Decentralized spatial algorithm design

Matt Duckham

***Abstract*—Spatial computers present challenges to conventional distributed algorithm design. Substantive progress is being made in developing new algorithm design tools and techniques, for example in the development of the Proto language. This paper summarizes an alternative but complementary technique targeted at the specification of decentralized algorithms for spatial computing. The approach focuses on abstract, implementation-independent specification of designs, as opposed to more practical programming constructs. The aim is to speed the development and ease the communication of algorithms designs. This is achieved augmenting an established distributed algorithm design technique with the minimal additional constructs required to compute with diverse spatiotemporal objects and relationships. The paper illustrates the additional spatial and temporal structures using the running example of decentralized algorithms for spatial region boundary detection.**

I. Introduction

Spatial computing can be characterized as a special case of distributed computing, where additional *geographic* constraints to the generation and communication of information exist. The challenge set in [1] is "to conceive of how to re-formulate [distributed systems] applications for a continuous geometric world."

This paper describes an approach to designing *decentralized spatial algorithms*. A decentralized system is a distributed system where no single system element possesses global knowledge of the system state [2]. Consequently, decentralized spatial algorithms are well-suited to spatial computing environments, which present geographic constraints to both the generation and movement of information. Our approach is complementary to, but distinct from related approaches in spatial computing, in particular [1], [3], [4]. In comparison with [1], [3], [4], our approach focuses more strongly on the algorithm design and specification. We augment an established distributed algorithm design procedure with the spatiotemporal structures required for decentralized computing with many different types of spatiotemporal objects, references, and relations. As a consequence, however, our approach does not focus so strongly on practical programming architectures and implementation—something that is an important focus and contribution of [1], [3], [4].

Following a brief review of related work, the established distributed algorithm design technique upon which our approach is founded is introduced (Section III, after [5]). We then identify, with examples, the fundamental spatial and temporal structures required for decentralized spatial algorithm design (Sections IV and V). Finally, before concluding (Section VII) the paper looks briefly at the role of agent-based simulation in algorithm design (Section VI).

M. Duckham is with the Department of Infrastructure Engineering, University of Melbourne, VIC 3010, Australia. E-mail: matt@duckham.org

II. Related work

As already highlighted, this work shares similar aims to research on the definition of languages for spatial computing, including the development of the Proto language [1], [3], [4], [6] as well as more broadly (see [7] for a survey). The design process summarized in this paper is, we believe, complementary to these efforts. Our approach favors assisting designers with the construction and communication of algorithms; but places less emphasis on practical implementation of these algorithms within spatial computers.

In attempting to construct an implementation-independent decentralized spatial algorithm design framework, it is essential to draw on established design tools and techniques. Hoare's CSP (communicating sequential processes [8]) and Robin Milner's CCS (calculus of communicating systems, [9]) are two examples of influential formal models that deal explicitly with the interactions between processes, and so are highly relevant. More recently, Milner's CCS has been extended with additional structure in the pi-calculus and bigraphs [10], [11].

These formalisms are being applied to fundamental problems in geographic information science (e.g., [12]), but are relatively complex to apply to higher-level domains, like algorithm design. Similarly, related models like IOA (input-output automata, [2]), have been applied to decentralized spatial algorithms (e.g., [13], [14]), but have a strong focus on proving formal properties, like fairness and liveness, rather than ease of construction or communication of designs.

So, while alternative models have the advantage of more formal rigor, their substantial additional complexity makes them less well-adapted to supporting the algorithm design process. Hence, in this paper we argue that the less formal but more intuitive technique of Nicola Santoro [5] provides a practical compromise between complexity and rigor.

III. Santoro's distributed algorithm design

The distributed algorithm design approach of [5] is founded on four key structures:

1) *Restrictions* on the environment in which the algorithm is designed to operate, such as restrictions on the network structure and connectivity, communication reliability and synchronization, and so forth.
2) System *events* that occur to nodes, specifically *receipt* of a message; *triggered* events (such as a scheduled alarm or periodic sensor reading); and a *spontaneous* impulse, external to the system.
3) *Actions* that a node can perform in response to the different events that occur—actions must be atomic sequences of operations that cannot be interrupted by other events.

4) *States* for nodes, which allow nodes to retain knowledge of previous interactions, and respond in different ways to the same event depending on the context.

