Diagnosis of the Dynamics within an Organisation by Trace Checking of Behavioural Requirements

Catholijn Jonker
Vrije Universiteit Amsterdam
Department of Artificial Intelligence
De Boelelaan 1081a
1081 HV Amsterdam The Netherlands
Tel. +31 20 444 7743
jonker@cs.vu.nl

Ioan Alfred Letia
Technical University
Department of Computer Science
Baritiu 28
RO-3400 Cluj-Napoca Romenia
letia@cs-gw.utcluj.ro

Jan Treur
Vrije Universiteit Amsterdam
Department of Artificial Intelligence
De Boelelaan 1081a
1081 HV Amsterdam The Netherlands
Tel. +31 20 444 7763
treur@cs.vu.nl

Abstract

The main question addressed in this paper is how requirements on the dynamics of an organisation model can be specified and how the dynamics of such an organisation can be formally analysed. A specification language is proposed, and a number of different types of requirements for dynamics at different levels in the organisation are identified. Based on a logical analysis and a software environment to check requirements against traces of the dynamics, a diagnostic method is proposed to analyse the malfunctioning of an organisation, and pinpoint causes of malfunctioning.

1. INTRODUCTION

Organisation modelling aims at abstracting from agents and their interaction within a complex multi-agent system using notions such as role, interaction and group structure (cf. [8], [9]). The notion of role, for example, is independent of any particular agent fulfilling this role. Role interactions define the relationships between roles. A group structure is a set of roles and interactions between them. The advantage of organisation modelling is to deal more adequately with complexity, in particular for multi-agent systems with nontrivial global behaviour.

In recent years Requirements Engineering for distributed and agent systems has been studied in more depth, e.g., [3], [4], [11], [15]. At the level of the multi-agent system, requirements concern the dynamics of interaction and cooperation patterns. At the level of individual agents, requirements concern agent behaviour. Due to the dynamic complexity, specification and analysis of such requirements is a difficult process. The importance of using more abstract and intuitive notions in requirements specification, as opposed to more directly formulated behaviour constraints, is emphasised in, e.g., [3]. Below organisational concepts are used to serve this purpose. Because of their intuitive meaning and conciseness, such notions are easy to understand.

For description of multi-agent systems from the organisational point of view, the agent/group/role model, adopted from [8] is used. An organisation is based on a definition of groups, roles and their relationships within the organisation. In relation to an organisation model four different types of requirements are distinguished (Section 2), varying from global requirements for the organisation as a whole, to requirements for specific roles and for interactions between them. Section 3 briefly describes the example application used in the paper: a (simulated) organisation model for work flow related to a Call Center and a bank. In Section 4 a temporal trace language to formally specify behavioural requirements is briefly introduced. Using this language, in Section 5 a number of requirement specifications for the example organisation are presented.

For all different types of requirements discussed in this paper, for a given set of finite traces representing dynamics within the organisation, the requirements can be verified automatically. By specifying the refinement of a global requirement for the overall organisation in terms of more local requirements, in Section 6 it is discussed how it is possible to perform diagnosis of malfunctioning of the organisation. If the overall requirement fails on a given trace, then subsequently, all refined requirements for the parts of the organisation can be verified against that trace: the cause of the malfunctioning can be attributed to the part(s) of the organisation for which the refined requirement(s) fail(s). This diagnostic method is applicable both to data on the dynamics of simulated organisations and empirical data on the dynamics of real organisations. Section 7 is a discussion.

2. TYPES OF REQUIREMENTS

Based on an organisational structure, the following types of requirements are distinguished: single role behaviour requirements, intragroup interaction requirements, intragroup communication successfulness requirements, intergroup interaction requirements. To be able to specify ongoing interaction between two roles for which multiple appearances exist, the notion of *role instance* is used. This notion abstracts from the agent realising the role as actor, but enables to distinguish between appearances of roles.

For a given role within a group, role behaviour requirements specify the dynamics of the role within the group. They are typically expressed in terms of temporal relationships between the *input* and *output* of a role instance. Intragroup role interaction requirements specify the temporal constraints on the dynamics of the interaction protocol between two roles within a group. Intragroup role interaction requirements between two roles instances in one group instance are typically expressed in terms of the *output* of both role instances. Intragroup role interaction requires *communication* within the group. Therefore, in order to function properly, requirements are needed that communications are successful. These requirements relate *output* of one role instance to *input* of another role instance in the same group.

Intergroup role interaction requirements 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 in one group instance and the *output* of the other role instance in the other group instance.

