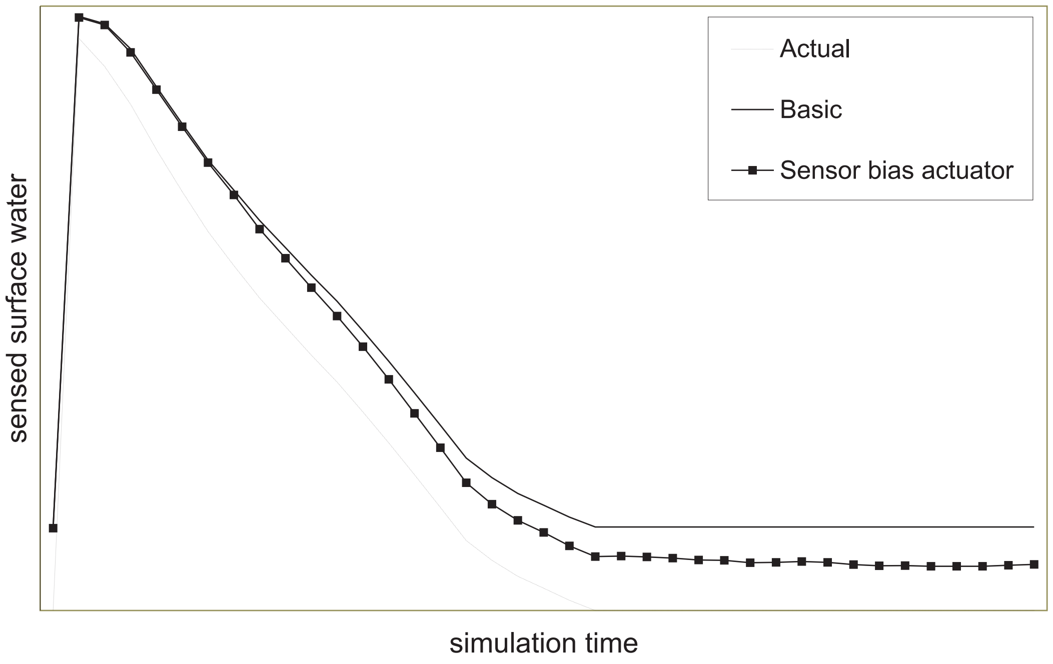


Figure 13: Global performance of basic and sensor bias actuator nodes over 10 simulations in the presence of sensor bias

6.3 Experiment #3: Sensor and model bias

Experiment #3 explored the potential for nodes to distinguish between different sources of error, specifically sensor and model bias. The general principle applied was that where all the neighbors of a node exhibit a similar discrepancy between sensed and predicted, this is a basis for attributing that discrepancy to *model* bias. Conversely, where neighboring nodes exhibit different discrepancies, that is a basis for attributing the discrepancy to *sensor* bias.

With this principle in mind, a further modification was made to the sensor bias actuator node design, leading to a new *sensor/model bias actuator* node. For brevity, the details of the IOA model for this sensor/model bias actuator node are omitted here, but follow a similar principle to the sensor bias actuator node described above: when a model bias is detected, the model is adjusted accordingly. Figure 15 again shows the global performance of these nodes alongside the basic and the sensor bias actuator nodes for a set of 10 simulations that included both model bias and sensor bias, but no sensor noise.

6.3.1 Discussion

The augmented sensor/model bias actuator nodes again out-perform the basic nodes. The total discrepancy between the actual and sensed values is again reduced by about 30% (significant at the 99% confidence level). This shows that the nodes are able to distinguish between and correct for sensor and model bias, although given the level of reduction in error, indicates they do not make further improvements upon those made by the sensor bias actuator nodes.

It is important to note that the sensor bias actuator nodes developed for the previous experiment (shown as a solid line with diamonds in Figure 15) perform extremely badly in the presence of model bias. Without the additional capability to distinguish between model and sensor bias, the more simple sensor bias actuator nodes mis-attribute model bias to a bias in their sensors and recalibrate accordingly. This in turn leads to increasing incorrect predictions, causing a feedback loop that decreases

