LEADSTO: A Language and Environment for Analysis of Dynamics by SimulaTiOn

Tibor Bosse[1], Catholijn M. Jonker[2], Lourens van der Meij[1], and Jan Treur[1]

[1] Vrije Universiteit Amsterdam, Department of Artificial Intelligence,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
{tbosse, lourens, treur}@cs.vu.nl
http://www.cs.vu.nl/~{tbosse, lourens, treur}
[2] Nijmegen Institute for Cognition and Information, Division Cognitive Engineering,
Montessorilaan 3, 6525 HR Nijmegen, The Netherlands
C.Jonker@nici.kun.nl

Abstract. This paper presents the language and software environment LEADSTO that has been developed to model and simulate the dynamics of Multi-Agent Systems (MAS) in terms of both qualitative and quantitative concepts. The LEADSTO language is a declarative order-sorted temporal language, extended with quantitative means. Dynamics of MAS can be modelled by specifying the direct temporal dependencies between state properties in successive states. Based on the LEADSTO language, a software environment was developed that performs simulations of LEADSTO specifications, generates simulation traces for further analysis, and constructs visual representations of traces. The approach proved its value in a number of projects within different domains of MAS research.

1 Introduction

Two important phases in the development of Multi-Agent Systems are the Design phase and the Implementation phase. In principle, the result of the Design phase is a high-level description (a model) of the system to be developed which, when encoded in some programming language, solves a particular problem. To this end, the problem is decomposed into modules, of which the functions and interfaces are specified in detail [10]. Then, the result of the Design phase, the (technical) specification, can serve as a starting point for the Implementation phase. However, an important problem is the validation of this specification: can it be proven that the specification shows the expected behaviour (e.g. as described by requirements) before it is actually implemented? Especially when the specification is given in terms of abstract high-level concepts this is a non-trivial task.

To contribute to the validation of Multi-Agent System specifications, this paper introduces the language and software environment LEADSTO. LEADSTO can be used to model the *dynamics* of systems to be designed, on the basis of highly abstract process descriptions. If those dynamics are modelled correctly, the LEADSTO software environment can use them for *simulation* of the desired behaviour of the system. Although such simulations are no formal proof, they can contribute to an informal vali-

T. Eymann et al. (Eds.): MATES 2005, LNAI 3550, pp. 165 – 178, 2005.

dation of the specification: by performing a number of simulations, it can be tested whether the behaviour of the specification is satisfactory. Therefore, LEADSTO may be an important tool to bridge the gap between the Design and the Implementation phase.

Generally, in simulations various formats are used to specify basic mechanisms or causal relations within a process, see e.g., [1], [5], [9]. Depending on the domain of application such basic mechanisms need to be formulated quantitatively or qualitatively. Usually, within a given application explicit boundaries can be given in which the mechanisms take effect. For example, "from the time of planting an avocado pit, it takes 4 to 6 weeks for a shoot to appear".

As mentioned above, in order to simulate a system to be designed, it is important to model its *dynamics*. When considering current approaches to modelling dynamics, the following two classes can be identified: *logic-oriented* modelling approaches, and *mathematical* modelling approaches, usually based on difference or differential equations. Logic-oriented approaches are good for expressing qualitative relations, but less suitable for working with quantitative relationships. Mathematical modelling approaches (e.g., Dynamical Systems Theory [9]), are good for the quantitative relations, but expressing conceptual, qualitative relationships is very difficult. In this article, the LEADSTO language (and software environment) is proposed as a language combining the specification of qualitative and quantitative relations.

In Section 2, the LEADSTO language is introduced. Section 3 provides examples from existing case studies in which LEADSTO has been applied. Section 4 describes the tools that support the LEADSTO modelling environment in detail. In particular, the LEADSTO Property Editor and the LEADSTO Simulation Tool are discussed. Section 5 compares the approach to related modelling approaches, and Section 6 is a conclusion.

2 Modelling Dynamics in LEADSTO

Dynamics can be modelled in different forms. Based on the area within Mathematics called calculus, the Dynamical Systems Theory (DST) [9] advocates to model dynamics by continuous state variables and changes of their values over time, which is also assumed continuous. In particular, systems of differential or difference equations are used. This may work well in applications where the world states can be modelled in a quantitative manner by real-valued state variables and the world's dynamics shows continuous changes in these state variables that can be modelled by mathematical relationships between real-valued variables.

