Auton Agent Multi-Agent Syst
DOI 10.1007/s10458-006-0014-y

Modeling dynamic environments in multi-agent simulation

Alexander Helleboogh · Giuseppe Vizzari · Adelinde Uhrmacher · Fabien Michel

Published online: ■

Abstract Real environments in which agents operate are inherently dynamic — the environment changes beyond the agents' control. We advocate that, for multi-agent simulation, this dynamism must be modeled explicitly as part of the simulated environment, preferably using concepts and constructs that relate to the real world. In this paper, we describe such concepts and constructs, and we provide a formal framework to unambiguously specify their relations and meaning. We apply the formal framework to model a dynamic RoboCup Soccer environment and elaborate on how the framework poses new challenges for exploring the modeling of environments in multi-agent simulation.

Keywords Multi-agent simulation · Dynamic environments · Dynamism

A. Helleboogh (✉)
AgentWise, DistriNet, Department of Computer Science, K.U.Leuven,
Leuven, Belgium
e-mail: alexander.helleboogh@cs.kuleuven.be

G. Vizzari
Laboratory of Artificial Intelligence (L.Int.Ar.), Department of Computer Science,
Systems and Communication (DISCo),
University of Milan-Bicocca,
Milan, Italy
e-mail: giuseppe.vizzari@disco.unimib.it

A. Uhrmacher
Department of Computer Science, University of Rostock,
Rostock, Germany
e-mail: lin@informatik.uni-rostock.de

F. Michel
CReSTIC/LERI – IUT de Reims-Châlons-Charleville,
51687 Reims Cedex 2, France
e-mail: fabien.michel@univ-reims.fr

1 Introduction

Simulation is the imitation of the operation of a real-world system or process over time [3]. A simulation model is a representation of a real-world system that incorporates time and the changes that occur over time [6].

In multi-agent simulation, the operation of a real-world system over time is imitated by means of executing a multi-agent model, i.e. a model comprised of a number of agents situated in a simulated environment. The agents represent the original actors present in the real-world system. The simulated environment represents the real environment the actors of the real-world system are situated in. The agents as well as the environment are essential parts of the *model* of a multi-agent simulation [18]. Consequently, both agents and environment must be supported explicitly in the development of a multi-agent model.

In case of a multi-agent system, the real environment is typically dynamic. A dynamic environment is an environment that changes in ways beyond the agents' control [23]. Each agent experiences dynamism in the environment that originates from various other sources. Dynamism in the environment can originate from other agents that are acting autonomously, or from various non-agent entities evolving on their own. Moreover, as all dynamism happens in a shared environment, it can interfere in complex ways, resulting for example in situations where actions do not yield their intended result [12, 27].

Modeling the environment has recently gained a lot of attention in multi-agent simulation. We advocate that for dynamic environments in multi-agent simulation, dynamism should be modeled explicitly as part of the simulated environment. In this paper, we propose a domain-specific formalism to model dynamic environments in multi-agent simulation. The concepts and constructs of the formalism are specifically aimed at modeling dynamic environments. The formalism represents the simulated environment as a dynamic system that encapsulates and regulates its own dynamism. The formalism provides concepts and constructs (1) to reify all dynamism in the simulated environment, (2) to delineate the way agents manipulate dynamism in the environment, and (3) to define how dynamism in the environment can interfere. As the concepts and constructs are complemented with a formal description, their meaning and relations are specified unambiguously. The formalism is applied to model a dynamic RoboCup Soccer environment. We elaborate on how the formalism poses new challenges for exploring the modeling of environments in multi-agent simulation to further extent.

This paper is structured as follows. In Sect. 2, we give an overview of related work and motivate our approach. In Sect. 3 we introduce concepts to model the structure of the environment, which serve as starting point to describe dynamism in the next sections. Section 4 describes the concepts to reify *dynamism* as first-class abstraction in the simulated environment. Section 5 focusses on concepts to specify the way agents manipulate dynamism in the environment and Sect. 6 on concepts to describe how dynamism in the environment interferes. Section 7 integrates the main concepts in a graphical overview and specifies the way a model based on the formalism can be executed. In Sect. 8, we discuss the added value of the formalism for modeling and simulation. Finally, Sect. 9 presents challenges for future research and we draw conclusions in Sect. 10.

2 Related work and motivation

Recent research puts forward that the environment in multi-agent simulation comprises two parts: the simulated environment and the simulation environment [18]. The *simulated environment* is part of the *model* and represents the real environment the agents are situated in. The *simulation environment* on the other hand, is the infrastructure for executing the simulation, for example a discrete event simulation engine. Making an explicit decoupling between the simulated environment (the model) and the simulation environment (the infrastructure) is a prerequisite for good engineering practice.

The simulated environment is an explicit building block at a modeling level. Research on environments in multi-agent simulation devotes a lot of attention to developing *domain-specific* constructs, i.e. constructs specifically aimed at modeling environments. For example, to support spatial and social structures of the environment, AGRE [11] relies on *spaces* as an explicit representation of physical (i.e. geometrical areas) as well as social structures (i.e. groups); MMASS [2] introduces a *multi-layered* model of the environment, with each layer an explicit representation of a particular spatial or conceptual structure of the real environment. To support interaction between agents, MIC* [13] introduces *interaction objects* and *interaction spaces* in the model of the environment; ELMS [22] is an environment description language with explicit support for specifying perception and interaction of cognitive agents. To present relevant information of the physical world to agents, Chang et al. [7] model a *cognitive middle layer* in the simulated environment that employs a shared ontology to present environmental information to the agents. To support actions in *dynamic* environments that change beyond the agents' control, the influence–reaction model [12] introduces the constructs *influences* and *reactions*. Influences model the attempts of agents, whereas reactions model the reaction of the environment in response to a set of simultaneously performed influences. In the influence–reaction model, the original attempt of the agent is decoupled from the actual outcome, which is essential in case of dynamic environments. To support interference in *dynamic* environments, in particular collisions of moving physical bodies, rigid body simulation [19] relies on very fine-grained models, expressed in terms of constructs from physics, such as masses, forces, rotational inertia, momentum and friction. Collisions are described in terms of interacting forces between bodies.

To execute a model, simulation environments typically require the model to be described in a particular *simulation formalism*. A simulation formalism provides constructs to specify a model as well as its evolution over time. For example, discrete-event simulators support the execution of models described in terms of the constructs *state* and *events* [4]. Hybrid simulators [20] support models expressed in terms of *state variables*, *equations*, *time events* and *state events*. Consequently, a model that is described in terms of *domain-specific* concepts must be translated into the constructs of a simulation formalism before it can be executed in a particular simulation environment.

The difficulty is that a simulation formalism typically fails to provide domain-specific support, i.e. it is *not* expressed in terms of concepts specifically aimed at modeling one particular domain. Simulation formalisms employ general constructs, such as state and events, which can be applied in a broad range of domains. Consequently, there is a lack of support for translating a model described in terms of domain-specific constructs into a general simulation formalism to allow model execution.

In this paper, we propose a simulation formalism that provides domain-specific support for modeling *dynamic* environments in multi-agent simulation. On the one hand, the formalism provides the modeler with a set of constructs that facilitate the modeling of dynamic environments. On the other hand, the formalism specifies how these constructs can be supported in an executable simulation. The formalism provides constructs for (1) representing in an explicit manner dynamism that happens in the environment, (2) specifying how dynamism in the environment is related to the agents, and (3) specifying how dynamism in the environment may arise, interfere and terminate. The formalism presents the simulated environment as a dynamic system that encapsulates and regulates its own dynamism.

The influence–reaction model also proposes a domain-specific simulation formalism for actions. The main difference between our formalism and the influence–reaction model is that our formalism employs a first-class representation of dynamism in the environment to express reactions.

3 Structure of the simulated environment

In this section, we introduce the basic concepts that are used to represent the structural parts of the environment. We deliberately kept this part of the formalism simple, as this suffices to discuss dynamism in the next sections.

The concepts, together with their formal description, are introduced in Sect. 3.1. Section 3.2 defines their state and Sect. 3.3 expresses their relation to the agents.

3.1 Environmental entities and properties

We represent the parts that constitute the simulated environment by means of two concepts: environmental entities and environmental properties. We do not address methodological issues on how to apply these concepts in practice, as this is highly dependent upon the objective of the simulation study [25].

– *Environmental entities.* Environmental entities represent entities characterized by their own, distinct existence in the real environment. The real environment typically contains numerous entities of different kinds that can be incorporated as environmental entities in the simulated environment. We define:

$$E = \{e_1, e_2, \ldots, e_n\} \mid \text{the set of environmental entities.}$$

Environmental entities can be partitioned into a set of disjoint subsets, with each subset grouping entities of the same kind. Formally:

$$Part_E = \{E_1, E_2, \ldots, E_k\} \mid \begin{array}{l}\text{a partition of environmental entities}\\ \text{with:}\\ E_i \subseteq E\\ E = \bigcup_{i=1\ldots k} E_i\\ E_i \cap E_j = \phi, \forall i \neq j\end{array}$$

For example, consider Fig. 1, depicting a part of a RoboCup Soccer environment [21]. The environmental entities we distinguish are two robots, a ball, a field and a goal:

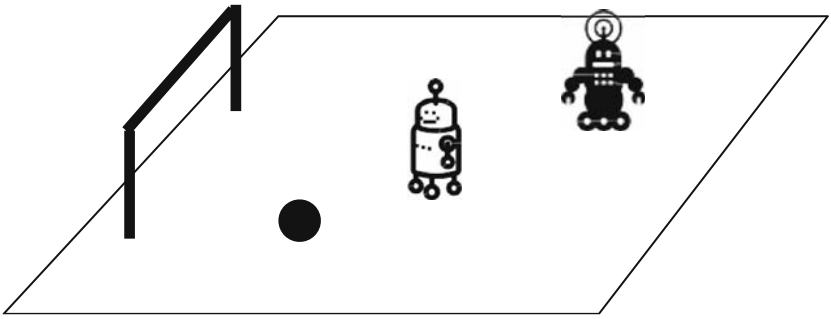

Fig. 1 A representation of a RoboCup Soccer environment

$E = \{robot_1, robot_2, ball, field, goal\}$	the set of entities.
$Part_E = \{Robot, Ball, Field, Goal\}$	a partition into four kinds of entities.
$Robot = \{robot_1, robot_2\}$	the set of robots.
$Ball = \{ball\}$	the set of balls.
$Field = \{field\}$	the set of fields.
$Goal = \{goal\}$	the set of goals.