3. THE EXAMPLE ORGANISATION MODEL

The example organisation model was inspired by the application addressed in [1]. The organisation consists of a Call Center together with a (large) number of local banks. For simulating this organisation and visualizing the experiments Swarm[1], an agent-based simulator [18], was used. A basic agent model has been developed with the core functionality required for the organisation; see Figure 1 for an overview. This agent model has then been specialized in a bank employee (BE agent), a bank distributor (BD agent) and the Call Center distributor (CD agent). The organisational structure of the bank and call center is hierarchical with communication between Call Center Distributor (CD) and bank distributors (BD_i) and between bank distributors (BD_i) and bank employees (BE_{ij}).

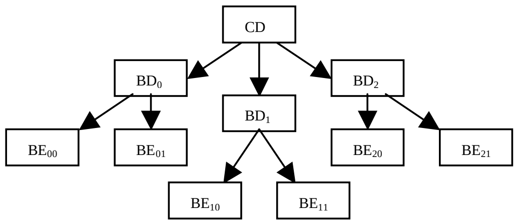

Figure 1 Call Center distributor (CD), bank distributors (BD_i) and bank employees (BE_{ij}).

In the experiments each local bank had nine employees. The tasks come from the Clients to the Call Center Distributor. The Call Center Distributor allocates the tasks for the three banks according to its policy and the three queues below are the ones sent to the Bank Distributors. The same procedure is repeated by the Bank Distributor for the Bank Employees that have to process the tasks.

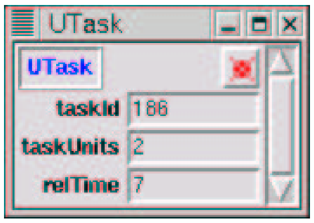
UTask
UTask
taskId 186
taskUnits 2
relTime 7

Figure 2 Task probe

A task within this organisation (shown as UTask in Figure 2) has an identifier (taskId), number of time units needed for its processing (taskUnits) and a relative time (relTime) specifying the time moment when the task has been promised to be processed. We can vary the policy used by distributors in the allocation of tasks for their subordinates. The availability of employees can also be varied with some degree of absenteeism. The distributors can all use the same policy for scheduling the tasks to their subordinates or can use different policies. The kind of exceptions that can be currently injected in the dynamics of the organisation are: (i) agent unavailable, (ii) task misrouted, and (iii) task delayed [6]. Currently the communication of tasks can be logged in both directions, to subordinates and to superiors. When the employees finish processing a task, they communicate it at the end of that particular time interval to their superior and from there it is communicated to the center distributor, also at the end of the same time interval. There is no time lag between accepting a task and its being sent to the bank which should process it, and similarly for acknowledging its finish back to the center distributor.

For tracing the communication of tasks between the Call Center Distributor and the Bank Distributors we have log files for the input and output of each of the role instances. These log files are used to generate the traces that are analysed by the checking software.

By testing for various situations, the designer of the multi-agent system can thoroughly check the requirements for interactions [17], actually see how well the organisation is able to manage exceptions that can occur during the enactment of a process [6], and uncover agents that do not comply with coordination protocols [19]. Unreliable coordination between agents and the effect of various decisionmaking policies on the behavior of the organisation as a whole is done by creating specific circumstances (e.g., exceptions) and observing behavior of the organisation under these circumstances. The tools presented in this paper provide support for such an analysis.

4. A TEMPORAL TRACE LANGUAGE

To specify requirements on the dynamics within the organisation, the temporal trace language used in [11], [12] is adopted. 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 |= between states and state properties is used: S |= p means that 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* $\mathcal{M}$ over an ontology Ont and time frame T is a mapping $\mathcal{M}$: T → STATES(Ont), i.e, a sequence of states $\mathcal{M}_t$ (t ∈ T) in STATES(Ont). The set of all traces over ontology Ont is denoted by TRACES(Ont) , i.e., TRACES(Ont) = STATES(Ont)^T.

Comparable to the approach in situation calculus, the sorted predicate logic temporal trace language TTL is built on atoms referring to, e.g., traces, time and state properties, such as state($\mathcal{M}$, t, output) |= p. Here |= is a predicate symbol in the language, 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

[1] Swarm home page http://www.santafe.edu/projects/swarm/

elements as abbreviations of formulae of the temporal trace language.

States of a trace can be related to state properties via the formally defined satisfaction relation $\models$ between states and formulae. If $\varphi \in$ SPROP(InOnt), then state($\mathcal{M}$, t, input) $\models \varphi$ denotes that φ is true in the state of $\mathcal{M}$ at time point t.

Ontologies can be specific for a role. In Section 5, for simplicity explicit reference to the specific ontologies per role are omitted; the ontology elements used can be read from the requirements themselves.

