Temporal Languages for Simulation and Analysis of the Dynamics Within an Organisation

Catholijn M. Jonker, Jan Treur, Wouter C.A. Wijngaards
Department of Artificial Intelligence, Vrije Universiteit Amsterdam,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Email: <{jonker,treur,wouterw}@cs.vu.nl>
URL: http://www.cs.vu.nl/~{jonker,treur,wouterw}

Abstract. In this paper a modelling approach to the dynamics within a multi-agent organisation is presented. A declarative, executable temporal modelling language for organisation dynamics is proposed as a basis for simulation. Moreover, to be able to specify and analyse dynamic properties, another temporal language is put forward, which is much more expressive than the executable language for simulations. Supporting tools have been implemented that consist of a software environment for simulation of multi-agent organisation models and a software environment for analysis of dynamic properties against traces organisation dynamics.

1 Introduction

Multi-agent systems often have complex dynamics, both in human society and in the non-human case. Organisational structure is used as a means to handle these complex dynamics. It provides a structuring of the processes in such a manner that an agent involved can function in a more appropriate manner. For example, at least partly the behavioural alternatives for the other agents are known. Put differently, the flow of dynamics within a given organisational structure is much more predictable than in an entirely unstructured situation. This assumes that the organisational structure itself is relatively stable, i.e., the structure may change, but the frequency and scale of change are assumed low compared to the more standard flow of dynamics through the structure. Both types of dynamics, dynamics within an organisational structure, and dynamics of an organisational structure are quite relevant to the area of organisation modelling. In this paper, for reasons of focussing the former type of dynamics is addressed, leaving the latter type out of consideration.

Modelling of dynamics within an organisation has at least two different aspects of use. First, models for the dynamics can be specified to be used as a basis for simulation, also called *executable models*. These types of models can be used to perform (pseudo-)experiments. Second, modelling dynamics can take the form of specification of *dynamic properties* of the organisation. These properties can play the role of requirements, and can be used, for example, in evaluation of sample behaviours of (real or simulated) organisations. These two different uses of models of dynamics impose different requirements on the languages in which these models are to be expressed.

A language for executable organisation models should be formal, and as simple as possible, to avoid computational complexity. Expressivity can be limited. Software tools to support such a language serve as *simulation environment*. A language to

specify and analyse dynamic properties of the flow within an organisation, on the other hand, should be sufficiently advanced to express various dynamic properties that can be identified. Expressivity should not be too limited; executability, however, is not required for such a language. What is important, though, is that properties specified in such a language can be checked for a given sample behaviour (e.g., a simulation run) without much work, preferably in an automated manner. Moreover, it is useful if a language to specify properties provides possibilities for further analysis of logical relationships between properties, and to generate theories about organisation dynamics. For these reasons also a language to specify properties of the dynamics within an organisation should be formal, and at least partly supported by software tools (*analysis environment*).

In this paper, for the Agent-Group-Role (AGR) organisation modelling approach (formerly called the meta-model Aalaadin) introduced in [4], two different temporal specification languages are put forward and illustrated for an example organisation. In Section 2 the static and dynamic view of the AGR modelling approach is briefly introduced. In Section 3 it is shown how the two languages (one aiming at simulation, the other one aiming at analysis) to model dynamics within such a model can be defined. In Section 4 the analysis and simulation software environment is discussed.

2 The Agent/Group/Role Organisation Modelling Approach

To obtain a useful specification of an organisation model not only the static, structural aspects, but also the dynamics within the organisation have to be taken into account. Thus, according to our view, an organisation model consists of a specification of the static *organisation structure* (Section 2.1), and a specification of the *organisation dynamics* (Section 2.2).

2.1 Static View

To model an organisation, the Agent/Group/Role (AGR) approach, adopted from [4] is used. The *organisational structure* is the specification of a specific multi-agent organisation based on a definition of groups, roles and their relationships within the organisation. A central notion is the *group structure*. It identifies the roles and (intragroup) interaction between roles within a group. The group structure is defined by the set of roles, interaction schemes and an (intragroup) transfer or communication model within the group. In addition, (intergoup) role relations between roles of different groups specify the connectivity of groups within an organisation.

