VU VRIJE UNIVERSITEIT AMSTERDAM

VU Research Portal

Specification, analysis and simulation of the dynamics within an organisation

Jonker, Catholijn; Treur, Jan; Wijngaards, Wouter

published in
Applied Intelligence
2007

DOI (link to publisher)
10.1007/s10489-006-0024-0

document version
Publisher's PDF, also known as Version of record

Link to publication in VU Research Portal

citation for published version (APA)
Jonker, C., Treur, J., & Wijngaards, W. (2007). Specification, analysis and simulation of the dynamics within an organisation. *Applied Intelligence*, *27*(2), 131-152. https://doi.org/10.1007/s10489-006-0024-0

General rights
Copyright and moral rights for the publications made accessible in the public portal are retained by the authors and/or other copyright owners and it is a condition of accessing publications that users recognise and abide by the legal requirements associated with these rights.

- Users may download and print one copy of any publication from the public portal for the purpose of private study or research.
- You may not further distribute the material or use it for any profit-making activity or commercial gain
- You may freely distribute the URL identifying the publication in the public portal

Take down policy
If you believe that this document breaches copyright please contact us providing details, and we will remove access to the work immediately and investigate your claim.

E-mail address:
vuresearchportal.ub@vu.nl

Download date: 28. Mar. 2024

Appl Intell (2007) 27:131–152
DOI 10.1007/s10489-006-0024-0

Specification, analysis and simulation of the dynamics within an organisation

Catholijn M. Jonker · Jan Treur · Wouter C. A. Wijngaards

Published online: 24 January 2007
© Springer Science + Business Media, LLC 2007

Abstract In this paper a modelling approach to the dynamics within a multi-agent organisation is presented. A declarative, executable specification language for dynamics within an organisation is proposed as a basis for simulation. Moreover, to be able to specify and analyse dynamic properties within an organisation, another declarative specification 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 organisation models and a software environment for analysis of dynamic properties against traces of dynamics within an organisation.

Keywords Organisation modelling · Dynamics · Simulation

1 Introduction

Cooperative activities of multiple agents often have complex dynamics, both in human society and in the non-human, computational 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 roles that other agents play within the organisation 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 the research reported in this paper, for reasons of focussing the former type of dynamics is addressed, leaving the latter type for future research.

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 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 not too complex, to avoid computational complexity. Expressivity can be limited to the aim of execution. Software tools to support such a language serve as *simulation environment*. Examples can be found in [27, 29].

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

C. M. Jonker · J. Treur (✉) · W. C. A. Wijngaards
Dept. of AI, Vrije Universiteit Amsterdam, De Boelelaan 1081, 1081 HV Amsterdam, The Netherlands
e-mail: treur@cs.vu.nl
URL: http://www.few.vu.nl/∼treur

C. M. Jonker
e-mail: jonker@cs.vu.nl
URL: http://www.few.vu.nl/∼jonker

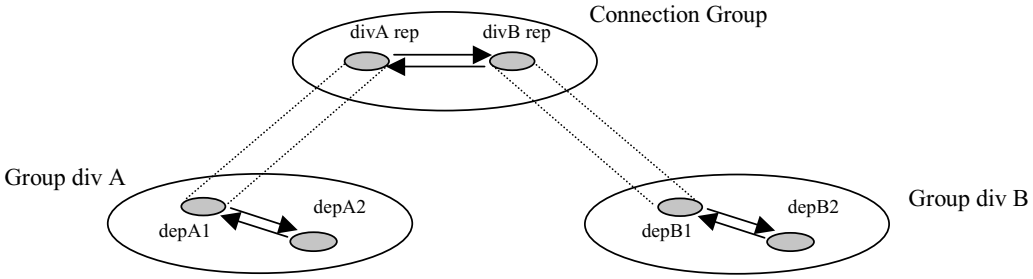


Fig. 1 An example AGR organisation structure

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 dynamic properties provides possibilities for further analysis of logical relationships between these properties, and enables formulation of theories about dynamics within an organisation. 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, as an extension of the Agent-Group-Role (AGR) organisation modelling approach (formerly called the meta-model Aalaadin) introduced in [8] and extended in [11], two different declarative specification languages for dynamics within an organisation are put forward. It is shown how these languages can be used for analysis and simulation of these dynamics, which is illustrated for an example organisation. In Section 2 the AGR modelling approach for organisational structures is briefly introduced. To obtain a useful specification of an organisation model not only the structural aspects, but also the dynamics within the organisation have to be taken into account. Thus, according to our view, an organisation model, on the one hand consists of a specification of the organisation *structure* as covered, for example, by AGR (see Section 2), but on the other hand of a specification of the *dynamics* in the form of dynamic properties at different levels in the organisation. The latter form of specification is discussed in Section 3.

In Section 4 it is shown how an expressive specification language can be used to specify these dynamic properties in a declarative manner. In Section 5, a less expressive executable specification language is introduced which allows for declarative specification of organisation simulation models. In Section 6 the analysis approach and its supporting software environment is discussed for a case study. Section 7 addresses the simulation environment, for the same case study. In Section 8 the analysis approach is extended by proposing a method to perform diagnosis of disfunctioning of the dynamics within an organisation. Section 9 concludes the paper by a discussion.

2 The Agent/Group/Role organisation modelling approach

To model an organisation, the Agent/Group/Role (AGR) approach, adopted from [8, 11] is used. The *organisational structure* is the specification of a specific multi-agent organisation based on three aggregation levels: (1) the *organisation* as a whole, which consists of (2) a set of *groups*, and each group consists of (3) the *roles* in that group. Furthermore, *connections* between roles and between groups are possible; see Fig. 1 for a simple example. Here the smaller ovals indicate roles and the bigger ovals indicate groups. Connections are indicated by the two types of arrows (dashed indicates an intergroup interaction, not dashed indicates a transfer between roles). The information on which role belongs to which group, is depicted by drawing the smaller role oval within the bigger group oval. The organisation is *realized* by *agents* allocated to roles. Notice that roles and groups can be considered as functions of an organisation, whereas agents are the entities realising these functions. At a functional level the organisation is described in a generic manner, abstracting from these agents.

As a simple example, a factory is considered that is organised at the highest aggregation level according to two divisions: division A that produces certain components and division B that assembles these components to (composite) products. At one aggregation level lower the division A is organised according to two departments: department A1 (the work planning department for division A) and department A2 (component production department). Similarly, division B is organised according to two department roles: department B1 (for assembly work planning) and department B2 (product assembly department).

The two divisions are modeled as *groups* (depicted by the larger ovals), with the departments as their *roles* (depicted by smaller ovals within larger ones). A third group, the Connection Group C, models the cooperation between the two divisions. This group consists of the two *roles* 'division A representative' and 'division B representative'. *Intergroup role interactions* (depicted by pairs of dotted lines)

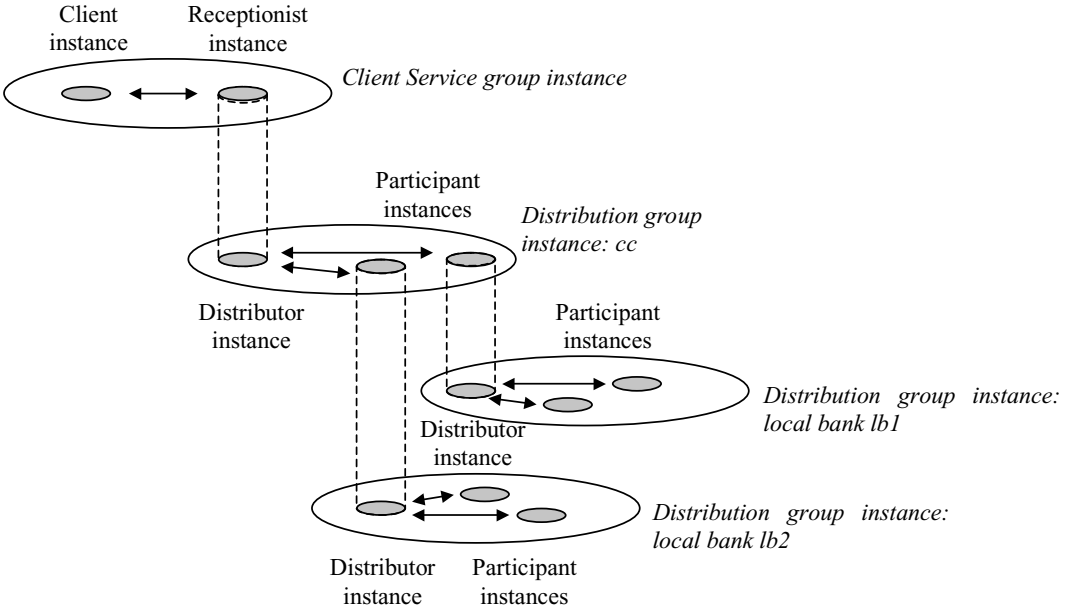


Fig. 2 The AGR-organisation structure for the bank Call Center organisation

are modeled between the role 'department A1' in the division A group and the role 'division A representative' within the connection group, and between the role 'department B1' in the division B group and the role 'division B representative' within the connection group. *Intragroup role transfers* model communication between the two roles within each of the groups (depicted by the arrows). Such transfers have destination roles (indicated by the arrow points) and source roles (no pointing).

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 [5]. 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: Receptionist, Client
Distribution: Distributor, Participants

Within the Client Service group the service requests of clients are identified in an interaction between Receptionist and Client (this takes place within the Call Centre). See Fig. 2; in this figure

- the big ovals denote *group instances*,
- small grey ovals denote *role instances* within a group instance,
- arrows denote *interactions between role instances* in *one* group instance, and
- dashed vertical lines denote *relations* between role instances in *different* group instances.

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 group (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. Similarly an agent with role Receptionist in the Client Service group has a role of Distributor in the first line Distribution group. At the level of the organisation model, this means that between these pairs of roles within the organisation model, intergroup role relations are defined.

3 Modelling dynamics within an organisation

An organisation model according to the AGR-approach discussed above specifies an organisation structure, but provides no dynamics. In a sense it abstracts from the dynamics. However, to be able to analyse and/or simulate dynamics within an organisation, as part of an organisation model, also a specification of properties of the dynamics within the organisation is needed, for short called *dynamic properties*. Dynamic properties relate states of the organisation over time. Usually one

particular dynamic property refers not to the whole state but to a limited set of specific elements or aspects of these states. This set can be viewed as the scope of a dynamic property. Depending on the property at hand, this scope can be more broad (or *global*), or more narrow (or *local*). For example, global properties of an organisation as a whole may refer to a number of different aspects (sometimes such a property is called *integrative*), whereas properties on the interaction between two specific roles within an organisation will refer only to aspects related to these roles (e.g., a *role interaction* protocol).