Not for all applications dynamics can be modelled in a quantitative manner as required for DST. Sometimes qualitative changes form an essential aspect of the dynamics of a process. For example, to model the dynamics of reasoning processes in Intelligent Agents usually a quantitative approach will not work. In such processes states are characterised by qualitative state properties, and changes by transitions between such states. For such applications often qualitative, discrete modelling approaches are advocated, such as variants of modal temporal logic; e.g., [6]. However, using such non-quantitative methods, the more precise timing relations are lost too.

For the approach used in this paper, it was decided to consider time as continuous, described by real values, but to allow both quantitative and qualitative state properties. The approach subsumes approaches based on simulation of differential or difference equations, and discrete qualitative modelling approaches, but also combines them. For example, it is possible to model the exact (real-valued) time interval for which some qualitative property holds. Moreover, the relationships between states over time are described by either logical or mathematical means, or a combination thereof. This is explained below in more detail.

Dynamics is considered as evolution of states over time. The notion of state as used here is characterised on the basis of an ontology defining a set of properties that do or do not hold at a certain point in time. For a given (order-sorted predicate logic) ontology Ont, the propositional language signature consisting of all *state ground atoms* (or *atomic state properties*) based on Ont is denoted by APROP(Ont). The *state properties* based on a certain ontology Ont are formalised by the propositions that can be made (using conjunction, negation, disjunction, implication) from the ground atoms. A *state* S is an indication of which atomic state properties are true and which are false, i.e., a mapping S: APROP(Ont) → {true, false}.

To specify simulation models a temporal language has been developed. This language (the LEADSTO language) enables one to model direct temporal dependencies between two state properties in successive states, also called *dynamic properties*. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can often easily be depicted graphically. The format is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In the LEADSTO language the notation $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$ (also see Figure 1), means:

If state property α *holds for a certain time interval with duration g, then after some delay (between e and f) state property* β *will hold for a certain time interval of length h.*

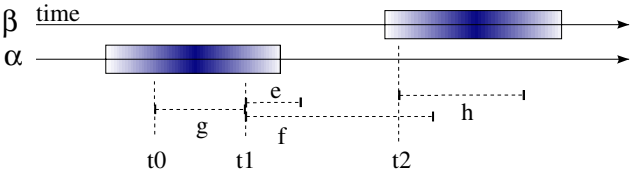

Fig. 1. The timing relationships

An example dynamic property that uses the LEADSTO format defined above is the following: "observes(agent_A, food_present) $\rightarrow_{2, 3, 1, 1.5}$ belief(agent_A, food_present)". Informally, this example expresses the fact that, if agent A observes that food is present during 1 time unit, then after a delay between 2 and 3 time units, agent A will believe that food is present during 1.5 time units. In addition, within the LEADSTO language it is possible to use sorts, variables over sorts, real numbers, and mathematical operations, such as in "has_value(x, v) $\rightarrow_{e, f, g, h}$ has_value(x, v*0.25)".

Next, a *trace* or *trajectory* γ over a state ontology Ont is a time-indexed sequence of states over Ont (where the time frame is formalised by the real numbers). A LEADSTO expression $\alpha \rightarrow\!\!\!\!\rightarrow_{e, f, g, h} \beta$, holds for a trace γ if:

$\forall t1$: [$\forall t$ [t1–g ≤ t < t1 ⇒ α holds in γ at time t] ⇒ $\exists d$ [e ≤ d ≤ f & $\forall t'$ [t1+d ≤ t' < t1+d+h ⇒ β holds in γ at time t']

An important use of the LEADSTO language is as a specification language for simulation models. As indicated above, on the one hand LEADSTO expressions can be considered as logical expressions with a declarative, temporal semantics, showing what it means that they hold in a given trace. On the other hand they can be used to specify basic mechanisms of a process and to generate traces, similar to Executable Temporal Logic (cf. [1]).

Finally, the LEADSTO format can be graphically depicted in a causal graph-like format, such as in Figure 2. Here, state properties are indicated by circles and LEADSTO relationships by arrows. An arc denotes a conjunction between state properties. Agents are indicated by dotted boxes. Circles that are depicted within an agent denote its internal (mental) state properties. Circles that are depicted on the left or right border of an agent denote, respectively, its input and output state properties, and circles that are depicted outside an agent denote state properties of the external world. Notice that this simple form leaves out the timing parameters e, f, g, h. A more detailed form can be obtained by placing the timing parameters in the picture as labels for the arrows. For more details about the LEADSTO language, see Section 4.