– *Environmental properties.* An environmental property is a distributed quantity that represents a measurable, system-wide characteristic of the real environment. Environmental properties can be directly represented in the simulated environment if needed. We define:

$P = \{p_1, p_2, \ldots, p_m\}$	the set of environmental properties.
$Part_P = \{P_1, P_2, \ldots, P_l\}$	a partition of properties in different kinds.

Examples of environmental properties are gravitation and magnetic fields in the environment. In the RoboCup Soccer environment, the environmental properties we distinguish are temperature and humidity:

$P = \{temp, hum\}$	the set of environmental properties.
$Part_P = \{Heat, Humidity\}$	a partition in two kinds of properties.
$Heat = \{temp\}$	the set of temperature properties.
$Humidity = \{hum\}$	the set of humidity properties.

The set of all constituents is defined as the union between the set of environmental entities and properties: $C = E \cup P$. Constituents can be partitioned according to their kind, respecting the partitions of entities and properties: $Part_C = Part_E \cup Part_P$. After relabeling: $Part_C = \{C_1, C_2, \ldots, C_{k+l}\}$.

3.2 The state of the constituents

The state of the simulated environment is implicitly defined by the state of all its constituents, i.e. the state of all environmental entities and properties. We describe

the state of a constituent of the simulated environment as a list of values that are sufficient to define the status of the constituent at any point in time. We introduce the following definitions to describe the state of constituents:

S_{C_i}	the set of all possible states of constituents of kind C_i.
$s_c \in S_{C_i} = \langle v_1, v_2, \ldots, v_r \rangle$	the actual state of a particular constituent $c \in C_i$, represented as a tuple of values $v_j \in V_j$, with V_j a value domain.
$S = \bigcup_{C_i \in Part_C} S_{C_i}$	the set of all possible states of constituents of any kind.

To specify the initial state of each constituent, we define a function *Init* which maps a constituent on its initial state:

$$Init : C \rightarrow S$$
$$Init(c) = s_c$$

For the RoboCup Soccer environment we have for example:

$S_{Robot} = \mathbb{R}^2$	the set of all possible states for a robot.
$s_{robot_1} \in S_{Robot} = \langle \overrightarrow{pos} \rangle$	with $\overrightarrow{pos} \in \mathbb{R}^2$ a coordinate in the two dimensional space that indicates the actual position of $robot_1$.
$S_{Ball} = \mathbb{R}^2$	the set of all possible states for a ball.
$s_{ball} \in S_{Ball} = \langle \overrightarrow{pos} \rangle$	with $\overrightarrow{pos} \in \mathbb{R}^2$ a coordinate in the two dimensional space that indicates the position of the ball.

In the rest of the paper, we use the shorthand notations $s_r|_{pos}$ to select position of a robot $r \in Robot$, and $s_b|_{pos}$ to select the position of a ball $b \in Ball$.

3.3 Agents embodied as environmental entities

Embodiment is central in describing the relation between agents and the environment [5]. Agents are not external to the simulated environment [8]; agents are *embodied* as environmental entities. The environmental entity represents the tangible part, i.e. body [10], by means of which an agent exists in a particular environment.

Let $Ag = \{ag_1, ag_2, \ldots, ag_n\}$ be the set of agents that live in the simulated environment. The embodiment of agents as environmental entities is defined as the *Embody* function that maps an agent to the environmental entity that embodies the agent:

$$Embody : Ag \rightarrow E$$
$$Embody(ag) = e$$

For the RoboCup Soccer environment, we consider $Ag = \{alice, bob\}$ and:

$$Embody(alice) = robot_1$$
$$Embody(bob) = robot_2$$

Embodiment of an agent determines the way the agent can affect the environment (see Sect. 5) and vice versa (see Sect. 6).

4 Dynamism in the simulated environment

Starting from the basic model of the structure of the environment, we now elaborate on dynamism.

Dynamism is the evolution of the simulated environment *over time*. Agents are embodied in a dynamic environment where they experience dynamism happening beyond their own control. For example, an agent embodied as a robot in a RoboCup Soccer environment may experience the ball rolling in front of it and another robot moving on its side. To support the modeler in describing dynamism, we provide a first-class representation of dynamism in the simulated environment.

In this section, we describe the concepts for capturing dynamism in the simulated environment (Sect. 4.1), explain how they can be used to describe scenarios (Sect. 4.2) and how they specify the state (Sect. 4.3). Currently, we make abstraction from the way dynamism is initiated or how it interferes, as these will be the topics of the next two sections.

4.1 Activities

Dynamism in the environment comprises the evolution of environmental entities and properties over time. We introduce *activities* as a construct for representing dynamism in the simulated environment in an explicit manner. An activity describes a well-specified evolution of a particular constituent of the simulated environment, that is active over a specific time interval. An example of an activity in the RoboCup Soccer environment is a robot that is driving during a particular time interval. An activity comprises the following: the constituent involved, the time interval of occurrence and the evolution strategy.

- *The constituent involved.* Dynamism has an impact on particular parts of the simulated environment. Each activity is associated with the environmental entity or property it describes the evolution of. For example, in Fig. 2, the activity a_3 represents the rolling of the ball. The activities a_1 and a_2 represent the driving of one of the robots.
- *The time interval of occurrence.* Dynamism occurs over time. Consequently, each activity is characterized by a specific time interval. The time interval of an activity specifies the point in time the activity starts and the time its evolution completes. In case the activity never ends, the time interval is infinitely long. For example, in Fig. 2, activity a_1 representing the driving of a RoboCup robot $robot_1$, starts at time $t = 3$ after the start of the game, until $t = 5$. Activity a_2 starts at time $t = 6$ after the start of the game, until $t = 10$. Activity a_3 represents the rolling of the ball, starting at time $t = 7$, until the moment it stops rolling, at time $t = 9$.
- *The evolution strategy.* Dynamism evolves in a particular way. Consequently, activities are characterized by an evolution strategy that describes the specific way the status of the involved constituent changes over the time interval of occurrence. For example, for activity a_2 in Fig. 2, this could correspond to a change of the position of the robot according to a constant velocity vector.

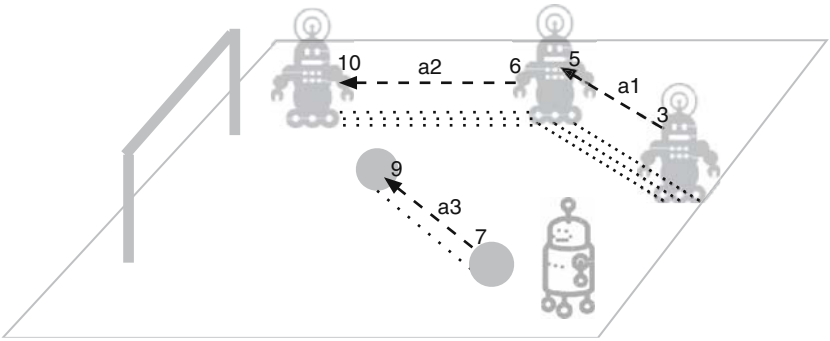

Fig. 2 Activities a_1, a_2 and a_3 in a RoboCup Soccer environment

Before giving a formal description of activities, we first introduce a number of general definitions:

$t \in T$	a particular time instant, with T the time domain.
$\Delta t \in \Delta T : t + \Delta t = t'$	a particular duration, with ΔT the set of all possible durations, including ∞.
$\Delta s_c \in \Delta S_{C_k}$	a state change for a constituent c of kind C_k, with ΔS_{C_k} the set of all possible state changes for constituents of kind C_k.
$\Delta S = \bigcup_{C_i \in Part_C} \Delta S_{C_i}$	the set of all possible state changes of constituents of any kind.
$\oplus : S \times \Delta S \rightarrow S$ $s \oplus \Delta s = s'$	the state-composition operator $\oplus$ defines a new state from a given state and a state change. Note that the operator is overloaded for each kind of constituent.

An activity a is defined as tuple:

$a = \langle c, t, \Delta t, par, F \rangle$	an activity with the following characteristics: $c \in C$: the constituent involved. $t \in T$: the starting time. $\Delta t \in \Delta T$: the duration until completion. $par = \langle v_1, \ldots, v_r \rangle \in V_1 \times \cdots \times V_r$: the parameters of the evolution strategy. $F : V_1 \times \cdots \times V_r \times \Delta T \rightarrow \Delta S$: the state change function, returning a state change $\Delta s \in \Delta S$ relative to the start of the activity, given a tuple of parameters and any duration not greater than the duration Δt of the activity.

In the rest of the paper, we use the following shorthand notations: $a|_c$ denotes the constituent, $a|_t$ and $a|_{\Delta t}$ denote the begin time and duration of activity a, respectively.

Furthermore, $a|_{par}$ denotes the parameters and $a|_F$ denotes the state change function of activity a.

For the RoboCup Soccer scenario in Fig. 2, consider the following activity as an example:

$$a_1 = \langle robot_1, 3, 2, \langle \overrightarrow{v_1} \rangle, Driving \rangle$$

Activity a_1 represents that constituent $c = robot_1$ starts driving at time $t = 3$ for a duration of $\Delta t = 2$. The state change is defined by the velocity vector $\overrightarrow{v_1}$ and the function *Driving*.

The function *Driving* is defined as:

$$Driving : \mathbb{R}^2 \times \Delta T \rightarrow \Delta S_{Robot}$$
$$Driving(\vec{v}, \Delta t) = \langle \vec{v} * \Delta t \rangle$$

Driving returns a state change $\Delta s \in \Delta S_{Robot}$ for robots that drive with a given velocity vector $\vec{v} \in \mathbb{R}^2$ during a given duration $\Delta t \in \Delta T$. As the state of a robot is a tuple $\langle \overrightarrow{pos} \rangle$ (see Sect. 3.2), the state change returned by *Driving* is the change of the position $\overrightarrow{pos}$. The change of the position $\overrightarrow{pos}$ is expressed as the function $\vec{v} * \Delta t$ with Δt the duration since the robot started driving with velocity $\vec{v}$.

As another example, consider the following activity from the RoboCup Soccer scenario in Fig. 2:

$$a_3 = \langle ball, 7, 2, \langle \overrightarrow{v_3}, \overrightarrow{d_3} \rangle, Rolling \rangle$$

Activity a_3 represents that constituent $c = ball$ starts rolling at time $t = 7$ for a duration of $\Delta t = 2$. The state change is defined by velocity vector $\overrightarrow{v_3}$ and deceleration vector $\overrightarrow{d_3}$, and the function *Rolling*.