Related to these differences between types of properties, partly following and extending [10, 23], in this section six different types of dynamic properties are distinguished: single role behaviour properties, intragroup role interaction properties, intragroup transfer properties, global group properties, intergroup role interaction properties, global organisation properties. Each of these types is discussed below in more detail. A specification language for such properties is discussed in Section 4. How these properties can be used in an analysis of the dynamics within an organisation is addressed in Section 6. How to obtain some of these properties in executable format is addressed in Section 5, and how to use these executable dynamic properties for simulation can be found in Section 7.

To be able to specify ongoing interaction between two roles for which multiple appearances exist, the notion of *role instance* is used. This notion still abstracts from the agent realising the role as actor, but enables to distinguish between appearances of roles. For example, the role 'secret agent' has role instances like 'secret agent 001' to 'secret agent 009', and the role instance 'secret agent 007' is sometimes realised by the agent James Bond, sometimes by the agent Jack Jackson (his predecessor), and so on. If R is a role, then a role instance for R is denoted by I:R; so, for example, 007:secret agent denotes the role instance 'secret agent 007'. Throughout this paper the identifiers used to denote instances are considered to be unique (this leads to better readable properties). As for roles, also *group instances* are used. In our example, the Distributor role instance within the Distribution group instance has an intergroup role interaction with the Receptionist role instance within the Client service group instance.

3.1 Single role behaviour properties

For a given role within a group, role behaviour properties specify the dynamics of the role within the group. They are typically expressed in terms of temporal relationships between the *input* and *output* of the role instance, often according to a pattern like the following:

if role instance I:R receives as input ....
[and in the past as input of I:R it was received ...
and in the past at the output of I:R it was generated ...]
then some time later role instance I:R generates as output ....

The following are examples of role behaviour properties (within the Client Service group). Note that the clause 'some time later' can easily be made more specific by referring to a certain maximal response time. The same applies to the other examples in this paper. For simplicity of presentation such more precise references to response times have been left out.

Client role behaviour

If the Client receives a request for additional information
then the Client either provides this information
or the Client provides a 'bye-bye'.
If the Client receives a proposal for a service requested by him
then the Client either accepts the proposal
or the Client rejects the proposal.

Receptionist role behaviour

If a Receptionist receives a request for a service from a Client
and the necessary information regarding this Client is **not** available
then the Receptionist issues a request for this information to that Client.
If the necessary information regarding a Client has become available
then the Receptionist communicates a request to that Client to wait.

3.2 Intragroup role interaction properties

Intragroup role interaction properties express temporal constraints on the dynamics of the interaction protocol between two roles within a group. Intragroup role interaction properties between two roles instances *I*:*R*1 and *J*:*R*2 in one group instance are typically expressed in terms of the *output* of both role instances, often according to a pattern as:

if role instance I:R1 generates as output ....
[and in the past at the output of I:R1 it was generated ...
and in the past at the output of J:R2 it was generated ...]
then some time later role instance J:R2 generates as output ....
or role instance I:R1 generates as output ....

The last conclusion enables, for example, to specify that a request from one role instance to another role instance is withdrawn before the request actually was answered, thus taking away the urge to answer it. In the simplest situations no references to the past are made, and the pattern takes the form of a direct reactivenes relation:

if role instance I:R1 generates as output ....
then some time later role instance J:R2 generates as output ....

Client Service group role interaction
The Client role instances interact with Receptionist role instances.

If a Client provides on his output a service request
then some time later:
the Receptionist provides for that Client on his output a proposal for that request
or that Client provides on his output a 'bye-bye'.

If a bye-bye has been communicated by a role instance I:R1 to a role instance J:R2, it is assumed that from that moment on I:R1 will not listen to communication directed from J:R2 to I:R1, e.g., because the telephone connection has been closed. This has impact on the communication successfulness properties that will be discussed in Section 3.3.

If the Receptionist provides on his output a request for information to a Client
then some time later:
that Client provides on his output either the requested additional information
or that Client provides on his output a 'bye-bye'.

3.3 Intragroup communication successfulness properties

Intragroup role interaction requires communication within the group. Therefore, in order to function properly, properties are needed that express that communications are succesful. These properties have the following pattern:

if role instance I:R1 generates as output a communication directed to J:R2
[and ...]
then some time later role instance J:R2 receives as input this communication

Such a property may not hold unconditionally; it may as well be domain and time dependent. In particular, if the other role instance communicated earlier a bye-bye, then we don't assume it is listening anymore (it may have put the telephone off). We also include the condition that the previous communication of role instance J:R2 was not a request to wait (hold on); it might be the case that after such a request J:R2 will not listen until it resumes communication. A Communication Successfulness property then specifies, for example, the following:

If communication is directed by a role instance I:R1 to a role instance J:R2
then role instance J:R2 will always receive this communication, unless its previous communication was either 'hold on' or 'bye-bye'.

3.4 Group properties

It is also possible to express dynamic properties within a group as a whole. For example, such properties may involve a number of (inputs and outputs of) roles within this group. An example of such a property for the Distribution group is as follows.

If a role instance I:Participant exists who has communicated to the role instance M:Distributor that he accepted task tid with deadline tf,
then a role instance J:Participant exists which at some point in time before tf communicates to the M:D that the task tid was finished

This property expresses that the group as a whole takes the responsibility to finish an accepted task before its deadline.

3.5 Intergroup role interaction properties

To express dynamics of connections between groups, intergroup role interaction properties are used; they specify the temporal constraints on the dynamics of the interaction protocol between two role instances within two different group instances. They are typically expressed in terms of the *input* of one of the role instances and the *output* of the other one, for example, according to the following pattern, in which I:R1 is a role instance within group instance GI1:G1 of group G1, and J:R2 is a role instance within group instance GI2:G2 of group G2:

if role instance I:R1 receives as input ....
[and in the past at the input of I:R1 it was received ...
and in the past at the output of J:R2 it was generated ...
and in the past at the input of J:R2 it was received ...]
then some time later role instance J:R2 generates as output ....

Note that more role instances (of more group instances) may be involved.

Client Service group - Distribution group role interaction properties

The interaction between Client Service group and Distribution group is realised by one agent to fulfill both roles. However, independent of knowledge of the assignment of agents to roles, the required intergroup role interaction can be expressed. Here I:C is a Client role instance, J:CR is a Receptionist role instance within the Client Service group instance and K:D a Distributor role instance, and L:P a Participant role instance within the Distribution group instance.

The first intergroup interaction property between the Client Service and Distribution group instances states:

If a Receptionist role instance J:CR receives a service request as input within the Client Service group instance,
then this request is put forward within the Distribution group instance cc by the Distributor role instance K:D to all Participant role instances L:P.

The second intergroup role interaction property between the Client Service and Distribution group instance cc states:

If a Distributor role instance K:D receives as input within the Distributor group instance a service proposal on a request,
then this proposal will be put forward within the Client Service group instance by the Receptionist role instance J:CR that received the request earlier, and is directed to the Client role instance I:C from which the request originates.

3.6 Organisation properties

To be able to express that an organisation as a whole functions properly, dynamic properties for the organisation can be expressed. Such properties may refer to any part (relating to any role within any group) of the organisation. An example of such a global property expresses:

At any point in time,
if the Receptionist communicates to a Client that a request was accepted,
then at some later time point in one of the Distribution group instances a Participant communicates to the Distributor that the task was finished.

Note that a property may be classified as global, or an organisation property, if only some but not all aspects of the state of the organisation are involved.

The AGR organisation modelling approach itself does not make commitments nor provides support for the use of any particular language to specify the dynamics within the organisation model. In Sections 4 and 5 below two specification 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.

4 Specification of dynamic properties within an organisation model

In this section the Temporal Trace Language TTL to specify and analyse dynamic properties is introduced. This temporal trace language is adopted from [3, 22] see [17, 18] for application of this language in the context of requirements engineering. It is a language in the family of languages to which also situation calculus [26, 30], event calculus [24], and fluent calculus [19] belong. See also [13, 14, 15, 20] for more background in temporal modelling. The language is defined as follows.

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. A *trace* $\mathcal{T}$ over an ontology Ont and time frame T is a mapping

$$\mathcal{T}: \mathsf{T} \rightarrow \mathsf{STATES(Ont)}$$

i.e., a sequence of states

$$\mathcal{T}_{\mathsf{t}}(t \in T)$$

in STATES(Ont). The set of all traces over ontology Ont is denoted by TRACES(Ont), i.e., $\mathsf{TRACES(Ont)} = \mathsf{STATES(Ont)}^{\mathsf{T}}$.

States of a trace can be related to state properties via the formally defined satisfaction relation $\models$ between states and formulae. The sorted predicate logic *temporal trace language* TTL is built on atoms referring to traces, time and state

properties, such as

state($\mathcal{T}$, t, output(R)) $\models$ p.

This expression denotes that state property p is true at the output of role R in the state of trace $\mathcal{T}$ at time point t. Here $\models$ 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 left out; the ontology elements used can be read from the property specifications themselves. As an example, the following dynamic property for the dynamics within the organisation as a whole is shown.

GP2 All accepted jobs are finished

The next property expresses that for all traces and task identifiers, if at any point in time the receptionist communicates to a client that a request was accepted, then at some later time point in one of the Distribution group instances a Participant communicates to the Distributor that the task was finished.

∀ $\mathcal{T}$: TRACES ∀ id : TaskId ∀ t, t1 : T
∀ C: CLIENT: open_group ∀ R: RECEPTIONIST:
open_group
[state($\mathcal{T}$, t1, input(C)) $\models$ comm_from_to(accepted(id, t),
R, C)
⇒ ∃ t2 : T ∃ P : PARTICIPANT, D : DISTRIBUTOR : DG :
DISTRIBUTION_GROUP
[t2 ≥ t1 & state($\mathcal{T}$, t2, input(C)) $\models$ comm_ from_
to (finished(id), P, D)]]

Another example of a dynamic property is the following.

IaRI1 Client-Receptionist Intragroup Role Interaction

As an example of a more local property, within the Client Service group (or open group) instance the following 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.

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

Note that this property is a relevant property, 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 sub-language introduced in the next section is more restrictive.

The temporal trace language TTL 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. Furthermore, the explicit reference to *time points and time durations* offers the possibility of modelling the dynamics of real-time phenomena, as often are essential in organisations where real-time constraints are important, or organisations with dynamics that are adaptive in a continuous manner. Another feature of the language TTL is that it is possible to define *local languages for parts* of an organisation. For example, the distinction between internal, external and input and output 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.