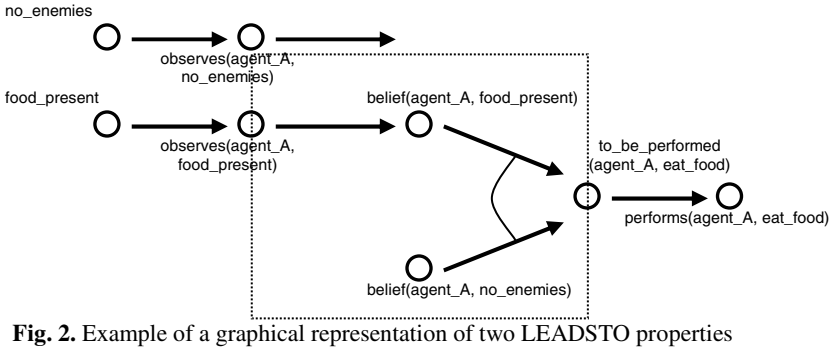

Fig. 2. Example of a graphical representation of two LEADSTO properties

3 Applications

The LEADSTO environment has been applied in a number of research projects in different domains. In this section, an example LEADSTO specification is given for a specific domain: a Multi-Agent System for ant behaviour, adopted from [3]. The world in which the ants live is described by a labeled graph as depicted in Figure 3. Locations are indicated by A, B,…, and edges by E1, E2,… The ants move from location to location via edges; while passing an edge, pheromones are dropped. The objective of the ants is to find food and bring this back to their nest. In this example there is only one nest (at location A) and one food source (at location F).

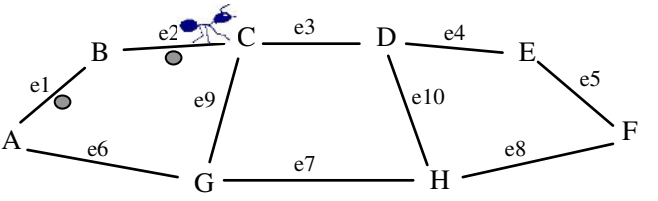

Fig. 3. An ants world

In [3], the dynamics of this system are formalised in LEADSTO, and some simulations are shown for different situations. A number of LEADSTO expressions that have been used for the simulation are shown in Box 1. For the complete specification, see [3].

In Figure 4 an example of a resulting simulation trace is shown. The upper part of the figure shows qualitative information; the lower part shows quantitative information. Time is on the horizontal axis. In the upper part, the state properties are on the vertical axis. Here, a dark box on top of the line indicates that the property is true during that time period, and a lighter box below the line indicates that the property is false. For example, the state property to_be_performed(ant2, pick_up_food) is true from time point 20 to 21. Because of space limitations, only a selection of important state properties was depicted. In the lower part, different instantiations of state property pheromones_at_E1(X) are shown, with different (real) values for X. For example, from time point 1 to 7 the amount of pheromones on E1 is 0.0.

LP5 (Selection of Edge)

This property models (part of) the edge selection mechanism of the ants. It expresses that, when an ant a observes that it is at location l coming from edge e0, and there are two other edges connected to that location, then the ant goes to the edge with the highest amount of pheromones. Formalisation:

observes(a, is_at_location_from(l, e0)) and neighbours(l, 3) and connected_to_via(l, l1, e1) and observes(a, pheromones_at(e1, i1)) and connected_to_via(l, l2, e2) and observes(a, pheromones_at(e2, i2)) and e0 ≠ e1 and e0 ≠ e2 and e1 ≠ e2 and i1 > i2 $\rightarrow\!\!\!\!\rightarrow_{0,0,1,1}$ to_be_performed(a, go_to_edge_from_to(e1, l1))

LP9 (Dropping of Pheromones)

This property expresses that, if an ant observes that it is at an edge e from a location l to a location l1, then it will drop pheromones at this edge e. Formalisation:

observes(a, is_at_edge_from_to(e, l, l1)) $\rightarrow\!\!\!\!\rightarrow_{0,0,1,1}$ to_be_performed(a, drop_pheromones_at_edge_from(e, l))

LP13 (Increment of Pheromones)

This property models (part of) the increment of the number of pheromones at an edge as a result of ants dropping pheromones. It expresses that, if an ant drops pheromones at edge e, and no other ants drop pheromones at this edge, then the new number of pheromones at e becomes i*decay+incr. Here, i is the old number of pheromones, decay is the decay factor, and incr is the amount of pheromones dropped. Formalisation:

to_be_performed(a1, drop_pheromones_at_edge_from(e, l1)) and ∀l2 not to_be_performed(a2, drop_pheromones_at_edge_from(e, l2)) and ∀l3 not to_be_performed(a3, drop_pheromones_at_edge_from(e, l3)) and a1 ≠ a2 and a1 ≠ a3 and a2 ≠ a3 and pheromones_at(e, i) $\rightarrow\!\!\!\!\rightarrow_{0,0,1,1}$ pheromones_at(e, i*decay+incr)

Box 1. Example LEADSTO specification

Although this picture provides a very simple example (involving only three ants), it demonstrates the power of LEADSTO to combine (real-valued) quantitative concepts with (conceptual) qualitative concepts.