The function *Rolling* is defined as:

$$Rolling : \mathbb{R}^4 \times \Delta T \rightarrow \Delta S_{Ball\rightarrow}$$
$$Rolling(\vec{v}, \vec{d}, t) = \langle \vec{v} * \Delta t - \frac{\vec{d}}{2} * \Delta t^2 \rangle$$

The function *Rolling* returns a state change for a ball that rolls with a initial velocity vector $\vec{v} \in \mathbb{R}^2$ and deceleration $\vec{d} \in \mathbb{R}^2$ during a given duration $\Delta t \in \Delta T$. The state of a ball is a tuple $\langle \overrightarrow{pos} \rangle$ (see Sect. 3.2). The change of the position returned by *Rolling* is expressed as the function $\vec{v} * \Delta t - \frac{\vec{d}}{2} * \Delta t^2$ with Δt the duration since the ball started rolling with velocity $\vec{v}$ and deceleration $\vec{d}$.

4.2 Scenarios

We now focus on how scenarios can be described. Scenarios describe a particular evolution of the environment and can be expressed in terms of activities for the various constituents. We define:

$a \in A^\Omega$	A^Ω is the set of all possible activities.
2^{A^Ω}	the powerset of A^Ω, i.e. the set of all possible subsets of A^Ω
$A = \{a_1, a_2, \ldots, a_r\}$	a scenario described by a set of activities, with $A \in 2^{A^\Omega}$

The scenario for the RoboCup Soccer environment depicted in Fig. 2, can be expressed as:

$A = \{a_1, a_2, a_3\}$, with:
$a_1 = \langle robot_1, 3, 2, \langle \overrightarrow{v_1} \rangle, Driving \rangle$
$a_2 = \langle robot_1, 6, 4, \langle \overrightarrow{v_2} \rangle, Driving \rangle$
$a_3 = \langle ball, 7, 2, \langle \overrightarrow{v_3}, \overrightarrow{d_3} \rangle, Rolling \rangle$

4.3 Dynamism and state

So far, we introduced activities and illustrated how activities can be used to describe scenarios. We now elaborate on how a scenario specifies the state of each constituent at any point in time.

We first define a function *Active* that checks whether an activity is active for a given constituent and a given time instant:

$$Active : A^{\Omega} \times C \times T \rightarrow Boolean$$
$$Active(a, c, t) = (a|_c = c) \wedge (a|_t < t \leq (a|_t + a|_{\Delta t})$$

For an activity to be active, it is required that the given constituent c is the constituent involved in the activity, and that the given time instant t is situated between the begin and end of the activity.

For a given initial state and a given scenario, we define a function *State* that calculates in a recursive manner the state of any constituent at any particular point in time.

$$State : 2^{A^{\Omega}} \times (C \rightarrow S) \times C \times T \rightarrow S$$
$$State(A, Init, c, t) = \begin{cases} State(A, Init, c, a|_t) \oplus a|_F(a|_{par}, t - a|_t) \\ \quad \text{if } \exists a \in A : Active(a, c, t); \\ \\ State(A, Init, c, t_x) \text{ with:} \\ \quad t_x = max\{t_k \in T | (t_k < t) \wedge (\exists a \in A : Active(a, c, t_k))\} \\ \quad \text{if } (\forall a \in A : \neg Active(a, c, t)) \ \wedge \\ \qquad (\exists t_k \in T; \exists a \in A : t_k < t \wedge Active(a, c, t_k)); \\ \\ Init(c, t) \\ \quad \text{otherwise;} \end{cases}$$

For a particular scenario A and initial state *Init*, the state of a constituent c at time t can be derived in the following way. Note that in the formalism, we assume that at most one activity can be active for any constituent at any time. We explain the three cases in the domain of the *State* function:

1. In the first case, there exists an activity a that is active for the given constituent c and time t. In this case, the state is recursively defined as the state $State(A, Init, c, a|_t)$ at the start $a|_t$ of the activity, composed with the state change specified by the activity. This state change is obtained by applying the function $a|_F$ with the following arguments: on the one hand, the parameters $a|_{par}$ specified by the activity, and on the other hand the duration $t - a|_t$, i.e. the time elapsed from the time $a|_t$ the activity started, until time t.

2. In the second case, there is no activity that is active for constituent c and time t; however, there exists an earlier time instant t_k at which there is an activity that is active for c. In this case, the state of the constituent c at time t is the same as the state of the constituent c at time t_x, where t_x is the latest time instant before t at which an activity was active for constituent c.
3. Otherwise, i.e. when there is no activity active for constituent c and time t, and there are no earlier activities that describe the evolution of c, then the state of c at time t is the initial state of c.

We illustrate the *State* function by means of the scenario in the RoboCup Soccer environment depicted in Fig. 2. We expand the recursion for:

$$\begin{aligned}
&State(A, Init, robot_1, 9)\\
&\quad = State(A, Init, robot_1, 6) \oplus Driving(\overrightarrow{v_2}, 3)\\
&\quad = (State(A, Init, robot_1, 5)) \oplus Driving(\overrightarrow{v_2}, 3)\\
&\quad = (State(A, Init, robot_1, 3) \oplus Driving(\overrightarrow{v_1}, 2)) \oplus Driving(\overrightarrow{v_2}, 3)\\
&\quad = ((Init(robot_1)) \oplus Driving(\overrightarrow{v_1}, 2)) \oplus Driving(\overrightarrow{v_2}, 3)
\end{aligned}$$

We explain the four expansions:

1. In the first expansion, $State(A, Init, robot_1, 9)$ is expressed in terms of $State(A, Init, robot_1, 6)$. At time $t = 9$, activity a_2 is active, indicating the $robot_1$ is driving. Consequently, the expansion is obtained by applying the first case of the *State* function.
2. In the second expansion, $State(A, Init, robot_1, 6)$ is expressed in terms of $State(A, Init, robot_1, 5)$. At time $t = 6$, activity a_2 is not active yet, i.e. $robot_1$ is not driving. However, there exists an earlier time at which another activity, i.e. activity a_1, is active. As the robot is not driving between $t = 5$, i.e. the end of a_1, and $t = 6$, i.e. the beginning of a_2, $State(A, Init, robot_1, 6)$ is the same as $State(A, Init, robot_1, 5)$, according to the second case of the *State* function.
3. For $State(A, Init, robot_1, 5)$, the first case of the *State* function applies, and it is expanded in terms of $State(A, Init, robot_1, 3)$.
4. For $State(A, Init, robot_1, 3)$, only the third case of the *State* function applies, which specifies that $State(A, Init, robot_1, 3)$ is equal to the initial state $Init(robot_1)$.

5 Agents manipulating dynamism

We now focus on the way scenarios arise by relating activities to agents.

Dynamism in the real environment is manipulated by the various actors present in that environment. In multi-agent simulation, the actors in the environment are represented as embodied agents. For example, in a RoboCup Soccer game, the agents *alice* and *bob* playing the soccer game are embodied as $robot_1$ and $robot_2$, respectively. It is clear that an agent must have a means to manipulate the entity it embodies, as the agents can make their robot start driving around, stop driving, etc.

However, not only agents can initiate activities, as entities not embodying any agent can also be involved in activities. For example, it is obvious that a ball can roll. Such activities are typically initiated indirectly, i.e. by other activities. For example, a ball rolls because it is hit by a robot. Indirect initiation of activities, i.e. by means of interference, is not considered for now, as it is dealt with in Sect. 6.

5.1 Influences

An environmental entity embodying an agent, mediates that agent's access to the environment (see Sect. 3.3). The influence–reaction model [12] introduces influences and reactions to model the agents' mediated access. Agents can only perform influences. An influence represents the *attempt* of the agent to manipulate the environment. The reaction models what actually happens in the environment in response to the attempts. In contrast to the influence–reaction model, we express the reaction of the environment in terms of manipulation of activities. Influences initiate and terminate dynamism in the environment.

Each agent autonomously decides at what time to perform an influence. The amount of time it takes an agent to decide upon what to do, results in a cost, i.e. a delay for all its subsequent influences. To determine the time instant an influence occurs, an explicit mapping between the internal process of an agent and time is necessary so as to determine how long an agent has been thinking or waiting [1,15].

We define an influence as a tuple:

$f \in Inf^{\Omega} = \langle ag, t, name, \langle v_1, \ldots, v_r \rangle \rangle$	an influence performed by agent $ag \in Ag$ at time $t \in T$, with name *name* and parameters $\langle v_1, \ldots, v_r \rangle \in V_1 \times \cdots \times V_r$. The set of all possible influences is Inf^{Ω}.

In the rest of the paper, we use shorthand notations $f|_{ag}$, $f|_t$, $f|_n$ and $f|_{v_i}$ to refer to the agent, time, name and a parameter, respectively.

For the RoboCup Soccer environment, consider the following influences as an example:

$$f_1 \in Inf^{\Omega} = \langle alice, t_a, startDriving, \langle \overrightarrow{v_0} \rangle \rangle$$
$$f_2 \in Inf^{\Omega} = \langle alice, t_b, stopDriving, \langle \rangle \rangle$$

The influence f_1 represents agent *alice* attempting to start driving with velocity vector $\overrightarrow{v_0}$ at time t_a. The influence f_2 represents agent *alice* attempting to stop driving at time t_b.

From the point of view of the simulated environment, influences are external inputs, generated by executing the agents. Let Ag^{Ω} be the set of all possible agents, $2^{Ag^{\Omega}}$ is the powerset, i.e. the set of all subsets of all possible agents, and $2^{Inf^{\Omega}}$ the powerset of all possible influences. We define a function *ExecAgents* which executes all agents in the simulation until the next time instant t_a at which one or several agents perform an influence:

$$\begin{aligned} &ExecAgents : 2^{Ag^{\Omega}} \rightarrow 2^{Inf^{\Omega}} \\ &ExecAgents(Ag) = \{ \quad \langle ag_1, t_a, name_0, \langle v_1, \ldots, v_r \rangle \rangle, \\ &\qquad\qquad \langle ag_2, t_a, name_1, \langle v_1, \ldots, v_u \rangle \rangle, \\ &\qquad\qquad \ldots, \\ &\qquad\qquad \langle ag_k, t_a, name_q, \langle v_1, \ldots, v_w \rangle \rangle \quad \} \end{aligned}$$

5.2 Reaction laws

Because an agent's access to the environment is mediated, an influence can lead to a different result than the one intended by the agent. For example, consider an agent embodied as a robot, that performs an influence at a particular point in time to start moving with a particular velocity in a particular direction. The activity that is initiated in response to this influence, represents the robot moving forward. However, the precise characteristics of the activity are determined by the characteristics of the robot that embodies the agent. For example, due to jitter in the hardware, the direction and velocity of the activity can slightly differ from the ones specified in the influence. Moreover, a robot is not able to travel at a higher velocity than the one it is physically able to achieve, even if an agent attempts to travel faster by performing an influence specifying a higher velocity.