5. EXAMPLE REQUIREMENTS

In this section behavioural requirements related to the example are fomalised using the temporal trace language introduced in Section 4. Within these requirements specifications universal and existential quantification over role instances can occur. Role instances are denoted by I:R where R is a role. To be able to specify requirements for the example organisation, first the following terms are introduced. The organisation model used for the bank and Call Center consists of two groups: DISTRIBUTION, OPEN_GROUP. For the group OPEN_GROUP only one instance exists, open_group, standing for the Call Center. For the group DISTRIBUTION the following instances exist, cc, lb1, ..., lbn, standing for the cooperation between Call Center and local bank managers (cc), and for each of the local banks and their employees (lbi). Within the group DISTRIBUTION two roles exist: DISTRIBUTOR, PARTICIPANT. For these roles the following instances are distinguished: d: cc (distributor role instance within group instance cc), pcc1, ..., pccn : cc (n participant role instances within group instance cc), d: lb1 (distributor role instance within group instance lb1), …, d: lbn, p11: lb1 (first participant role instance of group instance lb1), …, p1m: lb1 (m-th participant role instance of group instance lbn), …, pn1: lbn, ..., pnm: lbn. Within the group OPEN_GROUP, the following roles exist: RECEPTIONIST, CLIENT. Finally, the following role instances are distinguished: cl: open_group (client role instance within group instance open_group), and rec: open_group (receptionist role instance within group instance open_group).

Variable introduction at the beginning of a formula is done as $\forall$ V: S, or $\exists$ V: S, where S is the sort of the variable, and V is the variable name. Introducing several variables of the same sort can be done like: $\forall$ V1, V2, ..., Vn: S. Variables used in the article are:

- GI: GROUP is a variable ranging over all possible group instances of group GROUP. Examples: GI: DISTRIBUTION, GI: OPEN_GROUP.
- RI: ROLE: gi is a variable ranging over all possible role instances of role ROLE within group instance gi. Examples: P: PARTICIPANT: cc, P: PARTICIPANT: GI: DISTRIBUTION.
- t: T is a variable ranging over time.
- $\mathcal{M}$: TRACES is a variable ranging over traces.
- id : TaskId is a variable ranging over task identifiers.

When used within a formula, with the exception of the introduction of the variable, only the variable name of the variable is used. Example:

$\forall \mathcal{M}$: TRACES, $\forall$ t : T nr_of_finished($\mathcal{M}$, t, t) = 0

It is assumed that job names are unique. One job is presented to the organisation at a time.

The requirements are specified below in a form that is specific for the application. However, it is not difficult to reuse them over different applications of the same type of organisation.

5.1 Global Requirements

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

GR1. Every request is answered (either by rejecting or by accepting and finishing it)

GR2. No accepted jobs are lost: for every accepted job there is a time that that job is finished.

GR3. The ratio of accepted jobs over requested jobs is at least r.

GR4. The average delay of jobs is at most m.

These global requirements can be formalised as follows. The first requirement 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 finished to that client.

GR1 All requests answered

$\forall \mathcal{M}$: TRACES, $\forall$ tid : TaskId, $\forall$ t1, tf : T,

$\forall$ C: CLIENT:open_group, $\forall$ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t1, output(C)) $\models$ comm_from_to(requested(tid, tf), C, R)

$\Rightarrow$ $\exists$ t2 : T [t2 $\geq$ t1

& [state($\mathcal{M}$, t2, input(C)) $\models$ comm_from_to(rejected(tid), R, C)

$\vee$ state($\mathcal{M}$, t2, input(C)) $\models$ comm_from_to(finished(tid), R, C)]]

The next requirement 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 the receptionist communicates to the same client that the task was finished.

GR2 No lost jobs

$\forall \mathcal{M}$: TRACES, $\forall$ id : TaskId, $\forall$ t, t1 : T,

$\forall$ C: CLIENT: open_group, $\forall$ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t1, input(C)) $\models$ comm_from_to(accepted(id, t), R, C)

$\Rightarrow$ $\exists$ t2 : T [t2 > t1

& state($\mathcal{M}$, t2, input(C)) $\models$ comm_from_to(finished(id), R, C)]]

For the next two requirements additional specifications are needed to define the frequency functions used. For shortness' sake these are left out; see, however, the Appendix. The first one, G3, can be viewed as a liveness property: it indicates that at least a certain amount of 'good events' in the sense of request acceptances must happen.

GR3 Acceptable ratio of accepted jobs over [t1, t2]

$\forall \mathcal{M}$: TRACES

nr_of_ accepted($\mathcal{M}$, t1, t2) / nr_of_ requested($\mathcal{M}$, t1, t2) $\geq$ r

requirement G4 below can be viewed as a safety requirement: it indicates that only limited 'bad events' in the sense of delays can happen.

