Proceedings of the 2012 Winter Simulation Conference
C. Laroque, J. Himmelspach, R. Pasupathy, O. Rose, and A. M. Uhrmacher, eds.

MODELING AGENTS AND THEIR ENVIRONMENT IN MULTI-LEVEL-DEVS

Alexander Steiniger
Frank Krüger
Adelinde M. Uhrmacher

University of Rostock
Albert-Einstein-Straße 22
D-18059 Rostock, GERMANY

ABSTRACT

Environments play an important role in multi-agent systems. They present the context agents operate in. When testing multi-agent systems by simulation, the environment and partly agents have to be modeled. We explore the potential of Multi-Level-DEVS to serve as a modeling formalism for agents, their environment, and the interaction between them. Multi-Level-DEVS combines a modular, hierarchical modeling with variable structures, dynamic interfaces, and explicit means for describing up- and downward causation between different levels of the compositional hierarchy. The modeling in Multi-Level-DEVS emphasizes the role of the environment to provide information for and enforce constrains on the situated agents. A smart meeting room scenario is modeled, and an approach aimed at recognizing user activities in smart environments is tested and evaluated in a simulation study.

1 INTRODUCTION

Environments play an important role in the functioning of multi-agent systems, as agents being situated is one of their salient features (Weyns et al. 2005; Platon et al. 2006; Wooldridge 2009). The environment can be perceived as constraining the interaction between agents (Platon et al. 2006), but also as a shared medium for interaction, e.g., deploying pheromones in the environment (Parunak 1997). In both cases it is important to treat the environment as a first-order abstraction in designing effective multi-agent systems (Weyns et al. 2005).

To model and simulate interacting agents in their environment, both, a modular construction of models via building blocks and globally accessible information and global constraints appear equally important. For instance, building blocks foster the reuse of models and ease the adaption to test scenarios, e.g., by exchanging different human behavior models (Gierke et al. 2006). Whereas the ability to access information globally and define global constraints facilitates the consideration of physical and spatial constraints of interactions (Helleboogh et al. 2007). However, the question is how to support both? Therefore, we will take a closer look at the modeling formalism *ml*-DEVS.

The remainder of this paper is structured as follows: In Section 2, a revision of *ml*-DEVS and its execution semantics is presented. Then, a simple use case in the domain of smart meeting rooms is given. Finally, Section 4 concludes and discusses our approach and gives an outlook on future work.

2 MULTI-LEVEL-DEVS

Multi-Level-DEVS (Multi-Level Discrete Event System Specification), or short *ml*-DEVS, is based on the modular, hierarchical modeling formalism DEVS (Zeigler et al. 2000). First ideas of *ml*-DEVS were proposed in the context of computational systems biology (Uhrmacher et al. 2007), where an increasing need to combine different levels of organization, ranging from proteins and cells to cell populations, and

to describe the up- and downward causation between those levels exist (Maus et al. 2011). We explore the potential of *ml*-DEVS to serve as a formalism to model agents, their environment, and the interaction between them.

Multi-Level-DEVS distinguishes between MICRO-DEVS and MACRO-DEVS models, which refer to atomic and coupled models in other DEVS variants. However, in *ml*-DEVS, macro models have a state and behavior of their own. The definitions of both models are based on those given in (Uhrmacher et al. 2007), which have been revised to cope with the needs of multi-agent systems, e.g., the introduction of component interfaces that are accessible by the macro level and thus a clear separation between private and public information of components.

2.1 MICRO-DEVS Models

Let XY and S be structured sets $\{(n,v)|n \in \mathcal{N} \wedge v \in Dom_n\}$, where Dom_n is the domain of n and $\mathcal{N}$ refers to all available names including port and state variable names. A MICRO-DEVS or micro model is then defined as follows:

Definition 1 (MICRO-DEVS) A MICRO-DEVS model is defined as structure

$$\langle id, XY, S, s_{\text{init}}, p, \delta, \lambda, ta \rangle$$

with

id	is a unique identifier and $id \in \mathcal{N}$
XY	is the structured set of in- and outputs
S	is the structured set of states
$s_{\text{init}} \in S$	is the initial state
$p : S \to 2^{\mathcal{N}}$	is the port selection function
$\delta : XY^b \times Q \to S \times XY^b$	is the state transition function, where XY^b is a multiset over elements in XY, Q is a set of tuples $\{(s,e)\|s \in S, 0 \leq e \leq ta(s)\}$, and e is the time elapsed since the last state transition
$\lambda : S \to XY^b$	is the output function
$ta : S \to \mathbb{R}_0^+ \cup \infty$	is the time advance function

The model communicates with its surroundings via the set XY. The time advance function ta associates with each state a time interval the state will persist per Multimodel and after which an internal event (if no input event has arrived) or a confluent event (an input event occurs at the same time) will be triggered. In both cases, the output function λ is invoked and creates outputs for the given state. Subsequently, the transition function δ is invoked changing the model's current state. In contrast to Uhrmacher et al. (2007), the model can create additionally information in δ, which is then accessible at the macro level. If an input event arrives at a time, when the time defined by ta has not yet elapsed, only δ will be invoked. The port selection function p determines, which ports are available in a given state. Only those can be used for receiving input events, sending output events, and propagating information upwards to the macro level.