To capture the agents' mediated access in a model, we introduce *reaction laws*. A reaction law is a rule that specifies the reaction of the environment in response to an influence. To determine a reaction, a reaction law takes into account (1) the characteristics of the influence the agent performed and (2) the characteristics of the environmental entity that embodies the agent. The reaction specified by a reaction law manipulates the activities that involve the environmental entity that embodies the agent. Note that a reaction law does not take into account other constituents, nor does it manipulate activities of other constituents. Dealing with interference between several constituents is the topic of Sect. 6.

5.2.1 Formal description of reaction laws

To formalize reaction laws, we first define a transformation of activities:

$trans \in Trans^{\Omega} = \langle t, Rem, Add \rangle$	an activity transformation at time $t \in T$, which removes the activities of set $Rem \in 2^{A^{\Omega}}$ and adds the activities of set $Rem \in 2^{A^{\Omega}}$. The set of all possible transformations is $Trans^{\Omega}$.

In the rest of the paper, the shorthand notation $trans|_t$ will be used to select the time of an activity transformation *trans*.

We define a function *ApplyTrans* which returns a set of activities representing the result of applying a given transformation on a given set of activities:

$$ApplyTrans : 2^{A^{\Omega}} \times Trans^{\Omega} \rightarrow 2^{A^{\Omega}}$$
$$ApplyTrans(A, \langle t, Rem, Add \rangle) = (A \setminus Rem) \cup Add$$

Reaction laws determine the way the environment reacts to the influences performed by the agent. We define:

$$Rlaw : 2^{A^{\Omega}} \times (C \rightarrow S) \times Inf^{\Omega} \rightarrow Inf^{\Omega} \times Trans^{\Omega}$$
$$Rlaw(A, Init, f) = \langle f, trans \rangle$$

A reaction law *Rlaw* is represented as a function that, for a given scenario A, a given *Init* function and a given influence f, returns a tuple containing the influence and the transformation in response to the influence.

In addition, we define:

$Rlaw \in Rlaws^{\Omega}$	$Rlaws^{\Omega}$ is the set of all possible reaction laws.
$Rlaws = \{Rlaw_1, Rlaw_2, \ldots, Rlaw_r\}$	a set of reaction laws, $Rlaws \in 2^{Rlaws^{\Omega}}$

We define a function *ApplyRlaws* that applies a set of reaction laws to a given set of influences to determine a set of reactions to the influences:

$$ApplyRlaws : 2^{A^{\Omega}} \times (C \to S) \times 2^{Rlaws^{\Omega}} \times 2^{Inf^{\Omega}} \to 2^{Inf^{\Omega} \times Trans^{\Omega}}$$
$$ApplyRlaws(A, Init, Rlaws, \{f_1, \ldots, f_r\}) = \{\langle f_1, trans_1\rangle, \ldots, \langle f_k, trans_k\rangle\}$$

5.2.2 RoboCup Soccer: reaction laws

In the RoboCup Soccer environment, we consider the following reaction law as example. The reaction law *StartDriveLaw* is responsible for initiating, in response to an influence, an activity that represents a robot driving in the environment.

$$StartDriveLaw : 2^{A^{\Omega}} \times (C \to S) \times Inf^{\Omega} \to Inf^{\Omega} \times Trans^{\Omega}$$
$$StartDriveLaw(A, Init, f) =$$
$$\begin{cases} \langle f, \langle f|_t, \phi, \{a\}\rangle\rangle \\ \quad a = \langle Embody(f|_{ag}), f|_t, \infty, \langle Jitter(f|_{\overrightarrow{v_0}})\rangle, Driving\rangle \\ \quad \text{if } (f|_n = startDriving) \wedge \\ \qquad (Embod(f|_{ag}) \in Robot) \wedge \\ \qquad (\forall a \in A : \neg Active(a, Embod(f|_{ag}), f|_t)) \wedge \\ \qquad (\|f|_{\overrightarrow{v_0}}\| \leq v_{max}); \\ \\ undefined \\ \quad \text{otherwise}; \end{cases}$$

The law *StartDriveLaw* can be understood as follows. The outcome of the law is undefined, unless the condition, described after *if*, is valid. This condition is a conjunction of four subconditions:

1. The first subcondition expresses that the name $f|_n$ of the influence f must be equal to *startDriving*.
2. The second subcondition expresses that the agent $f|_{ag}$ that performed the influence, must be embodied as a robot.
3. The third subcondition states that the robot that embodies the agent, i.e. *Embody* $(f|_{ag})$ may not yet be driving at the time $f|_t$ the influence is performed. The condition states that no activity is active for the robot at the time the influence is performed.
4. The fourth subcondition states that the length (expressed by the vector norm $\|.\|$) of the velocity vector $f|_{\overrightarrow{v_0}}$ of the influence must be smaller than or equal to the maximum velocity $v_{\max}$ the robot can achieve. Otherwise the robot will not start driving.

If all these conditions are valid, the robot starts driving. This is expressed in the outcome of the reaction law *StartDriveLaw*. The outcome is a tuple consisting of the influence f, and an activity transformation $\langle f|_t, \phi, \{a\}\rangle$ in response to f. The activity transformation occurs at the time $f|_t$ of the influence, and results in an extra activity a that is initiated. This activity describes the driving of the robot that embodies the agent. The activity starts at the time $f|_t$, i.e. the same time as the influence, and never ends (its duration is ∞), as the robot will continue driving until instructed otherwise. The velocity of the robot while driving corresponds to the velocity $f|_{\vec{v_0}}$ described in the influence, after applying a stochastic perturbation, as described by the *Jitter*-function:

$$\begin{aligned} &\mathit{Jitter} : \mathbb{R}^2 \to \mathbb{R}^2 \\ &\mathit{Jitter}(\vec{v}) = \vec{v}' \end{aligned}$$

We consider a second reaction law *StopDriveLaw* that stops a driving robot in response to an influence:

$$\begin{aligned} &\mathit{StopDriveLaw} : 2^{A^\Omega} \times (C \to S) \times \mathit{Inf}^\Omega \to \mathit{Inf}^\Omega \times \mathit{Trans}^\Omega \\ &\mathit{StopDriveLaw}(A, \mathit{Init}, f) = \\ &\quad \begin{cases} \langle f, \langle f|_t, \{a_1\}, \{a_2\}\rangle\rangle \\ \quad \text{with: } \ a_2 = \langle a_1|_c, a_1|_t, f|_t - a_1|_t, a_1|_{par}, a_1|_F\rangle \\ \quad \text{if } (f|_n = \mathit{stopDriving}) \wedge \\ \qquad (\mathit{Embody}(f|_{ag}) \in \mathit{Robot}) \wedge \\ \qquad (\exists a_1 \in A : \mathit{Active}(a_1, \mathit{Embody}(f|_{ag}), f|_t) \wedge \\ \qquad\qquad a_1|_F = \mathit{Driving}) \\ \\ \mathit{undefined} \\ \quad \text{otherwise;} \end{cases} \end{aligned}$$

The law *StopDriveLaw* can be understood as follows. The outcome of the law is undefined, unless the condition described after *if* is valid. This condition is a conjunction of three subconditions:

1. The first subcondition expresses that the name $f|_n$ of the influence f must be equal to *stopDriving*.
2. The second subcondition expresses that the agent $f|_{ag}$ that performed the influence, must be embodied as a robot.
3. The third subcondition states that the robot that embodies the agent performing the influence, i.e. $\mathit{Embody}(f|_{ag})$ is driving at the time $f|_t$ the influence is performed. The condition states that there exists an activity a_1 such that a_1 is active for the robot at that time, and such that a_1 is a *Driving* activity.

If all these conditions are valid, the law *StopDriveLaw* stops the robot from driving any further: the original driving activity a_1 is removed, and a new one a_2 is inserted that ends at the time the influence with name *stopDriving* is performed. Consequently, this outcome of the law is a tuple consisting of the influence f, and an activity transformation $\langle f|_t, \{a_1\}, \{a_2\}\rangle$ in response to f. This transformation removes the activity a_1 that represents the driving of the robot. An extra activity a_2 is added. a_2 is the same activity as a_1, i.e. the elements in its tuple correspond to the elements of a_1, except for

its duration. The duration of a_2 is such that the robot stops driving at time $f|_t$, i.e. the time the influence made it stop.

5.2.3 RoboCup Soccer: scenario

As an example of both laws, we illustrate how an agent can generate the scenario of the moving robot in Fig. 2. We start from the initial situation where both robots are standing still on their initial position, as defined by *Init*: we have $A = \phi$, i.e. no robot is moving yet. The agents are executed:

$$\begin{aligned} &\mathit{ExecAgents}(\{\mathit{alice}, \mathit{bob}\}) = \{\, f_1 \,\} \\ &\quad f_1 = \langle \mathit{alice}, 3, \mathit{startDriving}, \langle \overrightarrow{v_1'} \rangle \rangle \end{aligned}$$

By means of this influence, *alice* indicates it wants to start driving. *ApplyRlaws* determines and applies all reaction laws that are applicable. In response to this influence, *StopDriveLaw* is undefined, but *StartDriveLaw* is applicable, and returns:

$$\begin{aligned} &\mathit{StartDriveLaw}(\phi, \mathit{Init}, f_1) = \langle f_1, \langle 3, \phi, \{a_0\} \rangle \rangle \\ &\quad a_0 = \langle \mathit{robot}_1, 3, \infty, \langle \overrightarrow{v_1} \rangle, \mathit{Driving} \rangle \end{aligned}$$

We apply the proposed transformation on $A = \phi$:

$$\mathit{ApplyTrans}(\phi, \langle 3, \phi, \{a_0\} \rangle) = \{a_0\}$$

This represents the situation in Fig. 3, where robot_1 is driving infinitely long.