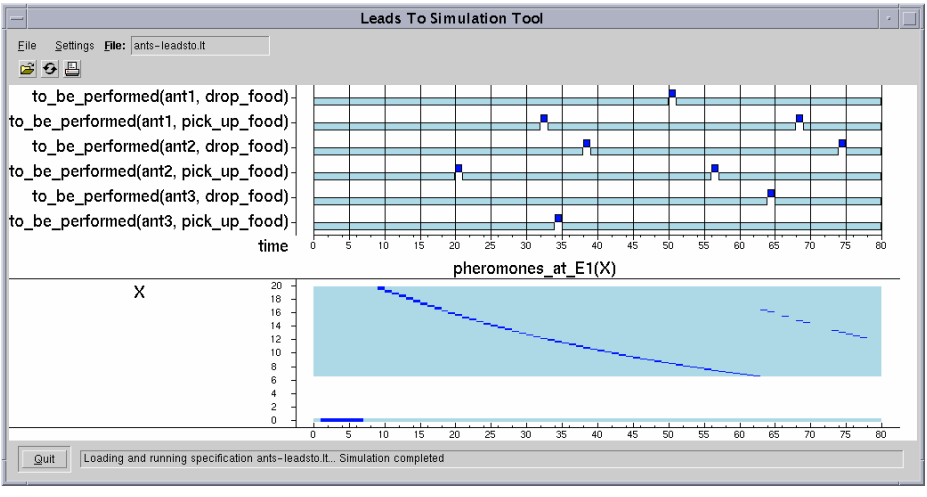

Fig. 4. Example simulation trace

Thus, Figure 4 shows an easy to read (important for the communication with the domain expert), compact, and executable representation of an informal model for ant behaviour. Moreover, the example demonstrates the power of conceptual modelling based on highly abstract process descriptions. In less than 3 pages of code, the global dynamics of ant behaviour are so well defined that the specification actually runs. The specification took only a couple of days to construct, making the LEADSTO approach valuable for proof-of-concept simulations, thus important for Agent-Oriented Software Engineering.

Finally, note that the ant example does not fully exploit the power of to use real-valued time parameters (in fact, most of the rules use the values 0,0,1,1 for the parameters e, f, g, h, see Box 1). Nevertheless, in a number of other domains the use of real-valued time parameters turned out to be beneficial, since it allows for more realistic simulations of dynamic processes. An example domain where this was the case, is the domain of adaptive agents based on classical conditioning, see [2].

4 Tools

In this section, the LEADSTO software environment is presented. Basically, this environment consists of two programs: the *Property Editor* (a graphical editor for constructing and editing LEADSTO specifications) and the *Simulation Tool* (for performing simulations of LEADSTO specifications, generating data-files containing traces for further analysis, and showing traces). Apart from the LEADSTO language constructs introduced in Section 2 the LEADSTO software has a number of other

language constructs. Section 4.1 discusses some details. Next, Section 4.2 introduces the Property Editor and Section 4.3 deals with the Simulation Tool. Section 4.4 describes the algorithm used to generate simulations. Finally, Section 4.5 provides some implementation details and discusses possible improvements for the future.

4.1 Details of the LEADSTO Language

There are various representations of LEADSTO specifications. A graphical representation is shown in Section 4.2 when discussing the Editor. In this section all language constructs are discussed using a formal representation, based on the way specifications are stored.

Variables. The language uses typed variables in various constructs. A variable is represented as <Var-Name>:<Sort>.

Sorts. Sorts may be defined as a set of instances that may be specified: sortdef(<Sort-Name>, [<Term>,...]). There are also built-in sorts such as integer, real, and ranges of integers represented as for example between(2,10).

Atoms. Atoms may be terms built up from names with argument lists where each argument must be a term or a variable, for example: belief(x:AGENT, food_present).

LEADSTO rules. LEADSTO rules are introduced in Section 2. They are represented as:

leadsto([<Vars>,] <Antecedent-Formula>, <Consequent-Formula>, <Delay>, where
<Delay> := efgh(<E-Range>,<F-Range>, <G-Range>,<H-Range>))[1]
<Vars> := "[" <Variable>,... "]"

For example, $\alpha \rightarrow\!\!\!\!\rightarrow_{0,\,0,\,1,\,1} \beta$ is represented as leadsto(alfa, beta, efgh(0,0,1,1)). Variables occurring in LEADSTO rules must be explicitly declared as <Variable> entries.