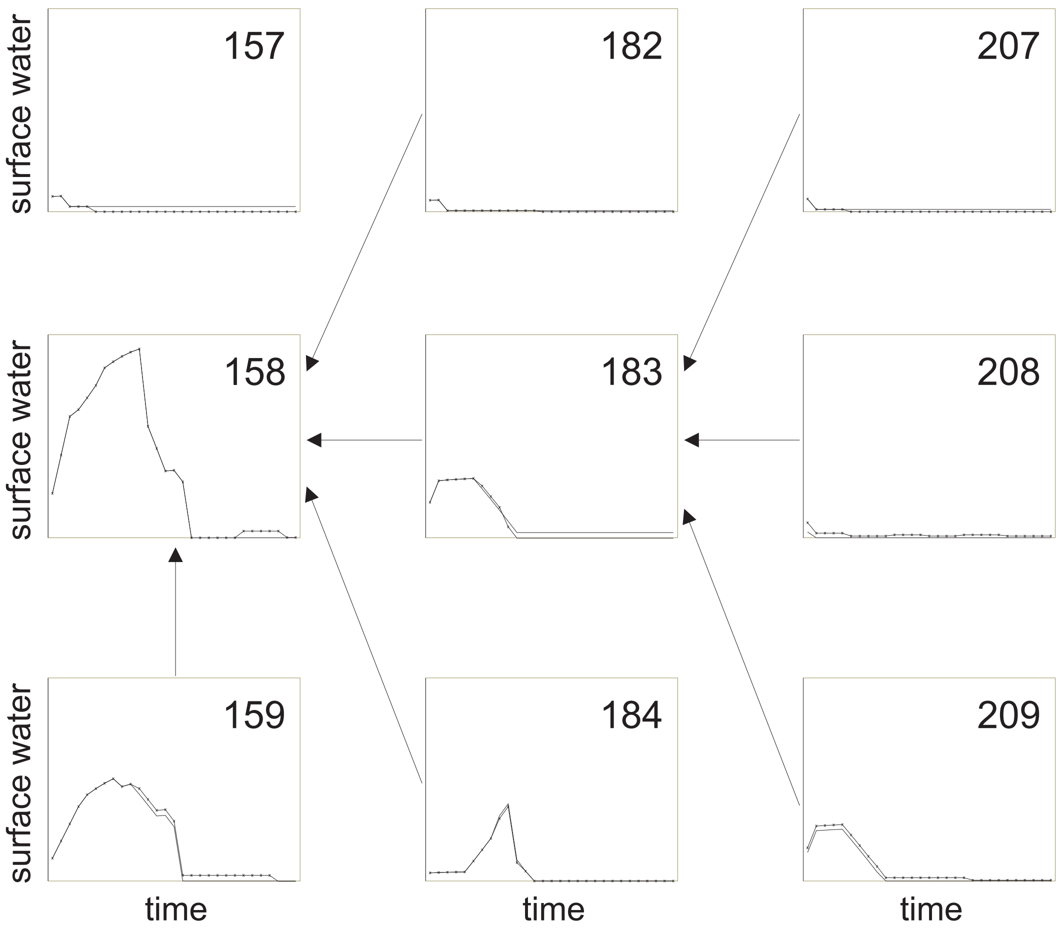


Figure 14: Individual sensed values for 9 neighboring nodes nodes across basic (solid line) and sensor bias actuator (solid line with cross) simulations

overall sensor performance by more than 200%. This example illustrates the importance in sensor/actuator DINEM systems of correctly identifying the source of error before taking actions based on observed errors.

While the IOA model helped in developing the foundations of the sensor/model bias actuator nodes, it was again found that unpredicted emergent global properties necessitated further ad hoc changes to achieve a stable simulation. In this case, stable and effective systems only resulted if 1) all changes to the model bias were made globally; and 2) sensor and model bias were not corrected simultaneously. Consequently, the augmented sensor/model bias actuator nodes first went through a period of model bias correction, and then switched to the sensor bias correction. The exact point at which a sensor switched was set as about 1/4 way through the simulation (a valued determined empirically to give nodes time to initially identify mode bias, and then counteract sensor bias for the remaining simulation). However, further work is required to identify analytically the best balance between model and sensor bias correction. Since model biases must be corrected globally (for all the nodes in the simulation), model bias correction is potentially very expensive in terms of system resources (requiring something akin to "flooding" in wireless sensor networks), and therefore also requires further investigation to minimize the potential drain on network resources.

