An Application Framework for Loosely Coupled Networked Cyber-Physical Systems

Minyoung Kim, Mark-Oliver Stehr
SRI International, Menlo Park, CA 94025 USA
Email: mkim,stehr@csl.sri.com

Jinwoo Kim, Soonhoi Ha
Seoul National University, Seoul, South Korea
Email: jwkim,sha@iris.snu.ac.kr

***Abstract*—Networked Cyber-Physical Systems (NCPSs) present many challenges since they require a tight combination with the physical world as well as a balance between autonomous operation and coordination among heterogeneous nodes. These fundamental challenges range from how NCPSs are architected, implemented, composed, and programmed to how they can be validated. In this paper, we describe a new paradigm for programming an NCPS that enables users to specify their needs and nodes to contribute capabilities and resources. This new paradigm is based on the partially ordered knowledge sharing model that makes explicit the abstract structure of a computation in space and time. Based on this model, we propose an application framework that provides a uniform abstraction for a wide range of NCPS applications, especially those concerned with distributed sensing, optimization, and control. The proposed framework provides a generic service to represent, manipulate, and share knowledge across the network under minimal assumptions on connectivity. Our framework is tested on a new distributed version of an evolutionary optimization algorithm that runs on a computing cluster and is also used to solve a dynamic distributed optimization problem in a simulated NCPS that uses mobile robots as controllable data mules.**

I. INTRODUCTION

Unlike traditional embedded systems with the emphasis on the computational elements, *Networked Cyber-Physical Systems (NCPSs)* are typically designed as a network of interacting nodes that sense and affect their environments often under harsh and dynamic network conditions. Hence, an NCPS requires the coordination of computational and physical parts of a system, and a balance between autonomy and cooperation among nodes that by their nature are unreliable and can be only loosely synchronized. An NCPS can provide complex, situation-aware, and often critical services in applications such as distributed surveillance and control, crisis response, medical systems, or networked space/satellite missions. General principles and tools are needed for building robust, effective NCPSs using individual cyber-physical nodes as building blocks. In this paper, we propose an application framework for an NCPS based on a partially ordered knowledge sharing model. We show how typical applications centered around distributed sensing, optimization, and control can be implemented in this framework so that local decisions and actions improve the solution quality of the overall NCPS.

The distributed and dynamic nature of the problem presents many challenges in developing a framework for an NCPS. The system should be able to take advantage of opportunities for communication and must be robust against delays and disruptions due to, e.g., mobility, failures, or nodes entering and leaving the system. Information fusion, optimization, and control in an NCPS should take place locally at any node, but a certain degree of global awareness may be needed. A seamless transition between autonomy and cooperation is needed to ensure no dependency on the existence or connectivity of other nodes, so that the local operation can always proceed, although possibly in a less optimal fashion. Strategies for local optimization and cooperation must lead to useful anytime solutions and enable adaptation to new situations based on newly available information. To enable efficient use of the distributed computing resources, an NCPS needs to allow the composition of solutions generated by different nodes.

We address the core problem of distributed sensing, optimization, and control in the NCPS by developing an application framework based on partially ordered knowledge sharing for loosely coupled systems without persistent network connectivity. In our approach, knowledge is semantically meaningful information that can be generated, stored, processed, aggregated, and communicated to other nodes. It is important to have a framework that captures and exploits the semantics of knowledge, the heterogeneous capabilities and resources of the nodes and the network, the diverse requirements from applications and users, and the physically limited interactions among potentially unreliable nodes. Using such a framework, we can uniformly address the core challenges of the NCPS by an integrated treatment of system monitoring, balancing of resources, and adapting to ever-changing situations.

In an NCPS, various kinds of knowledge need to be expressed including sensor readings at specific locations, results of information fusion and aggregation, queries for information, or requests to actuators, and locally computed solutions. The proposed application framework operates on top of a delay- and disruption-tolerant network (DTN) model, which makes minimal assumptions on the network connectivity and avoids expensive multi-round protocols that are frequently used in distributed and fault-tolerant algorithms. We implemented a distributed dynamic optimization algorithm as a test case and performed a comprehensive analysis of performance, solution quality, and corresponding trade-offs. Furthermore, thanks to the randomized nature of our algorithms these solutions are anytime solutions that are robust under failures and resource limitations (e.g., individual node/thread failures), making our approach applicable in a realistic dynamic setting.

This paper presents the following contributions: In Section II, we describe our partially ordered knowledge sharing model. Section III illustrates our application framework that provides

a uniform interface to all capabilities of the NCPS. A prototype implementation and details of the proposed framework are presented in Section IV. In Section V, our framework is tested using a simulated NCPS scenario centered around a team of mobile robots operating as controllable data mules. We also report on an experimental deployment on PlanetLab to evaluate the performance in a distributed computing setting.

II. Knowledge Sharing as a Basis for Distributed Computing

We use a generalization of a distributed computing model based on partially ordered *knowledge sharing* that we have used in earlier work [1] as the basis for DTN. The knowledge sharing model is asynchronous and can make explicit the structure of a distributed computation in space and time, and hence is less abstract than many other models of distributed computing, e.g., those abstracting from the network topology by assuming direct end-to-end channels.

In a nutshell, we assume NCPSs containing a finite set of so called *cyber-nodes* that provide *computing* resources, can have volatile and/or persistent *storage*, and are all equipped with *networking* capabilities. Cyber-nodes can have additional devices such as *sensors* and *actuators*, through which they can observe and control their environment to a limited degree, including possibly their own physical state, e.g., their orientation/position. Cyber-nodes can be fixed or mobile, and for the general model no assumption is made about the computing or storage resources or about the network and the communication capabilities or opportunities that it provides. Hence, this model covers a broad range of *heterogeneous* technologies (e.g., wireless/wired, unicast/broadcast) and potentially challenging environment conditions, where networking characteristics can range from high-quality persistent connectivity to intermittent/episodic connectivity. The cyber-physical system is *open* in the sense that new nodes can join and leave the network at any time. Permanent or temporary communication or node *failures* are admitted by this model. As a consequence, many forms of network dynamics — including partitioning, merging, message ferrying, group mobility — are possible.

In the following, we give an informal characterization of an individual cyber-node that will be sufficient for our purposes. First, each cyber-node has a unique *name*, which in practice can be generated locally by a random choice if we ensure that uniqueness holds with very high probability. Each cyber-node has a *local clock*, which increases monotonically by at least one unit in each instruction and is loosely synchronized with other nodes in the network whenever admitted by the networking conditions. Each implementation of time synchronization must satisfy Lamport's axioms of logical time. We also assume that each node has access to a source of randomness with a uniform distribution, with the idea that typical applications of this model make heavy use of randomization techniques.