Formulae. LEADSTO rules contain formulae. The current implementation allows conjunctions and universal quantification over typed variables. Some variables are global, encompassing the whole rule. Other - local - variables are part of universal quantification of some conjunction. The first kind of variables may be of infinite types. Currently, local variables must be of finite types. Some of these restrictions – such as on not allowing disjunction – will be removed in a next version. This will have no effect on the performance of the algorithm discussed in Section 4.4, but will make the details of the algorithm more complex. Other restrictions with respect to variables of infinite type will remain.

Time/Range. Time and Range values occurring in LEADSTO rules and interval constructs may be any number or expression evaluating to a number.

Constants. Constants may be defined using the following construct: constant(<Name>, <Value>). A constant(C1, a(1)) entry in a specification will lead to C1 being substituted by a(1) everywhere in the specification.

Intervals. During simulation, some atom values will be derived from LEADSTO rules. Others are not defined by rules but represent constant values of atoms over a certain time range. They are expressed as: interval([<Vars>,]<Range>,<LiteralConjunction>).

[1] The reason for grouping the delay is to make it easier to use delay constants.

Periodically reoccurring constant values are represented as: periodic([<Vars>,]<Range>,<Period>,<LiteralConjunction>), where

<Range> := range(<Start-Time>,<End-Time>)
<Vars> := "[" <Variable>,... "]"
<Period> : an expression or constant or variable representing a number.
<LiteralConjunction> := <Literal> { and <Literal> }*
<Literal> := <Atom> | not <Atom>

For example, an entry interval([X:between(1,2)], range(10,20), a(X)) makes a(1) and a(2) true in the time range (10,20). Likewise, an entry periodic(P, range(0,1), 10) makes P true in time ranges (0,1), (10,11), (20,21), and so on.

Simulation Range. The time range over which the simulation must be run is expressed by means of the constructs start_time(<Time>) and end_time(<Time>).

Visualisation of Traces. The construct display(<Tag-Name>, <Property>) is used to specify details of how to display the traces. The <Tag-Name> argument makes it possible to define multiple views of a trace. The active view may be specified from within the User Interface of the Simulation Tool. A number of properties may be specified, for showing or hiding certain atoms, for sorting atoms, for grouping atoms into a graph, and so on.

4.2 Property Editor

The Property Editor provides a user-friendly way of building and editing LEADSTO specifications. It was designed in particular for laymen and students. The tool has been used successfully by students with no computer science background and by users with little computer experience. By means of graphical manipulation and filling

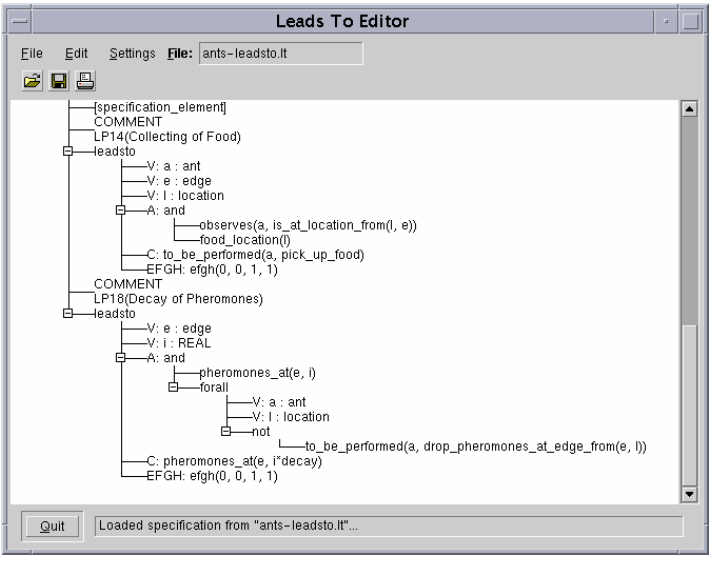

Fig. 5. The LEADSTO Property Editor

in of forms a LEADSTO specification may be constructed. The end result is a saved LEADSTO specification file, containing entries discussed in section 4.1. Figure 5 gives an example of how LEADSTO specifications are presented and may be edited with the Property Editor. This screenshot corresponds to (part of) the specification given in Box 1.

4.3 Simulation Tool

Figure 6 gives an overview of the Simulation Tool and its interaction with the LEADSTO Property Editor.

