Cyber-Physical Systems and Events

Carolyn Talcott

SRI International
Menlo Park, CA 94025-3493, USA
clt@cs.stanford.edu

Abstract. This paper discusses *event-based semantics* in the context of the emerging concept of Cyber Physical Systems and describes two related formal models concerning policy-based coordination and Interactive Agents.

1 Introduction

Cyber-physical systems (CPSs) integrate computing and communication with monitoring and/or control of entities in the physical world. Sensing and manipulation of the physical world occurs locally, system behavior emerges as a result of communication and other forms of interaction. Example CPSs include automobiles, aircraft, air traffic control, power grids, oil refineries, medical devices, patient monitoring, and smart structures. Software is becoming an increasingly important element of the operation of these systems, and must do so dependably, safely, securely, efficiently and in real-time.

CPSs go beyond traditional embedded and distributed systems. They are often long lasting, with 24x7 operation and must evolve without losing stability. Some CPSs or their components have stringent QoS requirements, others are more flexible. Traditional embedded and critical systems are closed, not only in the sense of closed physical locations or dedicated networks, but also closed with respect to their computational boundaries, i.e., all the participating elements in the systems are known initially. Thanks to network technology and mobility, today's embedded systems are shifting towards openness and federation This leads to multi-scale, wide area critical systems with real-time requirements, all this still with certification requirements, a major verification challenge. The openness brings more convenience and flexibility for controlling the systems. However, it also introduces extra complexity into the systems: large scale, uncertainty, and dynamics—entities can come into or leave from the systems, and makes the coordination of entities even harder.

QoS requirements, such as timing properties, fault-tolerance, security, etc., dictate how individual entities of the system being considered coordinate with each other. For instance, a deadline constraint on a task indicates that there must exist another entity that is coordinated with the constrained task. Or the deadline misses its meaning. If we consider computation is to achieve the system's functional requirements, QoS requirements are reflected through coordination among computational entities. With this view, embedded or critical systems are compositions of two main elements—computation and coordination.

A solid semantic foundation is crucial for design, deployment, monitoring and adaptation of CPSs. Such a foundation must support reasoning locally about individual components and globally about system wide properties. Our hypothesis is that *event-based* *semantics* can provide such a foundation. In Section 2 we discuss the general idea of event-based semantics. A sampling of related work is presented in Section 3. This is followed by brief overviews of two specific ideas: PAGODA—a policy and goal-based approach to modeling autonomous system components, and Interactive agents—which combines a policy based coordination model and an extension of the notion of actors to include interactions other than messages.

2 Event-Based Semantics

We begin with a discussion of various notions of event, followed by essential features, and challenges to be met in developing event-based semantics for CPSs.

2.1 What Is an Event?

An instance of an action? An occurrence in time and space? A change of condition? There are many notions of event, different notions being useful for different purposes, including:

– In the actor model of computation an event is a message send or receive.
– In the process algebra model of computation an event is an action shared by two processes.
– In the world of linguistics events may happen over time or be nested. For example: Tim and Ben played World of WarCraft. They completed a quest.

We can classify event models along several dimensions:

– punctual (for example, message send/receive) vs durative (for example, filling the tank, attending a class)
– single vs stream (for example, card reader readings, periodic chemical sensor readings, or object tracking)
– change vs action/observation.

In addition, event models may have different underlying temporal models: causal ordering (before/after), discrete time, continuous time.

It is important for an event-based semantic to include many different notions of event, and to allow moving from one to another in meaningful ways.

2.2 Why Event-Based?

What are the essential features of event-based semantics? One key feature of events is that they concern interactions between components and observations rather that internal state. This enables specification and reasoning at higher-levels while integrating easily with more detailed information. Another key feature of events is the notion of causal partial order that reflects the physical reality that for events separated in space we may not be able to decide a linear order (and should not depend on it).

– Events are a natural way to think about reactive systems, such as CPSs. They provide a natural way to specify components of open systems in terms of interfaces and observable behavior. Events also can form the basis for specifying coordination and composition of components.
– Global state and local state can be abstracted to event partial orders. The causal partial order of an event semantics captures dependencies/consequences and allows reasoning about what must have happened in the past, given some reasonable assumptions about the behavior of system components. Further, by (logically) locating events, reasoning can be localized, and will scale.
– Event models can be used to give semantics to specifications, as well and to support runtime observation/monitoring adaptation, security decisions, trust building. They support new programming abstractions that deal with actions and interactions rather than state transformations.
– Events may have associated evidence, for example the sensor generating low level events, the algorithm used to extract information (colors, shapes, faces, sound patterns, . . .), rules used to infer higher level events from lower level events (location of a person, end of lecture), or human input.

