Logic-Based Modeling of Information Transfer in Cyber-Physical Multi-Agent Systems

Christian Kroiß[1](✉) and Tomáš Bureš[2]

[1] Ludwig Maximilian University of Munich,
Institute for Informatics, Munich, Germany
kroiss@pst.ifi.lmu.de

[2] Charles University in Prague,
Faculty of Mathematics and Physics, Prague, Czech Republic
bures@d3s.mff.cuni.cz

Abstract. In modeling multi-agent systems, the structure of their communication is typically one of the most important aspects, especially for systems that strive toward self-organization or collaborative adaptation. Traditionally, such structures have often been described using logic-based approaches as they provide a formal foundation for many verification methods. However, these formalisms are typically not well suited to reflect the stochastic nature of communication in a cyber-physical setting. In particular, their level of abstraction is either too high to provide sufficient accuracy or too low to be practicable in more complex models. Therefore, we propose an extension of the logic-based modeling language SALMA, which we have introduced recently, that provides adequate high-level constructs for communication and data propagation, explicitly taking into account stochastic delays and errors. In combination with SALMA's tool support for simulation and statistical model checking, this creates a pragmatic approach for verification and validation of cyber-physical multi-agent systems.

Keywords: Statistical model checking · Cyber-physical systems · Situation calculus · Discrete event simulation

1 Introduction

With SALMA (Simulation and Analysis of Logic-Based Multi-Agent Systems) [2], we have recently introduced an approach for modeling and analysis of multi-agent systems that is aimed to provide a lightweight solution for approximated verification through *statistical model checking* [4] with the system model still being grounded on a rigorous formal foundation. SALMA's modeling language is based on the well-established *situation calculus* [7], a first-order logic language for describing dynamical systems.

In this paper, we provide an extension of SALMA (and the situation calculus in general) to explicitly address one aspect that is particularly important for *cyber-physical* multi-agent systems, namely the distributed gathering and

P.C. Vinh et al. (Eds.): ICTCC 2014, LNICST 144, pp. 42–52, 2015.
DOI: 10.1007/978-3-319-15392-6_5

transfer of information. Agents not only have to continuously sense their environment, but also share these readings with other agents, acquire information of others, and participate in coordination activities. In the cyber-physical context, these information transfer processes are subject to stochastic effects, e.g. due to sensor errors or unreliable communication channels. Furthermore, accuracy and timing of information transfer processes can strongly influence the behavior of the whole system. In particular, the efficacy of mechanisms for self-adaptation or optimization typically degrades when certain time-constraints are violated or the accuracy of sensors is insufficient.

Using pure logical formalisms like the basic situation calculus for describing such scenarios results in rather verbose and low-level representations that are not practicable in more complex cases. What is needed instead are high-level constructs that establish a bridge between the underlying logical semantics and the typical requirements for modeling information transfer in multi-agent CPS. Although higher-level extensions on top of the situation calculus have been designed for related aspects like sensing and knowledge (e.g. [9]), there has, to our knowledge, not been a detailed reflection of information propagation in CPS in the context of the situation calculus.

We have therefore developed a generic model of information transfer that is based on a stochastic timed version of the situation calculus and allows capturing a wide range of effects that may be imposed on information transfer processes. Additionally, we have defined a set of macro-like abstractions for common information transfer scenarios within CPS, such as message passing or sensor data propagation. This creates a concise interface for the modeler that hides the stochastic details of information propagation but makes them fully accessible in simulation and verification. The following sections introduce both the generic model and the high-level language and demonstrate their use by means of an example.

2 Example: Optimized Parking Lot Assignment

As a running example to illustrate our approach, we employ the e-mobility case-study of the ASCENS EU project[1] that has been described before, e.g., in [1]. The case study focuses on a scenario in which electric vehicles compete for parking lots with integrated charging stations (PLCS) in an urban area. The goal is to find an optimal assignment of PLCS to vehicles. Technically, the assignment is performed by an agent called super-autonomic manager (SAM) that coordinates a number of PLCS. The basic idea is that vehicles send *assignment requests* to the SAM, including a start time, a duration, and a list of preferred PLCS that is compiled by the vehicle's on-board computer. The SAM tries to find optimal suggestions for parking lot assignments, based on the knowledge about driver's intentions, and on occupancy information that is sent repeatedly by the PLCS.