Locally, each cyber-node uses a computationally universal event-based model that is conceptually sequential (i.e., without an explicit notion of thread or process). The model is based on the dual notions of local events and distributed knowledge. Two key services are provided by each node. First, timed events can be posted, i.e., scheduled to be activated at any (possibly randomized) local time in the future. Second, knowledge can be posted, i.e., submitted for dissemination in the network. All local computation is event-based, where corresponding to the two services above, events include *timed events* and *knowledge events*, with the latter representing the reception of a new unit of knowledge. Timed events and units of knowledge have the following attributes. They are equipped with a creator (i.e., the name of the creating node), the creation time, an application-defined activation time (the earliest time when they should be handled), and an expiration time (after which they become obsolete and are discarded, even if they have not been handled). Additional local services can be provided by attached devices that also use an event-based interface. Similar to existing middleware frameworks for messaging or group communication, knowledge dissemination can take place independently in different logical *cyber-spaces*, but a unit of knowledge is a more state-like entity that should not be confused with the notion of a message. Furthermore, no reliability, delivery order, or atomicity guarantees are provided to the applications, because they would severely limit the scalability of the model in terms of the network size.

Partially ordered knowledge sharing is asynchronous and each node may use some of its storage as a cache, which we also refer to as a *knowledge base*. Implementations based on network caching allow the system to support communication even if no end-to-end path exists at a single point in time. Different from a shared-memory model, partially ordered knowledge sharing allows each node to have its own (typically partial and delayed) view of the distributed state of knowledge. Different from an asynchronous message-passing model, knowledge is not directed toward a particular destination. Instead, each node decides based on the knowledge content (or its embedded type) if it wants to use the unit of knowledge.

Epidemic and (spatial) gossiping techniques can be used to implement knowledge sharing, but unlike (synchronized) gossiping, which is often based on the exchange of cache summaries, knowledge sharing can also be implemented by single-message protocols based on unidirectional communication [1]. Besides, epidemic computing covers a very broad class of algorithms, whereas partially ordered knowledge-sharing is a more restricted model that makes specific use of the abstract semantics of knowledge that is given in a very specific way, namely in terms of partial orders. The consideration of the partial-order semantics of knowledge by all (and in particular intermediate) nodes is of key importance for scalable implementations, because it enables the implementation to discard information in a semantically meaningful way, e.g., to bound the amount of knowledge that needs to be stored at each node. The use of a partial order is the reason why knowledge-sharing is fundamentally different from asynchronous/unreliable or even epidemic/probabilistic broadcast.

Specifically, we assume an application-specific partial order $\leq$ on all knowledge items together with its induced equiva-

lence relation. We refer to $\leq$ as the *subsumption order* given that the intuitive meaning of $K \leq K'$ is that K' contains at least the information contained in K. With this interpretation the induced equivalence $K \equiv K'$, defined as $K \leq K'$ and $K \geq K'$, means that K and K' have the same semantics, even if they are represented in different ways. In this situation, that is, if $K \equiv K'$, the knowledge sharing model may (but does not have to) discard K' without delivering it to the application, if K has already been delivered. In addition to $\leq$, we assume an application-specific strict partial order $\prec$ that is compatible with $\leq$ and we refer to it as *replacement order*, with the intuition that $K \prec K'$ means that K' replaces/overwrites K, and hence if K has not been delivered yet to the application, the knowledge sharing model may (but does not have to) discard it, if K' has already been delivered. Similar partial orders can be specified for events, but the case of knowledge is far more interesting due to its distributed nature.

Our model generalizes the knowledge-based networking approach of [1], because instead of specific types of knowledge, e.g., to support routing decisions, knowledge becomes an entirely application-defined concept. Further details, including a formal definition of the partially ordered knowledge sharing model, can be found in [2]. The model can be specialized by imposing local and global resource bounds as well by more specific environment (and hence network) models. The abstract dynamic network model and the wireless model that we discuss in Section IV are such examples.

III. CYBER-FRAMEWORK

From a software engineering perspective, the major challenges of an NCPS include (i) the design of an underlying software architecture with the most appropriate primitives for communication that can shield the applications from the complexities of dealing with dynamic topologies, delays/disruptions, and failures of all kinds, (ii) the careful design of APIs that are simple, flexible, and platform-independent, but at the same time easily extensible to capture the wide range of capabilities provided by cyber-physical devices, and (iii) the identification of typical application patterns that exploit the capabilities of the architecture to solve problems that involve the distributed and dynamic composition of the node capabilities to satisfy high-level global objectives, given the underlying hardware and network limitations. As a first step toward (iii), we show how a common pattern of distributed optimization and control can be expressed in the framework.

A. Preliminaries

We informally described the knowledge sharing model as a basis for distributed computing in Section II. Here, we present the cyber-application framework as one possible implementation of this model. We define the terminology and assumptions underlying our approach followed by the architecture of the proposed framework with its API and services.

The cyber-application framework (cyber-framework) consists of cyber-hosts, cyber-engines, cyber-nodes, and cyber-applications as depicted in Figure 1. A *cyber-host* and a *cyber-engine* correspond to a specific machine and a process on which cyber-applications are running, respectively. A unique identifier is randomly generated by each cyber-engine and is in turn used to generate unique identifiers for all (virtual) cyber-nodes that it is managing. Each *cyber-application* runs on a *cyber-node*, i.e., the smallest managed computational resource with or without attached *cyber-physical devices* (e.g., sensor, actuator). For example, a cyber-application for a mobile robot that needs to take a picture at a specific location could be decomposed into a parent cyber-application that contains the local decision and control logic, and two children, e.g., camera and positioning applications, that only manage the corresponding devices, as depicted by dotted lines in Figure 1. The camera application runs on a node that manages the camera device, and in the simplest possible case it just provides an abstract knowledge-based interface to the exposed device capabilities. Similarly, a positioning application uses the resources of its underlying node to manage the positioning device, e.g., to post position updates as knowledge and react to knowledge that represents positioning goals. One advantage of using knowledge-based interfaces is the potential for a logical interpretation in terms of facts and goals [3], which can lead to a better understanding and a more principled treatment of an NCPS, even if they are not developed in a declarative paradigm.

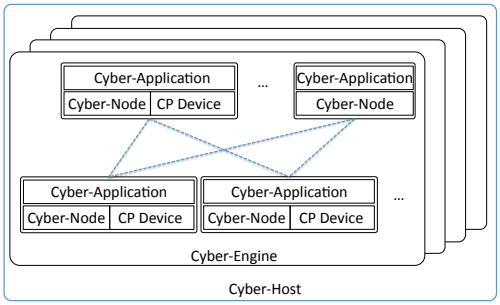


Fig. 1. Cyber-framework presented as a hierarchy of cyber-hosts, cyber-engines, cyber-nodes, CP (cyber-physical) devices (e.g., sensor, actuator), and cyber-applications.