2.3 Challenge Areas

There are many challenges to realizing the full promise of event-based semantics. The overall challenge is to develop general mathematical models together with domain specific refinements that are both natural and expressive. Beyond models its is crucial to develop logics and reasoning principles. Finally these models and logics need tools to make them usable for analysis, synthesis, and transformations. Below we discuss some of things an appropriate event model should capture and some of the issues that will be faced in doing so.

Identifying, Modeling and Reasoning About Interdependencies. It is critical to enable increasing dynamic (computer) control that is safe and without unpleasant surprises. Consider for example, a power grid versus a transportation system. One the one hand, some elements of the transportation system depend on the power grid: trolly cars, fuel pumps, logistics planning. On the other hand changes in the transportation system may affect loads on the power grid. How can event partial orders combined with event timelines enable effective modeling and analysis? For example, an event based model could express dynamic consequences of dependencies, not just static relations.

Dealing with Time, Space, Scale, and Uncertainty. Time resolution of observation and actions. Events must be communicated in time to be useful. An example is sub second control on a power grid. Local event and control models may change over time. One example is different aircraft flight modes—takeoff, landing, or cruising. Another example is traffic control for high density landing of aircraft, weather and traffic patterns can change things substantially. Thus we must be able to model and reason about changes of event structures over time.

Systems operate at multiple time scales, for example realtime control of a single device in the context of scheduling of train/air traffic or coordination of human activity. An event-based semantics is needed that supports reasoning at each scale and integrating multiple scales.

Event hierarchies and rules for deriving high level events from lower level evidence or refining high level events to lower level events such as actions are needed.

Privacy Issues for Human Centric CPS. Access to dynamic data raises new issues as patterns over time, together with context, can enable unexpected inference of information. An example: the water company may monitor patterns of water usage, for the purpose of optimizing flow control. Maybe be able to detect activity such as shower, toilet flush, running a dishwasher. If the electricity company and the water company shared information, it might be possible to infer more refined differences. Electricity patterns for a dishwasher might be different than those for a washing machine. Clearly there is the possibility of invasiveness if such inferences are made (and exposed) .

Notice that the dynamic data can be modeled as event streams and transformations on event streams can be used to control what information is exposed. Much work developing formal 'threat' models (knowledge context and ability of entity accessing data) is needed to realize this possibility.

Composition Composing is not just putting things together in parallel! It is also necessary to provide a means of interaction, and a means of constraining possible interactions. For example,

- What network and communication protocols are needed to enable interacting with physical systems?
- What coordination primitives are needed to describe event-based compositions that involve physical systems?
- What properties of the components and their composition are important?

Incomplete specifications are often more elegant and easier for a designer or implementor to work with, but they are generally not composable—as the missing information leaves open the possibility of interference or unexpected combined behaviors. Arguments for composition properties typically assume all events are known, while in a given event model some information will be implicit in the model. Assume/guarantee formalizations can help, but when composing using multiple models it will be crucial to make explicit all information relevant to the composition. A possible approach to cross model composition is to develop meta-models that make explicit model assumptions.

Another aspect of composition is composing evidence—proofs, statistical confidence levels, trust. For example, low level events or event streams may be combined an abstracted to infer higher level events. Event models are a good basis for thinking about situation awareness, and it may be important to know how and event was detected, before taking action.

New Models for Thinking About Things Top to Bottom. Currently embedded systems have their control loop in the hardware or a realtime operating system. This does not scale and does not work well in open systems. How can thinking in terms of events lead to better models?

New Languages Based on New Models of Computation and Interaction. Languages are needed for event-based requirements, executable specification, composing, monitoring, and even programming. Considerations include

- the ability to change the way instructions/descriptions are interpreted
- scoping visibility and effects of actions
- containing effects of errors or unexpected events—both physical and cyber
- programming concepts with resource sensitivity built in
- what can be monitored, detected and/or controlled?

3 Related Work

In the following we review a sampling of work related to event-based semantics.

Events Vs. State, Partial Order Vs. Interleaving. In [26,25] an argument is presented that the traditional computer science model of concurrent programming using state-based models and threads incurs unnecessary complexity and results in code that is difficult to debug. For sequential computation and function composition they work nicely. When deterministic sequential threads are composed in parallel they become non-deterministic and difficult to manage. A tag based signal model is proposed building on [24]. The domain of tags comes equipped with an ordering relation, events are tag-value pairs, and signals are sets of events describing incremental evolution of a system. Components modeled in terms of signals compose naturally. The model is elaborated to model both components and connectors, thus capturing interactions and also introducing the possibility of feedback loops. A mathematical theory based on topological concepts has been developed to give a compositional semantics to the components-connector wiring diagrams [27].