[1] www.ascens-ist.eu

True to the distributed CPS principle, all the agents (vehicles, PLCS, SAM) are autonomous and communicate via some wireless data transmission infrastructure like a VANET or 3G/4G network. This implies that neither transmission delays nor the possibility of errors can be neglected. However, timing clearly plays an important role in the scenario described above. First of all, the reservation service would simply not be accepted if the delay between reservation requests and reservation responses was too high. Also, the communication timing affects the convergence of the optimization, thus directly it influences the functionality of the distributed CPS.

3 Background: Situation Calculus

The situation calculus [7] is a first-order logic language for modeling dynamic systems. Its foundation is based on the notion of *situations*, which can be seen as histories of the world resulting from performing *action sequences*. The state of the world in a given situation is defined by the set of all *fluents*, which are situation-dependent predicates or functions. Since the models discussed here are meant to be used in *discrete event simulation*, time itself is simply modeled as an integer fluent named *time* that is increased with each simulation step. How other fluents are affected by *actions* and *events* is defined by *successor state axioms (SSAs)*. Additionally, a situation calculus model also contains *precondition axioms* that define whether or not an action or event is possible in a given situation. Actions can either be deliberately executed by agents or *exogenous*, i.e. external events caused by the environment. In general, both the effects of actions and events, and also the occurrence of exogenous actions, are of stochastic nature. Consequently, simulation involves sampling from a set of probability distributions that the modeler can define as part of the simulation's configuration (cf. section 6).

One of the most prominent applications of the situation calculus is GOLOG [5], a language that combines elements from procedural with logic programming. It has been used for modeling and implementation in various domains, ranging from robotics to the semantic web. In particular, GOLOG's core principles have strongly inspired the SALMA approach, which is introduced in the next section.

4 The SALMA Approach

In [2], we introduced SALMA (Simulation and Analysis of Logic-Based Multi-Agent Systems), an approach that adapts the concepts of the situation calculus and GOLOG for discrete event simulation and *statistical model checking*. The approach is outlined in Figure 1. The *domain* model, i.e. the general mechanisms of the simulated world, is described by means of situation calculus axioms that are encoded in Prolog. Based on this axiomatization, the modeler defines the behavior of the agents in the system with a Python-based procedural language that provides access to the situation calculus model. Finally, a concrete simulation model instance is created by defining initial values and *probability distributions* for stochastic actions and events.

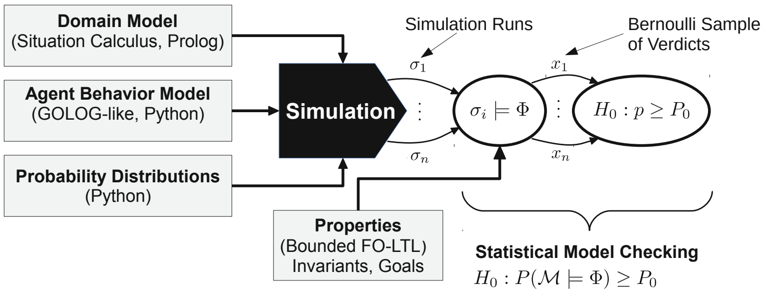

Fig. 1. Overview of the SALMA Approach

In addition to the system model, a set of invariants and goals can be specified with a language that is mainly a first-order version of linear temporal logics (LTL) with time-bounds for the temporal modalities. Since the simulated system model is also described by means of first-order logics, the property specification language is able to provide a very detailed and direct access to the system's state (i.e. fluents), actions, and events.