GR4 Acceptable average delay of accepted jobs over [t1, t2]

$\forall \mathcal{M}$: TRACES average_delay($\mathcal{M}$, t1, t2) $\leq$ m

In the next four subsections the different requirements on parts of the organisation are identified. In Figure 3 below an overview can be found. In this figure three group instances are depicted, together with two role instances in each of them. Requirements 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 requirement 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 requirement refers. The requirements distinghuished in Figure 3 and specified in detail below are selected in order to be able to derive global requirement GR1 from them. In Section 6.1 this will be addressed in more detail.

5.2 Intragroup Role Interaction Requirements

Intragroup role interaction requirements specify the cooperation within a group. Within each group instance at least one intragroup role Interaction requirements is specified. The first one specifies that within the open group proper interaction takes place: if a client communicates a request, then some time later, either the request will be rejected, or finished.

IaRI1 Client-Receptionist Intragroup Interaction

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[state(ℳ, t1, output(C)) |= comm_from_to(requested(tid, tf), C, R)

⇒ ∃ t2 : T [t2 ≥ t1 &

[state(ℳ, t2, output(R)) |= comm_from_to(rejected(tid), R, C)

∨ state(ℳ, t2, output(R)) |= comm_from_to(finished(tid), R, C)]]]

The next requirement 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 having it finished.

IaRI2/IaRI3 Distributor-Participant Intragroup Interaction

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T, ∀ GI: DISTRIBUTION,

∀ D: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P: PARTICIPANT: GI: DISTRIBUTION

[state(ℳ, t1, output(D)) |= comm_from_to(requested(tid,tf), D, P)

⇒ ∃ t2 : T [t2 ≥ t1 &

[state(ℳ, t2, output(P)) |= comm_from_to(rejected(tid), P, D)

∨ state(ℳ, t2, output(P)) |= comm_from_to(finished(tid), P, D)]]]

5.3 Intergroup Role Interaction Requirements

Intergroup role interaction requirements 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 requirement 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

∀ ℳ: 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(ℳ, t1, input(R)) |= comm_from_to(requested(tid, tf), C, R)]

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ, t2, output(D)) |= comm_from_to(requested(tid, tf), D, P)]]

The next intergroup role interaction requirement 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 requirement 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 requirement (and other requirements) info ranges over { finished(tid), rejected(tid), accepted(tid) }.

IrRI2 Distributor-Receptionist Intergroup Interaction

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ D: DISTRIBUTOR: cc, ∀ P: PARTICIPANT: cc,

∀ R: RECEPTIONIST: open_group, ∀ C: CLIENT: open_group

[[state(ℳ, t1, input(D)) |= comm_from_to(info, P, D)

& intergroup_role_relation(D, R)]

⇒ ∃ t2 : T

[t2 ≥ t1 & state(ℳ, 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

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ D1: DISTRIBUTOR: cc, ∀ P1: PARTICIPANT: cc,

∀ GI: DISTRIBUTION, ∀ D2: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P1: PARTICIPANT: GI: DISTRIBUTION,

[[state(ℳ, t1, input(P1)) |= comm_from_to(requested(tid, tf), D1, P1)

& intergroup_role_relation(P1, D2)]

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2,output(D2)) |= comm_from_to(requested(tid,tf),D2,P2)]]

IrRI4 Distributor-Participant Intergroup Interaction

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ D1: DISTRIBUTOR: cc, ∀ P1: PARTICIPANT: cc,

∀ GI: DISTRIBUTION, ∀ D2: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P1: PARTICIPANT: GI: DISTRIBUTION,

[[state(ℳ, t1, input(D2)) |= comm_from_to(info, P2, D2)

& intergroup_role_relation(D2, P1)]

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2,output(P1)) |= comm_from_to(info,P1,D1)]]

5.4 Transfer Requirements

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

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[state(ℳ, t1, output(C)) |= comm_from_to(requested(tid, tf), C, R)

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2, input(R)) |= comm_from_to(requested(tid, tf), C, R)]]

Moreover, they also have to communicate about acceptance, rejectance or finishing of tasks:

TR2 Client-Receptionist communication

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1: T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[state(ℳ, t1, output(R)) |= comm_from_to(info, R, C)

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2, input(C)) |= comm_from_to(info, R, C)]]

Similarly within the distribution groups proper communication has to take place about requests and what comes back for them:

TR3/TR5 Distributor-Participant communication

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ GI: DISTRIBUTION, ∀ D: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P: PARTICIPANT: GI: DISTRIBUTION

[state(ℳ, t1, output(D)) |= comm_from_to(requested(tid, tf), D, P)

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2, input(P)) |= comm_from_to(requested(tid, tf), D, P)]]