In [9] Clinger proves the existence of global times for event diagrams (a form of event partial orders) corresponding to possible interleavings. This provides an associated interleaving model allowing one to reason sequentially or about partial orders.

Rewriting logic [28,30] extends equational logic with local rewrite rules that model change over time. Proofs in rewriting logic can also be thought of as computations. Since rules are applied locally, a computation step may involve multiple parallel rewrites, while an equivalent computation carries these out one step at a time. In [29] it was shown that in a restricted class of rewrite theories modeling object / actor systems, there is an isomorphism between equivalence classes of proofs/computations and the event partial order generated by the computation.

In [36] an abstract interpretation of time is proposed to model systems involving preemptive scheduling. In this approach, models of the individual process are composed into a single time domain with the result being an infinite state timed automaton called a time domain automaton. Each state of such an automaton represents an equivalence class of all possible execution interleavings that result in that particular event ordering. Abstract interpretation combined with constraint solving techniques are used to make the model amenable to analysis.

Event Models for Actors. The actor model [20,2,1] is a model of concurrent and distributed computation based on asynchronous message passing. Actors are reactive entities that encapsulate state and control and interact with other actors only by sending and receiving messages. Events are the basis of semantics of actor languages and systems. Grief [18] introduced the notion of *event diagram* which captures the linear order of events at each actor and the causal order between message sends and corresponding receives. Baker and Hewitt [5] proposed a set of laws characterizing these event partial orders. In [33] a compositional notion of Actor Algebra is developed. Actor Algebra models include interfaces, specifications, event diagrams, and interaction paths with mappings between the different algebras.

PastTime Distributed Temporal Logic (PtDTL). PtDTL, a variant of PastTime Temporal logic was introduced in [32]. This logic reasons not over interleavings and linear sequences of past states, but over partially ordered sets of events causally in the past. As for event diagrams, events are located and the logic introduces epistemic operators that allow reasoning about what holds at the most recent causally previous state of another actor or process. The logic is used as the basis of an efficient algorithm for distributed monitoring.

Causal Logic of Events. Causal Logic of Events (CLE) [8,6,7] is a logic for distributed computing that has the explanatory and technical power of constructive logics of computation. CLE provides a proof technology that supports correct-by-construction programming based on the notion that concurrent processes can be extracted from proofs that specifications are achievable. A methodology for specifying distributed systems in CLE has been developed and implemented in NuPrl [3]. Requirements for a distributed system are expressed in terms of events, these requirements are then refined to collections of constraints called Message Automata (MAs) that imply the original requirements. MAs can be compiled to standard languages such as Java. Models of message automata are event diagrams, with events localized and the event order at each location a total order. Working bottom up, system properties can be inferred from MAs. Event classes and laws for composition allow specification and reasoning at a higher level of abstraction. The logical framework also supports timing properties, for example using variables that are trajectories of values rather than discrete values. The methodology has been applied to a variety of networking and security protocols.

Strand Spaces as an Event Model for Security and Location. Key exchange has logically simple goals, agnostic to communication concerns. In contrast, location protocols have quantitative goals, and models must consider transmission properties and use geometry. Strand spaces are a mathematical model that provides a special-purpose execution semantics, called Bundles, based on a causal partial order, that is complete for symbolic analysis of key exchange [16]. Reasoning about properties such as authentication or confidentiality makes combined use of causality and cryptographic properties. Strand spaces have been used to model and analyze a variety of security related protocols, including key exchange, contract negotiations, and secure payments systems.

Metric strand spaces are introduced in order to also reason about space and time. Bundles in a metric strand space have a distance and time elapse measures on some pairs of events that obey axioms reflecting by a model of transmission speed [19]. Secure location protocols combine cryptography with the physics of message transmission. The cryptographic operations authenticate the principals and preserve confidentiality, while the physics of message transmission constrain their possible locations. In the case of metric strand spaces, the strand space model is enriched by associating a space-time location with each node. The strands follow the world lines of principals. Some bundles are compatible with the physics of message transmission – e.g. the maximum message transmission speed – while others are not. An assertion true in every bundle compatible with the physics is a valid conclusion of a secure location protocol.

CEL and strand spaces are similar in a number of ways. They share notions of causal order and the need to express limitations on the adversary. Strand spaces have a notion of unique origination that is similar to the CEL notion of nonce. The two formalism differ in their treatment of (logical) locality: in CEL locality encompasses multiple activities of a single principle or actor and may allow sharing of information across multiple threads/activities. In contrast the Strand space model explicitly isolates each activity of a principal (a strand) enforcing further localization.