For example, Algorithm 1 provides a simple decentralized algorithm in the style of [5] for identifying nodes at the boundary of a spatial region. With reference to the four key structures identified above:

Algorithm 1: Determining the (inner) region boundary

```
Restrictions: Reliable, fully asynchronous communication;
  undirected communication graph, $G = (V, E)$; sensor
  function $s : V \to \{0, 1\}$
State Trans. Sys.:
  $(\{\text{INIT}, \text{IDLE}, \text{BNDY}\}, \{(\text{INIT}, \text{IDLE}), (\text{IDLE}, \text{BNDY})\})$
Initialization: All nodes in state INIT
INIT
  Spontaneously
    broadcast (ping, $\mathring{s}$) {Broadcast sensed value}
    become IDLE
  Receiving (ping, $s'$)
    defer until IDLE
IDLE
  Receiving (ping, $s'$)
    if $s' \neq \mathring{s}$ and $\mathring{s} = 1$ then
      become BNDY
```

- Restrictions: The algorithm makes no restrictions on synchronization between nodes (communication may be fully *asynchronous*), but does require *reliable* communication (messages sent will be received within some finite amount of time). Communication is assumed to be mediated through a bidirected communication graph G, but again no further restrictions on the communication graph structure are required. Finally, the algorithm does require a Boolean sensor on each node (capable of sensing either 1 or 0, e.g., in or out of a region of interest, "hot" or "cold," presence or absence of pollutant), represented as a sensor function $s : V \to \{0, 1\}$.
- States: The state transition system specifies at the beginning of the algorithm the defined states (INIT, IDLE, and BNDY) and allowable transitions (INIT to IDLE to BNDY). The initial states for all nodes are also specified in the algorithm header. The algorithm proper then defines for each state a (possibly empty) set of events and associated actions.
- Events and actions: Three events are defined in the algorithm. Nodes in the INIT state can spontaneously perform an action to broadcast a `ping` message, before transitioning to an IDLE state. Nodes in the IDLE state respond to `ping` messages received by checking if adjacent nodes sense a different value. If so, IDLE nodes transition into a BNDY state. Nodes in the INIT state receiving a `ping` message defer this event until the node is in the IDLE state (i.e., received message is placed on a stack and treated as received only after the node has transitioned into a IDLE state). Other possible events (e.g., receiving a `ping` message in the BNDY state) are by default associated with the empty action ("do nothing").

The intuition behind Algorithm 1 is that even without geometric information, based purely on communication neighborhoods, nodes can locally determine whether they are at a region boundary by comparing their local sensed data with that of their immediate one-hop neighbors.

While Algorithm 1 is kept deliberately simple, it does illustrate several key features of the approach. Most importantly, although the algorithm header specifies the *global* restrictions and states, the algorithm body only specifies *local* rules that each individual node executes in parallel. To enforce the rigid separation between local and global knowledge, we use the overdot notation to distinguish between a globally defined function, and an individual node's local knowledge about that function. For example, in the algorithm body, we write $\mathring{s}$ (read "my" s or "local" s) in place of $s(\circ)$, where $\circ \in V$ is the local node currently under consideration. A failure to adequately distinguish between the local information available to an individual node, and the global information available across the network is a major source of design errors in decentralized algorithms (e.g., writing an action for node v that attempts to access information that is not local to v, like $s(v')$).

In summary, the algorithm specification procedure adopted in this paper offers three main features:

- an established and standard toolkit for abstract and implementation-independent specification of decentralized algorithms, supporting improved communication between designers;
- an unambiguous specification of the computational and sensed environment in which the algorithm is designed to operate; and
- a rigid separation of local and global knowledge, helping to protect against design errors arising from incorrectly referring to inaccessible global information in a local protocol.

IV. Spatial extensions

The key question underpinning all of spatial information science is "What makes spatial special?" Similarly, extending the general distributed algorithm design technique in Section III to the special case of decentralized *spatial* algorithms requires the identification and selection of those characteristics that are "special" to spatial information.

Clearly, the most important spatial structure is location. However, "location" does not necessarily imply the *coordinate* location (usually termed *position*). Location may also involve a diversity of less detailed quantitative information about, for example, the relative distances or directions (bearings) between nodes, or even *qualitative* information about a node's proximity to other nodes or known locations.