Given the system model together with invariants and goals, the SALMA interpreter performs *discrete event simulations*. For each simulation run, the engine eventually decides whether it satisfies the given properties or not. The set of resulting verdicts yields a *Bernoulli sample* that is used to test the statistical hypothesis $H_0 : p \geq P_0$ which asserts that the probability of a success (a run fulfills the property) is at least as high as a given lower bound. By using the sequential probability ratio test (SPRT) by A. Wald [10], the number of required simulation runs for given statistical error bounds can be determined dynamically. This way of approximative assertion of properties defined by temporal logics is generally called *statistical model checking* [4] and provides a pragmatic alternative to exact model checking techniques that does not suffer from the same scalability problems since only individual simulation runs are inspected instead of the complete state space.

5 A Generic Model for Information Transfer

In order to use SALMA for analyzing scenarios like the one described in Section 2, concepts like sensing and communication have to be mapped to SALMA's modeling language framework. As a first step, we propose a generic model for information transfer in the situation calculus. This model is able to describe both sensing and inter-agent communication in a unified way and allows capturing stochastic effects with a variable level of detail.

In general, our approach is based on the notion that information is transferred from a *source fluent* to a *destination fluent* that is directly accessible by the receiving agent. The source fluent can either represent a *feature of the physical world* or data created by some artificial process, e.g. a message queue. A *connector* defines modalities of an information transfer process, including the

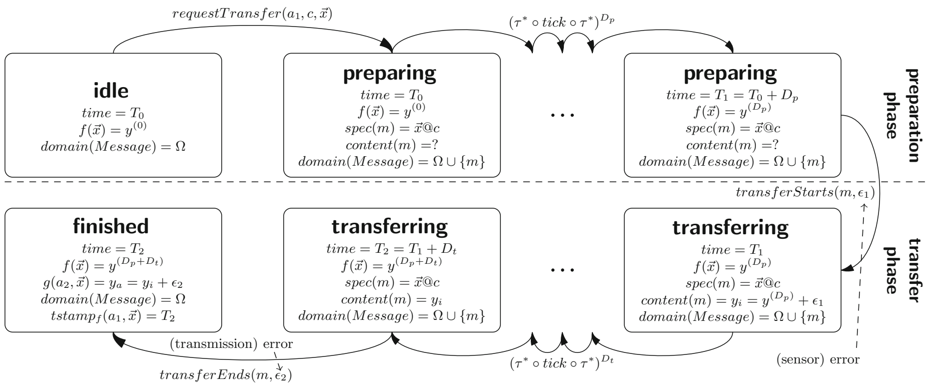

Fig. 2. General Information Transfer Model

fluent endpoints and the types and roles of participating agents. The *messages* that are transmitted over connectors are treated as first-level model citizens by representing them as entities of the dedicated sort $Message$. Both the content and the state of each message are stored separately by a set of fluents and evolve independently as result to several types of events. This representation provides great flexibility for the realization of arbitrary propagation structures. However, it requires that message entities can be created and removed dynamically as effect to actions and events. Unlike traditional realizations of the situation calculus, SALMA supports this by using a special (meta-)fluent $domain(sort)$ to store the sets of entities that manifest the current *domains* of all sorts in the model. Creation and destruction of entities can therefore be controlled through regular successor state axioms.

Based on the foundational concepts described above, we distinguish two phases of information transfer that are sketched in Figure 2:

a) The **preparation phase** starts when an agent (a_1 in Figure 2) executes a *transfer request action*, specifying a *connector* (c) and a parameter vector ($\vec{x}$) that fully qualifies the information source and, in case of a point-to-point transmission, contains the identity of the receiving agent. In response to this action, a new message (m) is created and initialized with the transfer metadata but without content yet. Depending on the concrete scenario, there can be various reasons for the actual transfer being delayed, e.g. initialization of sensors or communication devices. This means that there may be an arbitrary sequence of time steps (*tick* events) interleaved with actions and events (denoted as τ in Figure 2) that may change the information source (f) but are not recognized by the agent. After that sequence, the actual value that is eventually used as message content can deviate from the information source value present at the time when the transfer was initiated.

b) The **transfer phase** follows the preparation phase and begins when a $transferStarts$ event occurs. At that point, the current value of the source fluent f is fixated as the content of the message that is now actually transferred to its destination over the connector c whose stochastic characteristics are specified within the simulation model. Like above, this phase may take an arbitrary amount of time during which unrecognized or unrelated actions and events occur. Eventually, a $transferEnds$ event finishes the transfer process. Thereupon, the destination fluent instance $g(a_2, \vec{x})$ is updated and the message entity is removed. This moment, as well as the starting points of both phases, are memorized in timestamp fluents that can, for instance, be used to reason about the age of a measurement.