A further, more sophisticated feature of TTL is the possibility to quantify over traces (which are considered first class citizins in the language) allows for specification of *more complex types of dynamics*. As within most temporal logics, reactiveness and pro-activeness properties can 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.

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.

5 Declarative specification of organisation simulation models

Temporal specifications of dynamic properties can sometimes be used as a basis for simulation. To this end the properties have to be expressed in a specific (sometimes called executable) format. This perspective has been developed within the paradigm of *executable temporal logic*; cf. [1]. A temporal formula in executable format is one according to the pattern

past and current implies future

Here the time frame is assumed to be discrete. A simple example of an executable temporal formula is (with C the current operator and X the next operator)

$$Ca \wedge Cb \rightarrow Xc$$

which states that always if in a state the state properties a and b hold, then in the next state property c holds. Simulation based on such a temporal formula can be performed by executing it in the following inductive sense:

1. *Check the antecedents on the last generated state of the simulation trace*: If a trace has been generated up to time point t, determine whether the conditions a and b hold in the state at t.
2. *Collect the consequents for those antecedents that hold at the last generated state*: Examining in an exhaustive manner all temporal formulae in executable format defining a specification, a number of properties for the state at time $t + 1$ are determined; e.g., if a and b hold, then for the state at the next time point $t + 1$ the property c is to hold.
3. *Build the next state by derived state properties*: All collected consequents together provide a (partial) description of the next state at time $t + 1$.
4. *Complete the next state*: By some form of completion (e.g., by a closed world assumption, making state properties false that are not derivable in a positive manner), this description can be made complete, obtaining the complete next state of the trace for $t + 1$.

An advantage of this paradigm of Executable Temporal Logic is that simulation models are specified not in an algorithmic manner, but in a declarative logical manner. The relation between the specification and the constructed trace is that the trace is a model (in the logical sense) of the theory defined by the specification, i.e., all temporal formulae of the specification hold in the trace. A disadvantage of the discrete time frame assumption is that it does not allow specification of simulation models where variable real-valued time periods between the transitions play a role.

The language TTL described in Section 4 is a very expressive declarative language. This expressivity makes it inappropriate for simulation. To obtain an executable language, in comparison with the temporal trace language discussed above some constraints have to be imposed on what can be expressed. However, some of its advantages can be kept, such as the use of real-valued time periods between transitions. The constraints imposed define a temporal language within a real-valued time extension of the paradigm of executable temporal logic. 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 $\bullet\!\!\rightarrow\!\!\rightarrow$ (*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 $\bullet\!\!\rightarrow\!\!\rightarrow$ 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

$$P1{:}\alpha \rightarrow\!\!\rightarrow_{e,\,f,\,g,\,h} P2{:}\beta$$

with time delay interval [e, f] and duration parameters g and h if and only if:

for all traces and time points t1,
  if α holds in P1 for the interval [t1 - g, t1),
then a delay $\lambda \in$ [e, f] exists such that β holds in P2 for
  the interval [t1 + λ, t1 + λ + h).

Formally:

$$\forall\, \mathcal{T} \in \text{TRACES}\ \forall\, t1{:}\ [\forall t \in [t1 - g, t1) :$$
$$\text{state}(\mathcal{T}, t, P1) \models \alpha \quad \Rightarrow$$
$$\exists \lambda \in [e, f]\ \forall t \in [t1 + \lambda, t1 + \lambda + h) : \text{state}(\mathcal{T}, t, P2) \models \beta]$$

Conversely, the state property β *originates from* state property α, denoted by

$$P1{:}\alpha \bullet\!\!\!-_{e,\,f,\,g,\,h} P2{:}\beta$$

with time delay in [e, f] and duration parameters g and h if and only if:

for all traces and time points t2,
  if β holds in P2 for the interval [t2, t2 + h),
then a delay $\lambda \in$ [e, f] exists such that α holds in P1 for
  the interval [t2 - λ - g, t2 - λ).

Formally:

$$\forall \mathcal{T} \in \text{TRACES}\ \forall\, t2{:}\ [\forall\, t \in [t2, t2 + h) :$$
$$\text{state}(\mathcal{T}, t, P2) \models \beta \quad \Rightarrow$$
$$\exists \lambda \in [e, f]\ \forall\, t \in [t2 - \lambda - g, t2 - \lambda)\ \text{state}(\mathcal{T}, t, P1) \models \alpha]$$

If both $P1{:}\alpha \rightarrow_{e,f,g,h} P2{:}\beta$, and $P1{:}\alpha \bullet\!\!\!-_{e,f,g,h} P2{:}\beta$ hold, this is called a *leads to* relation and denoted by $P1{:}\alpha \bullet\!\!\rightarrow\!\!\rightarrow_{e,f,g,h}$

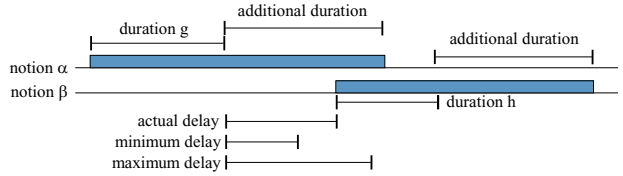


Fig. 3 Temporal relationships for longer durations

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 result for a longer duration of α for P1:$\alpha \bullet\!\!\rightarrow\!\!\rightarrow$P2:$\beta$ is depicted in Fig. 3. 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 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.

The procedure used for simulation is a variation on the procedure for Executable Temporal Logic shown above. For example, for step 1 not the last generated state is taken, but the past time interval is considered. Moreover, in step 3 and 4. the state properties are fixed for certain future time intervals instead of one state. For more details, see [2] and Section 7.3 below.

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 can be used.

∀ R : RECEPTIONIST:open_group: OPEN_GROUP,
∀ C : CLIENT:open_group:OPEN_GROUP,
  ∀ info:TASKINFORMATION
∀ D : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P:PARTICIPANT:cc:DISTRIBUTION,
INTERGROUP_ROLE_RELATION(R, D) ⇒
  [input(D):comm_from_to(info, P, D) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,10,10}$
    output(R):comm_from_to(info, 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 properties must be specified. Examples are:

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.

∀ tid : TASKID, ∀ tf : COMPLETIONTIME, ∀ f : FIFOSLOTS,
  ∀ GI : DISTRIBUTION,
∀ D : DISTRIBUTOR:GI:DISTRIBUTION,
  ∀ P : PARTICIPANT:GI:DISTRIBUTION
[GI ≠ cc] ⇒ [input(P):comm_from_to(requested(tid, tf), D, P)
  & internal(P):fifo_empty(f) $\bullet\!\!\rightarrow\!\!\rightarrow_{0,0,10,10}$
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.

∀ tid : TASKID, ∀ tf : COMPLETIONTIME,
  ∀ GI : DISTRIBUTION,
∀ D : DISTRIBUTOR:GI:DISTRIBUTION,
  ∀ P : PARTICIPANT:GI:DISTRIBUTION
[GI ≠ cc] ⇒ [input(P):comm_from_to(requested(tid, tf), D, P)
  & not(internal(P):fifo_empty(fifolast)) $\bullet\!\!\rightarrow\!\!\rightarrow_{0,0,10,10}$
output(P):comm_from_to(rejected(tid), P, D)]

6 Analysis of dynamic properties

By means of the example it is shown how dynamic properties can be analysed. First, in Section 6.1 an overview is presented of the dynamic properties involved in the example organisation. Next, in Section 6.2 the logical relationships between these properties are shown. Finally, in Section 6.3 it is discussed how these propereties can be automatically checked for given traces of organisation dynamics, and how properties in 'leads to' format can be proven from an executable specification.

6.1 Overview of dynamic properties

The dynamic properties are presented starting at the level of the organisation as a whole.

6.1.1 Global properties

At the level of the organisation as a whole the following properties can be identified:

- Every request is answered (either by rejecting or by accepting and finishing it)
- No accepted jobs are lost: for every accepted job there is a time that that job is finished.
- Every accepted job is finished before its deadline

These global properties can be formalised as follows.
The first 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 a rejection of the request or a notification that it was accepted to that client.

GP1 All requests answered

∀ $\mathcal{T}$: 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)]]

The next property expresses that if at any point in time the receptionist communicates to a client that a request was accepted, then at some later time point in one of the Distribution group instances a Participant communicates to the Distributor that the task was finished.

GP2 All accepted jobs are finished

∀ $\mathcal{T}$: TRACES ∀ id : TaskId ∀ t, t1: T ∀ C: CLIENT:
open_group ∀ R: RECEPTIONIST: open_group
[state($\mathcal{T}$, t1, input(C)) ⊨
comm_from_to(accepted(id, t), R, C)
⇒ ∃ t2 : T ∃ P : PARTICIPANT, D : DISTRIBUTOR :
DG : DISTRIBUTION_GROUP
[t2 ≥ t1 & state($\mathcal{T}$, t2, input(D)) ⊨
comm_from_to(finished(id), P, D)]]

Property GP3 expresses that if at any point in time the receptionist communicates to a client that a request was accepted, then at some later time point, but before the deadline, the receptionist communicates to the same client that the task was finished.

GP3 Meeting deadlines of accepted jobs

∀ $\mathcal{T}$: TRACES ∀ id : TaskId ∀ t, t1 : T ∀ C: CLIENT:
open_group ∀ R: RECEPTIONIST: open_group
[state($\mathcal{T}$, t1, input(C)) ⊨
comm_from_to(accepted (id, t), R, C)
⇒ ∃ t2 : T [t ≥ t2 ≥ t1 & state($\mathcal{T}$, t2, input(C)) ⊨
comm_from_to(finished(id), R, C)]]

In the next four subsections the different properties on parts of the organisation are identified. In Fig. 3 below an overview can be found. In this figure three group instances are depicted, together with two role instances in each of them. Properties of different types are depicted by arrows. The position from which an arrow starts indicates the role instance to which the *if*-part of the property refers. Whether the *if*-part refers to input or output is indicated by the start position of the arrow (resp. at the left hand side of the role instance or at the right hand side). In a similar manner the end point of an arrow indicates to which role instance the *then*-part of the property refers. The properties distinghuished in Fig. 3 and specified in detail below are selected in order to be able to derive global property GP1 from them. For briefness sake the other properties have been left out. In Sections 6.2 and 6.3 the logical relationships between the properties specified below and global property GP1 will be addressed in more detail.