Figure 1 summarizes six of the most common forms of location information. The six examples include: a. coordinate position with reference to some external coordinate framework, such as derived from GPS or virtual coordinate systems; b. relative *anchor* location, like proximity to some external

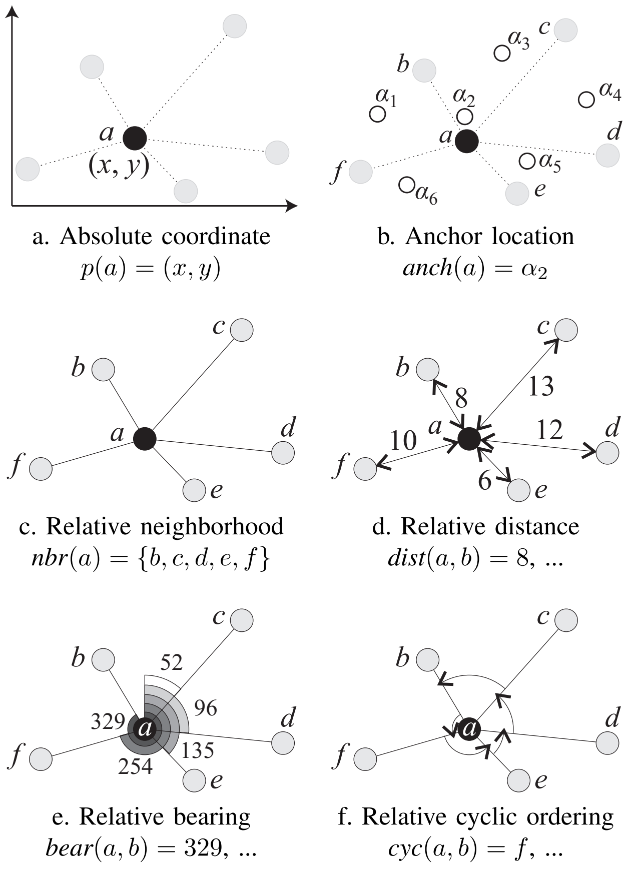


Fig. 1. Summary of six common types of location information available to a node.

"anchors" at known locations, such as might be derived from proximity-based RFID localization; c. relative neighborhood, such as knowledge of one-hop neighbors in a physical or overlay communication network; d. relative distance to neighbors, such as derived from range-finding technologies; e. relative bearing; and f. cyclic ordering, such as may be derived from direction-finding technologies.

The diversity of ways in which geographic location can be represented and related (e.g., cyclic ordering can be computed from bearings; coordinate position can be used to compute any of the other types of location information) is typical of problems in the spatial domain. Further information on localization techniques and technologies may be found in a range of literature on the topic, including [15]–[17].

Algorithm 2 provides an example extension to Algorithm 1 with more sophisticated spatial capabilities. The algorithm identifies not simply boundary nodes, but also a unique cycle of nodes around the region boundary. In practice, this requires each boundary node determine its next neighbor in the cycle, stored as local (i.e., to each node) data in the *wind* variable. Being able to cycle around a region boundary is a fundamental operation for a range of higher-level algorithms, such as computing the area or centroid of a region [18], testing containment between regions, efficient leader election for regions [19], or simply updating information stored at the region boundary [20]. Organizing communication around the boundary in this way is significantly more scalable than communicating over an entire spatial region [19].

Algorithm 2: Determining the (inner) boundary nodes and cycle for a region (cf. Algorithm 1).

```
Restrictions: Reliable, fully asynchronous communication;
    undirected planar communication graph, $G = (V, E)$;
    relative neighborhood, $nbr : V \to 2^V$; $s : V \to \{0, 1\}$;
    identifier function $id : V \to \mathbb{N}$; cyclic ordering
    $cyc : E' \to id_*$, where $E' = \{(v, id(v'))|(v, v') \in E\}$
State Trans. Sys.:
    $(\{\text{INIT}, \text{IDLE}, \text{BNDY}\}, \{(\text{INIT}, \text{IDLE}), (\text{IDLE}, \text{BNDY})\})$
Initialization: All nodes in state INIT
Local data: $wind : V \to V \cup \{\varnothing\}$, initialized $\mathring{wind} := \varnothing$;
    relation $D \subset \mathbb{N} \times \{0, 1\}$

INIT
  Spontaneously
    broadcast (ping, $\mathring{id}$, $\mathring{s}$)
    become IDLE
  Receiving (ping, $i$, $d$)
    defer until IDLE

IDLE
  Receiving (ping, $i$, $d$)
    set $D := D \cup (i, d)$ {Store id and sensed value}
    if $|D| = |\mathring{nbr}|$ then {Check if all ping received}
      Create function $data : I \to \{0, 1\}$, where
      $I = \{i'|(i', d') \in D\}$ and $data : i' \mapsto d'$
      if $\mathring{s} = 1$ and $0 \in data_*$ then
        set $\mathring{wind} := \mathring{cyc}(i'')$, where $data(\mathring{cyc}(i'')) = \mathring{s}$ and
        $data(\mathring{cyc}(i'')) \neq data(i'')$
        become BNDY
```