TR4/TR6 Distributor-Participant communication

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1: T,

∀ GI: DISTRIBUTION, ∀ D: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P: PARTICIPANT: GI: DISTRIBUTION

[state(ℳ, t1, output(P)) |= comm_from_to(info, P, D)

⇒ ∃ t2 : T [t2 ≥ t1

& state(ℳ,t2, input(D)) |= comm_from_to(info, P, D)]]

5.5 Single Role Behaviour Requirements

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. If they do not reject a task, they have to finish it, as is expressed below:

PB1 Participant behaviour

∀ ℳ: TRACES, ∀ tid : TaskId, ∀ t1, tf : T,

∀ GI: DISTRIBUTION, ∀ D: DISTRIBUTOR: GI: DISTRIBUTION,

∀ P: PARTICIPANT: GI: DISTRIBUTION

[state(ℳ, t1, input(P)) |= comm_from_to(requested(tid, tf), D, P)

⇒ ∃ t2 : T [t2 ≥ t1 &

[state(ℳ, t2, output(P)) |= comm_from_to(rejected(tid), P, D)

∨ state(ℳ, t2, output(P)) |= comm_from_to(finished(tid), P, D)]]]

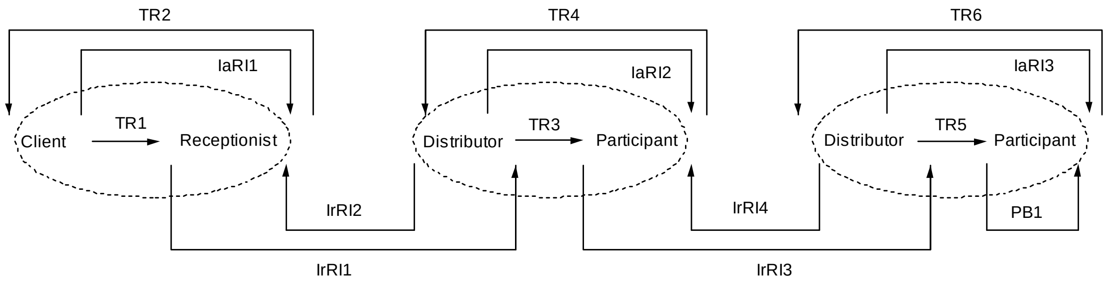

Figure 3 Overview of the non-global properties

6. DIAGNOSIS OF AN ORGANISATION

In this section it will be shown how the palette of requirements of different types identified in Section 4 can be used to perform diagnosis of the dynamics within an organisation. Before such a diagnostic process can be started, first a logical analysis is made of the relationships between global requirements and more local requirements for the organisation (Section 6.1). Next a software environment to check behavioural requirements against traces is briefly discussed (Section 6.2). Finally, in Section 6.3 it is discussed how the logical analysis and the checking software environment can be used within a systematic diagnostic process.

6.1 Logical Relationships between the Requirements

Figure 3 shows possible logical relationships between different types of requirements. For example, within the rightmost group instance, the arrows for transfer requirement TR5 and role behaviour requirement PB1 'chain' in an appropriate manner to intragroup interation requirement IaRI3. Indeed, logically the latter requirement can be derived from the former two. This obtains a proof pattern

TR5 & PB1 ⇒ IaRI3

In a similar manner other proof patterns have been identified and actually proven for the intragroup interaction requirements IaRI1 and IaRI2, making use of inter group interaction requirements, transfer requirements, and (other) intragroup requirements:

TR3 & IrRI3 &
IaRI3 &
TR6 & IrRI4 ⇒ IaRI2

TR1 & IrRI1 &
IaRI2 &
TR4 & IrRI2 ⇒ IaRI1

Finally, the global requirement GR1 can be derived from IaRI1 and TR2.

IaRI1 & TR2 ⇒ GR1

These proof patterns, depicted in Figure 4 as an AND-tree, 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 requirement GR1 is not satisfied, then by the last proof pattern, by a refutation process it can be concluded that either transfer does not function properly or IaRI1 does not hold. If IaRI1 does not hold, then by one of the other proof patterns either IaRI2 does not hold, or one of the intergroup interaction requirements IrRI1 or IrRI2 does not hold, (or transfer fails). If the intragroup requirement IaRI2 does not hold, then either either IrRI3, IrRI4 or IaRI3 does not hold (or transfer fails). Finally, if IaRI3 does not hold, then by the first proof pattern either role behaviour requirement PB1 does not hold or transfer is not properly functioning. By this refutation analysis it follows that if GR1 does not hold for a given trace, then, skipping the intermediate requirements, the cause of this malfunctioning can be found in the set (the leaves of the tree in Figure 4):

{IrRI1, IrRI2, IrRI3, IrRI14} ∪ {PB1} ∪ {TR1, .., 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.