6.1.2 Intragroup role interaction properties

Intragroup role interaction properties specify the cooperation within a group. Within each group instance at least one intragroup role interaction property is specified. The first one, within the Client Service group (or open group) instance, 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.

IaRI1 Client-Receptionist Intragroup Interaction

∀ $\mathcal{T}$: 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, output(R)) ⊨
comm_from_to(rejected(tid), R, C)
∨ state(T, t2, output(R)) ⊨
comm_from_to(accepted(tid), R, C)]]]

The next property expresses that within the distribution groups proper interaction takes place: if a request is communicated to a participant (by a distributor), then the participant will respond (eventually) by rejecting it or accepting it.

IaRI2/IaRI3 Distributor-Participant Intragroup Interaction

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T ∀ GI: DISTRIBUTION
∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[state($\mathcal{T}$, t1, output(D)) ⊨
comm_from_to(requested (tid,tf), D, P)

⇒ ∃ t2 : T [t2 ≥ t1 &
[state($\mathcal{T}$, t2, output(P)) ⊨
comm_from_to(rejected(tid), P, D)
∨ state($\mathcal{T}$, t2, output(P)) ⊨
comm_from_to(accepted(tid), P, D)]]]

6.1.3 Intergroup role interaction properties

Intergroup role interaction properties specify connectivity between the groups. This is achieved by an association between a role instance of one group and a role instance in another group, specified by the relation intergroup_role_relation(R, D). The first intergroup role interaction property specifies that an intergroup role relation between role instances of RECEPTIONIST and DISTRIBUTOR in open_group and cc exists, and, in particular that every request received by the role instance of RECEPTIONIST within open_group leads to a similar request of the role instance DISTRIBUTOR within cc.

IrRI1 Receptionist-Distributor Intergroup Interaction

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T ∀ R:
RECEPTIONIST: open_group ∀ C: CLIENT: open_group
∀ D: DISTRIBUTOR: cc ∀ P: PARTICIPANT: cc
[[intergroup_role_relation(R, D)
& state($\mathcal{T}$, t1, input(R)) ⊨
comm_from_to(requested(tid, tf), C, R)]
⇒ ∃ t2 : T [t2 ≥ t1 & state(T, t2, output(D)) ⊨
comm_from_to(requested(tid, tf), D, P)]]

The next intergroup role interaction property specifies that also the return path from group instance cc to group instance open_group is guaranteed. This is achieved by an intergroup role relation from the distributor instance to the receptionist instance. The explanation of this property is as follows. If within the distribution group instance cc the distributor role instance gets information communicated by a participant, then within the open group instance the related receptionist role instance will communicate this information to the client. In this property (and some of the other, subsequent properties) info ranges over {finished(tid), rejected(tid), accepted(tid)}.

IrRI2 Distributor-Receptionist Intergroup Interaction

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T ∀ D:
DISTRIBUTOR: cc ∀ P: PARTICIPANT: cc
∀ R: RECEPTIONIST: open_group ∀ C: CLIENT: open_group
[[state($\mathcal{T}$, t1, input(D)) ⊨
comm_from_to(info, P, D) & intergroup_role_relation(D, R)]
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, output(R)) ⊨
comm_from_to(info, R, C)]]

Similarly intergroup relations between the local bank group instances and the distributor group instance cc are specified:

IrRI3 Participant-Distributor Intergroup Interaction

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T
∀ D1: DISTRIBUTOR: cc ∀ P1: PARTICIPANT: cc
∀ GI: DISTRIBUTION ∀ D2: DISTRIBUTOR: GI:
DISTRIBUTION ∀ P1: PARTICIPANT: GI:
DISTRIBUTION
[[state($\mathcal{T}$, t1, input(P1)) ⊨
comm_from_to (requested(tid, tf), D1, P1)
& intergroup_role_relation(P1, D2)]
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, output(D2)) ⊨
comm_from_to(requested(tid, tf), D2, P2)]]

IrRI4 Distributor-Participant Intergroup Interaction

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T
∀ D1: DISTRIBUTOR: cc ∀ P1: PARTICIPANT: cc
∀ GI: DISTRIBUTION
∀ D2: DISTRIBUTOR: GI: DISTRIBUTION
∀ P1: PARTICIPANT: GI: DISTRIBUTION
[[state($\mathcal{T}$, t1, input(D2)) |= comm_from_to(info, P2, D2)
& intergroup_role_relation(D2, P1)]
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, output(P1))
⊨ comm_from_to(info, P1, D1)]]

6.1.4 Transfer properties

Successful cooperation within a group requires that communication takes place when needed. In particular this means that the two cooperating roles within the open group instance have to communicate successfully about requests, i.e., if a request is communicated by a client to the receptionist, this request will be received by the receptionist.

TR1 Client-Receptionist communication

∀ $\mathcal{T}$: 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(R)) ⊨
comm_from_to(requested(tid, tf), C, R)]]

Moreover, they also communicate about acceptance, rejectance or finishing of tasks; here, as before, info can be filled with any of these, related to tid:

TR2 Client-Receptionist communication

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1: T ∀ C: CLIENT:
open_group ∀ R: RECEPTIONIST: open_group
[state($\mathcal{T}$, t1, output(R)) ⊨ comm_from_to(info, R, C)
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, input(C)) ⊨
comm_from_to(info, R, C)]]

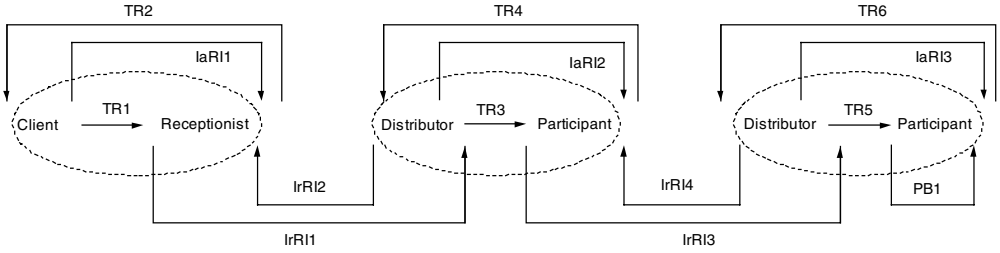


Fig. 4 Overview of non-global properties

Similarly within the distribution groups proper communication has to take place about requests and what comes back for them (e.g., acceptance, rejection, or finished notifications; again info can be filled with any of these, related to tid):

TR3/TR5 Distributor-Participant communication

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T
∀ GI: DISTRIBUTION
∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[state($\mathcal{T}$, t1, output(D)) ⊨
comm_from_to(requested(tid, tf), D, P)
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, input(P)) ⊨
comm_from_to(requested(tid, tf), D, P)]]

TR4/TR6 Distributor-Participant communication

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1: T ∀ GI: DISTRIBUTION
∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[state($\mathcal{T}$, t1, output(P)) ⊨ comm_from_to(info, P, D)
⇒ ∃ t2 : T [t2 ≥ t1 & state($\mathcal{T}$, t2, input(D)) ⊨
comm_from_to(info, P, D)]]

6.1.5 Single role behaviour properties

In this organisation model many of the roles just earn their money communicating. But at least at some place in the organisation the real work has to be done. This is performed by the participant roles in the local banks. A number of properties can be specified. As an example, the property 'if they do not reject a task, they have to finish it' is expressed below:

PB1 Participant behaviour

∀ $\mathcal{T}$: TRACES ∀ tid : TaskId ∀ t1, tf : T ∀ GI: DISTRIBUTION
∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[state($\mathcal{T}$, t1, input(P)) ⊨
comm_from_to(requested(tid, tf), D, P)
⇒ ∃ t2 : T [t2 ≥ t1 &
[state($\mathcal{T}$, t2, output(P)) ⊨
comm_from_to(rejected(tid), P, D)

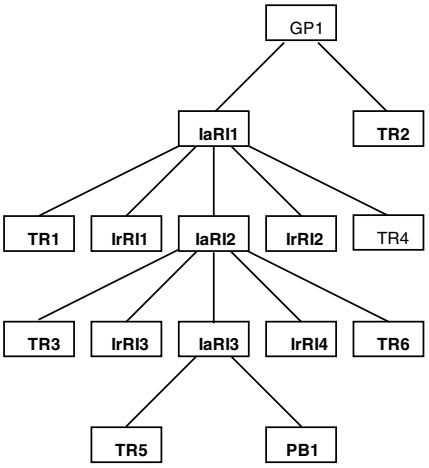


Fig. 5 AND-tree of properties of different types

∨ state($\mathcal{T}$, t2, output(P)) ⊨
comm_from_to(finished(tid), P, D)]]]

6.2 Logical relationships between the properties

Figure 5 shows logical relationships between different types of properties. For example, within the rightmost group instance, the arrows for transfer property TR5 and role behaviour property PB1 'chain' in an appropriate manner to intragroup interation property laRI3. Indeed, logically the latter property can be derived from the former two.
This obtains a proof pattern (see also Fig. 5).

TR5 & PB1 ⇒ laRI3

In a similar manner other proof patterns have been identified and actually proven for the intragroup interaction properties laRI1 and laRI2, making use of inter group interaction properties, transfer properties, and (other) intragroup properties:

TR3 & IrRI3 &
laRI3 &
TR6 & IrRI4 ⇒ laRI2

TR1 & IrRI1 &
laRI2 &
TR4 & IrRI2 ⇒ laRI1

Finally, the global property GP1 can be derived from IaRI1 and TR2.

IaRI1 & TR2 ⇒ GP1

These proof patterns are depicted in Fig. 5 as an AND-tree.

Note that due to these logical relations, the leaves of the tree together imply the top node. The other nodes in a sense have no independent staus; they can be considered intermediate steps. This observation will be exploited in two different manners. First it will be exploited in Section 7 by specifying a simulation model solely based on the leaves of this tree. The leaves include all information that is needed for the simulation. A second manner to exploit the structure of the AND-tree is in the diagnostic approach put forward in Section 8. Here the final result of a diagnosis of disfunctioning in an organisation is to pinpoint the particular leaf or leaves of the tree that is or are responsible for the disfunctioning. The other nodes of the tree are used as intermediate steps in the diagnostic process. This process is explained in more detail in Section 8.

6.3 Automated support for analysis

Apart from an editor to specify dynamic properties, the analysis environment includes two parts; all these tools assume a finite time frame:

1. a tool that, given a set of traces (for example, obtained by logging the activities over time in a real organisation, or generated by simulation based on the executable properties or on another simulation model), checks any dynamic property of an organisation expressed in the Temporal Trace Language TTL. In addition, a tool has been developed that, given a trace checks for any property expressed in terms of $\bullet\!\!\rightarrow\!\!\rightarrow$ where exactly in the trace the property fails, i.e., where the antecedent and the consequent of this property fail.
2. a tool that, given an executable specification, for any dynamic property in 'leads to' format proves or disproves whether it is entailed by the dynamic properties in 'leads to' format of the executable specification.