Ensuring a unique boundary cycle exists, and can be computed, requires: a. that the (overlay) network is planar[1]; and b. that nodes have access to local spatial information about the cyclic ordering of neighbors (listed in the restrictions to Algorithm 2). Information about the cyclic ordering of neighbors may be computed from geometric information, like absolute coordinates, or deduced via other means, such as direction finding. Irrespective of the details of the localization technology, it is possible to provide an abstract representation of the cyclic ordering as a function $cyc : E' \to id_*$, where $E' = \{(v, id(v'))|(v, v') \in E\}$. The function $id : V \to \mathbb{N}$ maps to an identifier for each node (such as hardware address), while id_* is the image of the *id* function (the set of identifiers mapped to by nodes). Making a clear distinction between a node itself, $v \in V$ (which cannot be communicated to neighbors), and the identifier of that node $id(v)$ (which can) is again important to accurate designs. It is never assumed that nodes have access even to immediate neighbors' states—

[1]Although simply stated, establishing and maintaining a planar network is often difficult in practice, for example due to positioning inaccuracies and environmental and energy fluctuations that affect network connectivity. Nevertheless, stating such restrictions in the algorithm header makes explicit and accessible those assumptions underlying an algorithm, assisting in subsequent comparison of different alternatives or in the context of specific deployment scenarios.

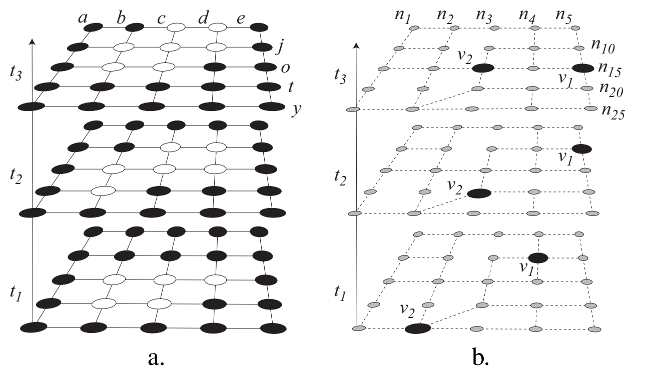


Fig. 2. Examples of spatiotemporal model of a. environmental dynamism ($s : V \times T \to \{\mathit{black}, \mathit{white}\}$) and b. node mobility ($\mathit{anch} : V \times T \to A$, where A is some set of known anchor locations, for example intersections in a transportation network).

all information that is not local to a node must be explicitly communicated to it by the algorithm before it can be used.

In summary, “spatial” involves more than coordinate position. Using a diversity of types of location information to efficiently construct higher-level spatial objects and relations, like boundaries and regions, groups and clusters, is a major challenge faced by decentralized spatial computing.

V. Temporal extensions

In a purely structural sense, time is a straightforward extension to our algorithm designs. Those functions that describe global restrictions to the algorithm can be easily augmented with an ordered set of discrete times T as part of their domain. For example, the atemporal positioning function $p : V \to \mathbb{R}^2$ can be extended to have time-varying positions as its domain, $p : V \times T \to \mathbb{R}^2$. Extensions to time-varying communication graphs can be similarly defined. In this way, both environmental dynamism and node mobility and volatility can be modeled (see Figure 2).

Algorithm 3 completes our boundary tracking example, showing an extension of the simple neighborhood-based boundary determination in Algorithm 1 to ongoing tracking of boundary status (this time through Boolean thresholding of a continuous sensor value, rather than a truly Boolean sensor). Currying allows time-varying functions to still be accessed locally. We adopt the database terminology *now* to indicate the current sensed value for a node (i.e., in Curried form $\mathring{s}(\mathit{now})$).