The agents are executed again:

$$\begin{aligned} &\mathit{ExecAgents}(\{\mathit{alice}, \mathit{bob}\}) = \{\, f_2 \,\} \\ &\quad f_2 = \langle \mathit{alice}, 5, \mathit{stopDriving}, \langle \rangle \rangle \end{aligned}$$

By means of this influence, *alice* indicates it wants to stop driving. *ApplyRlaws* determines and applies all reaction laws that are applicable. To this influence, *StartDriveLaw* is undefined, but *StopDriveLaw* is applicable, and returns:

$$\begin{aligned} &\mathit{StopDriveLaw}(\{a_0\}, \mathit{Init}, f_2) = \langle f_2, \langle 5, \{a_0\}, \{a_1\} \rangle \rangle \\ &\quad a_1 = \langle \mathit{robot}_1, 3, 2, \langle \overrightarrow{v_1} \rangle, \mathit{Driving} \rangle \end{aligned}$$

We apply the proposed transformation on $A = \{a_0\}$:

$$\mathit{ApplyTrans}(\{a_0\}, \langle 5, \{a_0\}, \{a_1\} \rangle) = \{a_1\}$$

The scenario now corresponds to Fig. 4, where the robot stops driving at time $t = 5$.

6 Interference of dynamism

Interference is the interaction of dynamism in the environment, which typically implies a discontinuity in its evolution [26].

Dynamism can interfere in complex ways. Interference of dynamism can be unwanted by the agents, typically because it causes an agent's actions not to yield the intended result [12,27]. For example, a robot that has started to drive forward in a straight line, hits an obstacle or is pushed aside by another robot. The result is that the robot

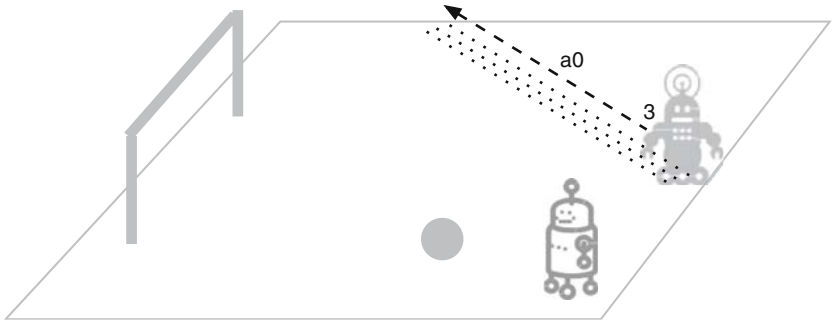

Fig. 3 Reaction of the environment: activity a_0 initiated in response to influence f_1

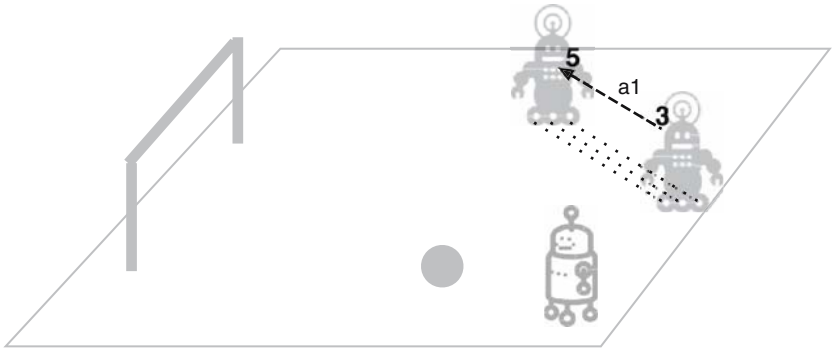

Fig. 4 Reaction of the environment: activity a_0 replaced by activity a_1 in response to influence f_2

reaches a position different from the one it intended to reach. However, interference of dynamism can also be desired by agents. For example, a robot moves forward into the path of a rolling ball, because it wants to deviate the ball in the direction of the goal.

Interference of dynamism plays a crucial role, particularly in *multi*-agent scenarios. Supporting the modeler to cope with interference of dynamism, requires an adequate means to represent interference at the modeling level.

In Sect. 6.1, we describe an example scenario of interference that will be used throughout the whole section. The concepts to support interference at a modeling level, are introduced in Sect. 6.2.

6.1 An example scenario

We illustrate interference of dynamism in the context of the RoboCup Soccer scenario in Fig. 5. In this scenario, it is clear that the robot driving with activity a_3 interferes with the ball. A robot and a ball are not allowed to penetrate: the robot kicks the ball when both come into contact, which happens at time $t = 5$. As a result of this contact, the ball will start rolling, denoted by activity a_4. At time $t = 8$, the rolling ball makes contact with the other robot, which is at that time driving according to activity a_2. As a result of this contact, the ball will be deviated towards the goal, as indicated by activity a_5.

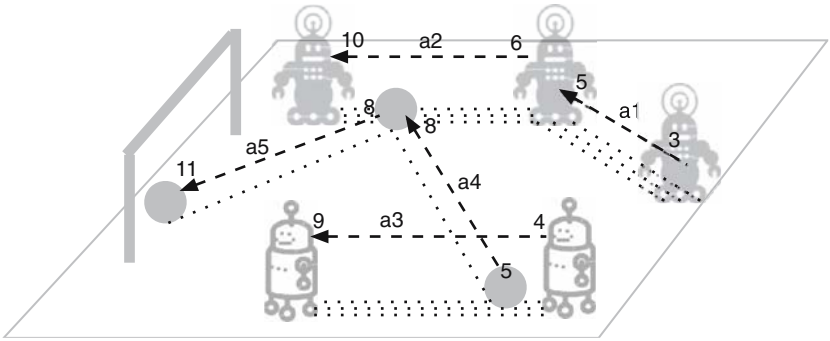

Fig. 5 An scenario with interference between robots and a ball

6.2 Interference laws

Whereas a reaction law determines a transformation of activities related to one environmental entity, without taking into account any others, an interference law determines the way several environmental entities can interact.

An *interference law* conceptualizes a domain-specific rule that specifies a way entities can interfere. As such, interference laws can be used to constrain dynamism in the simulated environment according to what is possible in the real environment. The description of an interference law comprises the following:

- *Interference conditions.* An interference condition specifies the circumstances for interference to occur. Interference conditions are used to check whether an activity interferes with other entities. An example of an interference condition is the penetration of a moving robot and the ball.
- *Activity transformations.* In case interference is detected at a particular time in a given scenario, the interference law specifies how the interaction affects the evolution described by the activities involved. An *activity transformation* specifies a particular transformation to be applied on the scenario.

Interference laws provide a modeler with an explicit means to define what kinds of interactions between entities are relevant for the simulation, as well as to specify the desired level of detail to describe the outcome of the interactions.

6.2.1 Formal description of interference laws

We define an interference law as a function that returns an activity transformation in case its interference conditions are met:

$$ILaw : 2^{A^{\Omega}} \times (C \rightarrow S) \rightarrow Trans^{\Omega}$$
$$Ilaw(A, Init) = trans$$

In addition, we define:

$Ilaw \in Ilaws^{\Omega}$	$Ilaws^{\Omega}$ is the set of all possible interference laws.
$Ilaws = \{Ilaw_1, Ilaw_2, \ldots, Ilaw_r\}$	a set of interference laws, $Ilaws \in 2^{Ilaws^{\Omega}}$

We define a function *ApplyIlaws* that applies a set of interference laws to a given scenario to determine a set of transformations:

$$ApplyIlaws : 2^{A^{\Omega}} \times (C \to S) \times 2^{Ilaws^{\Omega}} \to 2^{Trans^{\Omega}}$$
$$ApplyIlaws(A, Init, Ilaws) = \{trans_1, \ldots, trans_k\}$$

6.2.2 RoboCup Soccer: interference laws

For the RoboCup Soccer Environment, we give two examples of an interference law: *KickBallLaw* and *DeviateBallLaw*. The interference law *KickBallLaw* only applies to the case in which a stationary ball is hit by a moving robot. *KickBallLaw* enforces that the ball starts rolling by returning a transformation that initiates a new activity.

$$KickBallLaw : 2^{A^{\Omega}} \times (C \to S) \to Trans^{\Omega}$$
$$KickBallLaw(A, Init) =$$
$$\begin{cases} \langle t, \phi, \{a\}\rangle \\ \quad a = \langle b, t, \Delta t, \langle \vec{v}, \vec{d}\rangle, Rolling\rangle \\ \quad \text{if } \exists b \in Ball; \exists t \in T; \exists \Delta t \in \Delta T; \exists r \in Robot; \exists a_1 \in A; \exists \vec{v}, \vec{d} \in \mathbb{R}^2 : \\ \quad\quad Active(a_1, r, t) \wedge \\ \quad\quad \forall a_i \in A : \neg Active(a_i, b, t) \wedge \\ \quad\quad \|State(A, Init, b, t)|_{pos} - State(A, Init, r, t)|_{pos}\| = \epsilon \wedge \\ \quad\quad \|State(A, Init, b, t + dt)|_{pos} - State(A, Init, r, t + dt)|_{pos}\| < \epsilon \wedge \\ \quad\quad \vec{v} = Y_1(A, Init) \wedge \\ \quad\quad \vec{d} = Y_2(A, Init) \wedge \\ \quad\quad \Delta t = Y_3(\vec{v}, \vec{d}, Rolling) \\ \\ undefined \\ \quad \text{otherwise;} \end{cases}$$

The expression of the law can be understood as follows. The condition states that the law is applicable in case there is a ball b, a time instant t, a duration Δt, a robot r, an activity a_1 and velocity and deceleration vectors $\vec{v}$ and $\vec{d}$ such that:

1. The robot r is driving at time t, expressed by activity a_1.
2. The ball b is not rolling at time t, i.e. none of the activities is active for the ball at time t.
3. The distance between the position of the ball and the robot at time t is ϵ. ϵ represents the distance at which the bodies of the ball and the robot make contact.
4. The distance at time $t + dt$, with dt an infinitesimal small amount of time, between robot r and ball b is smaller than ϵ. This means that the robot would actually penetrate the ball after time t. This condition is not satisfied in situations where the robot stops driving right when it touches the ball.

5. The vectors $\vec{v}$ and $\vec{d}$ are determined by functions Y_1 and Y_2. We make abstraction on how their precise values are determined.
6. The duration Δt is determined by function Y_3, and corresponds to the time that elapses until the ball's speed reaches zero.