6.3.1 Checking a dynamic property against a set of traces

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 example, property GP1 from Section 6.1 is represented by

```
forall(M, T1, C:CLIENT, R:RECEPTIONIST, TID, TF,
  imp(holds(state(M, T1, output(C:CLIENT)),
      communication_from_to(requested(TID, TF),
        C:CLIENT, R:RECEPTIONIST), true),
  ex(T2 = T1,
    or(holds(state(M, T2, input(C:CLIENT)),
        communication_from_to(finished(TID),
          R:RECEPTIONIST, C:CLIENT), true),
holds(state(M, T2, input(C:CLIENT)),
  communication_from_to(rejected(TID),
    R:RECEPTIONIST, C:CLIENT), true)
) ) ) )
```

Traces are represented by sets of Prolog facts of the form

```
holds(state(m1, t(1), input(role1)), a)), true).
holds(state(m1, t(2), output(role1)), b)), true).
holds(state(m1, t(3), input(role2)), b)), true).
holds(state(m1, t(4), input(role2)), c)), true).
holds(state(m1, t(5), output(role2)), d)), true).
```

where m1 is the trace name, t(1) et cetera are time points, and a, b, c, d are state properties in the ontology of the roles's input or output states. E.g., in the second line it is indicated that state formula a is true in role1's output state within the organisation at time point 2. The Prolog programme for temporal formula checking uses Prolog rules such as

```
sat(and(F,G))          :-  sat(F), sat(G).
sat(not(and(F,G)))     :-  sat(or(not(F), not(G))).
sat(or(F,G))           :-  sat(F).
sat(or(F,G))           :-  sat(G).
sat(not(or(F,G)))      :-  sat(and(not(F), not(G))).
```

that reduce the satisfaction of the temporal formula to the satisfaction of less complex formulae, and finally to the satisfaction of atomic state formulae and their negations at certain time points. The latter satisfactions can be read from the trace.

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 leads to 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 GP1, GP2, 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.

These tools have been used to check a number of traces of organisation dynamics, both generated by the simulation environment discussed in Section 7 below, and traces generated by a Swarm implementation of an organisation model as described in [21].

6.3.2 Proving properties from an executable specification

A third software tool (about 300 lines of code in Prolog) addresses the *proving* of dynamic properties (expressed in terms of one of the formats $\rightarrow\!\!\!\rightarrow$, $\bullet\!\!-$ or $\bullet\!\!\rightarrow\!\!\!\rightarrow$ of an organisation from an (executable) specification of the dynamics within the organisation 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 specified in Section 7.1, an instantiated form of global level property GP1 (see Section 6.1) 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. Also properties can be disproven. Then the prover comes up with a trace that is a counter example against the disproven property. The efficiency of the prover is reasonable. The price that is paid to keep complexity limited is that only properties of the overall design can be proven that can be written in one of the formats $\rightarrow\!\!\!\rightarrow$, $\bullet\!\!-$ or $\bullet\!\!\rightarrow\!\!\!\rightarrow$.

7 Simulation environment

A simulation environment has been created to enable the simulation of the executable organisation models with all the usual benefits of rapid prototyping. In this section first an example of an executable organisation model is discussed. Then the simulation software is introduced. Subsequently the simulation algorithm is presented. Finally, some of the results of the experiments with the example organisation model are discussed. Input for the simulation environment is a set of executable temporal formulae expressed in terms of the 'leads to' relation $\bullet\!\!\rightarrow\!\!\!\rightarrow$, i.e., in the format defined in Section 3.2. Thus, the example executable organisation model is expressed in terms of such formulae.

7.1 Example of an executable organisation model

The Call Centre example introduced in Section 2.1 is examined further. For the Call Centre application there is one instance of the Client Service group (here called the Open Group), one group instance (cc), and for each local bank one instance of the Distribution group. In this section part of the dynamics of that example is determined in executable format. From a general perspective, as in Section 6, the dynamic properties that should hold within the Call Centre application are expressed in terms of single role behaviour properties, intragroup interaction properties, intragroup transfer properties, and intergroup interaction properties. However, for simulation, not all these properties are to be used. As already noticed in Section 6.2 in relation to Fig. 4, the dynamic properties at the leaves of the AND-tree already include all information that is needed. The other properties are implied by these leaf properties. Therefore specification of the simulation model can be restricted to specification of the leaf properties, i.e., intergroup role interaction properties, transfer properties, and role behaviour properties. These leaf properties are the more simple properties; they can be specified in 'leads to' format, whereas some of the other, derivable properties cannot. The three types of properties are discussed subsequently.

7.1.1 Intergroup interaction properties within the executable model

For our example organisation model the following intergroup interaction properties are part of the executable specification:

IrRI1 Receptionist-Distributor Intergroup Role Interaction

∀ tid : TASKID, ∀ tf : COMPLETIONTIME,
  ∀ R : RECEPTIONIST:open_group:OPEN_GROUP,
  ∀ r : REGION,
∀ C : CLIENT:open_group:OPEN_GROUP,
  ∀ D : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P : PARTICIPANT:cc:DISTRIBUTION
[INTERGROUP_ROLE_RELATION(R, D) &
  CLIENT_REGION_RELATION(C, r) &
  REGION_BANK_RELATION(r, P)] ⇒
[input(R):comm_from_to(requested(tid, tf), C, R) $\bullet\!\!\rightarrow\!\!\!\rightarrow_{5,5,10,10}$
  output(D):comm_from_to(requested(tid, tf), D, P)]

The above property expresses that if the receptionist of the open (or Client Service) group instance receives a request

from a client, then the distributor role instance of cc the group instance of the distribution group forwards this request to the participants in his group.

IrRI2 Distributor-Receptionist Intergroup Role Interaction

∀ tid : TASKID,
  ∀ R : RECEPTIONIST:open_group: OPEN_GROUP,
  ∀ C : CLIENT:open_group:OPEN_GROUP,
∀ D : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P : PARTICIPANT:cc:DISTRIBUTION,
  ∀ info : TASKINFORMATION
[INTERGROUP_ROLE_RELATION(R, D)] ⇒
[input(D):comm_from_to(info, P, D) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,10,10}$
  output(R):comm_from_to(info, R, C)]

This expresses that any information regarding the request of a client (i.e., info, that can be filled with acceptance, rejection and finished information related to tid) 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. If more clients are handled at the same time, an additional condition can be included to guarantee that the right client is notified.

IrRI3 Participant-Distributor Intergroup Role Interaction

∀ tid : TASKID, ∀ tf : COMPLETIONTIME,
  ∀ D1 : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P1 : PARTICIPANT:cc:DISTRIBUTION,
∀ GI : DISTRIBUTION,
  ∀ D2 : DISTRIBUTOR:GI:DISTRIBUTION,
  ∀ P2 : PARTICIPANT:GI:DISTRIBUTION
[GI ≠ cc & INTERGROUP_ROLE_RELATION(P1, D2)] ⇒
[input(P1):comm_from_to(requested(tid, tf), D1, P1)
  $\bullet\!\!\rightarrow\!\!\rightarrow_{0.5,0.5,1,1}$
  output(D2):comm_from_to(requested(tid, tf), D2, P2)]

Property IrRI3 denotes that the Distributor of a local bank group forwards requests to the Participants of that local bank group.

IrRI4a Distributor-Participant Intergroup Role Interaction

∀ tid : TASKID, ∀ D1 : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P1 : PARTICIPANT:cc:DISTRIBUTION,
  ∀ GI : DISTRIBUTION
∀ D2 : DISTRIBUTOR:GI:DISTRIBUTION,
  ∀ P2 : PARTICIPANT:GI:DISTRIBUTION
[GI ≠ cc & INTERGROUP_ROLE_RELATION(P1, D2)] ⇒
[input(D2):comm_from_to(finished(tid), P2, D2)
  $\bullet\!\!\rightarrow\!\!\rightarrow_{0.5,0.5,1,1}$
  output(P1):comm_from_to(finished(tid), P1, D1)]

When a Distributor of a local bank group instance hears that a task requested by a client is finished by a Participant of the local bank group instance, then this notice is forwarded to the Distributor of the cc group instance.

IrRI4b Distributor-Participant Intergroup Role Interaction

∀ tid : TASKID, ∀ D1 : DISTRIBUTOR:cc:DISTRIBUTION,
  ∀ P1 : PARTICIPANT:cc:DISTRIBUTION,
  ∀ GI : DISTRIBUTION
∀ D2 : DISTRIBUTOR:GI:DISTRIBUTION,
  ∀ P2 : PARTICIPANT:GI:DISTRIBUTION
[GI ≠ cc & INTERGROUP_ROLE_RELATION(P1, D2)] ⇒
[input(D2):comm_from_to(accepted(tid), P2, D2)
  $\bullet\!\!\rightarrow\!\!\rightarrow_{0.5,0.5,1,1}$
  output(P1):comm_from_to(accepted(tid), P1, D1)]

If a Distributor of a local bank group instance is notified that a task is accepted by a Participant of the local bank group instance, then this notification is forwarded to the Distributor of the cc group instance.

7.1.2 Transfer properties within the executable model

For an executable model, also the transfer of information between roles is important. Transfer properties are properties that ensure that information sent by A to B is received by B. They can easily be written in 'leads to' format. In Section 6 a number of transfer properties were identified. Rewritten in 'leads to' format they are as follows; here, as before, info can be filled with any of these, related to tid:

TR1 Client-Receptionist communication

∀ tid : TaskId ∀ t1, tf : T ∀ C: CLIENT: open_group,
  ∀ R: RECEPTIONIST: open_group
[output(C):comm_from_to(requested(tid, tf), C, R) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,1,1}$
  input(R):comm_from_to(requested(tid, tf), C, R)]

TR2 Client-Receptionist communication

∀ tid : TaskId ∀ t1: T ∀ C: CLIENT: open_group
  ∀ R: RECEPTIONIST: open_group
[output(R):comm_from_to(info, R, C) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,1,1}$
  input(C):comm_from_to(info, R, C)]

TR3/TR5 Distributor-Participant communication

∀ tid : TaskId ∀ t1, tf : T ∀ GI: DISTRIBUTION
  ∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[output(D):comm_from_to(requested(tid, tf), D, P) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,1,1}$
  input(P):comm_from_to(requested(tid, tf), D, P)]

TR4/TR6 Distributor-Participant communication

∀ tid : TaskId ∀ t1: T ∀ GI: DISTRIBUTION
∀ D: DISTRIBUTOR: GI: DISTRIBUTION
∀ P: PARTICIPANT: GI: DISTRIBUTION
[output(P):comm_from_to(info, P, D)
$\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,1,1}$ input(D):comm_from_to(info, P, D)]

TR7 Workmanager-Employee communication for assignments

∀ tid : TASKID, ∀ GI : DISTRIBUTION,
∀ D : DISTRIBUTOR:GI:DISTRIBUTION,
∀ P : PARTICIPANT:GI:DISTRIBUTION
[GI $\neq$ cc $\Rightarrow$
[output(D):comm_from_to(assigned(tid), P2, D2) $\bullet\!\!\rightarrow\!\!\rightarrow_{5,5,1,1}$
input(P):comm_from_to(assigned(tid), P1, D1)]]

This last transfer property states that if the Distributor of a local bank group instance wants to communicate to a Participant of the local bank group instance that he is assigned some task, then this communication will be received by that Participant of the local bank group instance some time later.