Event Streams and Uncertainty. Event-based semantics is natural for many real-time embedded applications. In such applications the issue of temporal uncertainty is a common and challenging problem. Temporal uncertainty comes from the inherent restrictions in the underlying sensing layer, such as the temporal/spatial limitations of sampling, inaccurate clocks and unpredictable network latency. Although events occur instantaneously (dense time) in the physical world, an event occurrence often cannot be assigned a precise time owing to the above limitations. PTMON (Probabilistic Timing MONitor) [42,41] is a generic framework for incorporating various uncertainty models on event time stamps, developed in the context of monitoring timing constraints on event streams. A monitor task is formulated by timing constraints in a simple real-time temporal logic and satisfaction/violation of these formulas is checked at run time. Given a probability model of the temporal distance from event occurrence to event detection, timing constraints based on event occurrences can be transformed to those based on event detection. This transformation enables the early detection of timing constraint violations. Applications include real-time baggage tracking, wireless process control, remote monitoring, online multimedia downloading, and teleconference.

Grounding High-Level Event Definition. High-level event-based models require a precise notion of events in the model. A formal way of defining high-level events in terms of low-level system state helps to define a faithful abstraction for an event-based model. The logic of events and conditions (LEC) [22] is a two-sorted logic bridging the gap between state-based formalisms, commonly found in low-level models, and higher-level event formalisms. Conditions represent an abstract view of the system state, with primitive conditions being state predicates over the observable state variables in the system. Primitive events can also be directly observed during a system run. The use of LEC for event definition was developed in the context of run-time verification. The same separation of concerns used in run-time verification can be applied to high-level modeling in general. The basic approach is to provide an event-based model that uses high-level events as atomic building blocks, reducing the size and complexity of the model. The event definition layer provides grounding of the high-level model in the implicit low-level model. It can be used to establish a mapping between behaviors of the high-level and low-level models, which can be used to demonstrate, without ever constructing the low-level model explicitly, that the high-level model is a faithful representation of the system.

Event Models for Pervasive Spaces. The Responsphere Infrastructure is a campus-level pervasive computing and communication infrastructure at University of California at Irvine (http://www.responsphere.org). It consists of a variety of sensors (video cameras, sensor mounted mobile robots, people counters, RFID, acoustic sensors, thermal and gas sensors) dispersed over approximately a third of the campus, connected via a variety of network and communication technologies (802.11, cellular, mesh, and power-line networks). It includes dense sensing in a few chosen buildings where it monitors all corridors, entries, exits, and public areas using cameras. In addition, some designated public spaces and laboratories are instrumented with RFID readers. Responsphere also includes mobile sensor mounted robots with communication capabilities that can be programmed for autonomous data collection. Responsphere serves as a test bed for developing a variety of pervasive functionalities, for example, using a mixture of video and RFID technologies to implement social policies of a shared common facility within a particular building. Examples include reminding people to switch off the coffee machine and conducting social experiments to study recycling behavior. In addition Responsphere has been used to conduct and monitor a variety of emergency drills such as building and region evacuations. The SATware System [21] (http://satware.ics.uci.edu) is a scalable middleware, that runs on Responsphere and provides seamless access to sensor and event level data. Applications access this information via a SQL style query language referred to as SATQL, at both the physical (e.g., raw sensor feeds) and semantic levels (i.e., at the level of entities, activities, and events). The key concept is that of a *virtual sensor* that empowers programmers to define and detect semantic concepts thereby realizing information abstraction. Virtual sensors are mapped (at run-time) to a graph of operators which are implemented over physical sensor streams. Challenges for management and programming of pervasive spaces include privacy and trustworthiness, evidence for judging semantic event reports, trading function for privacy, and self monitoring and adaptation.

4 Policy- and Goal-Based Operation of Autonomous Agents

There is a growing interest in autonomous agents that interact with and affect their environment, and have some ability to observe, reason, and adapt. As part of a larger system agents should also be able to compete for resources but also to cooperate for mutual benefit or to achieve an overall goal.

PAGODA (Policy And GOal based Distributed Autonomy) is a modular architecture for design of interactive autonomous systems. A PAGODA system is a collection of PAGODA nodes cooperating to achieve some mutual goal. A PAGODA node (agent) interacts with its environment by sensing and affecting, driven by goals to achieve and constrained by policies. The PAGODA architecture was inspired by studying architectures developed for autonomous space systems, especially the MDS architecture [15] and its precursors [31]. Software for deep space missions must be

– autonomous—operating remotely for extended time
– robust—operating under unpredictable conditions
– dependable—mission failure is costly

In PAGODA policy-based coordination is used at two levels: local modular combination of components making up an agents behavior; and coordination of a distributed system of agents constraining the possible interaction scenarios to meet end-to-end requirements.