The diagram in Figure 2 omits the fact that, due to malfunctions and disturbances in the environment, the transfer could *fail* at any time, which would be represented by an additional event $transferFails$. Additionally, the transfer process may be affected by stochastic errors that eventually cause the received value to deviate from the original input, which is reflected by the error terms ϵ_1 and ϵ_2 in the events $transferStarts$ and $transferEnds$.

In general, both stochastic errors and delays are governed by a set of probability distributions that are used during simulation to decide when the events mentioned above occur and which errors they introduce. By adjusting these parameters, a wide variety of different scenarios can be modeled, ranging from nearly perfect local sensing to wireless low-energy communication with interferences. The simulation engine supports probabilistic sampling both in an *anticipatory* and in a *momentary* way. In the first case, a random value is sampled in advance to set the time for which the corresponding event will be scheduled. By contrast, in momentary sampling mode, the effective delay is generated by stochastically choosing in each time step, whether the event should occur or not. While the second approach obviously increases computational effort, it is typically better suited for capturing highly dynamic effects, e.g. when the position of a moving agent has significant impact on communication quality.

6 A High-Level Modeling Language for Information Transfer Processes

To turn the generic information transfer model to a practical solution for modeling real-size systems, we provide high-level constructs that reflect the way a modeler normally thinks about information transfer processes in a CPS. These constructs can be seen as macros that are internally mapped to situation calculus axioms, agent process fragments, and probability distributions. How the variation points of the generic model are resolved depends largely on the type of information transfer that is modeled. In particular, we distinguish the following two core concepts:

Channel-Based Communication: an agent actively sends data to one or several other agents. The well-known channel paradigm fits well to the asynchronous communication style predominant in CPS and to the relational way of identifying information in the situation calculus.

Gerneralized Sensing: an agent acquires information about a feature of the world that can be assessed through sensing. In the case of **direct (local)** sensing, the querying agent can produce the desired result on its own, although the sensing process may take a considerable amount of time and can be disturbed by internal or external factors. **Remote sensing**, on the other hand, makes it possible to observe features that are not directly reachable by local sensors but have to be gathered from one or several other agents. The remote sensing abstraction reflects the delays and disturbances of the involved communication processes but abstracts away their technical details.

6.1 Usage of High-Level Constructs

SALMA's high-level language support for communication and generalized sensing spans across several sections of the model. First, all *connector types* for sensors and channels are declared in the domain model. As an example, the parking lot assignment model contains the following lines:

```
channel(assignments, v:vehicle, sam:sam).
sensor(freeSlotsL, plcs, freeSlots).
remoteSensor(freeSlotsR, sam, freeSlotsL, plcs).
```

Here, `assignments` is defined to be a type of channel over which agents of the sort `vehicle` can communicate directly with agents of the sort `sam` in order to request and receive a PLCS assignment. Each channel declaration actually specifies two *roles*, whose names are given on the left of the colons, that agents can play within the communication. The sensors of type `freeSlotsL` allow PLCS agents to count the current number of free slots at their station, i.e. access the fluent `freeSlots`. This information is propagated to the SAM via *remote sensors* of type `freeSlotsR` that effectively install unidirectional channels and periodic background processes at each SAM and PLCS agent which transmit and receive the content of `freeSlotsL`, respectively.