If all these conditions are satisfied, *KickBallLaw* proposes an activity transformation at time t. The transformation represents the addition of an activity a to the scenario. a describes the ball b rolling after time t. Note that according to *KickBallLaw*, the robot is not affected by hitting the ball.

In analogy with *KickBallLaw*, we define an interference law *DeviateBallLaw* that determines the outcome in case a rolling ball is hit by a robot that is driving:

$$
\begin{array}{l}
\mathit{DeviateBallLaw} : 2^{A^{\Omega}} \times (C \rightarrow S) \rightarrow \mathit{Trans}^{\Omega} \\
\mathit{DeviateBallLaw}(A, \mathit{Init}) = \\
\left\{ \begin{array}{l}
\langle t, \{a_1\}, \{a_2, a_3\}\rangle \\
\quad a_2 = \langle b, a_1|_t, t - a_1|_t, \langle a_1|\vec{v}, a_1|\vec{d}\rangle, a_1|_F\rangle \\
\quad a_3 = \langle b, t, \Delta t, \langle \vec{v}, \vec{d}\rangle, \mathit{Rolling}\rangle \\
\quad \text{if } \exists b \in \mathit{Ball}; \exists t \in T; \exists \Delta t \in \Delta T; \exists r \in \mathit{Robot}; \exists a_1, a_4 \in A; \exists \vec{v}, \vec{d} \in \mathbb{R}^2 : \\
\quad\quad \mathit{Active}(a_1, b, t) \wedge \\
\quad\quad \mathit{Active}(a_4, r, t) \wedge \\
\quad\quad \|\mathit{State}(A, \mathit{Init}, b, t)|_{pos} - \mathit{State}(A, \mathit{Init}, r, t)|_{pos}\| = \epsilon \wedge \\
\quad\quad \|\mathit{State}(A, \mathit{Init}, b, t + dt)|_{pos} - \mathit{State}(A, \mathit{Init}, r, t + dt)|_{pos}\| < \epsilon \wedge \\
\quad\quad \vec{v} = Y_1(A, \mathit{Init}) \wedge \\
\quad\quad \vec{d} = Y_2(A, \mathit{Init}) \wedge \\
\quad\quad \Delta t = Y_3(\vec{v}, \vec{d}, \mathit{Rolling}) \\
\\
\mathit{undefined} \\
\quad \text{otherwise;}
\end{array} \right.
\end{array}
$$

The expression of the law can be understood as follows. The condition states that the law is applicable in case there is a ball b, a time instant t, a duration Δt, a robot r, an activity a_1 and a_4, and velocity and deceleration vectors $\vec{v}$ and $\vec{d}$ such that:

1. The ball b is rolling at time t, expressed by activity a_1.
2. The robot r is driving at time t, expressed by activity a_4.
3. The distance between the position of the ball and the robot at time t is ϵ. ϵ represents the distance at which the bodies of the ball and the robot make contact.
4. The distance at time $t + dt$, with dt an infinitesimal small amount of time, between robot r and ball b is smaller than ϵ. This means that the bodies of the robot and the ball would actually penetrate at time t.
5. The vectors $\vec{v}$ and $\vec{d}$ are determined by functions Y_1 and Y_2. We make abstraction on how their precise values are determined.
6. The duration Δt is determined by function Y_3, and corresponds to the time that elapses until the ball's speed reaches zero.

If all these conditions are satisfied, *DeviateBallLaw* proposes an activity transformation at time t. The transformation performs two things. One the one hand, it removes the original activity a_1 of the ball, and replaces it with an activity a_2 which is identical to a_1, except that a_2 now stops at the moment ball and robot make contact.

Consequently, ball and robot no longer penetrate. On the other hand, the transformation adds a new activity a_3 that represents the new, deviated trajectory of the ball after the time t it hits the robot. Note that according to *DeviateBallLaw*, the robot is not affected by hitting the rolling ball.

From the definitions of *KickBallLaw* and *DeviateBallLaw*, it is clear that the interference scenario of Fig. 5 can be supported. In analogy to the *KickBallLaw* and *DeviateBallLaw*, laws can be defined to determine what happens in case a rolling ball hits a robot standing still, in case two robots collide, etc.

7 Towards a computational model

We now present the various parts in one integrated view and describe the evolution of the model as a whole.

In Sect. 7.1, we present a graphical overview that integrates all parts of the model. A formal way to specify the evolution of the model is described in Sect. 7.2.

7.1 Overview of dynamism in the simulated environment

Figure 6 gives a graphical overview that presents and relates the main concepts of the formalism. Depending on their purpose, the concepts to describe the simulated environment are organized in three main groups: concepts representing the *structure of the environment*, concepts to *represent dynamism* in the environment and concepts to *control dynamism* in the environment.

The concepts `Environmental Entity` and `Environmental Property` are used to model the structure of the environment. The association between `Environmental Entity` and `Agent` expresses that some environmental entities are embodied by an agent.

An `Activity` is used to represent dynamism in the simulated environment in an explicit manner. The association between `Activity` and `Environmental Entity` and the association between `Activity` and `Environmental Property` expresses that an activity describes the evolution of a particular environmental entity or property.

Activities themselves are subjected to evolution. We employ an `Activity Transformation` to reify how activities in the environment change. The association between `Activity Transformation` and `Activity` expresses that an activity transformation manipulates activities in the environment. The associations between `Activity Transformation` and `Reaction Law` on the one hand, and between `Activity Transformation` and `Interference Law` on the other hand, express that activity transformations are controlled by reaction laws and interference laws. A `Reaction Law` is used to specify how dynamism in the environment is affected by influences performed by agents. This is represented by the association between `Reaction Law` and `Influence`. An `Influence` is used to capture the *attempt* of an agent to affect dynamism going on in the environment. The association between `Agent` and `Influence` represents that agents can only manipulate activities indirectly, i.e. by means of performing influences in the environment. Finally, an `Interference Law` specifies a way in which several entities can interfere.

7.2 Evolution of the model

The evolution of the model is defined formally by means of the *Evol* function:

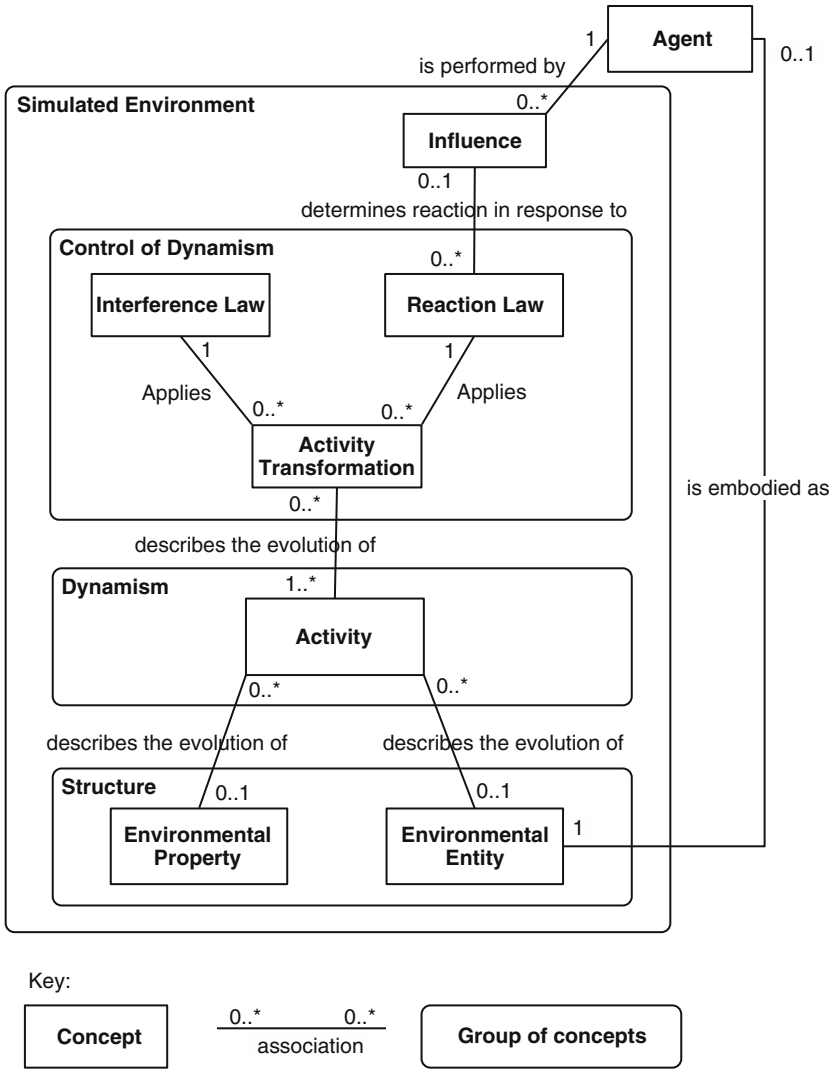

Fig. 6 Overview of the main concepts in the model and their associations

$$Evol : 2^{Ag^{\Omega}} \times 2^{A^{\Omega}} \times (C \rightarrow S) \times 2^{Rlaws^{\Omega}} \times 2^{Ilaws^{\Omega}} \rightarrow \perp$$
$$Evol(Ag, A, Init, Rlaws, Ilaws) =$$
$$Evol(Ag, Cycle(Exec(Ag), Rlaws, Ilaws, A, Init), Init, Rlaws, Ilaws)$$

The *evol* function is an infinite recursive function (denoted by the $\perp$). In each recursion, the *Exec* function executes all agents until the next time instant one or several agents perform an influence. Subsequently, the *Cycle* function returns an updated scenario A in reaction to this set of newly performed influences, by applying all transformations specified by the reaction laws and interference laws in the correct temporal order.