6.2 Checking the Temporal Trace Formulae

To check whether a given behavioural requirement is fulfilled in a given trace or set of traces, a Prolog programme has been developed. The temporal formulae are represented by nested term structures based on the logical connectives. For example, requirement GR1 from Section 4 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(2), input(role)), a)), true).
```

where m1 is the trace name, t(2) time point 2, and a is a state formula in the ontology of the agent's input. It is indicated that state formula a is true in the role's input 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).
```

that reduce the satisfaction of the temporal formula finally to the satisfaction of atomic state formulae at certain time points, which can be read from the trace.

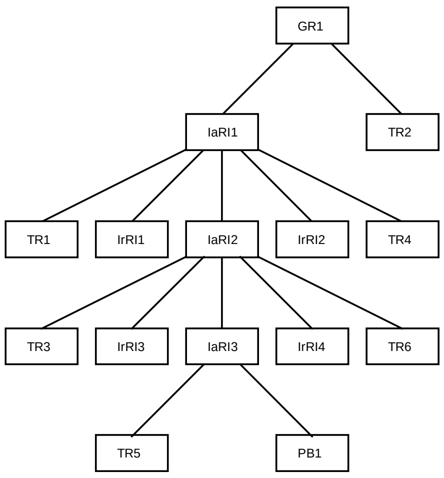

Figure 4 AND-tree of requirements of different types

6.3 Diagnostic Method

Returning to the verification of the global organisation property GR1, if the check shows that it is not satisfied, then subsequently, the candidate set of causes {IrRI1, IrRI2, IrRI3, IrRI14} ∪ {PB1} ∪ {TR1, .., 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 requirements 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 manner, based on the tree in Figure 4 (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 Figure 4)
2. Focus the subsequent checking process on only those more local properties that in view of the logical analysis relate to a 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 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 of depth n).

This method has been used to analyse the organisation simulation model presented in Section 3. 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. In particular, the tool can be used for debugging of the simulation model. Another type of application is to analyse empirical data on the dynamics within a real organisation. Because it is not easy to obtain such empirical data about the dynamics, this application has not been performed yet.

7. DISCUSSION

This paper contributes a framework to analyse the dynamics within an organisation. One part of the framework is a temporal trace language to formally specify behavioural requirements of different types within the organisation. Between different behavioural requirements specified in this language, logical relationships can be identified. A second part is a software environment to check behavioural requirements against a (set of) trace(s). The framework was tested by linking it to an organisation simulation model implemented in Swarm. Traces generated by the simulation model were automatically checked by the checking software. The interface between the two parts of the software is defined on the basis of the log files created within the simulation model of the states of the different parts of the organisation (i.e., input and output of the different role instances) over time. Since this is a very general notion, the approach can easily be applied using other simulation software. Another application is to use empirical traces of a real organisation.

By a systematic use of the framework a diagnostic method can be followed that is based on:

- a formal analysis of logical relationships between global behavioural properties and local behavioural properties; i.e., a tree such as the one depicted in Figure 4, obtained from a logical analysis of the requirements
- top down checking of behavioural requirements against traces

This diagnostic method for a malfunctioning organisation first checks the global properties, next focuses the subsequent checking process on only these more local properties that in view of the logical analysis relate to a global property that has turned out false, and finally identifies the most local properties that fail; they point at where the cause of malfunctioning can be found.

This method obtains its efficiency from the fact that all more refined properties that (within the tree) are not a refinement of a failing more global property can be left out of consideration. Depending on the shape of the tree, and assuming that only one failure arises (single fault hypothesis), this may obtain a linear versus exponential advantage in the number of properties to be checked, compared to simply checking all properties.

The experiments carried out so far have already shown the advantage of the rapid prototyping in analysing processes at various levels of abstraction [16]. It enables the designer to better understand the dynamic impact of organisational rules, organisational structures, and organisational patterns [20]. Having an abstract prototype at hand is of great help for communication between designers. Also, new agent-oriented modeling techniques can be tested before actually using them within a given methodology [7].

Monitoring of the multi-agent system is currently done within the environment of the simulated system. The next phase of our project will include monitoring agents (exception handling agents) for performing instrumentation, diagnosis and resolution tasks [5], [14] on the line of socially-attentive monitoring of failures in the social relationships between agents [13].

Future research will aim at building a library of different reusable organisation models together with associated sets of behavioural requirements at different organisational levels (and their logical relationships).