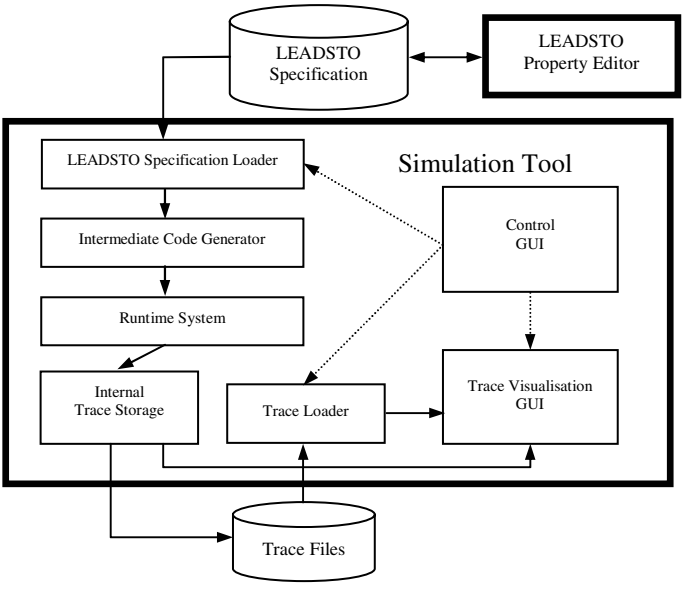

Fig. 6. Simulation Tool Architecture

The bold rectangular borders define the separate tools. The lines with arrows represent data transport; the dashed arrows represent control. The Property Editor is used to generate and store LEADSTO specification files. The Simulation Tool loads these specification files. The overall control of the Simulation Tool is handled by the *Control-GUI* component. The Simulation Tool can perform the following activities:

- Loading LEADSTO specifications, performing a simulation and displaying the result.
- Loading and displaying existing traces (without performing simulation).
- Adjusting the visualisation of traces.

Loading and simulating a LEADSTO specification is handled in four steps:

1. The *Specification Loader* loads the specification.
2. The *Intermediate Code Generator* initialises the trace situation with values defined by interval and periodic entries in the specification. The LEADSTO rules

are preprocessed: constants are substituted, universal quantifications are expanded and the rules are partially compiled into Prolog calls.

3. The actual simulation is performed by the *Runtime System*. This is the part that contains the algorithm, discussed in the next section.
4. At the end of a simulation the result is stored internally by the *Internal Trace Storage* component. The result can be saved as a trace file containing the evolution over time of truth values of all atoms occurring in the simulation, and will be visualised by the *Trace Visualisation GUI*. In principle, traces are three-valued, using the truth values true, false, and unknown. Saved trace files can be inspected later by the simulation tool and can be used by other tools, e.g., for automated analysis.

Note that visualisation of traces is integrated into the Simulation Tool through the *Trace Visualisation GUI* component. It is possible to select what atoms must be shown and in what order (sorting) etc. Figure 4 is an example of the visualisation of the result of a simulation.

4.4 Simulation Engine Algorithm

In this section a sketch of the simulation algorithm is given. The core of the semantics is determined by the LEADSTO rules, for example leadsto(alpha,beta, efgh(e, f, g, h)) or (in the notation of Section 2) $\alpha \twoheadrightarrow_{e, f, g, h} \beta$. The state properties α, β are internally normalised. Currently, only state properties that can be simplified to conjunctions of literals are allowed.

Restrictions on delays

The parameters g and h are time intervals, they must be >= 0. The algorithm allows only causal rules, e,f >= 0. Allowing e,f < 0 would lead to non-causal behaviour (any trace situation could have an effect arbitrarily in the past) and an awkward simulation algorithm. The causal nature of the semantics of LEADSTO rules results in a straightforward algorithm: at each time point, a bound part of the past of the trace (the maximum of all g values of all rules) determines the values of a bound range of the future trace (the maximum of f + h over all LEADSTO rules).

Outline of the algorithm

First all interval and periodic entries are handled by setting the ranges of atoms according to their definition. Next, for the algorithm a time variable HandledTime is introduced: all LEADSTO rules with antecedent range up to HandledTime have fired. The idea is to propagate HandledTime until HandledTime >= EndTime[2] via the following steps:

1. At a certain HandledTime, a value for NextTime is calculated. This will be the first time in the future after HandledTime that firing of a LEADSTO rule with its g-interval (see Figure 1) extending past HandledTime may have effect in the form of some consequent atom set. The time increment will be at least as big as the minimum of e + h over all LEADSTO rules.
2. An (optional) Closed World Assumption is performed for all selected atoms in the range (HandledTime, NextTime), i.e., all unknown atoms in this range are made false.

[2] EndTime is the time up to which the simulation should be run.