The *Cycle* function is defined as follows:

$$
\begin{aligned}
&Cycle : 2^{Inf^{\Omega}} \times 2^{Rlaws^{\Omega}} \times 2^{Ilaws} \times 2^{A^{\Omega}} \times (C \rightarrow S) \rightarrow 2^{A^{\Omega}} \\
&Cycle(Inf, Rlaws, Ilaws, A, Init) = \\
&\quad \begin{cases}
Cycle(Inf \setminus \{f\}, Rlaws, Ilaws, ApplyTrans(A, trans), Init) \\
\quad \text{if } \exists f \in F; \exists trans \in Trans^{\Omega} : \\
\quad\quad (\langle f, trans\rangle \in ApplyRlaws(A, Init, Rlaws, Inf)) \wedge \\
\quad\quad (\nexists \langle f_i, trans_i\rangle \in ApplyRlaws(A, Init, Rlaws, Inf) : \\
\quad\quad\quad trans_i|_t < trans|_t) \wedge \\
\quad\quad (\forall trans_j \in ApplyIlaws(A, Init, Ilaws) : \\
\quad\quad\quad trans|_t < trans_j|_t) \\
\\
Cycle(Inf, Rlaws, Ilaws, ApplyTrans(A, trans), Init) \\
\quad \text{if } \exists trans \in Trans^{\Omega} : \\
\quad\quad (trans \in ApplyIlaws(A, Init, Ilaws)) \wedge \\
\quad\quad (\nexists trans_i \in ApplyIlaws(A, Init, Ilaws) : \\
\quad\quad\quad trans_i|_t < trans|_t) \wedge \\
\quad\quad (\forall \langle f_j, trans_j\rangle \in ApplyRlaws(A, Init, Rlaws, Inf) : \\
\quad\quad\quad trans|_t \leq trans_j|_t) \\
\\
A \\
\quad \text{otherwise;}
\end{cases}
\end{aligned}
$$

The *Cycle* function takes as input a set of newly performed influences, the set of reaction laws and the set of interference laws, the actual scenario and a function that returns the initial state. The *Cycle* function then determines the updated scenario by applying laws in a recursive manner. There are three possible cases:

1. The first law that is applicable, is a reaction law. This is the case if there exists an influence f and an activity transformation *trans* for which the following three conditions are satisfied:

   (a) One of the reaction laws proposes the given transformation *trans* in reaction to the influence f of the set of influences. In other words, the tuple $\langle f, trans\rangle$ is an element of the set that is produced by the *ApplyRlaws* function.
   (b) There does *not* exist another transformation $trans_i$ that is also proposed by the *ApplyRlaws* function and that occurs earlier in time than *trans*.
   (c) All transformations $trans_j$ that are proposed by the interference laws occur later than the transformation *trans*.

   If all these conditions hold, the transformation *trans* in response to f is applicable before any other transformation. Consequently, the function *Cycle* is called recursively. The parameters reflect the outcome of the reaction law that proposed *trans*: (1) in the set of influences for which the reactions have yet to be determined, the influence f is removed, and (2) the scenario is now the one in which the transformation *trans* is already applied.
2. The first law that is applicable, is an interference law. This is the case if there exists an activity transformation *trans* for which the following three conditions are satisfied:

(a) One of the interference laws proposes the given transformation *trans* for the given scenario A. In other words, *trans* is an element of the set of transformations that is produced by the *ApplyIlaws* function.
(b) There does *not* exist another transformation $trans_i$ that is also proposed by the *ApplyIlaws* function and that occurs earlier in time than *trans*.
(c) All transformations $trans_j$ that are proposed by the reaction laws occur at the same time or later than the transformation *trans*.

If all these conditions hold, the transformation *trans* is applicable to A before any other transformation. Consequently, the function *Cycle* is called recursively. The parameters reflect the outcome of the interference law that proposed *trans*, i.e. the scenario is now the one in which the transformation *trans* is already applied.

3. No laws are applicable. The current scenario A is returned, as it already reflects all changes.

Note that all activity transformations are applied in increasing temporal order, which is required. With respect to activity transformations that occur at the same time, the *Cycle* function specifies the following arbitrary conventions. For activity transformations proposed by reaction laws that occur concurrently, the order is not specified. For activity transformations proposed by interference laws that occur concurrently, the order is not specified. Finally, in case an activity transformation proposed by a reaction law occurs concurrently with an activity transformation proposed by an interference law, the transformation of the interference law is applied first.

8 Discussion

In this section, we elaborate on the added value of the formalism for multi-agent simulation. We focus on the way the formalism provides support for describing a model in Sect. 8.1 and for executing a model in Sect. 8.2. In Sect. 8.3, we reflect upon factors that affect the computational cost.

8.1 Support for modeling

The concepts of the formalism offer domain-specific support to the modeler, i.e. they are aimed at facilitating the modeling of dynamic environments in MAS simulations. The concepts facilitate modeling not by hiding the complexity of dynamic environments, but by capturing this complexity in a clear and comprehensive way. The domain-specific nature of the concepts raises the abstraction level for the modeler, making it easier for the modeler to reason about dynamic environments.

For example:

- Activities support the modeler by representing in an explicit manner all dynamism that is happening in the environment.
- Embodiment, influences and reaction laws support the modeler in disentangling the complex relation between agents and the environment. The embodiment function specifies the way agents are associated with environmental entities. Influences reify the way an agent attempts to affect dynamism in the environment, whereas reaction laws specify the actual impact of influences on dynamism in the environment.

– Interference laws provide the modeler with a means for defining complex interactions in the environment.

8.2 Support for executing a model

The formalism provides support for translating a model described in terms of the domain-specific concepts into an executable simulation. The formalism specifies the functionality that is necessary to support the modeling concepts in an executable simulation. For example:

– The formalism unambiguously defines how each of the concepts can be expressed.
– The formalism defines how the state of any environmental constituent can be derived at any time in a given scenario (see Sect. 4.3).
– The formalism specifies the evolution of models that are expressed in terms of its concepts. In particular it specifies the synchronization between (1) the execution of the agents, (2) the enforcement of reaction laws and (3) the enforcement interference laws in the simulated environment (see Sect. 7.2).

Without the formalism, the translation of a model based on the concepts into an executable simulation would have to be reinvented from scratch for each simulation study. Consequently, the formalism promotes reuse, is less error-prone and accelerates the development of an executable simulation.

8.3 Computational cost

The formalism supports the modeler by providing a specification, without committing to specific algorithmic solutions to underpin it. As such, the formalism enables the use of custom algorithmic solutions that are optimized for a specific simulation study. The computational cost of executing a model that is based on the formalism, is dependent upon the following:

– *Complexity of the model.* The inherent complexity of the model has a significant impact on the computational cost to execute it. The complexity of a model is determined by the granularity of the activities, the quantity and complexity of the laws, etc.
– *Characteristics of the experiments.* For a particular model, the experimental setting can have an impact on the computational cost. For example, the initial position and the density of robots can have an impact on the frequency at which collisions occur.
– *Algorithmic solutions.* For a particular model and experiment, the chosen algorithmic solutions can have an impact on computational cost. Examples are fast collision detection algorithms, efficient mechanisms for matching reaction laws with influences, excluding activities that can never interfere from checks by the interference laws, etc.

A measurement of the computational cost, in the context of a specific simulation study based on the core ideas of this paper, is reported upon in [14]. The simulation study consists of building a simulation for testing the control software that steers automated guided vehicles (AGVs) in a dynamic warehouse environment.

9 Challenges

We discuss how the proposed formalism poses new challenges in the domain of multi-agent simulation. We concentrate on two main directions for future research: methodological challenges and engineering challenges.

9.1 Methodological challenges

Methodological challenges focus on using the formalism for a simulation study. The formalism presents a set of integrated concepts. Nevertheless, there are various possibilities for mapping a particular domain of interest on the concepts. The formalism provides a common ground for discussion, but future research is necessary to gain further insight on the various trade-offs when applying the formalism for modeling. Specific directions for future research include:

- How to devise an appropriate subset of relevant concepts for a particular simulation study? Not all concepts are necessarily needed when building a model for a simulation study. For example, not in all environments, activities can interference with each other.
- How well can the formalism be applied to more complex representations of the structure of the environment? We employed a simple model for representing the structure of the simulated environment, i.e. in terms of entities and properties. An interesting issue is investigating to what extent the formalism can be used to describe dynamism in an environment with a complex structure, for example an environment with an explicit spatial and social structure.
- How well is the formalism suited to describe dynamism other than physical movements? For example, activities could be used to represent the transmission of communication messages. In this case, interference laws could be used to model the quality of service of a communication channel, enforcing longer transmission durations in case of many transmissions. Further investigation is needed to explore the trade-offs in such settings.
- How to decide upon a suitable level of abstraction for applying the formalism? The concepts can be applied on various levels of abstraction, from very course grained to fined grained. Using a fine grained way to model activities and laws, will lead to more realistic scenarios, but will significantly increase the modeling effort and the computational cost to execute the model. Future research can focus on integrating experiences for gaining more insight on determining an appropriate trade-off.

9.2 Engineering challenges

The formalism defines the various concepts and relations with respect to dynamism, and specifies the functionality to support the concepts in an executable simulation. Engineering focusses on the design and development of a simulation environment according to the formal specification.

Developing a simulation environment that supports the formalism to its full extent is a substantial engineering challenge. We indicate two possible tracks that can serve as a basis for future research.

A first track is designing a *special-purpose simulation environment* based on the formal specification. The formal description defines the concepts in an unambiguous

way and specifies the evolution of simulation models that are expressed in terms of these concepts (see Sect. 7.2). As such, the formal specification provides guidance to the developer for building a special-purpose simulation environment for executing multi-agent simulations involving dynamic environments.

A second research track is investigating how the domain-specific constructs provided by the formalism can be translated into abstractions supported by mainstream simulation formalisms. Such a model translation would enable the reuse of *general-purpose simulation environments*. As a guideline for future research, we give a first indication on possible ways to relate the formalism to discrete event simulation and hybrid simulation.

9.2.1 Discrete event simulation

In discrete event simulation [4], models are described in terms of *state* and *events*. The state is a list of values that are sufficient to describe the state of the system at any point in time. An event is defined as a change of the state that occurs instantaneously at a specific point in time [24].

To support the formalism in discrete event simulation, we need to devise a way for mapping the concepts of the formalism onto *state* and *events*. There are various design choices. We briefly indicate two possible alternatives.

In a first alternative, the constituents are considered to form a state. Activities can be represented as a combination of state and events: the parameters of an activity are described as a state, attributed to a particular constituent, whereas the start and the end of each activity are translated into two successive events. Supporting activity transformations that result from the laws, is less trivial, as this requires additional infrastructure for inspecting and manipulating the event queue.

In a second alternative, the set of activities is considered to form a state, and the activity transformations proposed by the laws are represented as events. Consequently, the state of the constituents is defined implicitly, and the *State* function specified in Sect. 4.3 has to be implemented to derive it.

