Compositional Verification of Agents in Dynamic Environments: a Case Study

Catholijn M. Jonker, Jan Treur, Wieke de Vries

Vrije Universiteit Amsterdam
Department of Mathematics and Computer Science, Artificial Intelligence Group
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
URL: http://www.cs.vu.nl/~{jonker,treur,wieke} Email: {jonker,treur,wieke}@cs.vu.nl

Abstract. In this paper compositional verification of agents in dynamic environments is studied. Dynamic properties of an example agent in a dynamic environment are identified in relation to the different abstraction levels of the compositional structure of the system. The properties are formalised using temporal models. Mathematical proofs relate the properties at the different process abstraction levels. The dynamics of the environment has several consequences for the verification process. Properties often have to contain conditions concerning the dynamic behaviour of the world. In the proofs, the partly unpredictable behaviour of the word has to be taken into account. This complicates the verification process. A number of aspects of proof pragmatics (i.e., heuristics for finding proofs) identified during this analysis and aimed at controlling the proof complexity, are discussed.

1 Introduction

With the increase of the complexity of systems and the sensitivity of those systems with respect to security, safety, and costs, the need for verification becomes more important every day. The purpose of verification is to prove that, under a certain set of assumptions, a system will adhere to a certain set of properties, for example the design requirements; see also, e.g., (Fensel, 1995; Fensel and Benjamins, 1996; Fensel, Schonegge, Groenboom, and Wielinga, 1996; Harmelen and Teije, 1997). In our approach, a mathematical proof (i.e., a proof in the form mathematicians are accustomed to do) is given that the (detailed) specification of the system together with the assumptions implies the properties that it needs to fulfil. In this sense verification leads to a formal analysis of relations between properties and assumptions.

Given the increasing complexity of systems to be verified, the need for a systematic approach to verification that leads to a comprehensible proof is paramount. The first experiences with the compositional verification method introduced in (Cornelissen, Jonker, and Treur, 1997) for knowledge-based systems with a case study in diagnostic reasoning, and in (Jonker and Treur, 1998a) for multi-agent systems applied to a case study in reactiveness and pro-activeness of agents acquiring information about a static world, were very promising. The proofs are structured in comprehensible manner and constructed by making use of the compositional structure of the system being verified. Although the systems verified in these first attempts are reasonably complex, the material (or external) world considered is static. In this paper the compositional verification method is applied to formally analyse an agent system with a dynamic world.

In the verification process again the different abstraction levels of the compositional structure of the system were used to identify dynamic properties of the agent, of the external world and of the interaction between them. Mathematical proofs relate the properties at the different process abstraction levels. This mathematical style is used because it is the most general way to formalise and verify system behaviour; it provides maximal expressive freedom. Using a formal logic, with a limited number of operators, could be too constraining in the present phase of the research. But in the future, a formal logic will be chosen to conduct the verification. To explore demands on this logic, the mathematical proof style is useful. The properties are formalised using temporal models with incomplete states (to express ignorance). The dynamics of the external world proved to make verification more complex; more effort was needed to create comprehensible proofs. Valuable experience was gained during this analysis that resulted in a number of basic assumptions and workable heuristics for finding proofs.

This case study is used to identify possibilities as well as problems that are encountered in the verification of multi-agent systems in dynamic environments. The properties verified in this paper are relevant for multi-agent systems in dynamic environments, although their formalization is domain-dependent. For other agent systems in dynamic worlds, the properties only have to be adjusted to the particular application domain. A general strategy is used to obtain the properties and to prove them. Genericity and reusability are of major importance. The aim of the research is to find a general approach of verification of multi-agent systems that are developed in a compositional and conceptual manner. To obtain this approach, a series of case studies is being performed, including the one described in this paper. The approach should identify common aspects of verification of all kinds of multi-agent systems. For example, there are types of properties that play a role in many multi-agent systems. Also, general applicable heuristics to reduce the search space of the proofs to a manageable size have been found (and more will have to be found). Using this approach of multi-agent system verification, the verification process will become more structured and algorithmic in nature. Tools to execute and/or aid with the verification could then be created.