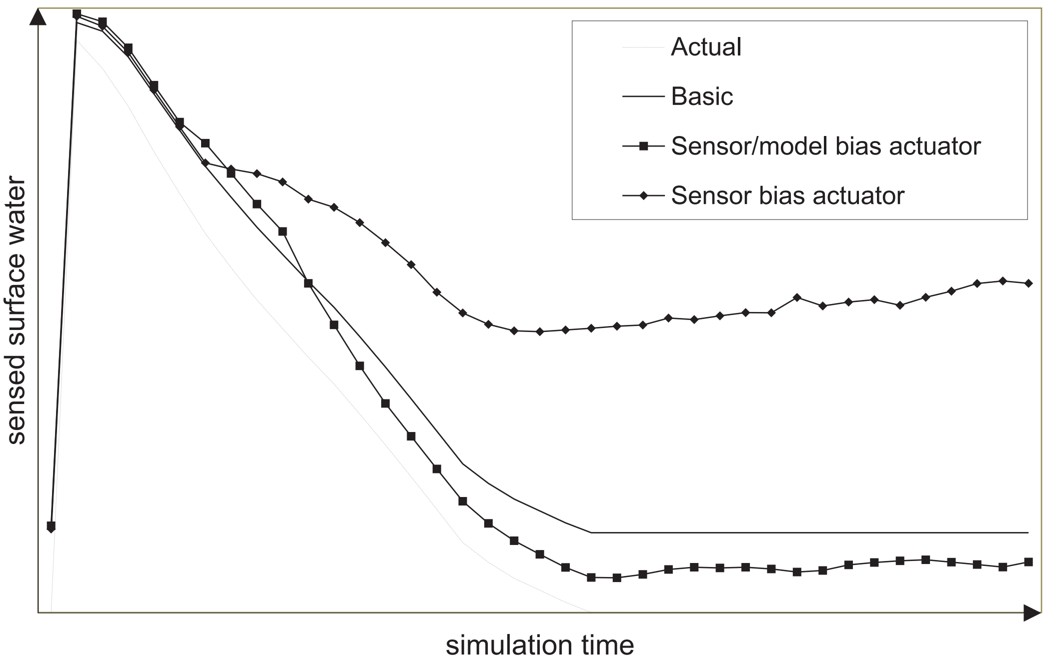


Figure 15: Global performance of basic, sensor bias actuator, and sensor/model bias actuator nodes over 10 simulations, in the presence of sensor and model bias

6.4 Summary

This section has presented three experiments using the simulated DINEM system which show how certain types of errors can be detected, corrected, and discriminated apart. In the simulations, simple heuristic bias correction routines in the DINEM nodes reduced error across the network by about 30% in certain cases. In addition, the experiments indicate the utility of the IOA model, and show how analytical IOA models can be combined with empirical investigation to improve DINEM performance and robustness. However, noise correction has not been investigated in this paper, although it is expected that existing filtering techniques (e.g., Kalman filters) would be likely to provide good conventional solutions to sensor noise.

The procedure for distinguishing between sensor and model bias is founded on the assumption of random sensor bias (i.e., correlated bias in a node's neighborhood is attributed to the model, uncorrelated bias in a node's neighborhood is attributed to the sensor). The reason for assuming uncorrelated sensor bias is that nodes are assumed to be manufactured with poor calibration, but randomly distributed through the study area. However, sensor biases that are associated with environmental conditions (say, operating temperature) might be expected to be spatially autocorrelated, demanding more sophisticated techniques to distinguish these error types.

7 Conclusions

This paper has shown how the incorporation of predictive environmental models into a geosensor network can be used to improve the robustness of the network to certain types of error. The method we have presented provides the opportunity for not only *detecting* problems that may be a result of sensor bias, sensor noise, or model bias, but also *acting* on any errors detected by recalibrating sensors or reparameterizing the model. Such so-called sensor/actuator networks are beginning to be recognized as an important next stage in the development of geosensor network theory and technology.

Although the DINEM system is empirically investigated using a simplified simulated watershed runoff model, a key contribution of this work is to show how analytical modeling of geosensor networks can support the system design process. Understanding the behavior of highly distributed systems like geosensor networks can be a highly complex task. Analytical tools can help identify solutions to problems more rapidly, as well as providing a common language for system analysts and designers to communicate ideas. At the same time, empirical simulation remains an important tool for informing analytical processes.

A range of further work is suggested by this initial research. For example, the hydrological models used in this work are greatly simplified, and the integration of realistic environmental models would be required before this work could find practical application to environmental monitoring. Furthermore, a variety of simplifying assumptions about the nature of geosensor networks were made in the course of this paper, including:

- Nodes are static and do not move in space;
- Communication between nodes is symmetric (bidirectional); and
- Nodes are aware of their precise coordinate location.

These assumptions often do not hold in practice for a range of technical reasons. Consequently, further work is required to relax these assumptions before the approach could be extended beyond simulations and implemented in actual geosensor networks.