2.2 MACRO-DEVS Models

MACRO-DEVS or macro models refer to coupled models of other DEVS variants as they comprise sub-models (components) and thus allow a composition of those. However, compared to coupled models, macro models have a state and behavior of their own, and they have functions that closely tie the behavior of the macro level to the behavior at the micro level. A MACRO-DEVS model is defined as follows:

Definition 2 (MACRO-DEVS) A MACRO-DEVS model is defined as structure

$$\langle id, XY, S, s_{\text{init}}, p, C, MC, \delta, \lambda_{\text{down}}, v_{\text{down}}, sc, act_{\text{up}}, \lambda, ta \rangle$$

with

C	is the set of sub-models, which can be of type MICRO-DEVS and MACRO-DEVS
MC	is the set of multi-couplings $\{mc \mid mc : \mathcal{N} \rightarrow 2^{\mathcal{N}} \setminus \emptyset\}$,
$\delta : XY^b \times Q \times 2^{\mathcal{I}_C} \rightarrow S \times XY^b$	is the state transition function, where $\mathcal{I}_C$ denotes the set of component interfaces $\{(id, XY) \mid \exists c \in C \wedge c = < id, XY, ... >\}$
$\lambda_{\text{down}} : S \rightarrow 2^{\mathcal{I}_C}$	is the downward output function
$v_{\text{down}} : S \rightarrow 2^{\{(n,v) \mid n \in \mathcal{N} \wedge v \in Dom_n\}}$	is the value coupling downward
$sc : S \rightarrow 2^C \times 2^{MC}$	is the structure change function
$act_{\text{up}} : S \times 2^{\mathcal{I}_C} \rightarrow \{true, false\}$	is the activation function

The identifier id, the structured sets XY and S, the initial state s_{init}, and the functions p, λ, and ta are defined as for MICRO-DEVS models. In addition, C denotes the set of potential components, which can be micro or macro models. The communication structure between components in conjunction with their variable ports is specified by a dynamic coupling mechanism called multi-couplings. Multi-couplings are defined by the set MC of functions that map one source port name to several target port names. The availability of ports, whose names match those specified in a multi-coupling, during execution implies the existence of a concrete coupling between those ports and thus the models they refer to as long as the ports are available. To support variable structures, the structure change function sc determines the available components and multi-couplings for a given state. The available components indicate the current composition structure, whereas the available multi-couplings define the current communication structure. Thereby, *ml*-DEVS stands in the tradition of other DEVS variants that support variable structures in a top-down manner, like the approach by Barros (1997).

However, in *ml*-DEVS, of particular interest are the questions: how does the macro model's state constrain activities at its micro level and how do activities at the micro level influence those at the macro level in return? Please note that models at the micro level, i.e., the macro model's components, can also be macro models that have components at their micro level and so on. Interdependencies between different organizational levels are typically referred to as down- and upward causation (Campbell 1974). In *ml*-DEVS, those can be expressed explicitly and split into information transfer and activation:

Downward information: The function v_{down} realizes the downward causation at information level. It couples relevant state variables of the macro model to ports of its components (value coupling). Values of those variables are made directly accessible to the micro level (for reading) and, in addition, those values can be manipulated by the macro model systematically, e.g., values that models at the micro level access can be distorted or uncertain.

Downward activation: Downward activation refers to the ability of macro models to initiate state transitions at the micro level. To do so, events need to be generated. This is realized by the function λ_{down}, which allows the macro model to send events directly to its components by accessing their ports.

Upward information: Information can be propagated upwards from the micro to the macro level by models changing their ports at the micro level. In the transition function of the MACRO-DEVS model, information about models at the micro level, i.e., which ports and, in contrast to Uhrmacher et al. (2007), also which values in the ports are available, are accessed and used to determine the next state of the macro model.

Upward activation: Upward activation refers to the ability to initiate changes at the macro level by the micro level by changing ports or port values. The function act_{up} guards the fulfillment of invariants that are defined at the macro level and refer to the macro model's components, their ports, and published values at those ports. If an invariant is violated, the macro model's transition function δ will be invoked.

2.3 Simulation

Multi-Level-DEVS models describe discrete event systems. Following the tradition of DEVS, an abstract simulator specifies how trajectories are produced according to the model description (Zeigler et al. 2000). Thereby, the changes made in the revised formalism are reflected by the abstract simulator that is presented next (cf. Uhrmacher et al. (2007)). Those changes include, no distinction between in- and outputs, the propagation of values in addition to port names up the model hierarchy, and the clear separation between private and public information of components, i.e., the introduction of component interfaces that are accessible by the macro level. A component interface is defined by the ports of the component and the values in those ports. The direction of a port is determined by its context, i.e., a port can be used as input and output port.

The abstract simulator of *ml*-DEVS is a treelike structure comprising of two processor types, i.e., "simulators" as leaves and "coordinators" as inner nodes. The former are responsible for executing MICRO-DEVS models, whereas the latter execute MACRO-DEVS models. To do so, coordinators and simulators communicate top-down and bottom-up the hierarchy via predefined messages, i.e., $\star$-messages, `x`-messages, `y`-messages, and `done`-messages. A third kind of processor, the "root-coordinator", that is not associated with a model forms the root of the processor tree. It initiates and controls the simulation cycles. In the following, simulators and coordinators of the revised version of *ml*-DEVS are briefly described.