Section 2 contains an overview of the compositional verification approach, which constitutes the foundation of the sought-for verification approach. In Section 3 a problem description (a description of a pseudo-experiment) of animal behaviour is presented, the basic requirements are formulated, and the model is presented, that is to be verified in the subsequent sections. Section 4 contains the top-level properties of the system, that are proven in Section 5, using assumptions on the behaviour of the dynamic world. Some of these assumptions appear in Section 4. Section 6 discusses basic assumptions that play a central role. In Section 7 proof heuristics that (may) play a role in the verification of dynamic systems are identified. Section 8 discusses the interaction between design processes and verification processes. Conclusions and further perspectives are discussed in Section 9.

2 Compositional Verification of Dynamic Properties

The complexity of the verification process is one of the major concerns in verification of non-trivial systems. In particular, for verification of dynamic properties of a system, a huge search space has to be faced. Compositional verification is an approach meant to handle this complexity, by structuring proofs according to different process abstraction levels; e.g., (Cornelissen, Jonker and Treur, 1997; Jonker and Treur, 1998a); see also (Abadi and Lamport, 1993; Hooman, 1994; Dams, Gerth and Kelb, 1996). A compositional multi-agent system can be viewed at different levels of process abstraction. Viewed from the top level, the complete system is one process (modelled by a component S), with interfaces, whereas internal information and processes are hidden (information and process hiding). At the next lower level of abstraction, the system component S can be viewed as a composition of agents and the world, and information links between them. Each agent is composed of its sub-components, and so on. Compositional verification takes this compositional structure into account: it plays a heuristic role in finding the properties and proofs.

2.1 Verification and Levels of Process Abstraction

Often the properties that need to be verified are not given at the start of the verification process. Actually, the process of verification has two main aims:

- to find the properties
- given the properties, to prove the properties

The verification proofs that connect one process abstraction level with the other are compositional in the following manner: any proof relating level i to level i+1 can be combined with any proof relating level i-1 to level i, as long as the same properties at level i are involved. This means, for example, that the whole compositional structure beneath level i can be replaced by a completely different design as long as the same properties at level i are achieved. After such a modification the proof from level i to level i+1 can be reused; only the proof from level i-1 to level i has to be adapted. In this sense the verification method supports reuse of verification proofs.

2.2 The Temporal Semantics Used

In principle, verification is always relative to semantics of the system descriptions that are verified. For our Compositional Verification approach, these semantics are based on compositional information states which evolve over time. In this subsection a brief overview of these assumed semantics is given.

An *information state* M of a component D is an assignment of truth values {true, false, unknown} to the set of ground atoms that play a role within D. The compositional structure of D is reflected in the structure of the information state. A formal definition can be found in (Brazier, Treur, Wijngaards and Willems, 1996). The set of all possible information states of D is denoted by IS(D).

A *trace* $\mathcal{M}$ of a component D is a collection of information states $(M^t)_{t \in \mathbf{T}}$ in IS(D) over a time structure **T**. For this paper **T** will be chosen as a dense ordering, e.g., the non-negative real numbers. The set of all traces is denoted by $IS(D)^{\mathbf{T}}$, or Traces(D). If C is a sub-component (or sub-sub-component, or ...) of D, by Traces(D)|C the restriction of the traces to C is meant, that is, only that part of each information state that pertains to C is considered. Given a trace $\mathcal{M}$ of component C, the information state of the input interface of component C' at time point t is denoted by state($\mathcal{M}$, t, input(C')), where C' is either C or a sub-component of C, a sub-sub-component of C, etc. Analogously, state($\mathcal{M}$, t, output(C')), denotes the information state of the output interface of component C' at time point t.

3 Problem Description

One of the most important aspects of agents (cf. (Wooldridge and Jennings, 1995)) is their behaviour. In the past, behaviour has been studied in different disciplines. In Cognitive Psychology the analysis of human behaviour is a major topic. In Biology, animal behaviour has been and is being studied extensively. In one approach animal behaviour is explained only in terms of a black box that for each pattern of *stimuli* (input of the black box) from the environment generates a *response* (output of the black box), that functionally depends on the input pattern of stimuli; i.e., if two patterns of stimuli are offered, then the same behaviour occurs if the two patterns of stimuli are equal. In this section a generic model of a *purely reactive agent* is briefly presented which is an adequate agent model to describe the (immediate) functional character of stimulus-response behaviour (Jonker and Treur, 1998b). The black box is represented by the agent component. The stimuli form the input (observation results), and the response is formed by the actions to be performed which are generated as output.