Despite the work still to be done, geosensor networks are an important emerging technology for environmental monitoring. By combining simulation and geosensor networks this paper takes a step toward the development of what can be termed “smart environments” or “ambient spatial intelligence”: the capability to identify and manage dynamic geographical processes with sensor-enabled computing devices embedded in the natural environment.

References

Akyildiz, I. F., Su, W., Sankarasubramaniam, Y., and Cayirci, E. (2002). A survey on sensor networks. *IEEE Communications Magazine*, 40(8):102–114.

Beckwith, R., Teibel, D., and Bowen, P. (2004). Unwired wine: sensor networks in vineyards. In *Proc. IEEE Sensors*, volume 2, pages 561–564.

Beven, K. J. (2002). Towards an alternative blueprint for a physically based digitally simulated hydrologic response modelling system. *Hydrological Processes*, 16:189–206.

Braginsky, D. and Estrin, D. (2002). Rumor routing algorthim for sensor networks. In *Proc. 1st ACM International Workshop on Wireless Sensor Networks and Applications*, pages 22–31.

Endreny, T. A. and Wood, E. F. (2003). Maximizing spatial congruence of observed and dem-delineated overland flow networks. *International Journal of Geographic Information Science*, 17(7):699–713.

Favis-Mortlock, D. T., Boardman, J., Parsons, A. J., and Lascelles, B. (2000). Emergence and erosion: a model for rill initiation and development. *Hydrological Processes*, 14:2173–2205.

Fonstad, M. A. (2006). Cellular automata as analysis and synthesis engines at the geomorphology-ecology interface. *Geomorphology*, 77:217–234.

Hart, J. K. and Martinez, K. (2006). Environmental sensor networks: A revolution in the earth system science? *Earth-Science Reviews*, 78:177–191.

Karp, B. and Kung, H. T. (2000). GPSR: Greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual International Conference on Mobile Computing and Networking*, pages 243–254, Boston, MA. ACM Press.

Kininmonth, S., Atkinson, I., Bainbridge, S., Woods, G., Gigan, G., and Freitas, D. (2005). The great barrier reef sensor network. In *Proc. Pacem in Maribus XXXI*, pages 361–369.

Lubick, N. (2005). Earth observing: Something to watch over us. *Nature*, 436:168–169.

Lynch, N. (1996). *Distributed Algorithms.* Morgan Kaufmann, San Mateo, CA.

Lynch, N. and Tuttle, M. (1989). An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246.

Marks, D. (2001). Introduction to special section: Reynolds Creek experimental watershed. *Water Resources Research*, 37(11):2817.

Muetzelfeldt, R. and Massheder, J. (2003). The simile visual modelling environment. *European Journal of Agronomy*, 18:345–358.

Perkins, C. E., Royer, E. M., Das, S. R., and Marina, M. K. (2001). Performance comparison of two on-demand routing protocols for ad hoc networks. *IEEE Personal Communications Magazine*, 8(1):16–28.

Pullar, D. (2003). Simulation modelling applied to runoff modelling using mapscript. *Transactions in GIS*, 7(2):267–283.

Szewczyk, R., Osterweil, E., Polastre, J., Hamilton, M., Mainwaring, A., and D., E. (2004a). Habitat monitoring with sensor networks. *Communications of the ACM*, 47(6):34–40.

Szewczyk, R., Polastre, J., Mainwaring, A., Anderson, J., and David Culler, D. (2004b). An analysis of a large-scale habitat monitoring application. In *Proc. 2nd ACM Conference on Embedded Networked Sensor Systems*, pages 214–226.

Torrens, P. M. (2006). Simulating sprawl. *Annals of the Association of American Geographers*, 96(2):248–275.

Vaandrager, F. (1991). On the relationship between process algebra and input/output automata. In *Proc. Sixth Annual IEEE Symposium on Logic in Computer Science (LICS '91)*, pages 387–398.

Wainright, J. and Mulligan, M., editors (2004). *Environmental Modelling: Finding simplicity in complexity.* John Wiley & Sons, Chichester, UK.

Worboys, M. (2005). Event-oriented approaches to geographic phenomena. *International Journal of Geographical Information Science*, 19(1):1–28.

Worboys, M. F. and Duckham, M. (2006). Formalizing mobility in dynamic location-aware sensor networks. In *MDM '06 MLASN (Mobile Data Management 2006 Workshop on Mobile Location-Aware Sensor Networks)*, page 157. IEEE.