The long term objective of the PAGODA project is to develop techniques for specification and analysis that take advantage of the modularity and the declarative nature of policy- and goal-based systems. PAGODA has been developed in the context of projects providing driving applications, including a rover (for example for exploration or patrol) [13,14] and software defined radios [40] supporting specific missions. Other potential applications include reactive/adaptive planners, cognitive radios, software assistants, and self-configuring systems.

Our approach is based on the Reflective Russian Dolls (RRD) model of distributed object reflection [29,34] which in turn is founded on the rewriting logic formal modeling framework [28,30]. In [34] a general approach to modeling policy-based coordination using RRD was presented. The question addressed by PAGODA is how to specify autonomous behavior that meets or achieves its goals (subject to constraints on external conditions) in a modular and declarative manner using models of its environment. Our solution is to factor the behavior into components, each with a specific role, that combine to achieve the desired result.

4.1 PAGODA Nodes

Figure 1 shows the principal components of a PAGODA node: a knowledge base (KB), a reasoner (R), a monitor (M), a learner (L), and a 'hardware' abstraction layer (HAL). These interact with each other and the environment under the control of a coordinator (C).

The knowledgebase (KB) is the centerpiece. It contains knowledge that is shared and updated by the remaining components. This knowledge includes a wide range of information:

– *Goals* that specify what the node or system is trying to achieve. A goal could be a very high-level goal such as carrying out a scientific experiment or tuning parameters to achieve a given quality of service; or lower level goals that correspond to actions that can be carried out.
– *Policies* that constrain the allowed actions / interactions of a node or system. A policy might reduce the number of choices for setting parameters, for example based on importance of different competing effects. Another policy might determine trade-offs between speed and power usage. Other policies might control aggregation and abstraction of information used locally or communicated to other agents.
– A *device model* that specifies the HAL interface: parameters/knob that can be set (effecting) and read (sensing) and their relationships. At the system level the model

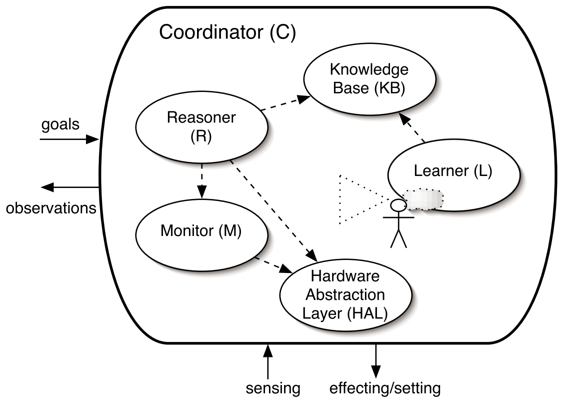

Fig. 1. PAGODA node architecture

should also specify how values sensed at different nodes can be combined to determine non-local system properties, and the relationships of such properties to higher level goals.

- An *environment model*, representing relevant features of the environment in which the node is operating, including information about other nodes. For a mobile node this could include terrain information or building maps.
- *Node state*, which includes values of variables determined by sensor readings and deduced from actions and information collected from other nodes. It also includes 'situation' information such as the stage in a complex task/mission or progress towards achieving a goal.
- *History*, a log of events–goals received, knob settings and sensor readings, monitor alerts, and so on.

The job of the *reasoner* component (R) is to determine proper parameter settings in response to goals requests: new goals, starting a new stage of a current goal, or alerts raised due to unexpected sensor values, indicating that adjustments need to be made. The reasoner uses information from the KB as a basis for its deductions: the device and environment model, the goals and policies, and the current state. When new parameter settings are determined, the reasoner also provides justifications such as what sensor values and/or what relationships from the device model were used to infer the new settings. This can be used for diagnostics if things don't go as expected. The reasoner also specifies sensors that should be monitored and conditions on sensor reading under which the reasoner to be alerted to take corrective action.

The *monitor* component (M) receives monitoring tasks from the reasoner, reads and evaluates specified sensors, and sends alerts to the reasoner when sensor readings are not within specified limits.

The job of the *learner* component (L) is to improve the model used by the reasoner to infer appropriate knob settings. In passive mode it observes events such as goals, settings, sensor readings and alerts and attempts to improve relationships specified by the model based on this information. A learner may also have an active mode where it is allowed to propose experimental settings and observe the results.