In this article a concrete example domain is considered that is taken from the discipline that studies animal behaviour; see e.g., (Vauclair, 1996).

3.1 The Domain

One type of experiment reported in (Vauclair, 1996) is set up as follows (see Figure 1). Separated by a transparent screen (a window, at position p0), at each of two positions p1 and p2 a cup (upside down) and/or a piece of food can be placed. At some moment (with variable delay) the screen is raised, and the animal is free to go to any position. Consider the following three possible situations:

Situation 1 At both positions p1 and p2 an empty cup is placed.

Situation 2 At position p2 a piece of food is placed, which is (and remains) visible for the animal. At position p1 there is nothing.

Situation 3 At position p1 an empty cup is placed and at position p2 a piece of food is placed, after which a cup is placed at the same position, covering the food. After the food disappears under the cup it cannot be sensed anymore by the animal.

In situation 1 the animal will not show a preference for either position p1 or p2; it may even go elsewhere or stay where it is. In situation 2 the animal will go to position p2, which can be explained as pure stimulus-response behaviour. In situation 3 the immediate stimuli are the same as in situation 1. Animals that react in a strictly functional stimulus-response manner will respond to this situation as in situation 1. Models of animals that show delayed response behaviour (and will go to p2, where food can be found) or other types of behaviour can be found in (Jonker and Treur, 1998b).

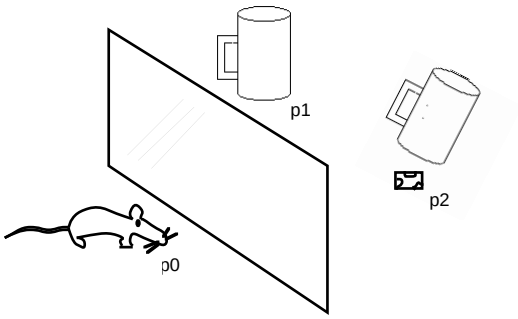


Fig. 1. Situation 3 of the experiment

3.2 The Requirements

In this paper a purely reactive agent model is described for the experiment. The following requirements on its behaviour are formulated: The agent should behave the same for the situations 1 and 3 described above: doing nothing, as if no food is present. Only in situation 2 should it go to the position of the food and eat it.

3.3 An Agent Model for Purely Reactive Behaviour

For the design and implementation of the different models the compositional development method for multi-agent systems DESIRE has been used; see (Brazier, Dunin-Keplicz, Jennings, and Treur, 1997) for more details. A generic agent model for purely reactive behaviour developed earlier within the DESIRE environment (and applied in chemical process control) was reused.

3.3.1 Process Composition

The (rather simple) agent system (denoted by S) consists of two components, one for the agent (denoted by M) and one for the external world (denoted by EW) with which it interacts (see Figure 2).

In the current domain, the observation information that plays a role describes that certain *objects* (cup1, cup2, food, screen, mouse, self) are at certain *positions* (i.e., p0, p1, p2). This is modelled by two sorts OBJECT and POSITION and a relation at_position between these two sorts. Moreover, two types of *actions* can be distinguished: eat and goto some position. The latter type of actions is parameterized by positions; this can be modelled by a function goto from POSITION to ACTION. E.g., goto(p1) is the action to go to position p1. The action eat that is specified assumes that if the animal is at the position of the food, it can have the food: if a cup is covering the food, as part of the action eat the animal can throw the cup aside to get the food. Variables over a sort, e.g., POSITION, are denoted by a string, e.g., P, followed by : POSITION, i.e., P : POSITION is a variable over the sort POSITION. The unary relation to_be_performed is used to express the information that the agent has decided to *perform an action*; for example, to_be_performed(goto(p1)) expresses that the agent has decided to go to position p1. The relation observation_result is used to express the meta-information that certain information has been acquired by *observation*; for example, observation_result(at_position(food, p1), pos) expresses that the agent has observed that there is food at position p1, whereas the statement observation_result(at_position(food, p1), neg) expresses that the agent has observed that there is *no* food at position p1.