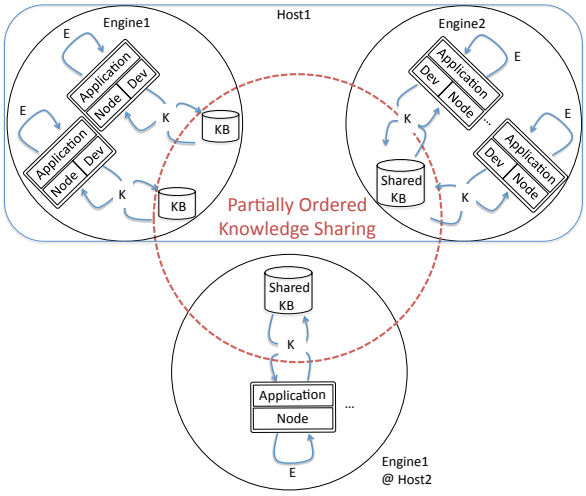


Fig. 2. Partially ordered knowledge sharing in a locally event-driven paradigm. Knowledge is stored redundantly in local knowledge bases (KBs) providing applications with the abstraction of a distributed network cache.

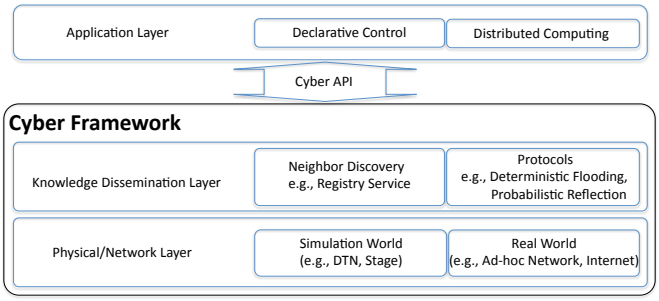


Fig. 3. Layered architecture of the cyber-framework.

As described in Section II, the computational model of each cyber-node is based on events and knowledge with corresponding equivalence and ordering relations. Each event is local in the sense that it is not distributed to other cyber-nodes, whereas knowledge is globally shared. Events can be posted by a device (e.g., for a sensor reading) or by a cyber-application (e.g., to schedule a local action) and will be handled by corresponding handlers. On the other hand, knowledge can be posted only by a cyber-application. In Figure 2, two cyber-hosts are sharing their knowledge. Cyber-host 1 has two cyber-engines. Each cyber-engine can have multiple cyber-applications with separate or shared knowledge bases, which serve as a network cache to enable opportunistic knowledge sharing with intermittent connectivity so that sender and receiver are completely decoupled.

B. Architecture

In an NCPS, information about the physical world (e.g., wireless signal strength, sensor readings) is usually acquired via physical devices, which can be geographically scattered. Hence, the management of an NCPS entails the integration between a physical layer, which includes the underlying physical devices as well as the network, and a higher layer, which includes the knowledge dissemination layer and the application layer that is designed to achieve global objectives. Figure 3 depicts the layered architecture of the cyber-framework.

To deal with uncertainty, complexity, and resource constraints in both communication and computation, the cyber-framework is concerned with decisions at each layer. For example, assume an application operating a networked team of mobile robots, i.e., mobile computational resources with sensors and actuators. At the application layer, it needs to decide when to perform a local action (e.g., move forward, sense the temperature of a certain area) to achieve local goals, which are typically obtained by decomposition from higher-level goals. Local computations are also needed for tasks such as information fusion, based on which new goals may be triggered. In [3], we presented a declarative approach to avoid low-level programming by combining distributed reasoning and asynchronous control. In this paper, we present a case study of a dynamic distributed optimization and control problem that is not solved using a distributed logic but directly built on top of the proposed cyber-framework in Section V.

At the knowledge dissemination layer, a specific strategy (e.g., deterministic flooding) is chosen to propagate knowledge on top of the underlying physical network layer. The global objective needs coordination among the nodes. Knowledge about local solutions needs to be disseminated through the network to improve overall solution quality. What kind of knowledge and how often the knowledge should be disseminated depends on the specific application and its objectives.

C. Cyber-API — Bridging the Gap

The rationale behind the API of our cyber-application framework is that it (i) should support simulation and analysis of systems in the context of suitable environment models as well as real-world implementations based on the same code; (ii) is reusable across a wide spectrum of platforms, networking technologies, and cyber-physical devices; (iii) is based on a sound mathematical foundation that enables reasoning about the complex dynamics of the composition of cyber- and physical components; and (iv) provides a suitable abstraction to implement robust strategies for distributed optimization and control, which are of key importance for nearly all NCPSs.

We briefly introduce the core interfaces of our API with some simplifications to focus on the key ideas. The full version is available from `http://ncps.csl.sri.com`. The API supports three different flavors of code organized in a sandwich-like approach: (i) at the lowest level, we have simulation code defining simulation models including local and global models, (ii) in the middle, we have application code extending application classes that cannot distinguish between simulated and real-world operation, (iii) at the top, simulation set-up code instantiates, composes, and configures simulation models and applications.

Each cyber-application runs on top of a cyber-node. The *CyberNode* interface gives applications access to the node's name and local time. The *parentname* is a way to exploit a hierarchy between nodes, e.g., several nodes that are physical subcomponents of a larger component. A cyber-node provides services to post events and knowledge (*postEvent* and *postKnowledge*, respectively). The *CyberNode* interface has a method *addApplication* to allow cyber-applications to register with their underlying cyber-nodes. The *CyberApplication* interface requires applications to define the local initialization and event/knowledge handling functions.

```
public interface CyberNode
{
    double time();
    String name();
    String parentname();

    void postEvent(ApplicationEvent e);
    void postKnowledge(ApplicationKnowledge k);

    void addApplication(CyberApplication app);
}
public interface CyberApplication
{
    void initialize();
    void handleEvent(ApplicationEvent e);
    void handleKnowledge(ApplicationKnowledge k);
}
```

A cyber-application may have access to additional services provided by attached cyber-physical devices. For example, the positioning device of a mobile robot may have the following

Position interface. Similar to *CyberNode*, the application needs to register with the device using *addApplication*. The positioning device provides access to its current position by *posx* and *posy* methods and supports the asynchronous initiation of a position change through the *move* service. The corresponding *PositionApplication* simply needs to implement a handler *handlePositionUpdate*, which is called when a position update occurs.

```
public interface Position
{
    String name();
    String parentname();

    double posx();
    double posy();
    void move(double newx, double newy);

    void addApplication(PositionApplication app);
}
public interface PositionApplication extends CyberApplication
{
    void handlePositionUpdate();
}
```