Within this paper examples are taken from a case study that has been performed in the context of the Rabobank, one of the largest banks in the Netherlands, see [3]. The case study addressed design and analysis of a multi-agent approach for a bank service provision problem using a Call Centre. However, in the reference mentioned no organisation modelling approach such as AGR was incorporated. From an organisation modelling perspective, an organisation model can be defined using the following two groups: Client Service (sometimes also called Open Group) and Distribution. The roles within the groups are as follows:

Client Service:	Group Manager, Customer Relator, Client
Distribution:	Group Manager, Distributor, Participants

Within the Client Service group the service requests of clients are identified in an interaction between Client Relator and Client (this takes place within the Call Centre). Within a Distribution group, an identified service request is allocated in mutual interaction between Distributor and Participants. Actually this process takes place in two steps, making use of different instantiations of the Distribution group: once in a first line Distribution groups (relating a Call Centre agent to local bank work manager agents) and once in second line Distribution groups (work manager and employees within a local bank). The agents with role Participant in the first line Distribution group have the role of Distributor in a second line Distribution group. Also an agent with role Customer Relator in the Client Service group has a role of Distributor in the first line Distribution group.

2.2 Dynamic View

To be able to simulate or analyse dynamics within an organisation, in addition to the static organisation structure specification discussed above, as part of an organisation model also a specification of the dynamics within the organisation is needed. To this end, partly based on the types of requirements identified in [6], in the specification of an organisation model six different types of specifications of *dynamic properties* are distinguished: single role behaviour properties, intragroup interaction properties, intragroup transfer properties, intergroup interaction properties, global group properties, global organisation properties. These properties serve as constraints on the dynamics of the respective roles and interactions. To be able to specify ongoing interaction between two roles for which multiple appearances exist, the notion of *role instance* is used. This notion abstracts from the agent realising the role as actor, but enables to distinguish between appearances of roles.

The AGR modelling approach itself does not make commitments nor provides support for the use of any particular language to specify the dynamic view within the organisation model. In Section 3 below two temporal languages are introduced to specify the different types of dynamic properties. These languages and the supporting software environments can be used as an extension to the AGR modelling approach.

3 Specification of Dynamics within an Organisation Model

First the temporal trace language TTL to specify and analyse dynamic properties is introduced (Section 3.1). Next, the executable temporal language to specify simulation models is introduced (Section 3.2).

3.1 The Temporal Trace Language to Specify Dynamic Properties

To specify properties on the dynamics within the organisation, the temporal trace language used in [11], [8], is adopted. This is a language in the family of languages to which also situation calculus [14], event calculus [13], and fluent calculus [9] belong.

An *ontology* is a specification (in order-sorted logic) of a vocabulary, i.e., a signature. A state for ontology Ont is an assignment of truth-values {true, false} to the set of ground atoms At(Ont). The *set of all possible states* for ontology Ont is denoted by STATES(Ont). The standard satisfaction relation $\models$ between states and state properties is used: $S \models p$ means that state property p holds in state S.

To describe behaviour, explicit reference is made to time in a formal manner. A fixed *time frame* T is assumed which is linearly ordered. Depending on the application, it may be dense (e.g., the real numbers), or discrete (e.g., the set of integers or natural numbers or a finite initial segment of the natural numbers), or any other form, as long as it has a linear ordering. For example, for human organisations discrete time may be suitable, but for our planned future work on modelling the organisation in the living cell, a dense time frame will be more appropriate. A *trace* M over an ontology Ont and time frame T is a mapping M : T → STATES(Ont), i.e., a sequence of states M_t (t ∈ T) in STATES(Ont). The set of all traces over ontology Ont is denoted by TRACES(Ont), i.e., TRACES(Ont) = STATES(Ont)^T.