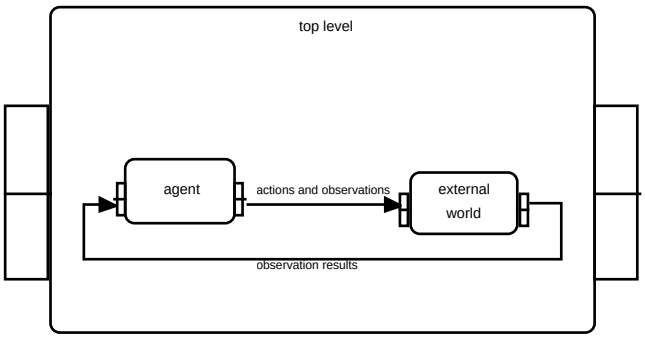


Fig. 2. A generic agent model for purely reactive behaviour

Within the process composition so far, the external world has been treated as a black box. This has the advantage that the system can easily be adapted to function either with a simulated world or with the real world. This approach is valuable for verification as the system can be verified up to the external world; leaving the external world as a black box assumed to satisfy certain properties. For a simulated external world the verification process can continue to be certain that the simulation has the required properties.

3.3.2 The Domain Knowledge

Assuming that food is offered at most one position (for example, position p2), the stimulus-response behaviour of agent model A expresses that if the agent observes that there is food at any position and that no screen at position p0 separates the agent from this position, then it goes to this position. Also, if the agent observes that it is at the position of the food, the agent decides to eat the food. This knowledge has been modelled in the following form:

if observation_result(at_position(food, P:POSITION), pos)
and observation_result(at_position(screen, p0), neg)
and observation_result(at_position(self, P:POSITION), neg)
then to_be_performed(goto(P:POSITION))

if observation_result(at_position(self, P:POSITION), pos)
and observation_result(at_position(food, P:POSITION), pos)
then to_be_performed(eat)

4 Different Types of Properties

As an example of our verification method for dynamic systems the behaviour of the system presented in Section 3 is to be verified for situations in which the food is visible at one of the positions; the proof obligation is that the food will disappear after the screen is gone (called screenrise). This is formalised in the following property:

S0. *The food has disappeared some time after screenrise:*

$$\forall \mathfrak{M} \in \text{Traces(S)|EW}\ \forall p \in \{p1, p2\}\ \forall t:$$
$$\text{state}(\mathfrak{M}, t, \text{output(EW)}) \models \neg \text{at\_position(screen, p0)} \wedge \text{at\_position(food, p)} \wedge$$
$$\forall t'' < t: \text{state}(\mathfrak{M}, t'', \text{output(EW)}) \models \text{at\_position(screen, p0)}$$
$$\Rightarrow$$
$$\exists t' > t: \text{state}(\mathfrak{M}, t', \text{output(EW)}) \models \neg \text{at\_position(food, p)}$$

A heuristic to prove properties like this one is to make use of a combination of top-down and bottom-up approaches. Top-down: With this property in mind formulate properties for the sub-components (behavioural properties) and for the co-operation between those sub-components (environment and interaction properties) that might be useful. Formalise them, and then use a bottom-up strategy to see whether or not these properties are enough to prove the main property. In this case, properties of the agent, of the world, of the co-operation between agent and world are in order.

4.1 Agent Properties

Property S0 is phrased in terms of the component EW, that is responsible for the maintenance of the correct state of the world. Therefore, the property describes correct behaviour purely in terms of world situations. But to obtain this behaviour, the agent component has to make the right decisions. Because S0 depends on correct agent behaviour and not just on correct world behaviour, it is a property of the entire system S. The correct reasoning of the agent is described with four properties.