2.3.1 Simulator

The simulator of *ml*-DEVS is shown in Listing 1 as pseudocode and similar to those of other PDEVS (Parallel DEVS) variants (Uhrmacher et al. 2007), in particular the processing of $\star$-messages (lines 10-12). However, there is only one transition function δ invoked (16) after receiving an `x`-message (14). Besides the actual model inputs, the `x`-message might contain information propagated downwards by the simulator's parent (denoted as `xy`). The state space, δ operates on, is structured into the usual (private) state S and the in- and outputs XY, which can be interpreted as public state. After executing δ, the in- and outputs are flushed to delete downward propagated information (17). The time of the last event (`tole`) is set to `t` and the time of the next internal event (`tonie`) is determined based on the time advance function ta (18-19). Finally, the port selection function p is executed to determine the active port names for the given model state (20) and a `done`-message is sent to the simulator's parent including the updated in- and outputs and the micro model's `tonie` (21). This allows to propagate information upwards to the macro level separated from regular outputs generated in the λ-function (12) and sent within a `y`-message (13).

2.3.2 Coordinator

The coordinator is shown in Listing 2 and combines functionality of other PDEVS coordinators and the *ml*-DEVS simulator. In a first step, $\star$-messages are sent to all imminents, i.e., sub-processors whose `tonie` equals `t` (line 18). Then, the coordinator waits for a `y`-message of each imminent. Outputs directed to the coordinator are stored in the current in- and outputs `m.xy` (20) of the macro model associated with the coordinator (external output couplings). Whereas, outputs directed to other children (internal couplings) are buffered in ic_{msg} (21). If the associated macro model is imminent, its λ-function will be invoked and the outputs will be merged with `m.xy` (23), which are then sent to the coordinator's parent in a `y`-message (24). As in other PDEVS variants, the coordinator waits for a `x`-message of its parent to proceed (25). The message contains, on the one hand, regular inputs `x`, i.e., outputs of other models generated by their λ-functions and directed to the coordinator, and value coupled information `xy` on the other hand. That information will be discarded, if there are no corresponding ports available in the associated macro model.

Afterwards, `x`-messages are sent to all imminent and influenced children, and those that should be downward activated referring to the function λ_{down} (31-33). Thus, `x`-messages contain the values buffered in ic_{msg} (21), inputs in `x` that are directed to the coordinator's children (external input couplings), events for activating children, and information defined by the v_{down}-function that should be propagated downwards.

Listing 1: Pseudocode of a simulator of *ml*-DEVS.

```
1  variables
2    m       // associated micro model
3    m.xy    // current inputs and outputs
4    m.state // current state
5    tole    // time of last event
6    tonie   // time of next internal event
7    x       // current external input events
8    xy      // downward propagated information
9
10 when receive *-message (t) at time t
11   m.xy = m.λ(m.state)
12   send y-message (m.id, m.xy, t) to parent
13
14 when receive x-message (id, x, xy, t) at time t with input value x
15   update m.xy according to x and xy
16   (m.state, m.xy) = m.δ(m.xy, m.state, t - tole)
17   flush(m.xy,x)
18   tole = t
19   tonie = tole + m.ta(m.state)
20   update m.xy according to m.p(m.state)
21   send done-message (m.id, m.xy, tonie) to parent
```

In contrast to Uhrmacher et al. (2007), λ_{down} is invoked after the coordinator received a `x`-message (32). Thus, a macro model can activate sub-models not only if it is imminent. After sending `x`-messages, the coordinator waits for `done`-messages of all imminent, influenced, and downward activated children (34) and $interface_C$ will be updated according to the upward propagated information from the micro level (35). Finally, it is checked whether the associated macro model has to be triggered, i.e., whether its δ- and structural change function have to be invoked (38-41). The macro model will be triggered, if there are inputs in the received `x`-message directed to it, if the model is imminent or if an invariant guarded by the activation function act_{up} is violated (39). At the end, the current in- and outputs are updated according to the active ports determined by the port selection function p (43), the `tonie` is updated (44), and a `done`-message including the public information is sent to the coordinator's parent (45).

3 APPLICATION: ACTIVITY RECOGNITION IN A SMART MEETING ROOM

The simulation model we have developed aims at producing synthetic sensor data for testing and evaluating model-based activity recognition in smart environments as described in (Krüger et al. 2012). The activity recognition system under test employs probabilistic plan recognition based on particle filter methods. We focused on forward estimation, which facilitates online recognition of user activities as basis for real-time assistance in smart environments. Thereby, the recognition system is parameterized referring to an assumed error in the sensor data, which reflects the sensor reliability. Interesting questions are, how this reliability affects the performance of the recognition system, and how the recognition systems deals with different accuracies of the actual sensors.