The *hardware abstraction layer component* (HAL) is an interface to the sensors and effectors used by the node. It plays the role of device driver, handling knob setting and sensor reading requests. In a real system the HAL might map requests to a format that is understood by the actual hardware, or even to a lower level abstraction layer. The intent is that these interactions should obey the 'physics' specified by the device model, but the node needs to be prepared for things to go wrong—some hardware component breaks, the environment is different than expected, it is being operated outside the expected operational mode, and so on.

The coordinator (C) controls message semantics for internal components and mediates interactions with the external world. The coordinator is responsible for ensuring specified relationships between the events (message deliveries) seen by different components, and for meeting logging and notification requirements. It also enforces component level synchronization constraints (only delivering messages for which the component is enabled). The coordinator actions are specified declaratively by policies. Note that coordinator policies are similar in spirit, to policies used by the reasoner, but different in detail.

Each PAGODA component type has an interface specified using events. The semantics of given component is an event-based semantics in the spirit of the Actor Algebra discussed in Section 2. This enables event-based composition of components and their semantics. Composition with a coordinator can be treated as a vertical composition in the spirit of [12].

This architecture provides a simple means of plugging in different component instances. PAGODA node components interact with other node components based on component type not on component instance identity. Thus it is easy to have multiple reasoners, knowledge bases, learners, etc., by simply modifying the coordinator policy to choose appropriate component instances. Different reasoners might be appropriate for different situations or goals, knowledge might be split into categories and stored in separate KB instances, or two KB instances might contain knowledge at different levels of abstraction appropriate for different situations.

Additional components types could be easily incorporated. For example a component capable of knowledge abstraction or aggregation could be invoked from time to time by the coordinator to infer higher-level information from sensor data or information received from peers. Such a component could be used to raise the level of abstraction at which the reasoner or learner operates.

5 Interactive Agents

Much has been written contrasting interactive computation and other models such as Turing machines and logic programming [37,38,39]. Our focus is on modeling and reasoning about the capabilities enabled by interactivity.

An interactive agent must be aware of its surroundings, and it may also affect its environment. It may need to negotiate, cooperate, or compete. A formal framework for modeling *interactive agents* was introduced in [35]. The framework was based on the need to consider the following features in the design of interactive agents.

- An agent has a boundary consisting of points of interaction with the environment. From the outside only what crosses the boundary is visible. Interaction points could be sensors, such as light detectors or thermometers, effectors such as switches or dials, or message queues for exchange of messages with other agents.
- An agent has actions that it can execute. It may also have goals, knowledge (about its environment and itself), policies constraining actions, or strategies for achieving goals.
- Internally an agent may have multiple concurrent activities; observing and processing sensory information; refining goals to subgoals, choosing actions, executing actions; evaluating and analyzing results: did actions have expected effect? updating knowledge by learning and inference;
- Interactivity means internal processes must be interruptible.

The framework is based on rewriting logic and a reflective model of coordination for managing an agents activities. New forms of interaction are introduced to model both message and channel/signal based interactions, and to pave the way for modeling continuous interactions. The compositional interaction semantics of [33,12] is extended to handle the new forms of interaction. The aims of the framework include:

- a higher level means of specifying and understanding agent behavior
- a place to classify agents with different 'skills'
- a formal design space to represent a variety of design decisions and to study tradeoffs resulting from decisions such as adaptability vs. predictability;

One advantage of the proposed framework is that specifications are executable, allowing prototyping of designs at many stages. In addition, such specifications are formally analyzable using the Maude rewriting logic system, and connections with other formal systems.

Briefly, interactive agents are formalized as actor like objects with rules for communication by messages, interaction through interface points, and policies for coordinating activities, also represented as (sub)agents. What the agent reads at an interaction point is controlled by the environment. What the environment can read is controlled by the agent (by a write action).

An *interaction path* is a (possibly infinite) sequence of interactions (events as viewed from an imaginary external observer). Each computation of an agent (allowed by the rewrite rules) gives rise to a set of interaction paths consisting of the (non-silent) interactions labeling the transitions (rewrite rule applications). The observable semantics of an interactive agent is thus the set of interaction paths of its possible computations. This definition derives from earlier work developing interaction semantics for actors [33,12], ideas from Timed Data Stream semantics for the Reo coordination model [4], and signal event semantics [23]. Interaction semantics is similar in spirit to the Interactive Stream Languages of [17]. The ideas are also related to work on *interfaces* of reactive and concurrent systems such as, [10,11].