M1. *Effective moving decision making of M: decisions of M to move are made if the circumstances are observed to be appropriate.*

$$\forall \mathfrak{M} \in \text{Traces(M)}\ \forall p \in \{p1, p2\}\ \forall t:$$
$$\text{state}(\mathfrak{M}, t, \text{input(M)}) \models \text{observation\_result(at\_position(screen,p0), neg)} \wedge \text{observation\_result(at\_position(food, p), pos)} \wedge \text{observation\_result(at\_position(self, p), neg)}$$
$$\Rightarrow$$
$$\exists t' > t: \text{state}(\mathfrak{M}, t', \text{output(M)}) \models \text{to\_be\_performed(goto(p))}$$

This property states that the agent decides to perform a goto-action when it observes that the screen is gone and that there is food at a position different from its own.

M2. *Justified moving decisions of M: decisions of M to move are only made if the circumstances are observed to be appropriate.*

$$\forall \mathfrak{M} \in \text{Traces(M)}\ \forall p \in \{p1, p2\}\ \forall t:$$
$$\text{state}(\mathfrak{M}, t, \text{output(M)}) \models \text{to\_be\_performed(goto(p))}$$
$$\Rightarrow$$
$$\exists t' < t: \text{state}(\mathfrak{M}, t', \text{input(M)}) \models \text{observation\_result(at\_position(screen, p0), neg)} \wedge \text{observation\_result(at\_position(food, p), pos)} \wedge \text{observation\_result(at\_position(self, p), neg)}$$

This property states that the agent only decides to move when there are good reasons, these being the absence of the screen and the presence of the food somewhere else.

M3. *Effective eating decision making of M: decisions of M to eat are made if the circumstances are observed to be appropriate.*

$$\forall \mathfrak{M} \in \text{Traces(M)}\ \forall p \in \{p1, p2\}\ \forall t:$$
$$\text{state}(\mathfrak{M}, t, \text{input(M)}) \models \text{observation\_result(at\_position(food, p), pos)} \wedge \text{observation\_result(at\_position(self, p), pos)}$$
$$\Rightarrow$$
$$\exists t' > t: \text{state}(\mathfrak{M}, t', \text{output(M)}) \models \text{to\_be\_performed(eat)}$$

The above property formalises that the agent decides to eat when it observes that it is with the food.

M4. *Justified eating decisions of M: decisions of M to eat are only made if the circumstances are observed to be appropriate.*

$$\forall \mathfrak{M} \in \text{Traces(M)}\ \forall t\ \exists p \in \{p1, p2\}:$$
$$\text{state}(\mathfrak{M}, t, \text{output(M)}) \models \text{to\_be\_performed(eat)}$$
$$\Rightarrow$$
$$\exists t' < t: \text{state}(\mathfrak{M}, t', \text{input(M)}) \models \text{observation\_result(at\_position(food, p), pos)} \wedge \text{observation\_result(at\_position(self, p), pos)}$$

This last agent-property states that the agent only decides to eat when it observes that it is with the food.

The agent component M is a primitive component; these properties can be proven directly from the knowledge base of the agent component (see Section 3.3.2), without using other properties. Such properties are called *basic properties.* As can be seen by comparing the knowledge base of the agent with the properties, the properties formalise the correct functioning of the rules of the knowledge base in the reasoning process. For each rule, there is a property stating that the conclusion will be drawn

if the premises hold, and one property stating that the conclusion will be drawn *only if* the premises hold.

In general, properties formalising the correct functioning of primitive components of arbitrary systems are very similar to the above properties.

4.2 Interaction Properties

The agent and the external world are connected through two information links, that transfer observation results from the world and actions for the world, respectively. Properties of information exchange have a general format, valid for every system containing links. There are two kinds of properties. Interaction effectiveness states that information from the source of a link is correctly delivered at the destination of the link some time later, and interaction groundedness states that when particular information is present in an interface, corresponding information must have been present in the source of one or more links some time earlier. One example of information exchange will be described, namely the information flow to the external world, with its two properties. Other interaction properties are analogous.

I1. *Interaction effectiveness from M to EW:*
$\forall \mathcal{M} \in$ Traces(S) $\forall$ L $\in$ groundliterals(action_info) $\forall$ t :
state($\mathcal{M}$, t , output(M)) $\models$ L $\Rightarrow$
$\exists$ t' > t : state($\mathcal{M}$, t' , input(EW)) $\models$ L

I2. *Interaction groundedness of input information of EW:*
$\forall \mathcal{M} \in$ Traces(S) $\forall$ L $\in$ groundliterals(action_info) $\forall$ t :
state($\mathcal{M}$, t , input(EW)) $\models$ L $\Rightarrow$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(M)) $\models$ L

Interaction properties are also basic properties.