3.1 Description of the Simulation Model

The model is composed of four different types of models reflecting our test scenario: coffee machine, printer, sensor mat, and user, which are distinct *ml*-DEVS models with their own states and behavior. These models are situated in the environment, which is a MACRO-DEVS model. Up- and downward causation are used for indirect communication between models, e.g., those representing users and sensors. Thereby, location information of the modeled users is globally available and accessed by the sensor models via value coupling (downward information). Although the environment model keeps track of those locations, the update of locations is triggered by the user model. Coffee machine and printer are represented by simple MICRO-DEVS models. If one of them misses a certain resource, e.g., water or paper, it will signal that to

Listing 2: Pseudocode of a coordinator of *ml*-DEVS.

```
variables:
  m       // associated macro model
  m.state // current state
  m.xy    // current inputs and outputs
  m.C     // current children
  m.MC    // current multi-couplings
  IMM     // imminent children
  INF     // influenced children
  ACT     // children to activate
  x       // current external inputs
  xy      // downward propagated information
  ic_msg  // internal coupling messages

when receive *-message (t) or x-message (id, x, xy, t) at time t
  // querying all imminent models and distributing their output
  if received message is *-message (t) then
    // IMM = c ∈ m.C | tonie(c) = t
    send *-message (t) to all d ∈ IMM
      wait for y-messages (d.id,d.xy,d.t) from all d ∈ IMM
        update m.xy with y-message directed to parent of m
        update ic_msg with y-message directed to other children
    if t = tonie(m) then
      m.xy = m.xy ∪ m.λ(m.state)
    send y-message (m.id,m.xy,t) to parent
      wait for x-message (id, x, xy, t) from parent

  // execute all imminent and influenced children
  update m.xy according to received x and xy
  // INF = {c ∈ m.C | ∃x ∈ m.mc(y) ∃d ∈ m.C ∪ {m} : (x,?) ∈ c.xy ∧ (y,?) ∈ d.xy ∧ c ≠ d}
  // ACT = {c ∈ m.C | ∃(id,?) ∈ m.λ_down(m.state) : id = c.id}
  for each d ∈ IMM ∪ INF ∪ ACT do
    msg = ∅ ∪ inputs(d, ic_msg, m.xy, m.λ_down(m.state))
    send x-message (d.id, msg, m.v_down(s), t) to d
      wait for done-message (d.id, d.xy, d.tonie) from all d
        update interface_C according to d.id and d.xy

  // execute associated macro model
  msg = getMessageForMe(x)
  if msg ≠ ∅ ∨ t = tonie(m) ∨ act_up(m.state, interface_C) with interface_C = {(id,XY) | c ∈ m.C : c =< id,XY... >} then
    (m.state,m.xy) = m.δ(m.xy, m.state, t-tole(m), interface_C)
    (m.C,m.MC) = m.sc(m.state)
  flush(m.xy,x)
  update m.xy according to m.p(m.state) and interface_C
  tonie = min_{c∈m.C∪{m}}(tonie(c))
  send done-message (m.id, m.xy, tonie) to parent
```

their surroundings by changing its ports, via which those resources can then be replenished by the modeled user.

An overview of the communication structure in our simulation model is shown in Figure 1. The blue boxes represent the agents identified by their labels. Edges between ports, depicted as white boxes with arrows, represent multi-couplings between those ports used for a direct communication and interaction between the agents associated with the ports. The arrows in the ports indicate the direction of this communication and interaction. The dashed edges represent information transfer via value coupling, whereas dashed arrows refer to up- and downward activation between the user agent and environment model (*Smart Meeting Room*).

In the following, the environment, user, and sensor model are described in more detail:

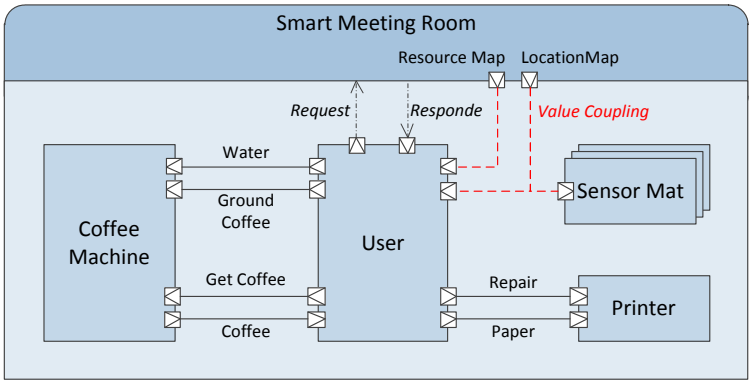


Figure 1: Simplified communication structure in the simulation model.

3.1.1 Environment Model

As mentioned before, the (smart) environment is explicitly modeled in our approach. It is represented by a MACRO-DEVS model that forms the root of the compositional hierarchy of our simulation model. This model might communicate with its surroundings via a well-defined interface (in terms of its ports). However, referring to the sub-models that are situated in it, the environment is more than merely a loosely coupled building block. It has access to information that sub-models would like to make globally available, and, in addition, it constrains interactions and controls the information flow between those sub-models. In our scenario this means that the environment model is restricting the intended movement of the user taking physical constraints into account, constraining intended interactions between user, printer, etc. according to environmental situations, and making information about locations available to the modeled sensors.