Interaction semantics is compositional both vertically and horizontally. The semantics of the horizontal (parallel) composition of two systems is done by zipping compatible paths, one from the semantics of each system. Two paths are compatible if their subsequences of complimentary interactions, such as out/write in one and in/read in the other match. In the composed path, these interactions become silent transitions and disappear, and the remaining interactions are merged. (See [33] for details in the case of horizontal, actor-actor composition, and [12] for vertical, actor-metaactor, composition.)

6 Conclusion

Cyber physical systems (CPSs) are an emerging phenomena. These systems are often not only software intensive, but also are tightly integrated with physical system, leading to many new challenges for design and development. We have proposed event-based semantics as a semantic foundation for Cyber physical Systems. We discussed a variety of notions of event, essential features and challenges for developing event-based semantics for CPSs. We also sketched two compositional models, one for autonomous agents and one for interactive agents. The latter providing forms of interaction such as needed in CPSs.

References

1. Agha, G.: Actors: A Model of Concurrent Computation in Distributed Systems. MIT Press, Cambridge (1986)
2. Agha, G.: Concurrent object-oriented programming. Communications of the ACM 33(9), 125–141 (1990)
3. Allen, S., Constable, R., Eaton, R., Kreitz, C., Lorigo, L.: The Nuprl open logical environment. In: McAllester, D. (ed.) CADE 2000. LNCS (LNAI), vol. 1831, pp. 170–176. Springer, Heidelberg (2000)
4. Arbab, F., Rutten, J.J.M.M.: A coinductive calculus of component connectors. In: Wirsing, M., Pattinson, D., Hennicker, R. (eds.) WADT 2003. LNCS, vol. 2755, pp. 34–55. Springer, Heidelberg (2003)
5. Baker, H.G., Hewitt, C.: Laws for communicating parallel processes. In: IFIP Congress, pp. 987–992 (August 1977)
6. Bickford, M.: Specification and derivation of distributed programs using a logic of events (2007), Invited lecture for Worshop on Event-based Semantics 2007, `http://blackforest.stanford.edu/eventsemantics`
7. Bickford, M.: Abstract sequential programs and a logic of events (2008), Invited lecture for Worshop on Event-based Semantics (2008), `http://blackforest.stanford.edu/eventsemantics`
8. Bickford, M., Constable, R.L.: A causal logic of events in formalized computational type theory. Technical Report Technical Report 2005-2010, Cornell University (2005)
9. Clinger, W.D.: Foundations of actor semantics. AI-TR- 633, MIT Artificial Intelligence Laboratory (May 1981)
10. de Alfaro, L., Henzinger, T.A.: Interface automata. In: Ninth Annual Symposium on Foundations of Software Engineering (FSE), pp. 109–120. ACM Press, New York (2001)
11. de Alfaro, L., Henzinger, T.A.: Interface theories for component-based design. In: Henzinger, T.A., Kirsch, C.M. (eds.) EMSOFT 2001. LNCS, vol. 2211, p. 148. Springer, Heidelberg (2001)
12. Denker, G., Meseguer, J., Talcott, C.L.: Rewriting semantics of distributed meta objects and composable communication services. In: Third International Workshop on Rewriting Logic and Its Applications (WRLA 2000). Electronic Notes in Theoretical Computer Science, vol. 36. Elsevier, Amsterdam (2000)