4.3 System Properties

Some properties are needed that seem to concern one component, however, in reality depend not only of the behaviour of that component, but also on the behaviour of the links and components that interact with it (this is called its environment). These properties are properties of the whole system S.

S9. *When the environment of EW is provided with the necessary observation results, it provides a goto-action:*
$\forall \mathcal{M} \in$ Traces(S)|EW $\forall$ p $\in$ {p1, p2} $\forall$ t :
state($\mathcal{M}$, t, output(EW)) $\models$
observation_result(at_position(screen, p0), neg) $\wedge$
observation_result(at_position(food, p), pos) $\wedge$
observation_result(at_position(mouse, p), neg)
$\Rightarrow$
$\exists$ t' > t : state($\mathcal{M}$, t' , input(EW)) $\models$ to_be_performed(goto(p))

S10. *The environment of EW only provides a goto-action when the necessary observation results were present:*
$\forall \mathcal{M} \in$ Traces(S)|EW $\forall$ p $\in$ {p1, p2} $\forall$ t :
state($\mathcal{M}$, t , input(EW)) $\models$ to_be_performed(goto(p))
$\Rightarrow$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(EW)) $\models$
observation_result(at_position(screen, p0), neg) $\wedge$
observation_result(at_position(food, p), pos) $\wedge$
observation_result(at_position(mouse, p), neg)

S11. *When the environment of EW is provided with the necessary observation results, it provides an eat-action:*
$\forall \mathcal{M} \in$ Traces(S)|EW $\forall$ p $\in$ {p1, p2} $\forall$ t :
state($\mathcal{M}$, t, output(EW)) $\models$
observation_result(at_position(food, p), pos) $\wedge$
observation_result(at_position(mouse, p), pos)
$\Rightarrow$
$\exists$ t' > t : state($\mathcal{M}$, t' , input(EW)) $\models$ to_be_performed(eat)

S12. *The environment of EW only provides an eat-action when the necessary observation results were present*:
$\forall \mathcal{M} \in$ Traces(S)|EW $\forall$ t $\exists$ p $\in$ {p1, p2}:
state($\mathcal{M}$, t, input(EW)) $\models$ to_be_performed(eat)
$\Rightarrow$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(EW)) $\models$
observation_result(at_position(food, p), pos) $\wedge$
observation_result(at_position(mouse, p), pos)

4.4 Properties of the External World

Even though the world is dynamic, this doesn't mean that everything is possible. Strange events should not occur: for example, the agent never ever disappears and the agent only moves according to the actions decided upon by the agent. These properties are formalised as follows:

W17. *The mouse is always somewhere:*
$\forall \mathcal{M} \in$ Traces(EW) $\forall$ t $\exists$ p $\in$ POSITION:
state($\mathcal{M}$, t, output(EW)) $\models$ at_position(mouse, p)

W26. *When the mouse changes position, there must have been a goto-action on the input of EW:*
$\forall \mathcal{M} \in$ Traces(EW) $\forall$ p $\in$ POSITION $\forall$ q $\neq$ p $\in$ POSITION $\forall$ t :
state($\mathcal{M}$, t, output(EW)) $\models$ at_position(mouse, q) $\wedge$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(EW)) $\models$ at_position(mouse, p) $\Rightarrow$
$\exists$ t'' < t : state($\mathcal{M}$, t'' , input(EW)) $\models$ to_be_performed(goto(q))

Furthermore, the observation results provided by the external world should correspond to the current world state.