3. All LEADSTO rules are applied for which the range of the antecedent ends before or overlaps with NextTime.
4. Set HandledTime := NextTime
5. Continue with step 1 until HandledTime >= EndTime

4.5 Implementation Details

The complexity of the current algorithm is proportional to the number of LEADSTO rules in the specification, to the number of incremental time steps of the algorithm (which is at most equal to the length of the simulation divided by the minimum of e + h over all LEADSTO rules) and (at most) to the number of matching antecedent atoms per LEADSTO rule (limited by the number of atoms set during the simulation). A number of optimizations already improve the performance, such as only considering antecedent atoms that have matching values in the (HandledTime, NextTime) time range and not considering LEADSTO rules that have been tested to not fire until some time in the future.

The software was written in SWI-Prolog/XPCE, and consists of approximately 20000 lines of code. The approach for the design and implementation has been to first focus on a complete implementation that is easily adaptable, with acceptable performance for the current users. For an impression of the performance: the simulation of Section 3 took two seconds on a regular Personal Computer. More complex LEADSTO simulations have been created that take about half an hour to run. For example: one simulation with 170 LEADSTO rules, 2000 time steps, with 15000 atoms set, took 45 minutes.

There is room for further performance improvement of the algorithm. One possible improvement is to increase the time increment NextTime – HandledTime introduced in the algorithm above. Global analysis of dependency of LEADSTO rules should improve the performance, for instance by trying to eliminate simple rules with small values of their e + h parameters. Furthermore, the LEADSTO language is being extended with constructs for probabilistic rules, and with constructs for systematically generating traces of LEADSTO specifications for a range of parameters.

5 Related Work

In the literature, a number of modelling approaches exist that have similarities to the approach discussed in this paper. Firstly, there is the family of approaches based on differential or difference equations (see, e.g., [9]). In these approaches, to simulate processes by mathematical means, difference equations are used, for example, of the form: $\Delta x = f(x)\,\Delta t$ or $x(t + \Delta t) = x(t) + f(x(t))\,\Delta t$. This can be modelled in the LEADSTO language as follows (where d is Δt): has_value(x, v) $\rightarrow\!\!\!\!\rightarrow_{d, d, d, d}$ has_value(x, v+f(v)*d). This shows how the LEADSTO modelling language subsumes modelling approaches based on difference equations. In addition to those approaches the LEADSTO language allows to express qualitative and logical aspects.

Another modelling approach, Executable Temporal Logic [1], is based on temporal logic formulae of the form $\varphi \,\&\, \chi \Rightarrow \psi$, where φ is a past formula, χ a present formula and ψ a future formula. In comparison to this format, the LEADSTO format is more

expressive in the sense that it allows order-sorted logic for state properties, and allows one to express quantitative aspects. Moreover, the explicitly expressed timing parameters go beyond Executable Temporal Logic. On the other hand, within Executable Temporal Logic it is allowed to refer to different past states at different points in time, and thus to model more complex relationships over time. For the LEADSTO language the choice has been made to model only the basic mechanisms of a process (e.g., the direct causal relations), like in modelling approaches based on difference equations, and not to model the more complex mechanisms.

The Duration Calculus [11] is a modal logic for describing and reasoning about the real-time behaviour of dynamic systems, where states change over time and are represented by functions from time (reals) to the Boolean values (0 and 1). It is an extension of Interval Temporal Logic [7], but with continuous time, and uses integrated durations of states as interval temporal variables. Assuming finite variability of state functions (i.e., between any two time points only a finite number of state changes occurs), the axioms and rules of Duration Calculus constitute a complete logic (relative to Interval Temporal Logic). A number of interesting tools have been created around (subsets of) Duration Calculus, see, e.g., [8] for information on model checking duration calculus formulae. Duration Calculus itself is not directly used for creating executable models, but environments for executable code exist (e.g., PLC automata, see [4]) for which a semantics is given in Duration Calculus.

Another family of modelling approaches based on causal relations is the class of *qualitative reasoning* techniques (see, e.g., [5]). The main idea of these approaches is to represent quantitative knowledge in terms of abstract, qualitative concepts. Like the LEADSTO language, qualitative reasoning can be used to perform simulation. A difference with LEADSTO is that it is a purely qualitative approach, and that it is less expressive with respect to temporal and quantitative aspects.

6 Conclusion

This article presents the language and software environment LEADSTO that has been developed to model and simulate the dynamics of Multi-Agent Systems on the basis of highly abstract process descriptions. If those dynamics are modelled correctly, the LEADSTO software environment can use them for simulation of the desired behaviour of the system. Although such simulations are no formal proof, they can contribute to an informal validation of the specification: by performing a number of simulations, it can be tested whether the behaviour of the specification is satisfactory. Therefore, LEADSTO may be an important tool to bridge the gap between the Design and the Implementation phase.