We realized two models of our smart environment: one without and one with an explicit representation of the physical environment. In the first model, user agents move from one location of interest to another by just consuming a certain time. In the second model (*Smart Environment 2*), spatial constraints of a physical environment are considered. Therefore, the state of the model contains a regular grid, in which information about locations of modeled users and their spatial extension is kept. The constraints that now apply referring to the users' activities are similar to those considered in many other multi-agent environments (Parunak 1997; Helleboogh et al. 2007). Printer, coffee-machine, and sensors are localized on the grid as well. Due to spatial constrains, moving towards the coffee machine is required to make coffee. To know this is the responsibility of the agent, however, the responsibility how to get to the coffee machine in terms of actual movement is the responsibility of the environment. This means that the model *Smart Environment 2* only expects from "its users" the ability to come up with a high-level plan, however, not the knowledge of how this plan can be achieved in this environment. In the current implementation, the path to the coffee machine or other locations is determined globally assuming that users tend to use the shortest path. Thereby, users can only move to an adjacent cell at a time, and cannot pass through "ger" cells, e.g., walls or furniture. The agent-environment communication and interaction is realized by means of up- and downward causation. Checking constraints and guarding invariants is part of the environment model's behavior.

3.1.2 User Model

At the moment, our user model is rather simple and a proof of concept, as the agent performs a predefined action sequence and does no decision making on its own. The actions a user agent can perform refer to those given in Table 1. Fetching resources, e.g., paper or water, is modeled implicitly. If the user is at a location that provides a certain resource, e.g., the paper stack or water sink, the user will receive the corresponding resource. Replenishing resources is realized as explicit interaction between the agents, by using variable ports and multi-couplings (see Figure 1). In case there is no physical representation of the

Table 1: User actions related to their locations ('-' denotes that the location of the action cannot be determined by the available sensors).

Action	Location	Action	Location
goto	-	start enter	-
end enter	door	start repair printer	printer
finish repair printer	printer	fetch paper	paper stack
fetch ground coffee	coffee jar	fetch water	water tap
replenish paper	printer	replenish ground coffee	coffee machine
replenish water	coffee machine	start exit	door
end exit	-		

user movement, the user is represented by a MICRO-DEVS model. If there is a physical environment, the user is represented by a MACRO-DEVS model that includes a locomotion sub-model. This model is responsible for requesting paths and moving along those cell by cell. However, in both cases the user agent does not update its location, but the environment does so to maintain consistency and check constraints, such that a cell is only occupied by one user at a time. As *ml*-DEVS is a general formalism and not restricted to any agent architecture, more sophisticated user agents, e.g., (Gierke and Uhrmacher 2005; Schattenberg and Uhrmacher 2001), can be realized.

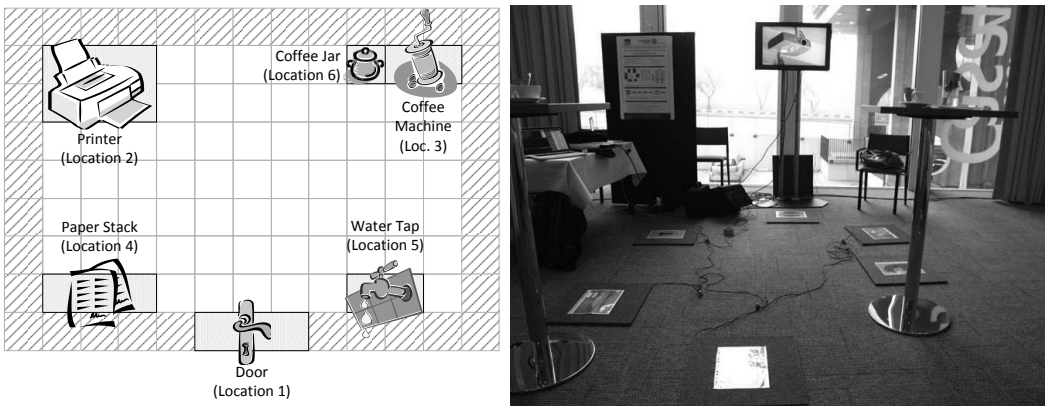
3.1.3 Sensor Model

Sensors are crucial components, as they provide the observations that are the basis for the activity recognition. We employ certain error models to produce realistic sensor readings, as context information in general and sensor readings in particular can be considered as imperfect (Henricksen and Indulska 2004). For each of the six locations of interest (see Table 1) an instance of the sensor model is created, which is customized by parameters, such as the actual location and frequency. The sensor model itself is a MACRO-DEVS model containing a sensing unit model. The macro model can activate or deactivate the sensing unit by downward activation after receiving corresponding input events or after a certain time has elapsed. The sensing unit is responsible for generating the actual output of the simulated sensor, which is then used for testing the activity recognition. The sensing process is modeled as indirect communication between user agents and the sensor models by using up- and downward causation. As mentioned before, location information is global and thus each sensor model can check whether there are any agents at its associated location or not. Instead of transmitting the ground truth, i.e., the presence or absence of agents at a location, an error model is applied on the ground truth. This error model refers to a given accuracy of the produced sensor readings.