13. Denker, G., Talcott, C.L.: Formal checklists for remote agent dependability. In: Fifth International Workshop on Rewriting Logic and Its Applications (WRLA 2004). Electronic Notes in Theoretical Computer Science. Elsevier, Amsterdam (2004)
14. Denker, G., Talcott, C.L.: A formal framework for goal net analysis. In: Workshop on Verification and Validation of Planning Systems. AAAI Press, Menlo Park (2005)
15. Dvorak, D., Rasmussen, R., Reeves, G., Sacks, A.: Software Architecture Themes In JPL's Mission Data System. In: IEEE Aerospace Conference, USA (2000)
16. Fábrega, F.J.T., Herzog, J.C., Guttman, J.D.: Strand spaces: Proving cryptographic protocols correct. Journal of Computer Security, 191–230 (1999)
17. Goldin, D., Smolka, S., Attie, P., Sonderegger, E.: Turing machines, transition systems, and interaction. Information and Computation Journal 194(2), 101–128 (2004)
18. Greif, I.: Semantics of communicating parallel processes. Technical Report 154, MIT, Project MAC (1975)
19. Guttman, J.: Strand spaces: From key exchange to secure location, Invited lecture for Worshop on Event-based Semantics 2008 (2008), `http://blackforest.stanford.edu/eventsemantics`
20. Hewitt, C., Bishop, P., Steiger, R.: A universal modular actor formalism for artificial intelligence. In: Proceedings of 1973 International Joint Conference on Artificial Intelligence, pp. 235–245 (August 1973)
21. Hore, B., Jafarpour, H., Jain, R., Ji, S., Massaguer, D., Mehrotra, S., Venkatasubramanian, N., Westermann, U.: Design and implementation of a middleware for sentient spaces. In: Proceedings of ISI 2007 (2007)
22. Kim, M., Kannan, S., Lee, I., Sokolsky, O., Viswanathan, M.: Java-MaC: A run-time assurance approach for Java programs. Formal Methods in Systems Design 24(2), 129–155 (2004)
23. Lee, E.A.: Concurrent semantics without the notions of state or state transitions. In: Formal Modeling and Analysis of Timed Systems. LNCS, pp. 18–31. Springer, Heidelberg (2006)
24. Lee, E.A., Sangiovanni-Vincentelli, A.: A framework for comparing models of computation. IEEE Transactions on Computer-Aided Design of Circuits and Systems 17(12), 1217–1229 (1998)
25. Lee, E.A.: Concurrent semantics without the notions of state or state transitions. In: Formal Modeling and Analysis of Timed Systems. LNCS, pp. 18–31 (2006)
26. Lee, E.A.: The problem with threads. IEEE Computer 39(5), 33–42 (2006)
27. Liu, X., Matsikoudis, E., Lee, E.A.: Modeling timed concurrent systems. In: CONCUR (2006)
28. Meseguer, J.: Conditional Rewriting Logic as a unified model of concurrency. Theoretical Computer Science 96(1), 73–155 (1992)
29. Meseguer, J., Talcott, C.L.: Semantic models for distributed object reflection (invited paper). In: Magnusson, B. (ed.) ECOOP 2002. LNCS, vol. 2374, pp. 1–36. Springer, Heidelberg (2002)
30. Meseguer, J.: A rewriting logic sampler. In: Van Hung, D., Wirsing, M. (eds.) ICTAC 2005. LNCS, vol. 3722, pp. 1–28. Springer, Heidelberg (2005)
31. Muscetolla, N., Pandurang, P., Pell, B., Williams, B.: Remote Agent: To Boldly Go Where No AI System Has Gone Before. Artificial Intelligence 103(1–2), 5–48 (1998)
32. Sen, K., Vardhan, A., Agha, G., Rosu, G.: Efficient decentralized monitoring of safety in distributed systems. In: ACM TOSEM (submitted, 2006) (invited papers)
33. Talcott, C.L.: Composable semantic models for actor theories. Higher-Order and Symbolic Computation 11(3), 281–343 (1998)
34. Talcott, C.: Coordination models based on a formal model of distributed object reflection. In: 1st International Workshop on Methods and Tools for Coordinating Concurrent, Distributed and Mobile Systems (MTCoord 2005) (2005)

35. Talcott, C.: A formal framework for interactive agents. In: Arbab, F., Golden, D. (eds.) Foundations of Interactive Computation (FInCo 2007). Electronic Notes in Theoretical Computer Science, vol. 203, pp. 95–106. Elsevier, Amsterdam (2007)
36. Tidwell, T., Gill, C.: Abstract interpretation of time for preemptive scheduling of cyber-physical systems. Position paper for Worshop on Event-based Semantics 2007 (2007), `http://blackforest.stanford.edu/eventsemantics`
37. Wegner, P.: Why interaction is more powerful than algorithms. CACM (May 1997)
38. Wegner, P., Goldin, D.: Computation beyond turing machines. CACM (April 2003)
39. Wegner, P., Goldin, D.: The church-turing thesis: Breaking the myth. In: Cooper, S.B., Löwe, B., Torenvliet, L. (eds.) CiE 2005. LNCS, vol. 3526, pp. 152–168. Springer, Heidelberg (2005)
40. Wirsing, M., Denker, G., Talcott, C., Poggio, A., Briesemeister, L.: A rewriting logic framework for soft constraints. In: WRLA 2006 (submitted, 2006)
41. Woo, H., Mok, A.K., Chen, D.: Realizing the potential of monitoring uncertain event streams in real-time embedded applications. Position paper for Worshop on Event-based Semantics 2007 (2007), `http://blackforest.stanford.edu/eventsemantics`
42. Woo, H., Mok, A.K., Lee, C.-G.: A generic framework for monitoring timing constraints over uncertain events. In: 27th IEEE International Real-Time Systems Symposium (2006)