9.2.2 Hybrid simulation

In hybrid simulation [9,20], the evolution of the system is mixed, i.e. continuous phases alternated by discrete events. In a continuous phase, time advances, and the values of the state variables are determined by equations as a function of time. When a discrete event occurs, the equations and the state variables are altered in a discontinuous manner.

Events can be of two kinds:

- *Time events.* Time events are scheduled at a predetermined time.
- *State events.* State events are scheduled at the occurrence of a particular condition, i.e. when the continuous phase exceeds certain thresholds. As such, it is not known a priori at what time a state event occurs. Consequently, the simulation engine has to detect whether state events occur, and at what time their occurrence takes place.

To support the formalism in hybrid simulation, we need to devise a way for mapping the concepts of the formalism onto *state variables*, *equations*, *time events* and *state events*. We give some further indications.

The state of the constituents could be translated into state variables, and activities represented by equations. As interference laws represent conditions on the continuous phase, they can be represented as state events. Supporting reaction laws requires further investigation. Reaction laws are triggered by influences, performed by the agents. However, agents autonomously decide when to perform an influence, whereas time events are predetermined. Moreover, state events are expressed as conditions on the continuous phase, whereas the agents are typically modeled in a discrete manner.

An efficient treatment of interference laws, represented as state events during continuous simulation, provides an interesting challenge. Existing approaches to detect state events are based on *retroactive detection* [17], i.e. checking after each integrated time step whether a state event occurred, or by *conservative advancement* [16], i.e. advancing the simulation conservatively by chosing the time step so that no collision will occur during it. Moreover, fast collision detection algorithms, impact models and methods to enforce general motion constraints — especially the non-penetration constraints — have been developed [19]. This illustrates the rich body of research results that can be exploited for building support to simulate interference in simulated environments in an accurate and efficient manner.

10 Conclusions

Research on environments in multi-agent simulation devotes a lot of attention to modeling environments, exploring new directions. In this paper, we proposed a formalism to model dynamic environments in multi-agent simulation. In contrast to general simulation formalisms, the formalism is domain-specific, i.e. it employs concepts that are specifically aimed at modeling dynamic environments in multi-agent simulation. This domain-specific nature enables the formalism to present and support in an explicit manner a number of relations that are pertinent for modeling dynamic environments in multi-agent simulation:

- A dynamic environment *encapsulates* its own dynamism. Activities are part of the simulated environment and model dynamism in the environment in an explicit manner.
- A dynamic environment *regulates* its own dynamism. By means of reaction laws, the environment governs how dynamism is affected in response to influences by the agents. By means of interference laws, a dynamic environment constrains dynamism according the domain.
- The relation between agents and dynamic environment is twofold. On the one hand, agents are inherently part of the environment, as they are embodied as environmental entities. On the other hand, however, agents are restricted in their ability to affect dynamism in the environment. Agents can only affect dynamism in an indirect manner, using influences.

For multi-agent simulation, the contribution of the formalism is twofold. On the one hand, the formalism provides the modeler with a set of unambiguously defined concepts that facilitate the modeling of dynamic environments. On the other hand, the formalism provides support for executing a model by specifying how the concepts can be supported in an executable simulation.

We illustrated the formalism by applying it for modeling dynamism in a RoboCup Soccer environment. We elaborated on how the formalism poses new challenges

to the multi-agent simulation community and hope that the formalism provides an interesting point of view to explore the modeling of environments for multi-agent simulation to further extent.

Acknowledgements We would like to express our appreciation to the attendees of the workshops on Environments for Multiagent Systems in New York 2004 and Utrecht 2005, and the AgentLink III Technical Forum in Ljubljana and Budapest, 2005 for the inspiring discussions that have considerably contributed to the work presented in this paper. We are also grateful to Danny Weyns, Tom Holvoet, Paul Valckenaers and Eric Platon for providing valuable feedback on drafts of this paper.

References

1. Anderson, S. D., & Cohen, P. R. (1996). Timed common lisp: The duration of deliberation. *SIGART Bull, 7*(2), 11–15.
2. Bandini, S., Manzoni, S., & Vizzari, G. (2005). A spatially dependent communiction model for ubiquitous systems. In D. Weyns, V. Parunak, & F. Michel (Eds.), *First international workshop on environments for multi-agent systems*. Vol. 3374 of *Lecture Notes in Computer Science*. New York, NY, USA: Springer-Verlag.
3. Banks, J. (1999). Introduction to simulation. In *WSC '99: Proceedings of the 31st conference on Winter simulation*. (pp. 7–13). New York, NY, USA: ACM Press.
4. Banks, J., Carson, J. S., Nelson, B. L., & Nicol, D. M. (2000). *Discrete-event system simulation*. (3rd ed.) Upper Saddle River, N.J.: Prentice-Hall.
5. Brooks, R. A. (1991). Intelligence without reason. In J. Myopoulos, & R. Reiter (Eds.), *Proceedings of the 12th international joint conference on artificial intelligence (IJCAI-91)*. Sydney, Australia (pp. 569–595). San Mateo, CA, USA: Morgan Kaufmann publishers Inc.
6. Carson, J. S. (2003). Introduction to simulation: Introduction to modeling and simulation. In *Winter simulation conference*, pp. 7–13.
7. Chang, P., Chen, K., Chien, Y., Kao, E., & Soo, V. (2005). From reality to mind: a cognitive middle layer of environment concepts for believable agents. In D. Weyns, V. Parunak, & F. Michel (Eds.), *First international workshop on environments for multi-agent systems*. Vol. 3374 of *Lecture Notes in Computer Science*. New York, NY, USA: Springer-Verlag.
8. Clark, A. (1996). *Being there: Putting brain, body, and world together again*. Cambridge, MA, USA: MIT Press.
9. Esposito, J. M., & Kumar, V. (2004). An asynchronous integration and event detection algorithm for simulating multi-agent hybrid systems. *ACM Transactions on Modeling and Computer Simulations, 14*(4), 363–388.
10. Ferber, J., Michel, F., & Báez-Barranco, J.-A. (2005a). AGRE: Integrating environments with organizations. In D. Weyns, H. V. D. Parunak, & F. Michel (Eds.), *Environments for multi-agent systems, first international workshop, E4MAS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*. Vol. 3374 of *Lecture Notes in Computer Science* (pp. 48–56). Springer.
11. Ferber, J., Michel, F., & Baez, J. (2005b). AGRE: Integrating environments with organizations. In D. Weyns, V. Parunak, & F. Michel (Eds.), *First international workshop on environments for multi-agent systems*. Vol. 3374 of *Lecture Notes in Computer Science*. New York, NY, USA: Springer-Verlag.
12. Ferber, J., & Müller, J. (1996). Influences and reaction: A model of situated multiagent systems. In *Proceedings of the 2nd international conference on multi-agent systems* (pp. 72–79). AAAI Press.
13. Gouaïch, A., Michel, F., & Guiraud, Y. (2005). MIC*: A deployment environment for autonomous agents. In D. Weyns, V. Parunak, & F. Michel (Eds.), *First international workshop on environments for multi-agent systems*, Vol. 3374 of *Lecture Notes in Computer Science*. New York, NY, USA: Springer-Verlag.
14. Helleboogh, A., Holvoet, T., & Berbers, Y. (2006). Testing AGVs in dynamic warehouse environments. In D. Weyns, V. Parunak, and F. Michel (Eds.), *Environments for multiagent systems II*, Vol. 3830 of *Lecture Notes in Computer Science* (pp. 270–290). Springer-Verlag.
15. Helleboogh, A., Holvoet, T., Weyns, D., & Berbers, Y. (2005). Extending time management support for multi-agent systems. In *Multi-agent and multi-agent-based simulation: Joint workshop MABS 2004, New York, NY, USA, July 19, 2004, Revised Selected Papers*, Vol. 3415 / 2005 of *Lecture Notes in Computer Science* (pp. 37–48). Springer-Verlag, GmbH.

16. Herzen, B. V., Barr, A. H., & Zatz, H. R. (1990). Geometric collisions for time-dependent parametric surfaces. In *SIGGRAPH '90: Proceedings of the 17th annual conference on Computer graphics and interactive techniques* (pp. 39–48). New York, NY, USA: ACM Press.
17. Kamat, V. V. (1993). A survey of techniques for simulation of dynamic collision detection and response. *Computers & Graphics 17*(4), 379–385.
18. Klügl, F., Fehler, M., & Herrler, R. (2005). About the role of the environment in multi-agent simulations. In *Environments for multi-agent systems*, Vol. 3374 of *Lecture Notes in Computer Science* (pp. 127–149). Springer-Verlag.
19. Mirtich, B. (2000). Timewarp rigid body simulation. In *SIGGRAPH '00: Proceedings of the 27th annual conference on computer graphics and interactive techniques* (pp. 193–200). New York, NY, USA: ACM Press/Addison-Wesley Publishing Co.
20. Mosterman, P. J. (1999). An overview of hybrid simulation phenomena and their support by simulation packages. In *HSCC '99: Proceedings of the 2nd international workshop on hybrid systems* (pp. 165–177). London, UK: Springer-Verlag.
21. Nardi, D., Riedmiller, M., Sammut, C., & Santos-Victor, J. (Eds.). (2005). RoboCup 2004: Robot Soccer World Cup VIII, Vol. 3276 of *Lecture Notes in Computer Science*. Springer.
22. Okuyama, F., Bordini, R. H., & da Rocha Costa, A. C. (2005). An environment description language for multiagent simulation. In D. Weyns, V. Parunak, & F. Michel (Eds.), *First international workshop on environments for multi-agent systems*. Vol. 3374 of *Lecture Notes in Computer Science*. New York, NY, USA: Springer-Verlag.
23. Russell, S., & Norvig, P. (1995). *Artificial intelligence: A modern approach*. Englewood Cliffs, NJ: Prentice-Hall
24. Schriber, T. J., & Brunner, D. T. (1999). Inside discrete-event simulation software: How it works and why it matters. In *WSC '99: Proceedings of the 31st conference on Winter simulation* (pp. 72–80). ACM Press.
25. Shannon, R. E. (1998). Introduction to the art and science of simulation. In *Winter simulation conference*, pp. 7–14.
26. Witkin, A., Gleicher, M., & Welch, W. (1990). Interactive dynamics. In *SI3D '90: Proceedings of the 1990 symposium on Interactive 3D graphics* (pp. 11–21). New York, NY, USA: ACM Press.
27. Wooldridge, M. J. (2001). *Introduction to multiagent systems*. New York, NY, USA: John Wiley & Sons, Inc.