7.1.3 Role behaviour properties within the executable model

Role behaviour properties specify dynamic properties of the behaviour of a role within a group. To obtain an executable specification, for each role within a group, and for all groups, executable properties of the role behaviour 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 used for simulations are presented here:

E1 Accepting jobs

∀ tid : TASKID, ∀ tf : COMPLETIONTIME, ∀ f : FIFOSLOTS,
∀ GI : DISTRIBUTION,
∀ D : DISTRIBUTOR:GI:DISTRIBUTION,
∀ P : PARTICIPANT:GI:DISTRIBUTION
[GI $\neq$ cc] $\Rightarrow$
[input(P):comm_from_to(requested(tid, tf), D, P) &
internal(P):fifo_empty(f) $\bullet\!\!\rightarrow\!\!\rightarrow_{0,0,10,10}$
output(P):comm_from_to(accepted(tid), P, D)]

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.

E2 Rejecting jobs

∀ tid : TASKID, ∀ tf : COMPLETIONTIME,
∀ GI : DISTRIBUTION,
∀ D : DISTRIBUTOR:GI:DISTRIBUTION,
∀ P : PARTICIPANT:GI:DISTRIBUTION
[GI $\neq$ cc] $\Rightarrow$
[input(P):comm_from_to(requested(tid, tf), D, P) &
not(internal(P):fifo_empty(fifolast)) $\bullet\!\!\rightarrow\!\!\rightarrow_{0,0,10,10}$
output(P):comm_from_to(rejected(tid), P, D)]

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.

7.2 The simulation software

A software environment has been made which implements the temporal formalisation of the dynamics as specified by an executable organisation model. First the approach is introduced, then the program will be briefly reviewed, after which some of the results are discussed.

The simulation determines the consequences of the temporal relationships forwards in time. Remember that α *leads to* β, is denoted by P1:$\alpha\bullet\!\!\rightarrow\!\!\rightarrow_{e,f,g,h}$ P2:β where the time delay λ is taken from the interval [e, f]. The duration parameter g denotes the time span that α must minimally hold, and h denotes the duration parameter that β must minimally hold. In order to make simulation efficient, long intervals of results are derived when starting from long intervals. By applying additional conditions (i.e., e + h $\geq$ f), the derivation of longer intervals becomes possible, see Section 3, Fig. 2. The logical relationships thus avoid unnecessary work for the simulation software.

The delay value λ can either be chosen randomly within the interval [e, f] each time a relationship is used, or the λ can be fixed to a value. Selecting either a random or fixed λ enables thorough investigation of the consequences of a particular model.

Extending the paradigm of executable temporal logic, cf. [1], a 17000 line simulation program was written in C++ to automatically generate the consequences of the real-value parameterised 'leads to' temporal relationships within the executable organisation specification. The program is a special purpose tool to derive the results reasoning forwards in time, as in executable temporal logic. Some additional information is required to define the time frame of simulation, picture preferences and text for the labels.

The program reads a specification of temporal rules from a plain text file. The maximum time for derivation is also specified in that file, the interval [0, MaxTime). In order to specify facts about the environment (world), (periodic) intervals can be given. The functions not(), and(,), and or(,) can be used to make more complex properties from atoms. The properties have and, or and not given in prefix ordering for the program (instead of infix), i.e., a function is given before its arguments, e.g.., and(a, b) instead of (a and b). The +(and +) brackets perform concatenation of their contents, in order to construct identifiers from variables and strings. The $\bullet\!\!\rightarrow\!\!\rightarrow$ relation is specified using LeadsTo, followed by the e,

f, g and h values. Note that the program does not use the •– (originates from) part of the relation as only forward derivation is performed. First the timing is given, then the variables are quantified. A restriction is put on the R and D variables; in this case they must have an intergroup role relation. Then the antecedent and consequent are given. For clarity, tokens are displayed boldface, values and identifiers are not. As an example of a rule, the IrRI2 distributor-receptionist intergroup interaction property, is presented as follows:

```
RuleVarLeadsTo delay 5 5 10 10
   Var ForAll tid : TASKID
      ForAll D : DISTRIBUTOR:cc:DISTRIBUTION
      ForAll P : PARTICIPANT:cc:DISTRIBUTION
      ForAll R : RECEPTIONIST:open_group:
         OPEN_GROUP
      ForAll C : CLIENT:open_group:OPEN_GROUP
      ForAll info : TASKINFORMATION
   MemberCheck +(R _related_D+) :
    INTERGROUP_ROLE_RELATION
EndVar
+(D _input_ comm_from_to( info (tid), P, D ) +)
o->>
+( R _output_ comm_from_to( info (tid), R, C) +)
```

7.3 The simulation algorithm

First a short look at the method of simulation by forward derivation is presented. In order to derive the consequences of the temporal relationships within a specific interval of time, a cycle is performed, starting at time 0. For the set of rules the earliest starting time of the antecedent for each rule, for which the consequent does not already hold, is computed. A rule with earliest start time of the antecedent is chosen. This rule is then fired at that time, adding the consequent to the trace. The cycle is restarted, only looking for antecedents at or after the fire time point, as effects are assumed to occur simultaneously or after their causes. This continues until no more rules can be fired, or the fire time is at or after the end time of the simulation interval. In more detail the algorithm is as follows. Note that also a closed world assumption is built in. This avoids the need to specify all facts that should not hold, which can lead to large executable specifications.

The derivation process uses a point in time called now. It starts at 0. The derivation will derive the trace from time 0 until a time called the maximum time. The trace is fully unknown for all atoms for all time points at the start of derivation. The derivation procedure is as follows:

Step 1. Find earliest eligible rule.
At now in time examine each rule to find the earliest eligible rule for firing. An eligible rule is a rule, for which the antecedent holds, and the consequent does not hold yet in the trace thus far; these rules are eligible to be fired. Eligible rules are thus rules whose consequent could be added to the trace under construction. If a rule 'fires' it adds its consequent to the trace. For a more elaborate account of rule selection see the section on rule selection below. For now, a single, 'earliest applicable', rule is selected. Call this rule r, and its start of the antecedent time t0.

At this point several alternatives exist. Each alternative must be treated correctly. No rules could be eligible for firing at all; this is treated in Step 2. If some rules are eligible, some rule r and some time t0 must have been found. If the rule fires sometime in the future, $t0 > now$, time must be advanced in Step 3. If time does not need to be advanced, the time t0 could be at the end, $t0 > maximum\ time$, and in this case it should be checked if the trace is already completed, in Step 4. In all other cases, when $t0 \leq now$, the rule r can be fired in Step 5 and the loop is continued at Step 1.

Step 2. No rules are eligible for firing.
If there are no eligible rules for firing, no rule r can be found. Perform the CWA (closed world assumption) procedure from now till the maximum time. This is done in order to provide CWA at the end of the trace as well. If t_{next}, the continuation time that is recommended by the CWA is smaller than the maximum time, $t_{next} < maximum\ time$, then CWA rules have fired; the trace is changed and now is set to t_{next} and the derivation process is continued at Step 1. If the CWA procedure recommends a continuation time at or after the maximum time, $t_{next} \geq maximum\ time$, derivation stops. Since this indicates that no rules will fire before the maximum time, the trace is complete.

Step 3. For a rule r, *time* $t0 > now$, *advance time.*
There is a rule r and a time t0 that it can fire. If the time that the rule will fire is in the future, $t0 > now$, then by firing this rule the time would skip to t0. Do not fire rule r in this case yet, it will be fired in Step 5 at some later time. The closed world assumption needs to be performed in the meantime, since no rule is affecting the interval from now to t0. Perform the CWA procedure from now to t0. The separate CWA procedure returns a recommended continuation value upon completion, taking the t0 into account. This is explained in the steps of the closed world assumption procedure below. Set the now to the recommended continuation time, t_{next}, and continue the derivation at Step 1.

Step 4. For a rule r, $t0 > maximum\ time$, *finished.*
If the time to fire rule r is after the end of time, $t0 > maximum\ time$, then do not fire the rule. The derivation stops, since the trace is already complete.

Step 5. For a rule r, t0 ≤ now*, fire the rule.*
Fire rule r with the antecedent starting at time t0. This entails applying the consequent of rule r to the trace. Loop back to Step 1 to continue derivation.

7.3.1 Rule selection

The trace under construction is expanded by firing one rule at a time. Therefore one rule must be selected for firing. In theory, each rule must be checked to see if it is *applicable*. A rule is applicable if its antecedent holds at a particular time. The rule must also, however, be *eligible*, its consequent must not already hold in the trace, otherwise the system would loop on adding only the first rule forever. The earliest such eligible rule must then be fired, its consequent interval must be added to the trace.

Using this method, all the rules would be checked against the entire trace to select a rule. This would work, but it would be inefficient. Since we assume that rules cause their effects to happen after their causes, we know that the effects of a rule happen in the future. Thus it is assumed that the delay values are not negative for any rule. If the rule antecedent starts at the now moment, then the rule consequent interval is always after the now moment. Therefore, if you only fire rules whose antecedent starts at the now moment, the only changes that will happen are in the future, never in the past.

This can be used to speed up derivation considerably, as the past need not be checked for applicable rules anymore. It is already known that all applicable rules in the past are not eligible, since their consequent will already hold. Their consequent will hold as it has been added when the rule was fired in the past, when the now was equal to the start of the rule's antecedent. So only some antecedent intervals need to be checked, starting with the intervals that contain the now moment.