Despite this apparent simplicity, dealing with time does introduce additional conceptual complexity into algorithms. A basic philosophical distinction is made usually between things that *endure* through time (called *endurants* or *continuants*), and things that *happen* in time (called *perdurants* or *occurrents*) [12]. Boundaries and regions are typical examples of geospatial endurants; the appearance, splitting, merging, and disappearance of regions are all examples of geospatial perdurants.

The distinction between endurants and perdurants maps directly to two fundamentally different types of information that may be generated by a decentralized spatiotemporal algorithm: *histories* and *chronicles* [21]. A history provides

Algorithm 3: Tracking the (inner) boundary of a region, with state maintenance.

```
Restrictions: Reliable, fully asynchronous communication;
    undirected planar communication graph $G = (V, E)$;
    $s : V \times T \to \mathbb{R}$; $id : V \to \mathbb{N}$; region threshold $r$
State Trans. Sys.:
    $(\{\text{INIT}, \text{IDLE}, \text{BNDY}\}, \{(\text{INIT}, \text{IDLE}), (\text{IDLE}, \text{BNDY}),$
    $(\text{BNDY}, \text{IDLE}), (\text{IDLE}, \text{INIT}), (\text{BNDY}, \text{INIT})\})$
Initialization: All nodes in state INIT
Local data: sensor reading at time of state change $s_l$;
    neighbor data $d : \mathring{nbr} \to \{-1, 0, 1\}$, initialized to
    $d(v) := -1$

INIT
  Spontaneously
    set $s_l := \mathring{s}(now)$ {Store last sensed value}
    broadcast (ping, $\mathring{s}(now)$, $\mathring{id}$)
    become IDLE

IDLE, BNDY
  Spontaneously
    if $\mathring{s}(now) = 1$ and $0 \in d_*$ then
      become BNDY
    else
      become IDLE
  Receiving (ping, $s'$, $i$)
    set $d(i) := s'${Store neighbor's sensed values}
  When $\mathring{s}(now) < r \leq s_l$ or $s_l < r \leq \mathring{s}(now)$
    become INIT
```

a spatiotemporal record of the states of monitored endurants (e.g., point locations, regions, boundaries, moving objects) through time. A chronicle provides a record of the occurrences (perdurants) that happened through time.

For example, imagine a spatial computer, like a sensor network, tasked with monitoring the spread of an oil spill (see Figure 3). We might wish the system to generate an alert when parts of the oil spill *appear* or *break up* (a chronicle). Alternatively, we might (also) wish the system to report on the *connectivity* of the oil spill every ten minutes over the course of a day (a history). We can expect to need to design decentralized spatiotemporal algorithms to monitor histories in some cases, and to monitor chronicles in other cases.

Thus, just as “spatial” involves more than simply coordinate location, so “temporal” involves more than simply timestamps; it means identifying and tracking salient spatial *events*.

VI. Algorithm simulation

Although our design approach aims to be implementation independent, we have developed a simulation system for implementing and evaluating decentralized spatial algorithm designs (see Figure 4), based on a popular agent-based simulation system called NetLogo [22]. A small number of additional keywords have been implemented as a NetLogo library, which make it possible to rapidly and directly translate pen-and-paper algorithm specifications into simulation models. A key advantage of using NetLogo is in its ability to simulate both decentralized spatial computing environments and dynamic