REFERENCES

[1] Brazier, F. M. T., Jonker, C. M., Jungen, F. J., and Treur, J., Distributed Scheduling to Support a Call Centre: a Co-operative Multi-Agent Approach. In: *Applied Artificial Intelligence Journal*, vol. 13, 1999, pp. 65-90. H. S. Nwana and D. T. Ndumu (eds.), Special Issue on Multi-Agent Systems.

[2] S. Bussmann, N.R. Jennings, and M. Wooldridge. On the identification of agents in the design of production control systems. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, LNCS 1957. Springer-Verlag, 2001.

[3] Dardenne, A., Lamsweerde, A. van, and Fickas, S. (1993). Goal-directed Requirements Acquisition. *Science in Computer Programming*, vol. 20, pp. 3-50.

[4] Darimont, R., and Lamsweerde, A. van (1996). Formal Refinement Patterns for Goal-Driven Requirements Elaboration. In: *Proc. of the Fourth ACM Symposium on the Foundation of Software Engineering (FSE4)*, pp. 179-190.

[5] C. Dellarocas and M. Klein. An experimental evaluation of domain-independent fault handling services in open multi-agent

systems. In: *Proceedings of the 4th International Conference on Multi-Agent Systems (ICMAS-2000),* Boston, MA, 2000.

[6] C. Dellarocas and M. Klein. A knowledge-based approach for handling exceptions in business processes. *Information Technology and Management* , 1:155--169, 2000.

[7] R. Depke, R. Heckel, and J.M. Kuster. Formal agent-oriented modeling with graph transformation. In: *Science of Computer Programming*, 2001, to appear.

[8] Ferber, J. and Gutknecht, O. (1998). A meta-model for the analysis and design of organisations in multi-agent systems. In: *Proc. of the Third International Conference on Multi-Agent Systems (ICMAS '98) Proceedings.* IEEE Computer Society, 1998

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

[10] Ferber, J., Gutknecht, O., Jonker, C.M., Mueller, J.P., and Treur, J., Organisation Models and Behavioural Requirements Specification for Multi-Agent Systems (extended abstract). In: *Proc. of the Fourth International Conference on Multi-Agent Systems, ICMAS 2000.* IEEE Computer Society Press, 2000. Extended version in: *Proc. of the ECAI 2000 Workshop on Modelling Artificial Societies and Hybrid Organisations*, 2000.

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

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

[13] G.A. Kaminka and M. Tambe. Robust agent teams via socially-atentive monitoring. In: *Journal of Artificial Intelligence Research*, 12:105--147, 2000.

[14] M. Klein and C. Dellarocas. Exception handling in agent systems. In O. Etzioni, J. Muller, and J. Bradshaw, editors, In: *Proceedings of the 3rd International Conference on Autonomous Agents (AA'99)*, pages 62--68, 1999.

[15] Kontonya, G., and Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques.* John Wiley and Sons, New York.

[16] T. W. Malone, K. Crowston, J. Lee, B. Pentland, C. Dellarocas, G. Wyner, J. Quimby, C. S. Osborn, A. Bernstein, G. Herman, M. Klein, and E. O'Donnell. Tools for inventing organisations: Toward a handbook for organizatinal processes. In: *Management Science*, 45:425-443, 2000.

[17] S. Miles, M. Joy, and M. Luck. Designing agent-oriented systems by analysing agent interactions. In P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, LNCS 1957. Springer-Verlag, 2001.

[18] M. Minar, R. Burkhart, C. Langton, and M. Askenazy. *The Swarm simulation system: A toolkit for building multi-agent simulations*. Technical report, Santa Fe Institute, 1996. http://www.santafe.edu/projects/swarm/.

[19] M. Venkatraman and M.P. Singh. Verifying compliance with commitments protocols: enabling open web-based multiagent systems. In: *Autonomous Agents and Multi-Agent Systems*, 2:217--236, 1999.

[20] F. Zambonelli, N.R. Jennings, and M. Wooldridge. Organisational abstractions for the analysis and design of multi-agent systems. In: P. Ciancarini and M. Wooldridge, editors, *Agent-Oriented Software Engineering*, LNCS 1957. Springer-Verlag, 2001.