In addition to initializers and handlers, applications need to specialize the abstract *ApplicationKnowledge* and *ApplicationEvent* classes to make use of the corresponding services. For example, *PositionKnowledge* can be posted by a cyber-node that is equipped with a positioning device by extending *ApplicationKnowledge*. Knowledge can be local (used by one application) or global (shared by multiple applications), whereas events are always local to applications. As explained in Section II, knowledge and events have a creator, a creation time, an activation time, and an expiration time. For activation and expiration of knowledge and events, we use a relative delay from current local time. Most importantly, applications need to specify subsumption and replacement orderings for knowledge and events by overwriting *subsumes* and *replaces*.

```
public abstract class ApplicationKnowledge
{
    public ApplicationKnowledge(CyberApplication app,
        double activationDelay, double expirationDelay);

    public String creator();
    public double creationTime();
    public double activationTime();
    public double expirationTime();

    public boolean subsumes(ApplicationKnowledge k);
    public boolean replaces(ApplicationKnowledge k);
}
public class ApplicationEvent extends Event
{
    public ApplicationEvent(CyberApplication app,
        double activationDelay, double expirationDelay);
    ...
}
```

Finally, the top-level simulation setup is defined by extending the *SimWorld* class of the cyber-framework. The following *SensorWorld* example instantiates five mobile robots that are each composed of two instances of *SimNode*, a robot *robot* and a subnode *posnode*. Here, *SimNode* is used as a simulation model for a cyber-node. We also define *posdev*, an instance of *SimPosition*, a simulation model of a robot positioning and localization device. At the end of the loop, a robot application is instantiated to run on the corresponding node $robot$, and similarly a positioning application is instantiated to run on $posnode$ and utilizes the device $posdev$.

```
class SensorWorld extends SimWorld
{
  public SensorWorld()
  {
    for(int i = 0; i < 5; i++)
    {
      SimNode robot =  new SimNode(this,"RobotNode"+i);
      SimNode posnode =
               new SimNode(this,robot[i],"PosNode"+i);
      SimPosition posdev =
               new SimPosition(this,posnode,"PosDev"+i);
      new RobotApp(robot);
      new PositionApp(posnode, posdev);
    }
  }
}
```

In a similar way, the *RealWorld* class can be extended to utilize the same application code with actual physical devices and knowledge dissemination on top of an actual network. Note that decomposing a robot into several nodes means that all communication between them must take place uniformly by means of the knowledge-based paradigm, but it does not require that these (virtual) nodes be mapped on different physical processors.

IV. SYSTEM IMPLEMENTATION

In describing our current prototypical implementation of the cyber-application framework, note that, although our current implementation is based on Java, implementations in other languages such as C or C++ are possible and of potential interest for better performance or for resource-constrained devices.

A. Simulation vs. Real World for Physical/Network Layer

Our current prototype of the cyber-framework includes two types of physical- and network-layer implementations depicted as simulation (*SimWorld*) and real world (*RealWorld*) in Figure 3, respectively. In the case of *SimWorld*, we support communication among cyber-nodes via (i) a DTN simulator that we previously developed [1] as well as (ii) Stage, a widely used multi-robot simulator [4] that supports a simple wireless network model. In the DTN simulator, we use an abstract topological mobility model instead of a model with actual coordinates. The DTN network model is a probabilistic dynamic graph-based model, where each link has a state, e.g., up or down, and is characterized by its abstract features such as bandwidth, latency, and error rate. To utilize the Stage simulator together with the Java-based implementation of the cyber-framework, we developed a *Stage API* using JNI (Java Native Interface) to control the robots, their devices, and the environment in the C++-based Stage simulator. In this way, we have access to the physical device models (e.g., SimPositionStage, SimCameraStage, SimPowerPackStage) that can be attached to robots in Stage. The Stage wireless network model uses the signal-to-noise ratio and interference from other robots to simulate losses and delays associated with packet transmissions. As an alternative, we also developed a hybrid simulation model that bypasses the Stage packet transmission (i.e., a packet is transmitted using the DTN simulator, yet link quality information is extracted from the Stage simulator).

Different from *SimWorld*, the *RealWorld* class utilizes an actual network among different cyber-engines to disseminate knowledge. For minimum overhead, our current protocols have been developed on top of UDP. Typically, many (virtual) cyber-nodes are executed by a cyber-engine, which is the smallest (real) computational resource distinguishable from outside. We also support multiple cyber-engines on a single cyber-host by dynamically allocating a different part to each cyber-engine.

A real-world deployment can run on top of multiple cyber-hosts, unlike *SimWorld* that is currently limited to a single cyber-engine on a single cyber-host. Across the network, time synchronization becomes an important issue. Currently, for *RealWorld* instances we have implemented a loose time synchronization scheme based on logical time to synchronize local clocks in different cyber-hosts, but for use in time-critical cyber-physical systems we plan to investigate protocols that can utilize clocks of different qualities (an information fusion problem) and use local adaptation to additionally adjust their drift rates (a distributed control problem).

B. Neighbor Discovery

To disseminate knowledge via opportunistic links, each cyber-engine needs to keep track of its *immediate neighborhood*, i.e., the set of cyber-engines that are currently reachable in a single hop. Each cyber-engine advertises its presence by periodically emitting a *hello* packet that contains its cyber-engine identifier, IP addresses of all network interfaces attached to its cyber-host, and its port number. This *hello* packet is broadcast on the same subnet and additionally unicast to a set of *potential neighbors* initially given by a set of user-defined IP addresses. To enable multi-hop discovery (e.g., for bootstrapping when broadcasting is not sufficient), *hello* packets can be forwarded until a small user-defined maximum hop count is reached (without counting subnet broadcasts). To enable the use of Network Address Translators (NATs), e.g., on the edge of private subnets, the receiving cyber-engine will additionally record the sender's public IP address.

Each cyber-engine discovers and refreshes its immediate neighborhood based on the incoming *hello* packets. Once a certain expiration time for a neighbor entry is reached, the corresponding cyber-engine is removed from the immediate neighborhood, but remains an element of the set of potential neighbors to which *hello* packets are sent so that a reconnection can be quickly detected.

If broadcasting is not available or not sufficient to enable full discovery, cyber-engines can send their *hello* packets (with maximum hop count 2) to one or multiple user-defined *facilitators*, i.e., cyber-engines that simply forward it one more hop, and in particular to those neighbors from which they have already received *hello* packets. In this way, potential two-hop neighbors can become immediate neighbors.