W21. *Observations from EW were facts:*
$\forall \mathcal{M} \in$ Traces(EW) $\forall$ A $\in$ groundatoms(world_info) $\forall$ t :
(state($\mathcal{M}$, t, output(EW)) $\models$ observation_result(A, pos)
$\Rightarrow$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(EW)) $\models$ A) $\wedge$
(state($\mathcal{M}$, t, output(EW)) $\models$ observation_result(A, neg)
$\Rightarrow$
$\exists$ t' < t : state($\mathcal{M}$, t' , output(EW)) $\models \neg$A)

Finally, some assumptions on the behaviour of the external world are necessary. Since, the goal is to prove the system correct for situation 2, the external world is assumed to behave according to situation 2, i.e., food does not move around, food does not disappear unless it is eaten, food remains visible at all times and the mouse is initially at p0. This last property is given:

W18. *Initially, the mouse is at p0:*
$\forall \mathcal{M} \in$ Traces(EW) :
state($\mathcal{M}$, 0, output(EW)) $\models$ at_position(mouse, p0) $\wedge$
$\neg$at_position(mouse, p1) $\wedge$
$\neg$at_position(mouse, p2)

Properties of the external world are not influenced by the agent. In the verification process they are used as assumptions to prove the system properties. If in the system design the external world is the real physical world, the properties have to be obeyed by the world in order to obtain the desired system behaviour. In case of a simulated external world, containing several sub-components, the properties can be proven from properties at a deeper level of abstraction.

4.5 Phase Properties

The properties in Section 4.1 through 4.4 are used to prove the main property S0. However, the proof still needs more structure in order to be easily comprehensible. In this case phasing is used. Property S0 spans all of the execution of the agent system, from time 0 when the screen was still down, until the disappearance of the food. A number of milestones can be distinguished in the behaviour of the system, such as screenrise, the arrival of the agent at the position of the food, and the disappearance of the food. These milestones divide the processing of the system in three phases. Each phase is formalised by one or two properties, that each can be proven from other properties, like those in Sections 4.1 through 4.4. The phase properties together are sufficient to prove the main property S0 of the system.

The first phase of the process is the phase before screenrise. In this phase, the agent is at p0, as S1 states:

S1. *The mouse is at p0 until screenrise:*

$$\forall \mathcal{M} \in \text{Traces}(S)|EW \;\; \forall t: \text{state}(\mathcal{M}, t, \text{output}(EW)) \models \neg \text{at\_position}(\text{screen}, p0) \;\wedge\; \forall t'' < t: \text{state}(\mathcal{M}, t'', \text{output}(EW)) \models \text{at\_position}(\text{screen}, p0) \Rightarrow \forall t' \leq t: \text{state}(\mathcal{M}, t', \text{output}(EW)) \models \text{at\_position}(\text{mouse}, p0)$$

The next phase is the phase between screenrise and the arrival of the agent at the position of the food. This phase is described by properties S2 and S3. Property S2 formalises that the agent actually will arrive at the spot where the food was some time before and S3 states that the food is still there when the agent arrives.

S2. *After screenrise, the mouse will eventually arrive at the position of the food some time:*

$$\forall \mathcal{M} \in \text{Traces}(S)|EW \;\; \forall p \in \{p1, p2\} \; \forall t: \text{state}(\mathcal{M}, t, \text{output}(EW)) \models \neg \text{at\_position}(\text{screen}, p0) \wedge \text{at\_position}(\text{mouse}, p0) \;\wedge\; \text{at\_position}(\text{food}, p) \;\wedge\; \forall t'' < t: \text{state}(\mathcal{M}, t'', \text{output}(EW)) \models \text{at\_position}(\text{screen}, p0) \Rightarrow \exists t' > t: \text{state}(\mathcal{M}, t', \text{output}(EW)) \models \text{at\_position}(\text{mouse}, p)$$

S3. *When at time t' the mouse has just arrived at the location where the food was at some time t before t', then the food has stayed at this location from t to t'.*

$$\forall \mathcal{M} \in \text{Traces}(S)|EW \;\; \forall p \in \{p1, p2\} \; \forall t', \forall t < t': \text{state}(\mathcal{M}, t, \text{output}(EW)) \models \text{at\_position}(\text{food}, p) \;\wedge\; \text{state}(\mathcal{M}, t', \text{output}(EW)) \models \text{at\_position}(\text{mouse}, p) \;\wedge\; \forall t'': [\, t \leq t'' < t' \Rightarrow \text{state}(\mathcal{M}, t'', \text{output}(EW)) \not\models \text{at\_position}(\text{mouse}, p)\,] \Rightarrow \forall t'': [\, t < t'' \leq t' \Rightarrow \text{state}(\mathcal{M}, t'', \text{output}(EW)) \models \text{at\_position}(\text{food}, p)\,]$$

The last phase is the period after the arrival of the agent at the location of the food; S4 states that the food will be eaten at some time in this phase.