Within LEADSTO, dynamics can be modelled in terms of both qualitative and quantitative concepts. It is, for example, possible to model differential and difference equations, and to combine those with discrete qualitative modelling approaches. Existing languages are either not accompanied by a software environment that allows simulation of the model, or do not allow the combination of both qualitative and quantitative concepts.

The language LEADSTO is a declarative order-sorted temporal language extended with quantitative notions (like integer, and real). Time is considered linear, continu-

ous, described by real values. Dynamics can be modelled in LEADSTO as evolution of states over time, i.e., by modelling the direct temporal dependencies between state properties in successive states. The use of durations in these temporal properties facilitates the modelling of such temporal dependencies. In principle, accurately modelling the dynamics of processes may require the use of a dense notion of time, instead of the more practiced variants of discrete time. The problem in a dense time frame of having an infinite number of time points between any two time points is tackled in LEADSTO by the assumption of "Finite Variability" (see Section 5 and, e.g., [11]). Furthermore, main advantages of the LEADSTO language are that it is executable and allows for graphical representation.

The software environment LEADSTO was developed especially for the language. It features a dedicated Property Editor that proved its value for laymen, students and expert users. The core component is the Simulation Tool that performs simulations of LEADSTO specifications, generates simulation traces for further analysis, and visualises the traces.

The approach proved its value in a number of research projects in different domains. It has been used to analyse and simulate behavioural dynamics of agents in cognitive science (e.g., human reasoning, creation of consciousness, diagnosis of eating disorders), biology (e.g., cell decision processes, the dynamics of the heart), social science (e.g., organisation dynamics, incident management), and artificial intelligence (e.g., design processes, ant colony behaviour). LEADSTO is so rich that it can be used to model phenomena from diverse perspectives. It has, for example, been used to model cognitive processes from a psychological/BDI perspective and from a physical/neurological perspective. For more publications about these applications, the reader is referred to the authors' homepages.

References

1. Barringer, H., M. Fisher, D. Gabbay, R. Owens, & M. Reynolds (1996). *The Imperative Future: Principles of Executable Temporal Logic*, Research Studies Press Ltd. and John Wiley & Sons.
2. Bosse, T., Jonker, C.M., Los, S.A., Torre, L. van der, and Treur, J., Formalisation and Analysis of the Temporal Dynamics of Conditioning. In: J.P. Mueller and F. Zambonelli (eds.), *Proc. of the Sixth Int. Workshop on Agent-Oriented Software Engineering, AOSE'05*. To appear, 2005.
3. Bosse, T., Jonker, C.M., Schut, M.C., and Treur, J, Simulation and Analysis of Shared Extended Mind. In: Davidsson, P., Gasser, L., Logan, B., and Takadama, K. (eds.), *Proc. of the First Joint Workshop on Multi-Agent and Multi-Agent-Based Simulation, MAMABS'04*, 2004, pp. 191-200.
4. Dierks, H. PLC-automata: A new class of implementable real-time automata. In M. Bertran and T. Rus, editors, *Transformation-Based Reactive Systems Development (ARTS'97)*, volume 1231 of Lecture Notes in Computer Science, pages 111-125. Springer-Verlag, 1997.
5. Forbus, K.D. *Qualitative process theory*. Artificial Intelligence, vol. 24, no. 1-3, 1984, pp. 85-168.

6. Meyer, J.J.Ch., and Treur, J. (volume eds.), *Agent-based Defeasible Control in Dynamic Environments.* Series in Defeasible Reasoning and Uncertainty Management Systems (D. Gabbay and Ph. Smets, series eds.), vol. 7. Kluwer Academic Publishers, 2002.
7. Moszkowski, B., and Manna, Z. Reasoning in Interval Temporal Logic. In Clarke, E., and Kozen, D., editors, *Proceedings of the Workshop on Logics of Programs*, volume 164 of LNCS, pages 371–382, Pittsburgh, PA, June 1983. Springer Verlag.
8. Pandya, P.K., Model checking CTL[DC]. In: *Proceedings of TACAS 2001*, Genova, LNCS 2031, Springer-Verlag, April 2001.
9. Port, R.F., Gelder, T. van (eds.) (1995). *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass.
10. Vliet, H., van. *Software Engineering: Principles and Practice*. John Wiley & Sons, Ltd, 2000.
11. Zhou, C., Hoare, C.A.R., and Ravn, A.P. *A Calculus of Durations*, Information Processing Letter, 40, 5, pp. 269-276, 1991.