States of a trace can be related to state properties via the formally defined satisfaction relation |= between states and formulae. Comparable to the approach in situation calculus, the sorted predicate logic *temporal trace language* TTL is built on atoms referring to traces, time and state properties, such as state(M , t, output(R)) |= p. This expression denotes that state property p is true at the output of role R in the state of trace M at time point t. Here |= is a predicate symbol in the language (in infix notation), comparable to the Holds-predicate in situation calculus. Temporal formulae are built using the usual logical connectives and quantification (for example, over traces, time and state properties). The set TFOR(Ont) is the set of all *temporal formulae* that only make use of ontology Ont. We allow additional language elements as abbreviations of formulae of the temporal trace language. Ontologies can be specific for a role. In Section 4, for simplicity explicit reference to the specific ontologies per role are omitted; the ontology elements used can be read from the property specifications themselves. As an example, a dynamic property for the dynamics within the organisation as a whole is shown.

GR1 All requests answered

The first global property specifies that at any point in time, if a client communicates a request to the receptionist, then at some later time point the receptionist will communicate either an acceptance or a rejection of the request to this client.

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId, ∀ t1, tf : T ∀ C: CLIENT:open_group ∀ R: RECEPTIONIST: open_group
   [state($\mathcal{T}$, t1, output(C)) |= comm_from_to(requested(tid, tf), C, R) ⇒
     ∃ t2 : T [t2 ≥ t1 & [state($\mathcal{T}$, t2, input(C)) |= comm_from_to(rejected(tid), R, C) ∨
                state($\mathcal{T}$, t2, input(C)) |= comm_from_to(accepted(tid), R, C)]]

Note that this property is relevant, but is not of a format that can be executed easily within a simulation. Although it generates properties of future states out of current states, it entails nontrivial complexity since, due to the disjunction in the consequent, two possibilities would have to be considered. To avoid this, the executable sublanguage introduced in the next section is more restrictive.

IaRI1 Client-Receptionist Intragroup Interaction

As an example of a more local property, within the Client Service group (or open group) instance the following intragroup role interaction property is shown. It specifies that within this group proper co-operation takes place: if a client communicates a request, then some time later, either the request will be rejected or accepted.

∀ 𝒯: TRACES, ∀ tid : TaskId ∀ t1, tf : T ∀ C: CLIENT: open_group,
∀ R: RECEPTIONIST: open_group
[state(𝒯, t1, output(C)) |= comm_from_to(requested(tid, tf), C, R)
⇒ ∃ t2 : T [t2 ≥ t1 &   [state(𝒯, t2, output(R)) |= comm_from_to(rejected(tid), R, C)
  ∨ state(𝒯, t2, output(R)) |= comm_from_to(accepted(tid), R, C)]]]

The temporal trace language used in our approach is much more expressive than standard temporal logics in a number of respects. In the first place, it has *order-sorted predicate logic* expressivity, whereas most standard temporal logics are propositional. Secondly, the explicit reference to *time points and time durations* offers the possibility of modelling the dynamics of real-time phenomena, such as sensory and neural activity patterns in relation to mental properties (cf. [16]). Especially when the language is applied to the internal organisation of (biological or chemical) processes within a biological organisation, as is planned for the near future, this is an important feature. Third, in our approach states are *three-valued*; the standard temporal logics are based on two-valued states, which implies that for a given trace a form of closed world assumption is imposed. This means that, for example, in Concurrent MetateM (cf., [7]), if the executable temporal logic specification leaves some atom unspecified, during construction of a trace the semantics will force it to be false. To avoid this, an atom has to be split into a positive and a negative variant.

Fourth, the possibility to quantify over traces allows for specification of *more complex types of dynamics*. As within most temporal logics, reactiveness and pro-activeness properties be specified. In addition, in our language also properties expressing different types of adaptive behaviour can be expressed. For example a property such as 'exercise improves skill', which is a relative property in the sense that it involves the comparison of two alternatives for the history. This type of property can be expressed in our language, whereas in standard forms of temporal logic different alternative histories cannot be compared. Fifth, in our language it is possible to define *local languages for parts* of an organisation. For example, the distinction between internal, external and interface languages is crucial, and is supported by the language, which also entails the possibility to quantify over organisation parts; this allows for specification of organisation modification over time. Sixth, since state properties are used as first class citizens in the temporal trace language, it is possible to explicitly refer to them, and to quantify over them, enabling the specification of what are sometimes called *second-order properties*, which are used in part of the philosophical literature (e.g., [12]) to expres functional roles related to state properties.