Appendix More details of Requirements Specifications

Relations used:

nr_of_requested:	TRACES x T x T → N
nr_of_accepted:	TRACES x T x T → N
nr_of_rejected:	TRACES x T x T → N
nr_of_finished:	TRACES x T x T → N
average_delay:	TRACES x T x T → N
acc_of_delay:	TRACES x T x T → N

Requested jobs

∀ $\mathcal{M}$: TRACES, ∀ t1, t2, t3 : T

[t1 ≤ t2 ≤ t3

⇒ nr_of_requested($\mathcal{M}$, t1, t3) = nr_of_requested($\mathcal{M}$, t1, t2) +

nr_of_requested($\mathcal{M}$, t2, t3)]

∀ $\mathcal{M}$: TRACES, ∀ t : T: nr_of_requested($\mathcal{M}$, t, t) = 0

∀ $\mathcal{M}$: TRACES, ∀ t : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[nr_of_requested($\mathcal{M}$, t, t+1) =

1 if ∃ id : TaskId

state($\mathcal{M}$, t, output(C)) |= comm_from_to(requested(id, t), C, R)

0 otherwise]

Accepted jobs

∀ $\mathcal{M}$: TRACES, ∀ t : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[nr_of_accepted($\mathcal{M}$, t, t+1) =

1 if ∃ id : TaskId

state($\mathcal{M}$, t, input(C)) |= comm_from_to(accepted(id, t), R, C)

0 otherwise]

Rejected jobs

∀ $\mathcal{M}$: TRACES, ∀ t : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[nr_of_rejected($\mathcal{M}$, t, t+1) =

1 if ∃ id : TaskId

state($\mathcal{M}$, t, input(C)) |= comm_from_to(rejected(id, t), R, C)

0 otherwise]

Finished jobs

∀ $\mathcal{M}$: TRACES, ∀ t : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[nr_of_finished($\mathcal{M}$, t, t+1) =

1 if ∃ id : TaskId

state($\mathcal{M}$, t, input(C)) |= comm_from_to(finished(id), R, C)

0 otherwise]

Delayed jobs

∀ $\mathcal{M}$: TRACES, ∀ id : TaskId, ∀ t, t1, t2 : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[[state($\mathcal{M}$, t1, input(C)) |= comm_from_to(accepted(id, t), R, C)

& state($\mathcal{M}$, t2, input(C)) |= comm_from_to(finished(id), R, C)

& t1 ≥ t2] ⇒ delay(id) = t2 - t]

Too soon jobs

∀ $\mathcal{M}$: TRACES, ∀ id : TaskId, ∀ t, t1, t2 : T

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[[state($\mathcal{M}$, t1, input(C)) |= comm_from_to(accepted(id, t), R, C)

& state($\mathcal{M}$, t2, input(C)) |= comm_from_to(finished(id), R, C)

& t2 ≥ t1]

⇒ tooSoon(id) = t – t2]

All jobs done

∀ $\mathcal{M}$: TRACES, ∀ id : TaskId, ∀ t, t1 : T,

∀ C: CLIENT: open_group, ∀ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t1, output(C)) |= comm_from_to(requested(id, t), R, C)

⇒ ∃ t2 : T [t2 > t1

& state($\mathcal{M}$, t2, input(C)) |= comm_from_to(finished(id), R, C)]]

Accumulation of delay

∀ $\mathcal{M}$: TRACES, ∀ t : T acc_of_delay($\mathcal{M}$, t, t) = 0

∀ $\mathcal{M}$: TRACES, ∀ t1, t2, t3 : T [t1 ≤ t2 ≤ t3

⇒ acc_of_delay($\mathcal{M}$, t1, t3) = acc_of_delay($\mathcal{M}$, t1, t2) +

acc_of_delay($\mathcal{M}$, t2, t3)]

version 2:

∀ $\mathcal{M}$: TRACES, ∀ t : T acc_of_delay($\mathcal{M}$, t, t+1) = delay($\mathcal{M}$, S($\mathcal{M}$,t))

where

S($\mathcal{M}$,t) = { id : TaskId | ∃ C: CLIENT: open_group,

∃ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t, input(C)) |= comm_from_to(finished(id), R, C)] }

∀ id : TaskId

[element_of(id, S($\mathcal{M}$, t))

⇒ ∃ C: CLIENT: open_group, ∃ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t, input(C)) |= comm_from_to(finished(id), R, C)]]

∀ id : TaskId

[not element_of(id, S($\mathcal{M}$, t))

⇒ ∃ C: CLIENT: open_group, ∃ R: RECEPTIONIST: open_group

[state($\mathcal{M}$, t, input(C)) |≠ comm_from_to(finished(id), R, C)]]

Let S be a variable over the sort of sets of job names.

∀ S delay(S) = $\sum_{id\,:\,S}$ delay(id)

∀ S, ∀ id : TaskId delay(S ∪ {id}) = delay(S) + delay(id)

delay(∅) = 0

element_of(id', S∪{id}) if element_of(id', S) ∨ id' = id

Average delay

∀ $\mathcal{M}$: TRACES, ∀ t1, t2 : T

[average_delay($\mathcal{M}$, t1, t2) =

acc_of_delay($\mathcal{M}$, t1, t2) / nr_of_finished($\mathcal{M}$, t1, t2)