C. Knowledge Dissemination Protocols

At present, two knowledge dissemination protocols are supported by the framework. Generalizing the traditional notion of acknowledgments, information about the awareness of nodes regarding units of knowledge, a form of knowledge about knowledge, is shared in both protocols. First, a simple optimized flooding mechanism disseminates knowledge to all neighbors that are not (known to be) aware of the particular unit of knowledge. Second, we have implemented a protocol based on a notion of probabilistic refection that we previously developed [1]. It is a single-message protocol that can operate under unpredictable dynamic network conditions with very small windows of communication opportunities. The knowledge dissemination protocols make essential use of the partial order defined on knowledge, by replacing and hence discarding a unit of knowledge whenever a unit that is higher in the ordering is received.

To illustrate the benefits of exploiting the partial-order semantics of knowledge for scalability, consider a simple randomized distributed optimization algorithm to find the best solution, i.e., with maximum fitness value. The knowledge dissemination layer in the cyber-framework will then replace and discard all instances of *SolutionKnowledge* if their fitness values are less than that of a new unit of knowledge. This is a very typical use of a total subsumption ordering. For the purpose of optimization a better solution always has a higher information content (formally it gives a better bound for the unknown optimum). Since any *SolutionKnowledge* with less optimal fitness value will be absorbed by the knowledge dissemination at each hop, the amount of knowledge at each node is bounded, and the algorithm becomes naturally scalable in number of nodes. Clearly, this is an extreme case, but it illustrates the possibility of scalable knowledge dissemination and the importance of the ordering.

In an actual deployment on a network, knowledge is normally disseminated at the cyber-engine level, since the neighborhood is maintained by each cyber-engine. Within a cyber-engine, the knowledge will be available to all cyber-nodes, usually by means of a shared knowledge base as depicted in Figure 2. Since the framework can also be used for the purpose of simulation where cyber-nodes are part of a network model, we additionally implemented the option of using knowledge dissemination protocols at the cyber-node level (inside a single cyber-engine).

D. Multi-threaded Execution and Simulation

In the cyber-framework, a local computation is triggered by processing an event from the event queue, which exists per cyber-engine, i.e., per process. To increase the performance through parallel processing of those events, we implemented two versions of multi-threaded execution in our framework. First, we exploit locality of events by executing each cyber-node in Figure 1 as a thread with its own event queue. This approach is appropriate when a relatively small number of cyber-nodes exists within the same cyber-host. However, as the number of cyber-nodes increases, thread management overhead increases, and in practice platform resource-dependent bounds limit the number of useful threads.

Hence, we alternatively allow each cyber-engine to maintain a single shared event queue with an associated thread pool. A thread pool is a fixed collection of threads that can be dynamically allocated to perform tasks in the background. Events are placed in the queue until they can be serviced as threads become available. In this manner, we can decrease the thread management overhead by avoiding the cost of creating a dedicated thread for each cyber-node. The thread pool approach also reduces synchronization overhead when multiple threads simultaneously attempt to access the event queue.

In summary, parallel execution can take place in a fine-grained manner, i.e., by multiple cyber-nodes cooperating in a multi-threaded way, as well as in a coarse-grained manner, i.e., by multiple cooperating cyber-engine processes. As discussed above the coarse-grained approach can again be used at different levels (and correspondingly at multiple timescales), e.g., on a single host (local communication), hosts on the same subnet (broadcasting), and beyond subnets (unicasting), e.g., on a computing grid. The cyber-framework supports all those configurations as well as arbitrary combinations.

E. Probabilistic Analysis

To analyze the behavior of NCPS (e.g., in terms of properties and other discrete or continuous observables) in a probabilistic sense, the cyber-framework implements various approximation algorithms. The generic approximation algorithm [5] provides a technique to constrain the error probability of an answer by calculating the sufficient sample size (i.e., number of obervations) from the given error bound (ϵ) and confidence interval (δ). Under mild boundedness conditions, the optimal (ϵ, δ)-approximation algorithm [6] uses a number of samples that can be proven to be optimal, i.e. within a constant factor relative to the minimum sample size. Given a property, the sequential probability ratio test [7] continues sample generation until its answer about accepting or rejecting the hypothesis can be guaranteed to be correct within the required error bounds. Black-box testing [8] instead computes the statistical significance (p-value) for a given number samples without having any control on the execution. In our recent work [9] we extended the quantitative approach of [10] by an on-demand sample generation that can compute the sample size sufficient to reach confidence in the normality of data, and then utilize the normal distribution to obtain the error bound and confidence interval for quantitative analysis.

With all of the above-mentioned approximation algorithms implemented, the cyber-framework can be used to quantify statistical performance (e.g. execution times) with a specific confidence and to verify properties (e.g. classical invariants such as mutual exclusion), which may only be satisfied in a probabilistic sense.

V. CASE STUDY

To demonstrate the applicability and performance of cyber-framework, we focus on a specific case study in the context of distributed sensing, optimization, and control. In this case study, we assume that various wireless sensors are deployed in space yet isolated in the sense that the sensors can not communicate directly with each other due to limited energy (e.g., small battery and long lifetime), which requires mobile robots to take the role of data collectors. This problem is representative of a class of challenging problems, since the network is highly dynamic and in the presence of uncertainties, delays, and failures it requires cooperation among mobile robots to achieve a globally approximate optimal solution.

We use a mapping into the multiple Traveling Salesman Problem (mTSP) [11], a well-known NP-complete problem. Multiple mobile robots attempt to find the most cost-efficient route assignment visiting all sensor locations and returning to a common starting point. In our case study, a robot can also encounter other robots on its traversal and exchange up-to-date information about each robot and about the sensors already covered, which leads to a dynamic version of mTSP with opportunistic knowledge sharing. The algorithm should be executed in a fashion that can adjust load distribution among robots in a fully decentralized manner to cope with failures of all kinds.

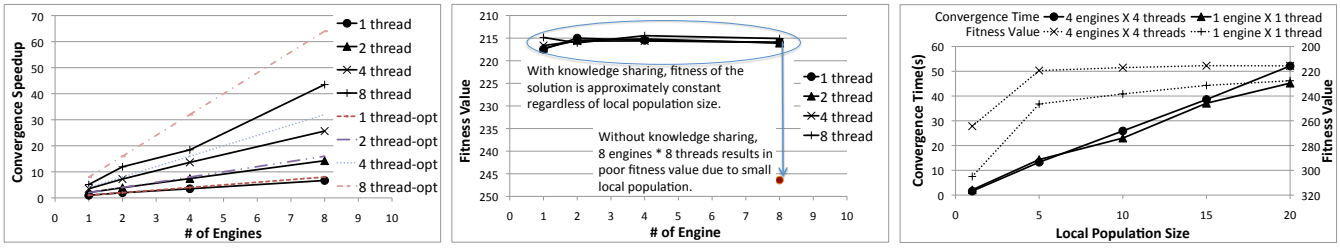
As an approach that can be naturally distributed, we use a quantum evolutionary algorithm (QEA) [12], which is based on the concepts borrowed from quantum computing such as the notions of quantum bit and superposition of states. The key ideas of QEA lie in the concept of a Q-bit individual, a compact probabilistic representation of an entire set defined as a string of Q-bits, and the operation of a Q-gate, designed to probabilistically adjust the Q-bit individual. However, it should be noted that we are not aiming to devise a better QEA for a specific problem. Instead, we developed a parallel and distributed version of an existing QEA implementation as a sample application of our framework to explore its performance, robustness, and scalability. Our current implementation of QEA can be further optimized and equipped with adaptive capabilities, which is beyond the scope of this paper.