3.2 Simulation and Results

To test the robustness of the activity recognition and evaluate its performance systematically, we produced synthetic data with a varying accuracy of the simulated sensors ranging from 0 to 100% and applied the recognizer to that data. Thereby, the accuracies for all six sensors were set globally to the same value. All other model parameters, e.g., action durations or sampling frequencies, remained unchanged for all accuracies chosen. Figure 2(a) illustrates the schematic layout of the modeled physical environment, which is considered in our model *Smart Environment 2* with the locations of interest. Each of these locations is associated with one simulated sensor. A test arrangement of real sensor mats for demonstration and on-site testing of the activity recognition is depicted in Figure 2(b).

Figures 3 and 4 show the trajectories of the simulated sensors, visualized as bichrome color maps, for single runs and the same scenario, but with an accuracy of 100% (perfect observations) and 90%. In both figures the x-axis denotes the time elapsed in simulation and the y-axis refers to the locations with which sensors are associated (see Figure 2(a)). A black patch indicates the presence of users at the corresponding location denoted at the y-axis and white areas indicate the absence of users. With decreasing sensor accuracy the degree of noise, i.e., the divergence between actual and perfect observation, is increasing.



(a) Schematic layout showing the locations of interest.

(b) Test arrangement of real sensor mats for on-site testing.

Figure 2: Physical environments.

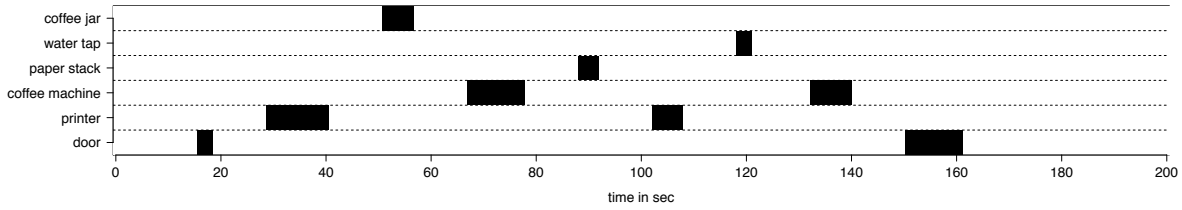


Figure 3: Output trajectories of all sensors for a simulated accuracy of 100% (0% error rate).

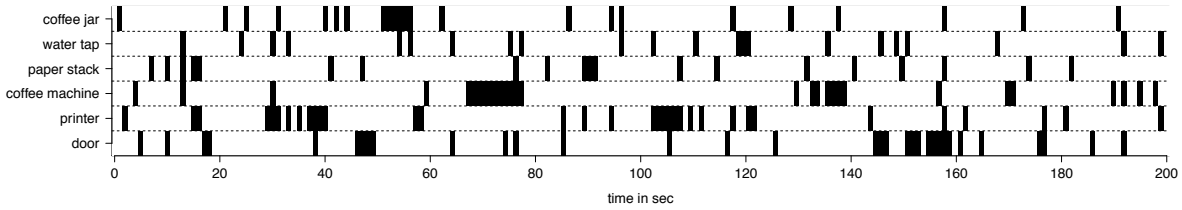


Figure 4: Output trajectories of all sensors for a simulated accuracy of 90% (10% error rate).

After producing synthetic sensor data, the activity recognition was applied to the data and the corresponding results, i.e., the recognized plan, were recorded for further analyses. Based on those results and the ground truth, i.e., the actual plan of the user agent, provided by the simulation, we started computing different performance metrics, e.g., the recognition rate, the Hamming distance (adapted for different lengths) and the Levenshtein distance (Levenshtein 1966) between the recognized and actual plan. Thereby, it became obvious that the use of a particular performance metrics has a significant impact on the evaluation. For instance, some metrics might be able to detect shifts in the plans. Also in many cases, weighting the deviations between estimated and actual plan and a weighting between different types of recognition errors is desirable. This is, for instance, possible with the dynamic time warping algorithm (Rabiner and Myers 1981) that is currently tested.

In our scenario, some of the user actions, e.g., *goto*-actions, cannot be distinguished by only using the data of the available sensors. However, the evaluation showed that the recognition system under test is able to resolve those ambiguities by using the underlying causal model for correct sensor data (accuracy of 100%). By decreasing the sensor accuracy to 90% the recognition rate reaches 92.63% for an assumed

sensor error of 10% in the activity recognition system. A further decrease of the accuracy down to 75% results in a recognition rate of still 80.36% for an assumed error of 25%. These results indicate that the recognition system under test seems to be robust referring to sensor errors to some extent. A thorough evaluation will be part of future work.

In Figure 5, the ground truth, i.e., the plan performed by the user agent during simulation, is shown in comparison to two recognized action sequences with the above mentioned sensor accuracies and assumed errors. For sake of readability, certain actions, e.g., *fetch water* and *fetch paper*, are combined and transitions between actions are shown in the figure. Because of the combination of actions, the given recognition rates are lower than Figure 5 indicates.