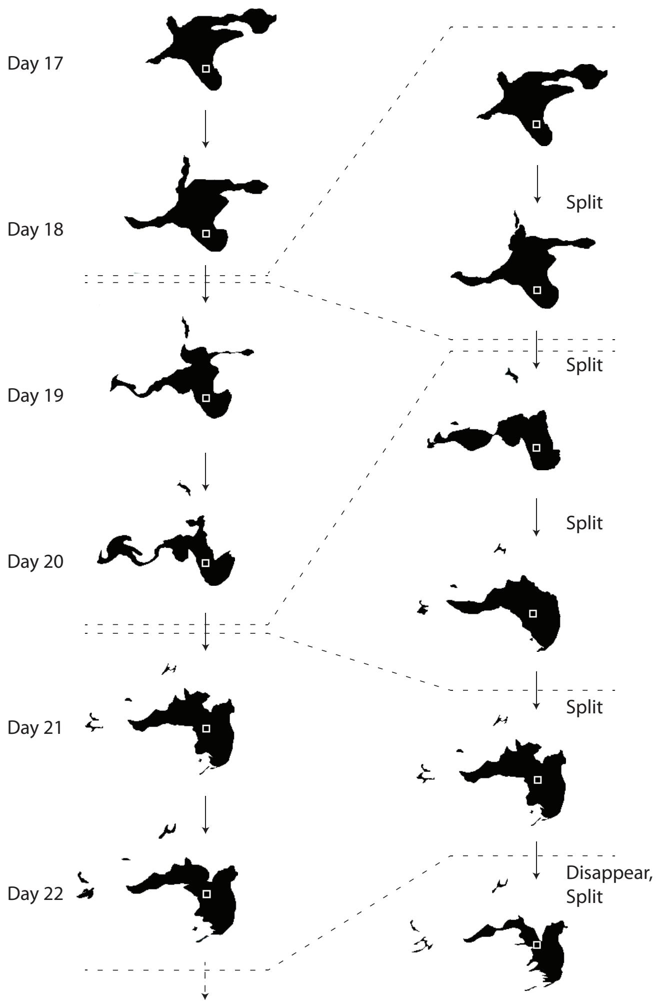
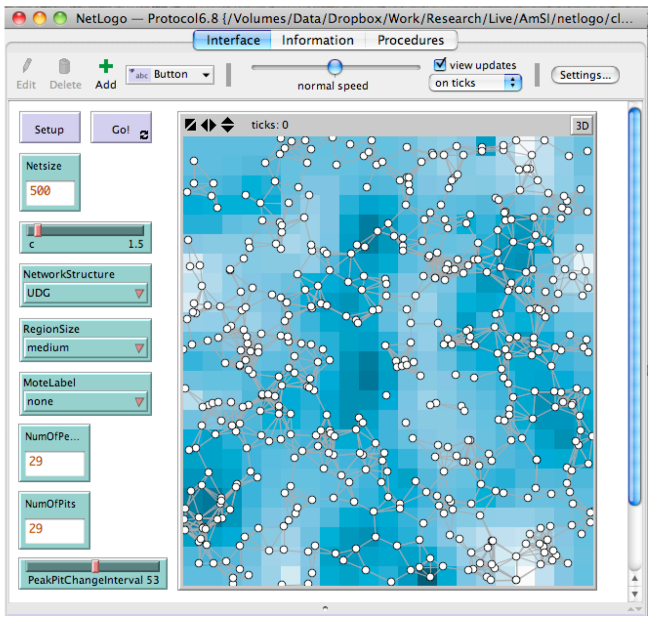


Fig. 3. Histories and chronicles: Two views of the changes in the extent of Deepwater Horizon Disaster oil slick, Gulf of Mexico, from Day 17, 7 May 2010. (Source: Times-Picayune).



Netsize 500
c 1.5
NetworkStructure UDG
RegionSize medium
MoteLabel none
NumOfPe... 29
NumOfPits 29
PeakPitChangeInterval 53

Fig. 4. Example NetLogo interface for decentralized spatial algorithm simulation

geographic environments, supported by NetLogo's large and diverse community of domain scientist users (for example in ecology, biology, geography, and social sciences).

The ability to simulate algorithms can greatly assist the designer, by generating rapid feedback on algorithm behavior and as a basis for adversarial analysis to identify design flaws. As well as providing for empirical evaluation of the global behavior of decentralized spatial algorithms, such as scalability, simulations also can also help to explore experimentally the robustness of decentralized spatial algorithms. Spatial information is inherently uncertain, subject to inaccuracy (lack of correctness), imprecision (lack of detail), and vagueness (existence of borderline cases). In the case of spatial computing technologies, like sensor networks, inaccuracy and imprecision are especially important. Low cost, poorly calibrated sensors typically have relatively low accuracy; sensor observations are inevitably discrete in both space and time, the source of imprecision. Further, many of the application-level geographic objects and events of interest, like "hot spots" or "traffic jams," are vague (e.g., some location may be definitely in a hot spot, others definitely not, but typically there will be borderline locations, for which it is indeterminate whether or not they are in or out of the hot spot). Decentralized spatial algorithms frequently need to demonstrate robustness to imprecision and inaccuracy in sensed information, and an ability to generate useful knowledge about vague geographic phenomena. Spatial computing under uncertainty is a key focus for current research.