With the necessary declarations in place, the communication and sensing infrastructure can be used in agent processes by means of several special statements of the SALMA process definition language. As an example, the following lines appear in the definition of the main SAM process that handles incoming requests from vehicles, calculates optimal assignments, and sends them back to the vehicles:

```
Receive("assignments", "sam", SELF, vehicle,
req), Assign(resp, optimizeAssignments, [req]),
Iterate(resp, (v, p), Send("assignments", "sam",
v, "v", ("aresp", p)))
```

First, all available assignment requests are retrieved from the agent's incoming message queue with a call to `Receive` which stores a message list in the variable `req`. The actual assignment optimization logic is integrated by means of an external Python function `optimizeAssignments` that is not shown here due to space limitations. Through the `Assign` statement, the function is called with the received request list as a parameter and the function's result is stored in the variable `resp`. One of the most important inputs for this optimization is certainly the number of free slots at each PLCS. This information is made available by the remote sensor `freeSlotsR` from above that transparently gathers occupancy information from all PLCS. The result of `optimizeAssignments`, stored in `resp`, is a list of tuples that assign each requesting vehicle to a PLCS. The agent process iterates over this list and sends the PLCS id to each corresponding vehicle.

6.2 Predicate-Based Addressing

An important concern that arises in modeling multi-agent information propagation is how the set of receiving agents is determined. In many cases, it is either impossible or impracticable to do this statically. A particularly elegant alternative, supported by SALMA, is *predicate-based addressing* [3]. In this approach, the set of recipients for each information transfer is determined by a *characteristic ensemble predicate* that is evaluated for each (properly typed) agent pair. An *ensemble predicate* may describe *intentional selection criteria* as well as *intrinsic constraints* imposed by agent attributes or the environment. For instance, the channel declaration from Section 6.1 could be accompanied by the following predicate that declares that assignment requests issued by vehicles are only received by SAM agents within a given maximal communication range:

```
ensemble(assignment, Vehicle, SAM, S) :-
  distance(Vehicle, SAM, D, S), D <max_comm_dist.
```

7 Statistical Model Checking for Information Transfer

Once a system model has been created and configured in the way described above, SALMA's statistical model checker can be used to approximately assert system properties based on simulation results (cf. Section 4). SALMA's property specification language provides deep access to all elements of the communication and sensing processes. This allows direct reasoning about various aspects that are particularly important in CPS. For instance, the following invariant requires that when any vehicle agent sends an assignment request to the SAM, it will not take longer than 55 time units until a target PLCS has been set:

```
forall(v:vehicle,
implies(messageSent(assignments, v, ?, ?, ?, ?),
    eventually(55, currentTargetPLCS(v) \= none)))
```

Here, `messageSent` is a predicate that is true when the message has been sent in the current time step and `eventually` is the time bounded version of the well-known temporal operator. The question marks serve as wildcard arguments for pattern matching, applied here to the recipient and arguments of the message.

As another example, the next invariants define for all entries of the remote sensor `freeSlotsR` a maximum value age of 10 time units and a maximum deviation of 1 from the original sensor `freeSlotsL`:

```
forall([s:sam, p:plcs], age(freeSlotsR, [s, p]) =<10)
forall([s:sam, p:plcs], abs(freeSlotsR(s, p) - freeSlotsL(p)) =<1)
```

8 Experiments and Preliminary Evaluation

In order to test the presented approach and its integration in the SALMA toolkit, we implemented a reduced version of the scenario introduced in Section 2. It contains only a simple mock-up version of the optimization mechanism but realizes the full communication structure according to the approach presented in this paper. Both the SALMA toolkit and the model are available at the SALMA website[2]. By varying parameters and replacing the Python optimization function, different optimization schemes can be tested and the impact of factors like delays or transmission errors can be analyzed. A detailed evaluation of the model is still ongoing and beyond the scope of this paper. However, first experiences show that our information transfer model is well applicable also for complex communication scenarios. In particular, our proposed declarative high-level language has proven to be able to significantly improve clarity and conciseness of the model. For instance, the declarative part related to communication and sensing in the model mentioned above requires only about 30 lines in the style of the examples in Section 6.1. In contrast, the corresponding part of a functionally equivalent model that employs a direct axiomatization instead of the high-level abstractions, contains 15 fluents and 21 actions and events together with their associated axioms, which requires more than 200 lines of Prolog code.