In the current paper only part of the features of the language as discussed above are exploited. Due to simplicity of the chosen example, for this focus the job could also be done by a less expressive language. However, then the approach is less generic and will not be extendable to more complex dynamics, such as, for example, relative adaptive organisational behaviours. The language used is meant to support a more *generic* perspective and anticipates on these types of more complex behaviours which are in the focus of our further research.

3.2 The Executable Temporal Language to Specify Simulation Models

To obtain an executable language, in comparison with the temporal trace language discussed above strong constraints are imposed on what can be expressed. These

constraints define a temporal language within the paradigm of executable temporal logic; cf. [1]. Roughly spoken, in this executable language it can only be expressed that if a certain state property holds for a certain time interval, then after some delay another state property should hold for a certain time interval. This specific temporal relationship •→→ (*leads to*) is definable within the temporal trace language TTL. This definition is expressed in two parts, the forward in time part and the backward in time part. Time intervals are denoted by [x, y) (from and including x, to but not including y) and [x, y] (the same, but includes the y value).

Definition(The •→→ relationship)

Let α and β be state properties, and let P1 and P2 refer to parts of the organisation model (e.g., input or output of particular roles). Then β *follows* α, denoted by P1:α →→$_{e, f, g, h}$ P2:β, with time delay interval [e, f] and duration parameters g and h if

$\forall \mathcal{T} \in$ TRACES $\forall$t1:   [$\forall$t $\in$ [t1 - g, t1) : state($\mathcal{T}$, t, P1) $\models$ α $\Rightarrow$

$\exists \lambda \in$ [e, f] $\forall$t $\in$ [t1 + λ, t1 + λ + h) : state($\mathcal{T}$, t, P2) $\models$ β]

Conversely, the state property β *originates from* state property α, denoted by P1:α •—$_{e, f, g, h}$ P2:β, with time delay in [e, f] and duration parameters g and h if

$\forall \mathcal{T} \in$ TRACES $\forall$ t2:   [$\forall$t $\in$ [t2, t2 + h) : state($\mathcal{T}$, t, P2) $\models$ β $\Rightarrow$

$\exists \lambda \in$ [e, f] $\forall$t $\in$ [t2 - λ - g, t2 - λ) state($\mathcal{T}$, t, P1) $\models$ α]

If both P1:α →→$_{e,f,g,h}$ P2:β, and P1:α •—$_{e,f,g,h}$ P2:β hold, this is called a *leads to* relation and denoted by P1:α •→→$_{e,f,g,h}$ P2:β. Sometimes also conjunctions or negations on one of the sides (or both) of the arrow are used.

The definition of the relationships as given above, can be applied to situations where the sources hold for longer than the minimum interval length g. The additional duration that the source holds, is also added to the duration that the result will hold, provided that the condition e + h ≥ f holds. This is because under the given constraint the definition can be applied at each subinterval of α, resulting in many overlapping intervals of β. The end result is that the additional duration also extends the duration that the resulting notion β holds.

To use the language for simulation of organisations only role behaviour, intergroup role interaction and transfer properties are specified in the executable language. The other types of properties emerge during the simulation process. An example property in the executable language is:

IrRI2 Distributor-Receptionist Intergroup Role Interaction

This expresses that any information regarding the request of a client that the distributor instance of the distribution group instance cc receives is forwarded to the client by the receptionist role instance of the client server group instance (also called open group). In the example, for reasons of presentation we assume that only one client exists. For more clients an additional condition is needed.