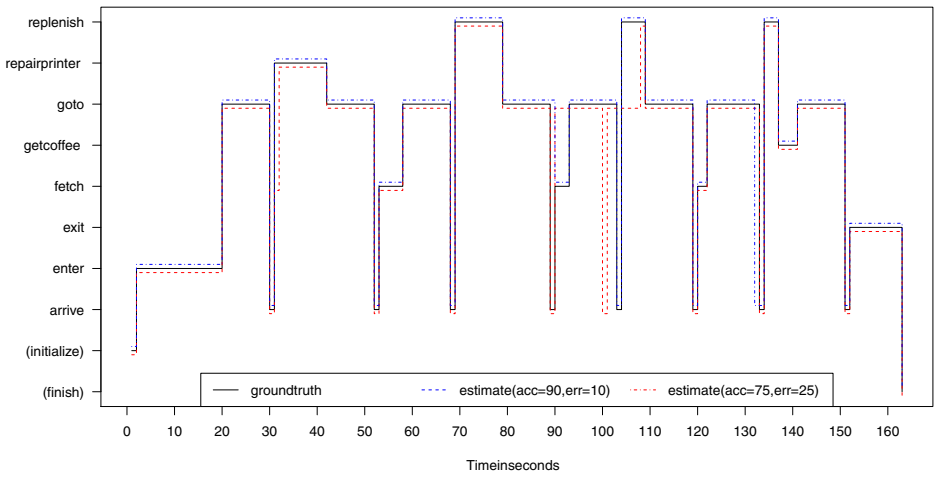


Figure 5: The solid line depicts the actual plan performed by the user agent during simulation (ground truth). The dashed lines illustrate two chosen recognized plans (estimates) for sensor accuracies of 90% and 75% and assumed errors of 10% and 25%, respectively.

4 CONCLUSION, DISCUSSION, AND OUTLOOK

We explored and revised *ml*-DEVS, a modeling formalism originally designed to support multi-level modeling (Uhrmacher et al. 2007), to model and simulate agents situated in their environment and the interaction between them. The revision includes, no separation between in- and outputs, the propagation of values upwards the compositional hierarchy, and a clear separation between private and public information of components, i.e., the introduction of component interfaces that are accessible by the macro level. Now, *ml*-DEVS allows an explicit modeling of the environment and its role in providing information and enforcing constraints for the situated agents. Therefore, the formalism supports representing the environment as first-order abstraction and explicit "building block" of multi-agent systems (Weyns et al. 2005), which encapsulates clearly defined aspects and responsibilities that differ from those of the situated agents (Weyns et al. 2005).

Multi-Level-DEVS has been realized in the modeling and simulation framework JAMES II (Java-based Multipurpose Environment for Simulation II (Himmelspach and Uhrmacher 2007)) and has been put to test in a small simulation study. The question we pursue is what influence has the sensor error rate on the performance of a system that is aimed at recognizing users' activities in a smart environment and that can be parameterized with an assumed error in the actual sensor data. This study, although still quite simple and in progress, has shown to be a valuable complement to on-site testing, as extensive test data can be produced systematically and in a controlled manner. Next we will analyze the influence of heterogeneous sensor errors, i.e., certain sensors have a different accuracy than the rest, and sensor failures on the activity

recognition. In addition, other model parameters, e.g., the action durations, and the action sequence itself shall be varied to create more realistic and complex simulation experiments. The implications of different metrics shall be analyzed as well.

The ingredients that have proven of benefit in our simulation study and that we deem of general importance are: nested models with a dynamic behavior of their own at each level, dynamic (yet explicitly defined) model interfaces, an intensional definition of interactions between models, and a combination of event processing, value couplings, and invariants to tie the different levels of nesting.

However, model descriptions in *ml*-DEVS are still rather verbose. Thus, we pursue ideas of a component-based model design to decrease the effort of creating models. To facilitate the development of models by reusing predefined model components, we are currently extending COMO (Component-based Modeling (Röhl and Uhrmacher 2008)), a component modeling and analysis framework, to support variable structures and *ml*-DEVS models in particular. COMO separates interface and composition descriptions from the actual implementation of model components and thus allows composing models solely based on this descriptions and check compositions for inconsistencies before executing them. Thereby, we will pursue a similar approach as (da Silva and de Melo 2011), which also focuses on the interactions between agents and between agents and their environment. However, whereas da Silva and de Melo (2011) do not take the implementation of the agents into account, we aim at achieving both by exploiting COMO in combination with *ml*-DEVS.

ACKNOWLEDGMENTS

This research is supported by the German Research Foundation (DFG) within the context of the project MuSAMA (Multimodal Smart Appliance Ensembles for Mobile Applications).

REFERENCES

Barros, F. J. 1997. "Modeling Formalisms for Dynamic Structure Systems". *ACM Transactions on Modeling and Computer Simulation* 7 (4): 501–515.

Campbell, D. T. 1974. "Downward causation in hierarchically organised biological systems". In *Studies in the philosophy of biology: reduction and related problems*, edited by F. J. Ayala and T. G. Dobzhansky, 179–186. Berkeley: University of California Press.

da Silva, P., and A. C. V. de Melo. 2011. "A Formal Environment Model for Multi-Agent Systems". In *Formal Methods: Foundations and Applications*, edited by J. Davies, L. Silva, and A. Simao, Volume 6527 of *Lecture Notes in Computer Science*, 64–79. Berlin, Heidelberg: Springer-Verlag Berlin Heidelberg.

Gierke, M., J. Himmelspach, M. Röhl, and A. M. Uhrmacher. 2006, September. "Modeling and Simulation of Tests for Agents". In *Multiagent System Technologies*, edited by K. Fischer, I. J. Timm, E. André, and N. Zhong, Volume 4196 of *Lecture Notes in Computer Science*, 49–60. Berlin, Germany: Springer-Verlag.