In derivation then, all antecedent intervals are checked for each rule, starting from intervals that contain now, and continuing with intervals after that. An antecedent interval is an interval where the logical statements of the antecedent of a leads-to relationship hold, for at least a duration g. See Fig. 6 for an explanatory example of an antecedent interval. These are applicable, both antecedent intervals in Fig. 6 are applicable. But only eligible rule antecedents are considered further, for those antecedents the consequent interval does not hold yet in the trace. For example the second antecedent interval in Fig. 6 is eligible, but the first antecedent intervals is not.

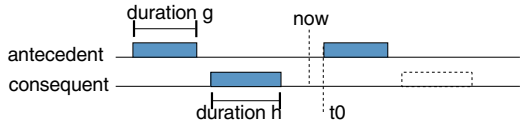


Fig. 6 An eligible antecedent interval at time t0

The eligible antecedent of a rule with the start of the antecedent the earliest in time is selected. In Fig. 6, the second antecedent interval would be selected, as the first is not eligible.

7.3.2 Closed world assumption

In order to derive simulation results a closed world assumption (CWA) has been employed. A closed world assumption assumes that properties at certain intervals that remain unknown are false. To apply the closed world assumption, a hypothetical trace is considered. The reason that the closed world assumption considers a hypothetical trace, is that it must be avoided that the now is advanced to a point in the future, where changes in the past (before the now) have to be made. The closed world assumption could cause changes in that past, because the trace is changed by applying the closed world assumption to unknown intervals. These changes in the past could trigger rules that could change the trace, also (partial) changes to the past. Such changes in the past would then cause inconsistency. The closed world assumption method described below will avoid making changes in the past, thus avoiding inconsistency during computation.

The closed world assumption procedure is performed from a time called t0 to a time called t2, and returns a recommended continuation time called t_{next}. The CWA procedure is:

Step 1. Construct a hypothetical trace.
This hypothetical trace is the same as the currently derived trace, but in all the time from t0 to t2 all the unknown intervals are made false. Each rule is examined for this hypothetical trace, from time point 0 onwards, to see if its antecedent holds and its consequent does not hold yet in the hypothetical trace. The rule with earliest start time of the antecedent is selected again, call this time t1.

Several alternatives exist at this point. If no rules can be found to fire in the hypothetical trace, the hypothesis is okay, and can be applied in Step 2. If rules can fire, some time t1 is found. Either t1 ≥ t2, and the hypothesis can be applied as well in Step 3, or t1 < t2, and a careful application of rules to avoid inconsistency must be done in Step 4. After handling the appropriate case, the closed world assumption procedure returns a continuation time and ceases.

Step 2. No rules can fire in the hypothetical trace.
If no eligible rule can be found, this means no rule will fire in the hypothetical trace. The closed world assumption is applied to the main trace (so the hypothetical trace becomes reality), t_{next} is set to t2 and the CWA procedure stops.

Step 3. A rule can fire at t1 ≥ t2.
If t1 ≥ t2, then no rules using the closed world assumption will fire before t2. The closed world assumption is

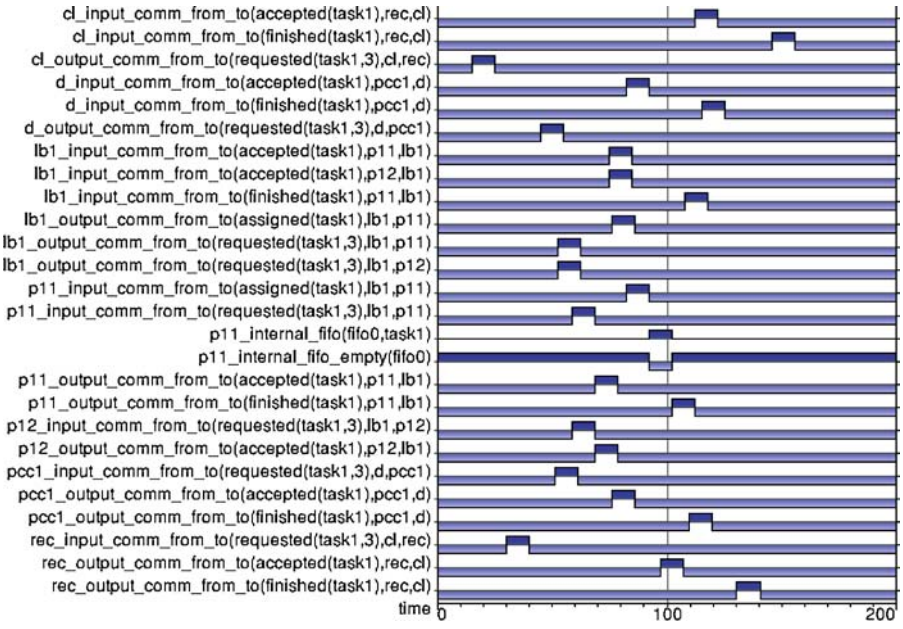


Fig. 7 Some simulation results considering only task 1

applied to the main trace (so the hypothetical trace becomes reality), and t_{next} is set to t2 and the CWA procedure stops.

Step 4. A rule can fire at $t1 < t2$.
If $t1 < t2$, then a rule that would use the intervals caused by the closed world assumption in the interval from t0 to t2 will fire before t2. The rule would not use only ordinary, non-hypothetical, parts of the trace as its antecedent. This must be the case, since the closed world assumption process is started only when the first rule will ordinarily fire at or after t2. The (direct and indirect) effects of the rule that fires at t1 may change the trace, and they may change the trace before time t2. Thus not all unknown intervals from t0 to t2 can be made false, since the effects could contradict this, and create inconsistency.

The closed world assumption can thus only be safely applied from t0 until the time point of the first change that the rule firing at t1 would cause, called time t3. Before this is applied, all the rules that would fire before time t2 are applied with a length as determined by the hypothetical trace, as this speeds up computation. Then in the interval t0 to t3 the unknown values are changed to false. The hypothetical trace is discarded. The CWA procedure stops with t_{next} set to t3.

After the derivation is complete, the results are output. An elaborate technical output is generated in a log file with exact times and intervals, as well as the sequence of rule firings that happened. A picture is drawn too, saved to a file and displayed for the user. See Section 7.4 for an example of pictorial output.

7.4 Some simulation results

Figure 7 shows some of the results of experimenting with the Call Centre organisation model. Time is on the horizontal axis. The properties are listed on the vertical axis. The λ is fixed at 0.5. A dark box on top of the line indicates the property is true during that time period, and a lighter box below the line indicates that the property is false during that time period. The first line, for example, contains the property that cl (the client) has on its input a communication from rec (the receptionist) that task1 has been accepted. This property is false most of the time, but true from approximately 110 to 120.

Figure 7 shows that the first event is that the client cl issues a request. The receptionist rec receives this request. The distributor d forwards the request to participant pcc1 of the cc group instance, as the client is in the region of pcc1. Participant pcc1 has it on its input, and role instance lb1 forwards it to its participants of the local bank group p11, and p12. Both participants accept the task, and thus d accepts the task. lb1 assigns the task to p11. After finishing the task, p11 notifies lb1, after which is passed on through the different group instances, finally reaching the client.

8 Diagnosis of disfunctioning within an organisation

Logical relatioships between properties, as depicted in the AND-tree of Fig. 5 can be very useful in the analysis of malfunctioning of the organisation in the following manner. For example, if for a given trace of the organisation the global property GP1 is not satisfied, then, given the last proof

pattern, by a refutation process it can be concluded that either transfer does not function properly or laRI1 does not hold. If laRI1 does not hold, then by one of the other proof patterns either laRI2 does not hold, or one of the intergroup interaction properties lrRI1 or lrRI2 does not hold (or transfer fails). If the intragroup property laRI2 does not hold, then either lrRI3, lrRI4 or laRI3 does not hold (or transfer fails). Finally, if laRI3 does not hold, then by the first proof pattern either role behaviour property PB1 does not hold or transfer is not properly functioning. By this refutation analysis it follows that if GP1 does not hold for a given trace, then, skipping the intermediate properties, the cause of this malfunctioning can be found in the set (the leaves of the tree in Fig. 5):

$$\{\text{lrRI1}, \text{lrRI2}, \text{lrRI3}, \text{lrRI14}\} \cup \{\text{PB1}\} \cup \{\text{TR1}, \ldots, \text{TR6}\}.$$

The logical analysis by itself does not pinpoint which one of these leaves actually is refuted. However, it shows a set of candidates that can be examined in more detail.

Returning to the verification of the global organisation property GP1, if the check shows that it is not satisfied, then subsequently, the candidate set of causes $\{\text{lrRI1}, \text{lrRI2}, \text{lrRI3}, \text{lrRI14}\} \cup \{\text{PB1}\} \cup \{\text{TR1},\ldots, \text{TR6}\}$ generated from the logical analysis in Section 6.1 can be checked. Due to the logical relationships given by the proof patterns, at least one of them must be not satisfied. After having them checked it will be found which one is the culprit. Since the set only contains specific properties which refer to local situations within the organisation, this localises the problem. Thus this approach provides a method of diagnosing malfunctioning in an organisation. In a more efficient, hierarchical, manner, based on the tree in Fig. 5 (obtained from the logical analysis resulting in the proof patterns in Section 6.1), this method for diagnosis of malfunctioning in an organisation runs as follows (according to a specific diagnostic method, sometimes called *hierarchical classification*):

1. First check the global properties
   (the top of the tree in Fig. 5)
2. Focus the subsequent checking process on only those more local properties that in view of the logical dependencies relate to a more global property that has turned out to be false
   (the branches in the tree under a failed node)
3. Repeat this procedure with the focused more local properties as top-node
4. The most local properties that fail point at where the cause of malfunctioning can be found
   (one or more of the leaves of the tree)

Note that in step 2 all local properties that do not relate to a failing global property can be left out of consideration, which may obtain an advantage in the number of properties to be checked, compared to simply checking all properties, of n over 2^n (if the property refinement graph would have the structure of a binary tree with all branches of depth n).

This method has been used to analyse the example organisation simulation model presented above. In the simulation software environment log files containing the traces were automatically created that were saved at a place where the checking software environment can automatically read in the files and perform the checking process. Thus an overall software environment was created that is an adequate tool to diagnose the dynamics within the organisation simulation model. As one of its uses, the tool can be used for debugging of the simulation model. The diagnostic method used is a simple one. More sophisticated diagnostic methods can be explored along the same lines, based on established logical relationships between dynamic properties such as expressed in the AND-tree of Fig. 5.