$\forall$ tid : TASKID, $\forall$ R : RECEPTIONIST:open_group:OPEN_GROUP,
$\forall$ C : CLIENT:open_group:OPEN_GROUP, $\forall$ info : TASKINFORMATION
$\forall$ D : DISTRIBUTOR:cc:DISTRIBUTION, $\forall$ P : PARTICIPANT:cc:DISTRIBUTION,
INTERGROUP_ROLE_RELATION(R, D) $\Rightarrow$

[input(D):comm_from_to(info(tid), P, D) •→→$_{5,5,10,10}$ output(R):comm_from_to(info(tid), R, C)]

Role behaviour properties specify the behaviour of a role within a group. For each role within a group, and for all groups, the role behaviour must be specified. Given that for the Call Centre application three types of groups have been identified with three roles each, nine role behaviours must be specified. The behaviour of one role can take more than one property to specify. For brevity, a few of the kernel role behaviours specified here:

E1 Accepting jobs
If a Participant of a local bank group instance is asked to perform some task, and he has time to do so, then he communicates to his Distributor of the local bank group instance that he accepts the task.

$\forall$ tid : TASKID, $\forall$ tf : COMPLETIONTIME, $\forall$ f : FIFOSLOTS, $\forall$ GI : DISTRIBUTION,
$\forall$ D : DISTRIBUTOR:GI:DISTRIBUTION, $\forall$ P : PARTICIPANT:GI:DISTRIBUTION

[GI $\neq$ cc] $\Rightarrow$ [input(P):comm_from_to(requested(tid, tf), D, P) & internal(P):fifo_empty(f) $\bullet\twoheadrightarrow_{0,0,10,10}$ output(P):comm_from_to(accepted(tid), P, D)]

E2 Rejecting jobs
If a Participant of a local bank group is asked to perform some task, and he has no time to do so, then he communicates to his Distributor of the local bank group that he rejects the task.

$\forall$ tid : TASKID, $\forall$ tf : COMPLETIONTIME, $\forall$ GI : DISTRIBUTION,
$\forall$ D : DISTRIBUTOR:GI:DISTRIBUTION, $\forall$ P : PARTICIPANT:GI:DISTRIBUTION

[GI $\neq$ cc] $\Rightarrow$ [input(P):comm_from_to(requested(tid, tf), D, P) & not(internal(P):fifo_empty(fifo2)) $\bullet\twoheadrightarrow_{0,0,10,10}$ output(P):comm_from_to(rejected(tid), P, D)]

4 Analysis Environment

The analysis environment includes three different tools:
1. a tool that, given a set of traces, checks any dynamic property of the organisation.
2. a tool that, given a trace and an executable specification checks any property expressed in terms of $\bullet\twoheadrightarrow$ against the trace and interprets which rules of the executable specification are satisfied in the trace and which are falsified by the trace.
3. a tool that, given an executable specification, proves or disproves any property expressed in terms of $\bullet\twoheadrightarrow$.

All three these tools assume a finite time frame. Simulation software has been created that allows the generation of traces using properties in the executable language.

To *check* whether a given behavioural property is fulfilled in a given trace or set of traces, a software environment based on some Prolog programmes (of about 500 lines) has been developed. The temporal formulae are represented by nested term structures based on the logical connectives. For more information see [10].

Another program, of about 4000 lines in C++, has been constructed that takes an existing trace of behaviour as input and creates an *interpretation* of what happens in this trace and a *check* whether all temporal relationships in a set of properties hold in that trace. The program marks any deficiencies in the trace compared with what should be there due to the temporal relationships. If a relationship does not hold completely, this is marked by the program. The program produces yellow marks for unexpected events. At these moments, the event is not produced by any temporal

relationship; the event cannot be explained. The red marks indicate that an event has not happened, that should have happened.

In addition to checking whether the rules hold, the checker produces an informal reading of the trace. The reading is automatically generated, using a simple substitution, from the information in the given trace. For example, the properties GR1, GR2, IaRI1 and a number of other properties (not shown in this paper) have been checked and shown to be valid.