A. Distributed QEA on the Cyber-Framework

Algorithm V.1: DISTRIBUTED QEA()

```
INITIALIZE()
{
  InitializePopulation(n)
  POSTEVENT(delay)
}
HANDLEEVENT()
{
  ∀Q_i ∈ P_r {
      q_i(t) ← Observe(Q_i(t))
      if(fitness(q_i(t)) > fitness(q_opt(t)))
        then { q_opt(t) ← q_i(t)
               Q_opt(t) ← Q_i(t)
      Q_i(t) ← AdjustQGate(q_i(t), q_opt(t), Q_i(t))
  if(SharingCondition)
     then POSTKNOWLEDGE(q_opt(t), Q_opt(t))
  POSTEVENT(delay)
}
HANDLEKNOWLEDGE(q(t), Q(t))
{
  RandomReplace(q(t), Q(t))
}
```

Algorithm V.1 describes the overall structure of the distributed QEA as a cyber-application. In the beginning, an initial population P_r of fixed size n is generated by each robot r. Next, an event is scheduled to trigger a computation



(a) (b) (c)

Fig. 4. Performance of distributed QEA on cyber-framework; (a),(b) on varying number of engines and threads per engine (with total population size of 320 that is evenly distributed to each thread), (c) on varying local population size. The best observed fitness value is 213.

after a $delay$, which bounds the local computational resources used. The main computation loop at each robot is expressed by the HANDLEEVENT procedure. Each robot r starts its local optimization based on its local population P_r and the locally best solution $q_{opt}(t)$ at time t. An individual solution $q_i(t)$ is obtained by observation (in the quantum mechanical sense) from $Q_i(t)$, the current Q-bit representation of each individual in P_r, and its fitness is compared with that of $q_{opt}(t)$. If $q_i(t)$ is better than $q_{opt}(t)$, then the algorithm records $q_i(t)$ as a best solution $q_{opt}(t)$. The algorithm updates the individual's Q-bit representation $Q_i(t)$ by means of the quantum gate, which amplifies the probability of observing the optimal solution from $Q_i(t)$.

The distributed QEA allows knowledge sharing when connectivity among robots is available. As formulated by the *SharingCondition*, when each robot has generated a sufficiently good solution (i.e., achieves a certain fitness improvement) or convergence is detected, the application posts knowledge about its locally best solution $q_{opt}(t)$ with the Q-bit representation $Q_{opt}(t)$ of its individual. At the end of HANDLEEVENT, an event is scheduled with $delay$ for the next optimization step. On the other hand, a random individual of the population will be replaced when incoming knowledge about the solutions from another robot arrives by using the same ordering relations as in the example of Section IV, i.e., suboptimal solutions can be discarded in favor of better solutions while they are cached in the network.

B. Experiments and Discussion

We implemented two test scenarios of utilizing our framework for distributed QEA. First, we installed the framework on PlanetLab, an Internet-wide testbed [13], and executed the distributed QEA with a *RealWorld* setup. In this case, the fairly stable computing cluster could be used as a command post for robots. Second, the distributed QEA was executed on Stage robots in a *SimWorld* setup. Here, robots are moving around based on knowledge about their progress in the simulated environment, and autonomy is essential due to intermittent connectivity. We evaluated the performance of distributed QEA on PlanetLab in terms of convergence speedup and fitness improvement in solving mTSP with 3 robots and 20 cities. To demonstrate the impact of multi-threading, we controlled the amount of parallelism by varying the number of engines and threads per engine. Each engine operates on a different cyber-host with a thread pool implementation. We experimented with fine-grained (i.e., single engine with multiple threads), coarse-grained (i.e., multiple engines, each with a single thread), and hybrid (i.e., multiple engines with multiple threads per engine) setups as discussed in Section IV-D.

Figure 4(a) shows that, with the same total population size, parallelism linearly improves convergence speed due to a smaller population per thread. From the comparison with the theoretically optimal speedup (as depicted by dashed lines), we also observe that the speedup due to multi-threading (i.e., with a shared memory knowledge base) can be less than the speedup resulting from distributed execution in spite of the fact that dual-processor quad-core machines were used in the experiment. Compared with the optimal case, the executions with 8 threads per engine achieve lower speedups (from 58% to 75%) than a single-threaded version with multiple engines (from 83% to 99%). Some of the performance variation is due to the use of a shared and heterogeneous testbed. We believe, however, that further optimizations of the multi-threaded execution, in particular of the knowledge base representation and its concurrent access, are possible and worthwhile. Even in the case with a small local population, our partially ordered knowledge sharing enables the distributed QEA to achieve approximately the same fitness value in all configurations as shown in Figure 4(b). Without knowledge sharing, the maximally concurrent execution (64 threads) results in a very poor solution due to the small local population in the bottom right of Figure 4(b). We also varied the local population size, which shows how a larger local population leads to higher fitness at the cost of convergence speed. Knowledge sharing mitigates the disadvantage of a smaller local population as illustrated by executions with single versus 16 threads in Figure 4(c).

For the *SimWorld* setup, we modified the distributed QEA to dynamically take into account partial information about the distributed progress by sharing knowledge on the robot's trajectory whenever connectivity is available. When connected, the robots cooperatively recompute a new solution for the reduced instance of mTSP reflecting all available knowledge, i.e., each robot executes the QEA with the union of its assigned but not yet visited cities according to its current solution and with their current positions. Robots return to the starting location after visiting their assigned cities and will repeat

the algorithm if not all cities are known to be covered. In this manner, uncertainties (e.g., battery depletion may slow a robot's movement) and runtime failures (e.g., robots are physically damaged) can be reflected in a new solution both during the normal operation and after returning to the starting location.

VI. RELATED WORK

Different from established and powerful frameworks for cyber-physical systems such as CybelePro [14] or Ptolemy [15], our framework is intended as a research tool to explore the feasibility, the possibilities, and the implications of a new distributed computing paradigm for NCPSs based on a partially ordered knowledge-sharing model that is loosely coupled in an extreme sense. This direction of research is motivated by current trends in networking and by the expectation that with the exponentially growing number of resource-constrained devices, issues such as failures, unreliability, and intermittent connectivity will become the norm rather than the exception so that new scalable foundations are urgently needed.