9 Related Work

Information in the situation calculus has traditionally been viewed from an epistemic perspective, i.e. as knowledge that agents gain through (communication) actions. In [6], the epistemic model has been extended to model inter-agent communication by means of channels in a similar way as in our model described above. However, neither time nor stochastic effects are covered. In contrast to that, the approach presented in [8] combines the epistemic model with time and concurrency and allows reasoning about time-related aspects like the age of measurements. Unlike the approaches mentioned above, our model does not consider knowledge in the epistemic sense but leaves the interpretation of transferred information to the agent processes. While we think that this perspective is better suited in the particular context of cyber-physical systems, it would be possible to combine both views in a straight-forward way.

In [3], the authors introduce a stochastically timed process calculus that is centered around predicate-based communication. Like our model, the most

[2] www.salmatoolkit.org

detailed semantical variant they describe distinguishes between a preparation and a transmission phase and allows assigning separate probability distributions for delays and errors to each of them. However, since the semantics is based on continuous time markov chains (CMTC), only exponential distributions can be used and delays or errors are effectively determined at the start of each phase. This can be too coarse-grained in very dynamic situations, e.g. when the movement of agents has significant effect.

10 Conclusion

We have presented a new logic-based approach for modeling channel-based communication, sensing, and other kinds of information transfer within cyber-physical multi-agent systems. The proposed high-level language provides means to embrace the stochastic nature of these systems, like transmission delays and errors. At the same time it has a precise formal semantics based on the first-order logic situation calculus. Therefore, it can be integrated in existing logic-based approaches for verification and validation, in particular SALMA, a framework for simulation and statistical model checking we have introduced earlier in [2]. A major advantage of this combination is that SALMA's property specification language, based on a first-order temporal logic, allows fine-grained reasoning about the inner details of information transfer processes.

First experiences show that our approach offers great flexibility with respect to the level of detail and accuracy with which both the system model and corresponding requirements are formulated. Altogether, we hope that, in the long run, SALMA will contribute to making verification and validation practicable for self-adaptive cyber-physical multi-agent systems.

Acknowledgments. This work has been partially sponsored by the EU project ASCENS, FP7 257414.

References

1. Bureš, T., et al.: A Life Cycle for the Development of Autonomic Systems: The e-Mobility Showcase. In: 3rd Workshop on Challenges for Achieving Self-Awareness in Automatic Systems, pp. 71–76. IEEE (2013)
2. Kroiß, C.: Simulation and Statistical Model Checking of Logic-Based Multi-Agent System Models. In: Jezic, G., Kusek, M., Lovrek, I., J. Howlett, R., C. Jain, L. (eds.) Agent and Multi-Agent Systems: Technologies and Applications. AISC, vol. 296, pp. 151–160. Springer, Heidelberg (2014)
3. Latella, D., et al.: Stochastically timed predicate-based communication primitives for autonomic computing. Technical report, QUANTICOL Project (2014)
4. Legay, A., Delahaye, B., Bensalem, S.: Statistical Model Checking: An Overview. In: Barringer, H., Falcone, Y., Finkbeiner, B., Havelund, K., Lee, I., Pace, G., Roşu, G., Sokolsky, O., Tillmann, N. (eds.) RV 2010. LNCS, vol. 6418, pp. 122–135. Springer, Heidelberg (2010)

5. Levesque, H.J., et al.: Golog: A logic programming language for dynamic domains. The Journal of Logic Programming **31**(1), 59–83 (1997)
6. Marcu, D., et al.: Distributed software agents and communication in the situation calculus. In: International Workshop on Intelligent Computer, Communication, pp. 69–78 (1995)
7. Reiter, R.: Knowledge in action: logical foundations for specifying and implementing dynamical systems. MIT Press (2001)
8. Scherl, R.B.: Reasoning about the interaction of knowledge, time and concurrent actions in the situation calculus. In: 18th International Joint Conference on Artificial Intelligence (IJCAI 2003), pp. 1091–1098 (2003)
9. Scherl, R.B., Levesque, H.J.: Knowledge, action, and the frame problem. Artificial Intelligence **144**(1), 1–39 (2003)
10. Wald, A., et al.: Sequential tests of statistical hypotheses. Annals of Mathematical Statistics **16**(2), 117–186 (1945)