The complexity of checking properties is limited. Let the number of properties be #p, the length of the properties be |m|, the number of atoms be #a and the number of value changes per atom in the trace be #v. The length of properties is measured by the total number of atoms and connectives in the antecedent and the consequent. A first inspection of the complexity of checking is that it is polynomial in #a, #p, #v and |m|. The complexity of simulation is comparable.

A third software tool (about 300 lines of code in Prolog) addresses the *proving* of dynamic properties (expressed in terms of •↠) of an organisation from an (executable) specification of the organisation dynamics without involving specific traces. This dedicated prover exploits the executable nature of the specification and the past/current implies future nature of the property to keep complexity limited. For example, given the executable specification of the Call Centre organisation model, an instantiated form of global level property GR1 (see Section 4.1) can be proven.

Using this prover, dynamic properties of the organisation can be checked that hold for all traces of the organisation, without generating them all by subsequent simulation. The efficiency of finding such a proof strongly depends on the complexity of the specifications of the role behaviour dynamics for the different roles.

5 Discussion

In this paper specification and uses of models of the dynamics within a multi-agent organisation are addressed. A declarative temporal language is proposed as a basis for simulation. This language belongs to the class of executable temporal languages; cf. [1]. Models can be specified in a declarative manner based on a temporal 'leads to' relation; within the simulation environment these models can be executed. Moreover, to specify dynamic properties another language is put forward: a temporal trace language that belongs to the family of languages to which also situation calculus [14], event calculus [13], and fluent calculus [9] belong. The executable language for simulations is definable within this much more expressive language. Supporting tools have been implemented that consist of:

- A software environment for simulation of a multi-agent organisation model
- A software environment for analysis of dynamic properties against traces for such a model

A simple example organisation model illustrates the use of both languages and of the software environments.

In [4] no commitment to a specific dynamic modelling approach is made. In contrast, the modelling approach put forward here, makes a commitment to particular temporal languages, and provides detailed support of the dynamic modelling aspect, both for (automated) simulation and evaluation. Within [6] also a temporal trace language to specify dynamic properties is used. However, in that paper no automated

support of specification, simulation and evaluation is included, which is the main subject of the current paper. A difference with [10] is that in the current paper at a conceptual level an executable declarative temporal language and an associated simulation environment are introduced, whereas simulation within [10] is based on an implementation environment such as Swarm, without conceptual specification language. Another difference is that in [10] a diagnostic method to analyse organisational dynamics is addressed, which is not the subject of the current paper.

In comparison to the environment SDML [15], our work differs in that a specific organisation modelling approach, AGR is taken as a basis, whereas SDML does not make such a commitment, nor provides specific support for such a more specific modelling approach enabled by this commitment. Moreover, in contrast to SDML (which is restricted to simulation), in our case in addition a specification language for dynamic properties is provided, and tools to check properties. As the multi-agent system design method DESIRE [2] is similar to SDML in perspective and scope, the approach proposed in the current paper has the same differences to DESIRE as it has to SDML.

In comparison to the executable temporal logic language Concurrent MetateM [7], this language is based on discrete time whereas our approach is able to use continuous time. Moreover, our temporal language is more expressive. The possibility to compare and quantify over traces allows for specification of more complex behaviours. Properties expressing different types of adaptive behaviour can be expressed. Consider, for example properties such as 'exercise improves skill', and 'the better the experiences, the higher the trust' (trust monotonicity) which are relative in the sense that they involve the comparison of two alternatives for the history. This type of property can be expressed in our language, whereas in Concurrent MetateM and standard forms of temporal logic different alternative histories cannot be compared. The same difference applies to situation calculus, event calculus, or fluent calculus.

In future research, the diagnostic method introduced in [10] will be integrated within the organisational modelling environment described above. Moreover, support will be developed for the identification and checking of logical relationships between dynamic properties specified at different levels within the organisation.