Gierke, M., and A. M. Uhrmacher. 2005. "Modeling Elderly Behavior for Simulation-based Testing on Agent Software". In *Proceedings of the Conceptual Modeling and Simulation Conference*, edited by F. Barros, A. Bruzzone, C. Frydman, and N. Giambiasi, 159–164: SCS.

Helleboogh, A., G. Vizzari, A. M. Uhrmacher, and F. Michel. 2007. "Modeling dynamic environments in multi-agent simulation". *Autonomous Agents and Multi-Agent Systems* 14 (1): 87–116.

Henricksen, K., and J. Indulska. 2004, March. "Modelling and Using Imperfect Context Information". In *Proceedings of the Second IEEE Annual Conference on Pervasive Computing and Communications Workshops*, edited by S. Das, M. Kumar, and C. Becker, 33–37. Los Alamitos, CA, USA: IEEE: IEEE Computer Society.

Himmelspach, J., and A. M. Uhrmacher. 2007, March. "Plug'n simulate". In *Proceedings of the 40th Annual Simulation Symposium*, edited by H. Karatza and T. F. Znati, 137–143. Washington, DC, USA: IEEE Computer Society.

Krüger, F., K. Yordanova, C. Burghardt, and T. Kirste. 2012, May. "Towards Creating Assistive Software by Employing Human Behavior Models". *Journal of Ambient Intelligence and Smart Environments* 4 (3): 209–226.

Levenshtein, V. I. 1966. "Binary Codes Capable of Correcting Deletions, Insertions and Reversals". *Soviet Physics Doklady* 10 (8): 707–710.

Maus, C., S. Rybacki, and A. M. Uhrmacher. 2011. "Rule-based multi-level modeling of cell biological systems". *BMC Systems Biology* 5 (166).

Parunak, H. V. D. 1997. ""Go to the ant": Engineering principles from natural multi-agent systems". *Annals of Operations Research* 75:69–101.

Platon, E., M. Mamei, N. Sabouret, S. Honiden, and H. V. D. Parunak. 2006, August. "Mechanisms for environments in multi-agent systems: Survey and opportunities". *Autonomous Agents and Multi-Agent Systems* 14 (1): 31–47.

Rabiner, L. R., and C. S. Myers. 1981, September. "A comparative study of several dynamic time-warping algorithms for connected word recognition". *The Bell System Technical Journal* 60 (7): 1389–1409.

Röhl, M., and A. M. Uhrmacher. 2008, December. "Definition and Analysis of Composition Structure for Discrete-Event Models". In *Proceedings of the 2008 Winter Simulation Conference*, edited by S. J. Mason, R. R. Hill, L. Moench, O. Rose, T. Jefferson, and J. W. Fowler, 942–950. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Schattenberg, B., and A. M. Uhrmacher. 2001. "Planning agents in JAMES". *Proceedings of the IEEE* 89 (2): 158–173.

Uhrmacher, A. M., R. Ewald, M. John, C. Maus, M. Jeschke, and S. Biermann. 2007, December. "Combining Micro and Macro-Modeling in DEVS for Computational Biology". In *Proceedings of the 2007 Winter Simulation Conference*, edited by S. G. Henderson, B. Biller, M.-H. Hsieh, J. Shortle, J. D. Tew, and R. R. Barton, 871–880. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Weyns, D., M. Schumacher, A. Ricci, M. Viroli, and T. Holvoet. 2005, January. "Environments in multiagent systems". *The Knowledge Engineering Review* 20 (02): 127.

Weyns, D., H. Van Dyke Parunak, F. Michel, T. Holvoet, and J. Ferber. 2005. "Environments for Multiagent Systems State-of-the-Art and Research Challenges". In *Environments for Multi-Agent Systems*, edited by D. Weyns, H. Dyke Parunak, and F. Michel, Volume 3374 of *Lecture Notes in Computer Science*, 1–47. Berlin, Heidelberg: Springer Berlin Heidelberg.

Wooldridge, M. 2009, May. *An Introduction to MultiAgent Systems*. 2nd ed. Chichester, UK: John Wiley & Sons.

Zeigler, B. P., H. Praehofer, and T. G. Kim. 2000, January. *Theory of Modeling and Simulation*. 2nd ed. San Diego, CA, USA: Academic Press.

AUTHOR BIOGRAPHIES

ALEXANDER STEINIGER is PhD student at the Institute of Computer Science at the University of Rostock. He is interested in a component-based design of environments for a simulation-based testing of agents. His email address is alexander.steiniger2@uni-rostock.de.

FRANK KRÜGER is a PhD student at the Institute of Computer Science at the University of Rostock. He is interested in a goal-based interactions and probabilistic plan recognition. His email address is frank.krueger2@uni-rostock.de.

ADELINDE M. UHRMACHER is professor at the Institute of Computer Science at the University of Rostock and head of the research group Modeling and Simulation. Her research interests are in developing effective modeling and simulation methods, particularly for multi-level, dynamic structure systems, and applying those methods in areas like smart environments, demography, and computational biology. Her email address is adelinde.uhrmacher@uni-rostock.de.