Knowledge sharing is a well known idea that has been investigated by Halpern [16] and in many subsequent works. Understanding knowledge sharing in distributed environments has lead to a complementary view providing new insights into distributed algorithms and a logical justification for their fundamental limitations.

As pointed out earlier, our partially ordered knowledge-sharing model can be implemented using gossip-style protocols [17], such as anti-entropy protocols [18] or bimodal multicast [19], but it provides a higher level of abstraction that enables many possible implementations with a wide

range of protocols and networking technologies.

Anti-entropy protocols [18] perform pairwise exchanges of so-called deltas, that is, differences in the local states of the two peers particpating in an interaction. At the core of the model it is assumed that each peer has a set of variables (keys) which not only have a value but also a version number. The ordering on version numbers is then be used by the anti-entropy protocol to discard old versions in faviour of new versions whenever information is merged and cached in the network. It is noteworthy that the concept of versions with with their total order is a special case of partially ordered knowledge sharing.

In contrast to anti-entropy gossip protocols, which are reconciling local state, gossip-based disemination protocols such as bimodal multicast [19] provide logically a broadcasting service for streams of messages, which are typically buffered only for short periods of time so that buffer space is available for new messages comming in. The network model is critical in gossip-based dissemination. For instance, network partiting is usually not considered, and it has been observed that gossip-based dissemination is not robust under correlated losses [17].

Semantically reliable multicast [20] is designed to make use of the semantics of messages to discard obsolete messages in overload situations. To this end the authors assume that messages are equipped with an obsolesence relation that is coherent with the causal order of events. As suggested by the authors this can be implemented by simply tagging each message at its source with all messages that it makes obsolete. Different from the general partially ordered knowledge-sharing model, the obsolesence relation is defined independently for each stream of messages by the sender. Probabilitic reliable multicast [21] is a combination of semantically reliable multicast and gossip-based probabilisitic multicast.

Delay-tolerant networking [22], [23], [24] evolved from early ideas on an interplanetary Internet architecture [25] and uses late binding and a store-and-forward appoach (potentially utilizing persistent storage) to deal with episodic and intermittent connectivity and to overcome delays and temporary disconnections. Network partitioning and merging is usually considered part of the normal operation, especially when nodes (or groups of nodes) are used as data mules or message ferries [26] to transport stored messages by means of physcial mobility. A related concept are throw-boxes [27], which can be placed in the environment as buffers to further improve temporal decoupling of nodes. More generally, and in contrast to traditional Internet or MANET protocols, delay-tolerant networking aims to support communication even if a simultaneous end-to-end path does not exist. Instead of operating at the packet level, its units of information are semantically meaningful bundles (or fragments) of variable and typically large size. In content-centric networking [28] the semantically meaningful unit is referred to as content, and the network is viewed as a content cache which is querried by the user.

Distributed tuple spaces [29], [30] may resemble our proposed loosely coupled paradigm based on partially ordered knowledge sharing, but the atomicity properties of tuples spaces cannot be implemented in highly dynamic environments and impose strong limits on their scalability. For instance, the LIME (Linda in a Mobile Environment) [29] is based on the idea that the tuple spaces of individual host conceptually merge when they come into contact, which can partition again when the conneciton is lost. A different approach is taken in the space-based computing architecture of [30] which organizes tuple spaces using distributed hash tables.

Blackbord systems [31] are a well-known paradigm in multi-agent systems that allows multiple agents to interact and collaborate by sharing knowledge through a so-called blackboard. Parallel and distributed implementations have been proposed in [32] and [33]. Consistency maintenence among replicated blackboard data and the implementation of blackboard transactions have been identified as major challenges in [32]. By explicitly admitting and handing errors, uncertainty, and temporal inconsistencies as part of the overall problem solving process, our partially ordered knowledge sharing model may be regarded as a theoretical and practical foundation for functionally accurate, cooperative distributed systems [34], a vision that anticipated many requirements of NCPS and was already far ahead of its time when published.

VII. CONCLUSIONS

As a first step, we have presented in this paper an application framework based on the partially ordered knowledge sharing model and an API for cyber-physical devices that enables interaction with the physical world. Key features of our framework are that it is network independent and that it enables the same application code to be used in various environments including simulation models and real-world deployments. As a test case, we have applied it to a quantum evolutionary algorithm for a dynamic distributed optimization and control problem. Our experiments on both a multi-robot simulator and an Internet testbed indicate that the framework can dynamically adapt to a wide range of operating points between autonomy and cooperation to overcome limitations in connectivity and resources, as well as uncertainties and failures.

In the future, we envision using a multi-robot testbed similar to that of SRI's Centibots [35] and Commbots [36] for realistic experiments. The combination of the cyber-framework with our distributed logic [3] will facilitate a new style of distributed, declarative control. The use of the framework for large-scale distributed simulation and analysis of the NCPS (beyond our multi-threaded implementation) is an unexplored possibility and has the potential for better scalability than conventional simulators due to the locality of events and immutability of knowledge. Applying the framework to emerging computing platforms (e.g., manycore architectures, cloud computing, instrumented smart spaces) as well as to social computing networking applications that utilize mobile devices is an interesting direction, especially if such platforms increasingly become part of NCPS.

Acknowledgments: We thank Dr. Hoeseok Yang for providing his implementation for the sequential version of QEA. We also thank Andy Poggio and Dr. Steven Cheung at SRI International for their constructive criticism and valuable suggestions. Support from National Science Foundation Grant 0932397 (A Logical Framework for Self-Optimizing Networked Cyber-Physical Systems) and Office of Naval Research Grant N00014-10-1-0365 (Principles and Foundations for Fractionated Networked Cyber-Physical Systems) is gratefully acknowledged. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of NSF or ONR.

REFERENCES

[1] M.-O. Stehr and C. Talcott, "Planning and learning algorithms for routing in disruption-tolerant networks," in *IEEE Military Communications Conference*, 2008.

[2] M.-O. Stehr, M. Kim, and C. Talcott, "A partially ordered knowledge sharing model for networked cyber-physical systems," July 2010, Draft available at http://www.csl.sri.com/~stehr/CPS/poksmodel.pdf.

[3] M-O. Stehr, M. Kim, and C. Talcott, "Towards distributed declarative control of networked cyber-physical systems," 2010, to appear in Ubiquitous Intelligence and Computing (UIC ' 10), Full version available at http://www.csl.sri.com/~stehr/CPS/cpslogic.pdf.

[4] R. B. Rusu, A. Maldonado, M. Beetz, and B. Gerkey, "Extending Player/Stage/Gazebo towards cognitive robots acting in ubiquitous sensor-equipped environments," in *IEEE International Conference on Robotics and Automation Workshop for Network Robot Systems*, 2007.