References

1. Barringer, H., Fisher, M., Gabbay, D., Owens, R. & Reynolds, M. (1996). The Imperative Future: Principles of Executable Temporal Logic, Research Studies Press Ltd. and John Wiley & Sons.
2. Brazier, F.M.T., Jonker, C.M. & Treur, J., Principles of Compositional Multi-agent System Development. In: J. Cuena (ed.), *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98,* 1998, pp. 347-360. To be published by IOS Press, 2002.
3. Brazier, F. M. T., Jonker, C. M., Jüngen, F. J. & Treur, J. (1999). Distributed Scheduling to Support a Call Centre: a Co-operative Multi-Agent Approach. In: Applied Artificial Intelligence Journal, vol. 13, pp. 65-90. H. S. Nwana and D. T. Ndumu (eds.), Special Issue on Multi-Agent Systems.
4. Ferber, J. & Gutknecht, O. (1998). A meta-model for the analysis and design of organisations in multi-agent systems. In: Proc. of the Third International Conference on Multi-Agent Systems (ICMAS '98) Proceedings. IEEE Computer Society, 1998

5. Ferber, J. & Gutknecht, O. (1999). Operational Semantics of a role-based agent architecture. Proceedings of the 6th Int. Workshop on Agent Theories, Architectures and Languages. Lecture Notes in AI, Springer-Verlag.

6. Ferber, J., Gutknecht, O., Jonker, C.M., Mueller, J.P. & Treur, J. (2000). Organisation Models and Behavioural Requirements Specification for Multi-Agent Systems (extended abstract). In: Proc. of the Fourth International Conference on Multi-Agent Systems, ICMAS 2000. IEEE Computer Society Press, 2000. Extended version in: Proc. of MAAMAW'01.

7. Fisher, M., A survey of Concurrent METATEM — the language and its applications. In: D.M. Gabbay & H.J. Ohlbach (eds.), Temporal Logic — Proceedings of the First International Conference, Lecture Notes in AI, vol. 827, pp. 480–505.

8. Herlea, D.E., Jonker, C.M., Treur, J. & Wijngaards, N.J.E. (1999). Specification of Behavioural Requirements within Compositional Multi-Agent System Design. In: F.J. Garijo & M. Boman (eds.), Multi-Agent System Engineering, Proc. of the 9th European Workshop on Modelling Autonomous Agents in a Multi-Agent World, MAAMAW'99. Lecture Notes in AI, vol. 1647, Springer Verlag, 1999, pp. 8-27.

9. Hölldobler, S. & Thielscher, M. (1990). A new deductive approach to planning. *New Generation Computing*, 8:225-244, 1990.

10. Jonker, C.M., Letia, I.A. & Treur, J. (2001). Diagnosis of the Dynamics within an Organisation by Trace Checking of Behavioural Requirements. In: Wooldridge, M., Ciancarini, P., and Weiss, G. (eds.), *Proc. of the 2nd International Workshop on Agent-Oriented Software Engineering, AOSE'01.* Lecture Notes in Computer Science, Springer Verlag, to appear.

11. Jonker, C.M., & Treur, J. (1998). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. In: W.P. de Roever, H. Langmaack, A. Pnueli (eds.), Proceedings of the International Workshop on Compositionality, COMPOS'97. Lecture Notes in Computer Science, vol. 1536, Springer Verlag, 1998, pp. 350-380. Extended version in: *International Journal of Cooperative Information Systems.* In press, 2002.

12. Kim, J. (1998). *Mind in a Physical world: an Essay on the Mind-Body Problem and Mental Causation*. MIT Press, Cambridge, Mass, 1998.

13. Kowalski, R. & Sergot, M. (1986). A logic-based calculus of events. *New Generation Computing*, 4:67-95, 1986.

14. McCarthy, J. & P. Hayes, P. (1969). Some philosophical problems from the standpoint of artificial intelligence. *Machine Intelligence*, 4:463--502, 1969.

15. Moss, S., Gaylard, H., Wallis, S. & Edmonds, B. (1998), SDML: A Multi-Agent Language for Organizational Modelling, *Computational and Mathematical Organization Theory* **4**, (1), 43-70.

16. Port, R.F. & Gelder, T. van (eds.). (1995) *Mind as Motion: Explorations in the Dynamics of Cognition*. MIT Press, Cambridge, Mass, 1995.