9 Discussion

In this paper specification and uses of models of the dynamics within an organisation are addressed. A declarative but executable language is proposed as a basis for simulation. This language, using real-valued time points and durations, extends the class of executable temporal languages; cf. [1, 12, 13]. In this way organisation simulation 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 of an organisation, another, very expressive declarative language is put forward: a temporal trace language that belongs to the family of languages to which also situation calculus [26, 30], event calculus [24], and fluent calculus [19] belong. The executable language for organisation simulations is definable within this much more expressive language. Supporting tools for both languages 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

In the paper a simple example organisation model illustrates the use of both languages and of the software environments.

In comparison, in [8, 11] no commitment to a specific dynamic modelling approach is made. In contrast, the dynamic modelling approach put forward here, makes a commitment to particular specification languages, and provides detailed support of the dynamic aspects, both for (automated) simulation and analysis. The use of a temporal trace language to specify dynamic properties within an organisation model was first put forward in [10, 23]. However, there no automated support of specification and analysis is included, which is

a main subject of the current paper. Moreover, specification in executable format and simulation was not addressed. In [9] a formalisation of the procedural semantic of an AGR-organisation model in terms of π-calculus (a wellknown general semantical technique in Theoretical Computer Science) is presented. A difference with the approach put forward here is that in [9] it is not addressed how from this semantical formalisation adequate dedicated organisation modelling languages can be developed. For example, the use of formal languages for simulation or for analysis is not worked out.

A difference with [21] 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 [21] is based on an example implementation using the simulation environment Swarm, without a conceptual specification language.

The organisation modelling environment SDML [27] has in common with our work that a declarative language for simulation is offered. In comparison, 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 perform analysis of properties. An interesting extension and connection would be to define a standard way of representing log files of simulation traces generated in SDML, so that these log files can be read by our analysis software. As the multi-agent system design method DESIRE [4] has a lot of similarities 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 temporal logics, the temporal trace language TTL used in our approach is much more expressive in a number of respects than standard or extended modal temporal logics as described, for example, in [6,7,12,13,16,25,31,32]. See also [14, 15] for a discussion about modal temporal logics and predicate logic-based temporal logics. 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. These first two points apply only partially to logics where it is possible to have real numbers for time and arithmetical operations and order relations to express constraints between time points, as in [6, 16, 32].

Third, the possibility to quantify over traces allows for specification of *more complex dynamics*. As within most temporal logics, reactiveness and pro-activeness properties can be specified. In addition, in our language also properties expressing different types of adaptive behaviour can be expressed. For example an adaptive property such as 'exercise improves skill', or 'the better the experiences, the higher the trust' (trust monotonicity) which both are 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. The same difference applies to situation calculus, event calculus, fluent calculus, and the languages in [6, 16, 32].

Fourth, in TTL it is possible to define *local languages for parts* of a system. For example, the distinctions between components, and between input and output languages are crucial, and are supported by the language, which also entails the possibility to quantify over system parts and describe changing system parts over time.

In future research, connections will be made to other simulation approaches, so that traces generated by these approaches can be analysed in our analysis environment, for example for SDML, as discussed above. Furthermore, more extensive support will be developed for the identification and checking of logical relationships between dynamic properties specified at different levels within the organisation. Moreover, the use of the approach put forward for the use of adaptive dynamic properties for role behaviour and role interactions will be explored. In addition, specification and analysis of organisational configuration dynamics over time will be addressed; for a first starting point, see (Dastani, Jonker and Treur, 2001). Another theme that can be considered for future research is how to incorporate deontic aspects in an organisation in models of the type discussed here. This can be addressed by specifying effects of deontic constraints on role behaviours.

Acknowledgements Stimulating discussions with Jacques Ferber, Olivier Gutknecht, Jean-Pierre Müller and Fredi Letia on parts of the material in this paper have contributed to an increased insight in the subject.

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. Bosse T, Jonker CM, van der Meij L, Treur J (2005) LEADSTO: a language and environment for analysis of dynamics by simulaTiOn. In: Eymann T, Kluegl F, Lamersdorf W, Klusch M, Huhns MN (eds) Proc. of the third german conference on multi-agent system technologies, MATES'05. Lecture Notes in Artificial Intelligence, vol. 3550. Springer Verlag, pp 165–178
3. Bosse T, Jonker CM, van der Meij L, Sharpanskykh A, Treur J (2006) Specification and verification of dynamics in cognitive agent models. In: Proceedings of the sixth international conference on intelligent agent technology, IAT'06. IEEE Computer Society Press, to appear.
4. Brazier FMT, Jonker CM, Treur J (2002) Principles of component-based design of intelligent agents. Data and Knowledge

Engineering, 41:1–28. Also In: Cuena J, Demazeau Y, Garcia A, Treur J (eds) (2004) Knowledge engineering and agent technology. IOS Press, pp 89–113
5. Brazier FMT, Jonker CM, Jüngen FJ, Treur J (1999) Distributed scheduling to support a call centre: a co-operative multi-agent approach. In: Nwana HS, Ndumu DT (eds) Applied artificial intelligence journal, vol. 13. Special Issue on Multi-Agent Systems, pp 65–90
6. Bouajjani A, Lakhnech Y, Yovine S (1996) Model checking for extended timed temporal logic. In: Jonsson B, Parrow J (eds) Proc. of the 4th international symposium formal techniques in real-time and fault-tolerant systems, FTRTFT'96, Uppsala, Sweden, September 1996. Lecture Notes in Computer Science, vol. 1135, Springer-Verlag, pp 306–326
7. Clarke EM, Grumberg O, Peled DA (2000) Model checking. MIT Press
8. Ferber J, Gutknecht O (1998) A meta-model for the analysis and design of organisations in multi-agent systems. In: Demazeau Y (ed) Proc of the third international conference on multi-agent systems (ICMAS '98) Proceedings. IEEE Computer Society, pp 128–135
9. Ferber J, Gutknecht O (2000) Operational semantics of a role-based agent architecture. In: Jennings NR, Lesperance Y (eds) Intelligent Agents IV, Proceedings of the 6th Int. Workshop on Agent Theories, Architectures and Languages. Lecture Notes in AI, vol. 1757, Springer-Verlag, pp 205–217
10. Ferber J, Gutknecht O, Jonker CM, Müller JP, Treur J (2000) Organisation models and behavioural requirements specification for multi-agent systems. In: Demazeau Y, Garijo F (eds) Multi-agent system organisations. Proc. of the 10th European workshop on modelling autonomous agents in a multi-agent world, MAAMAW'01, pp 1–19
11. Ferber J, Gutknecht O, Michel F (2004) From agents to organizations: an organizational view of multi-agent systems. In: Giorgini P, Muller JP, Odell J (eds) Agent-oriented software engineering IV: Proc. of the fourth international workshop, AOSE 2003, Lecture Notes in AI, vol 2935, Springer Verlag, Berlin, 2004, pp 214–230
12. Fisher M (1994) A survey of concurrent metateM—the language and its applications. In: Gabbay DM, Ohlbach HJ (eds) Temporal logic—proceedings of the first international conference, Lecture Notes in AI, vol. 827, pp 480–505
13. Fisher M (2005) Temporal development methods for agent-based systems. J Auton Agents Multi-Agent Syst 10:41–66
14. Galton A (2003) Temporal logic. Stanford Encyclopedia of Philosophy, URL: http://plato.stanford.edu/ entries/logic-temporal/#2
15. Galton A (2006) Operators vs arguments: the ins and outs of reification. Synthese 150:415–441
16. Henzinger T, Nicollin X, Sifakis J, Yovine S (1994) Symbolic model checking for real-time systems. Inf Comput 111(2):193–244, Academic Press
17. Herlea DE, Jonker CM, Treur J, Wijngaards NJE (1999) Specification of behavioural requirements within compositional multi-agent system design. In: Garijo FJ, Boman M (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, pp 8–27
18. Herlea Damian DE, Jonker CM, Treur J, Wijngaards NJE (2005) Integration of behavioural requirements specification within compositional knowledge engineering. Knowl-Based Syst J 18:353–365
19. Hölldobler S, Thielscher M (1990) A new deductive approach to planning. New Gener Comput 8:225–244
20. Hodkinson I, Reynolds M (2005) Separation – past, present and future. In: Artemov S, Barringer H, d'Avila Garcez AS, Lamb LC, Woods J (eds) We will show them: essays in honour of Dov Gabbay, Vol 2. College Publications, pp 117–142
21. Jonker CM, Letia IA, Treur J (2002) Diagnosis of the dynamics within an organisation by trace checking of behavioural requirements. In: Wooldridge M, Weiss G, Ciancarini P (eds) Proc. of the 2nd international workshop on agent-oriented software engineering, AOSE'01. Lecture Notes in Computer Science, vol. 2222. Springer Verlag, pp 17–32
22. Jonker CM, Treur J (2002) Compositional verification of multi-agent systems: A formal analysis of pro-activeness and reactiveness. Int J Coop Inf Syst 11:51–92. Earlier shorter version in: de Roever WP, Langmaack H, Pnueli A (eds) (1998) Proceedings of the international workshop on compositionality, COMPOS'97. Lecture Notes in Computer Science, Springer Verlag, vol. 1536, pp 350–380
23. Jonker CM, Treur J (2003) Relating structure and dynamics in an organisation model. In: Sichman JS, Bousquet F, Davidson P (eds) Multi-Agent-Based Simulation II, Proc. of the third international workshop on multi-agent based simulation, MABS'02. Lecture Notes in AI, vol. 2581, Springer Verlag, pp 50–69
24. Kowalski R, Sergot M (1986) A logic-based calculus of events. New Gener Comput 4:67–95
25. Manna Z, Pnueli A (1995) Temporal verification of reactive systems: safety. Springer Verlag.
26. McCarthy J, Hayes P (1969) Some philosophical problems from the standpoint of artificial intelligence. Mach Intell 4:463–502
27. Moss S, Gaylard H, Wallis S, Edmonds B (1998) SDML: A multi-agent language for organizational modelling. Comput Math Organ Theory 4(1):43–70
28. Port RF, van Gelder T (eds) (1995) Mind as motion: explorations in the dynamics of cognition. MIT Press, Cambridge, Mass
29. Prietula M, Gasser L, Carley K (1997) Simulating organizations. MIT Press
30. Reiter R (2001) Knowledge in action: Logical oundations for specifying and implementing dynamical systems. MIT Press, Cambridge, MA
31. Stirling C (2001) Modal and temporal properties of processes. Springer Verlag
32. Yovine S (1997) Kronos: a verification tool for real-time systems. Int J Softw Tools Technol Transfer 1:123–133