VII. Summary

This paper has demonstrated how established distributed algorithm design techniques can be adapted to decentralized spatial algorithm design. The approach identifies a small number of spatial and temporal structures from which more sophisticated spatial computing algorithms can be constructed. Our approach complements and contrasts with existing research in [1], [3], [4], [6], and aims to help human designers in specifying local protocols that will exhibit the desired global behaviors. By contrast, [1], [3], [4], [6] target the (automated) transformation of global constructs into local protocols. Further, our approach emphasizes abstract, implementation-independent algorithm specifications, but does not explicitly address practical implementation and programming languages.

References

[1] J. Beal and R. Schantz, "A spatial computing approach to distributed algorithms," in *45th Asilomar Conference on Signals, Systems, and Computers*, 2010.

[2] N. Lynch, *Distributed Algorithms*. San Mateo, CA: Morgan Kaufmann, 1996.

[3] J. Bachrach, J. Beal, and J. McLurkin, "Composable continuous-space programs for robotic swarms," *Neural computing & applications*, vol. 19, no. 6, pp. 825–847, 2010.

[4] J. Bacharach and J. Beal, "Building spatial computers," Tech. Rep. 2007-017, MIT CSAIL, March 2007.

[5] N. Santoro, *Design and Analysis of Distributed Algorithms*. New Jersey: Wiley, 2007.

[6] J. Beal and J. Bachrach, "Infrastructure for engineered emergence in sensor/actuator networks," *IEEE Intelligent Systems*, pp. 10—19, 2006.

[7] J. Beal, S. Dulman, K. Usbeck, M. Viroli, and N. Correll, "Organizing the aggregate: Languages for spatial computing," in *Formal and Practical Aspects of Domain-Specific Languages: Recent Developments* (M. Mernik, ed.), IGI Global, 2012, to appear. http://arxiv.org/abs/1202.5509.

[8] C. A. R. Hoare, "Communicating sequential processes," *Communications of the ACM*, vol. 21, no. 8, pp. 666—677, 1978.

[9] R. Milner, *A Calculus of Communicating Systems*, vol. 92 of *Lecture Notes in Computer Science*. Springer Verlag, 1980.

[10] R. Milner, *Communicating and Mobile Systems: The π-calculus*. Cambridge University Press, 1999.

[11] R. Milner, "Pure bigraphs: Structure and dynamics," *Information and Computation*, vol. 204, pp. 60–122, 2006.

[12] M. Worboys, "Event-oriented aproaches to geographic phenomena," *International Journal of Geographic Information Science*, vol. 19, no. 1, pp. 1–28, 2005.

[13] M. Duckham and F. Reitsma, "Decentralized environmental simulation and feedback in robust geosensor networks," *Computers, Environment, and Urban Systems*, vol. 33, pp. 256–268, 2009.

[14] S. Dolev and N. Tzachar, "Empire of colonies: Self-stabilizing and self-organizing distributed algorithm," *Theoretical Computer Science*, vol. 410, pp. 514–532, 2009.

[15] J. Hightower and G. Boriello, "Location systems for ubiquitous computing," *IEEE Computer*, vol. 34, no. 8, pp. 57–66, 2001.

[16] M. Worboys and M. Duckham, *GIS: A Computing Perspective*. Boca Raton, FL: CRC Press, 2nd ed., 2004.

[17] H. Karl and A. Willig, *Protocols and Architectures for Wireless Sensor Networks*. Chichester, England: Wiley, 2005.

[18] M. Sadeq and M. Duckham, "Decentralized area computation for spatial regions," in *Proc. SIGSPATIAL ACMGIS*, (New York), pp. 432–435, ACM, 2009.

[19] M. Duckham, D. Nussbaum, J.-R. Sack, and N. Santoro, "Efficient, decentralized computation of the topology of spatial regions," *IEEE Transactions on Computers*, vol. 60, no. 8, pp. 1100–1113, 2011.

[20] M. J. Sadeq and M. Duckham, "Effect of neighborhood on in-network processing in sensor networks," in *Geographic Information Science* (T. Cova, K. Beard, M. Goodchild, and A. U. Frank, eds.), no. 5266 in Lecture Notes in Computer Science, pp. 133–150, Berlin: Springer, 2008. (Conference accepted 31% of submitted papers).

[21] A. Galton, "Fields and objects in space, time, and space-time," *Spatial Cognition and Computation*, vol. 4, no. 1, pp. 39–68, 2004.

[22] U. Willensky, "Netlogo." Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., 1999. http://ccl.northwestern.edu/netlogo/.