[5] R. Lassaigne and S. Peyronnet, "Probabilistic verification and approximation," *Electron. Notes Theor. Comput. Sci.*, vol. 143, pp. 101–114, 2006.

[6] P. Dagum, R. Karp, M. Luby, and S. Ross, "An optimal algorithm for monte carlo estimation," *SIAM J. Comput.*, vol. 29, no. 5, pp. 1484–1496, 2000.

[7] H. Younes, "Ymer: A statistical model checker." in *17th International Conference on Computer Aided Verification (CAV '05)*, ser. LNCS, vol. 3576, pp. 429–433, http://www.tempastic.org/ymer.

[8] K. Sen, M. Viswanathan, and G. Agha, "Statistical model checking of black-box probabilistic systems," in *16th International Conference on Computer Aided Verification (CAV '04)*, ser. LNCS, vol. 3114, pp. 202–215.

[9] M. Kim, M.-O. Stehr, C. Talcott, N. Dutt, and N. Venkatasubramanian, "A probabilistic formal analysis approach to cross layer optimization in distributed embedded systems," in *9th IFIP International Conference on Formal Methods for Open Object-based Distributed Systems (FMOODS'07)*, ser. LNCS, vol. 4468, 2007, pp. 285–300.

[10] G. Agha, J. Meseguer, and K. Sen, "PMaude: Rewrite-based specification language for probabilistic object systems," in *3rd Workshop on Quantitative Aspects of Programming Languages (QAPL'05)*, 2005.

[11] T. Bektas, "The multiple traveling salesman problem: an overview of formulations and solution procedures," *Omega*, vol. 34, no. 3, pp. 209–219, June 2006.

[12] K. Han, K. Park, C. Lee, and J. Kim, "Parallel quantum-inspired genetic algorithm for combinatorial optimization problem," in *IEEE Congress on Evolutionary Computation*, 2001, pp. 1422–1429.

[13] PlanetLab, http://www.planet-lab.org.

[14] CybelePro, http://www.cybelepro.com.

[15] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong, "Taming heterogeneity - the Ptolemy approach," *Proceedings of the IEEE*, vol. 91, no. 1, pp. 127–144, 2003.

[16] J. Y. Halpern and Y. Moses, "Knowledge and common knowledge in a distributed environment," *Journal of the ACM*, vol. 37, pp. 549–587, 1984.

[17] K. Birman, "The promise, and limitations, of gossip protocols," *SIGOPS Oper. Syst. Rev.*, vol. 41, no. 5, pp. 8–13, 2007.

[18] R. van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," in *LADIS '08: Proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*. ACM, 2008, pp. 1–7.

[19] K. P. Birman, M. Hayden, O. Ozkasap, Z. Xiao, M. Budiu, and Y. Minsky, "Bimodal multicast," *ACM Trans. Comput. Syst.*, vol. 17, no. 2, pp. 41–88, 1999.

[20] J. Pereira, L. Rodrigues, and R. Oliveira, "Semantically reliable multicast: Definition, implementation, and performance evaluation," *IEEE Trans. Comput.*, vol. 52, no. 2, pp. 150–165, 2003.

[21] J. Pereira, R. Oliveira, L. Rodrigues, and A.-M. Kermarrec, "Probabilistic semantically reliable multicast," in *NCA '01: Proceedings of the IEEE International Symposium on Network Computing and Applications (NCA'01)*. Washington, DC, USA: IEEE Computer Society, 2001, p. 100.

[22] K. Fall, "A delay-tolerant network architecture for challenged internets," in *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2003, pp. 27–34.

[23] S. Farrell and V. Cahill, *Delay- and Disruption-Tolerant Networking*. Norwood, MA, USA: Artech House, Inc., 2006.

[24] Z. Zhang and Q. Zhang, "Delay-/disruption tolerant mobile ad hoc networks: latest developments: Research articles," *Wirel. Commun. Mob. Comput.*, vol. 7, no. 10, pp. 1219–1232, 2007.

[25] V. Cerf, S. Burleigh, A. Hooke, L. Torgerson, R. Durst, K. Scott, E. Travis, and H. Weiss, "Status of this memo interplanetary internet (ipn): Architectural definition," May 2001, internet Draft.

[26] W. Zhao and M. H. Ammar, "Message ferrying: Proactive routing in highly-partitioned wireless ad hoc networks," in *FTDCS '03: Proceedings of the The Ninth IEEE Workshop on Future Trends of Distributed Computing Systems*. Washington, DC, USA: IEEE Computer Society, 2003, p. 308.

[27] W. Zhao, Y. Chen, M. Ammar, M. Corner, B. Levine, and E. Zegura, "Capacity enhancement using throwboxes in dtns," in *In Proc. IEEE Intl Conf on Mobile Ad hoc and Sensor Systems (MASS)*, 2006, pp. 31–40.

[28] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*. ACM, 2009, pp. 1–12.

[29] A. L. Murphy, G. P. Picco, and G.-C. Roman, "Lime: A coordination model and middleware supporting mobility of hosts and agents," *ACM Trans. Softw. Eng. Methodol.*, vol. 15, no. 3, pp. 279–328, 2006.

[30] S. Bessler, A. Fischer, E. Khn, R. Mordinyi, and S. Tomic, "Using tuple-spaces to manage the storage and dissemination of spatial-temporal content," *Journal of Computer and System Sciences*, 2009.

[31] R. S. Engelmore and A. Morgan, Eds., *Blackboard Systems*. Addison-Wesley, 1988.

[32] D. D. Corkill, "Design alternatives for parallel and distributed blackboard systems," in *Blackboard Architectures and Applications*, V. Jagannathan, R. Dodhiawala, and L. S. Baum, Eds. Academic Press, 1989, pp. 99–136.

[33] J. R. Ensor and J. D. Gabbe, "Transactional blackboards," in *IJCAI'85: Proceedings of the 9th international joint conference on Artificial intelligence*. Morgan Kaufmann Publishers Inc., 1985, pp. 340–344.

[34] V. R. Lesser and D. D. Corkill, "Functionally accurate, cooperative distributed systems," in *Distributed Artificial Intelligence*. Morgan Kaufmann Publishers Inc., 1988, pp. 295–310.

[35] C. L. Ortiz, R. Vincent, and B. Morisset, "Task inference and distributed task management in the Centibots robotic system," in *AAMAS '05: Proceedings of the Fourth International Joint Conference on Autonomous Agents and Multiagent Systems*, 2005, pp. 860–867.

[36] B. P. Gerkey, R. Mailler, and B. Morisset, "Commbots: Distributed control of mobile communication relays," in *AAAI Workshop on Auction Mechanisms for Robot Coordination (AuctionBots)*, 2006, pp. 51–57.