S4. *The food disappears some time after both the mouse and the food are at the same position:*

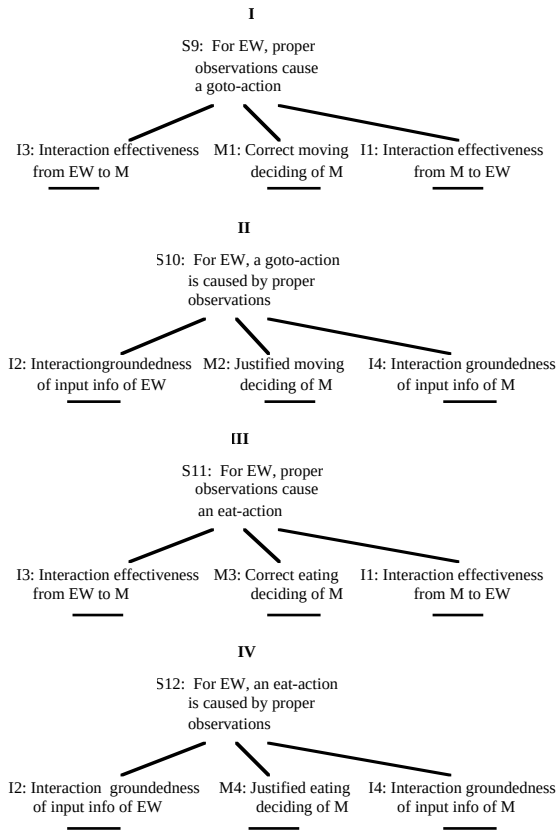
$$\forall \mathcal{M} \in \text{Traces}(S)|EW \;\; \forall p \in \{p1, p2\} \; \forall t \; \forall t' < t: \text{state}(\mathcal{M}, t, \text{output}(EW)) \models \text{at\_position}(\text{mouse}, p) \;\wedge\; \text{at\_position}(\text{food}, p) \;\wedge\; \forall t'': [\, t' \leq t'' < t \Rightarrow \text{state}(\mathcal{M}, t'', \text{output}(EW)) \not\models \text{at\_position}(\text{mouse}, p) \;\wedge\; \text{at\_position}(\text{food}, p)\,] \Rightarrow \exists t'' > t: \text{state}(\mathcal{M}, t'', \text{output}(EW)) \models \neg \text{at\_position}(\text{food}, p)$$

5 Verification of Top Level Properties

Verification of the properties in Section 4 follows the structure of Section 4. The properties of Sections 4.1 and 4.2 are basic properties. Their proof is straightforward from the knowledge bases and information links in the specification of the system and from the semantics of such a specification.

5.1 Verification of the System Properties

The proof trees of the four properties of Section 4.3 are as follows:



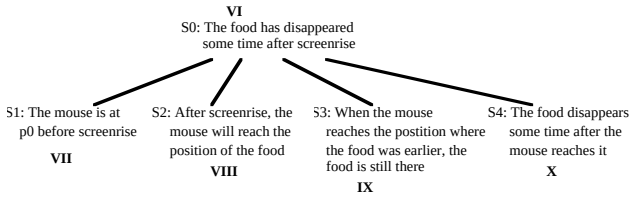
Every node in these trees can be proven from its children in a straightforward manner.
A line under a leaf property means that this property is basic. The roman numbers of the trees will be used to refer to them from other trees.

5.2 Verification of the External World

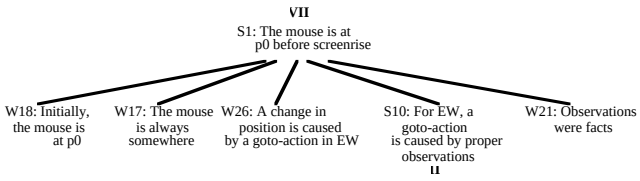
The properties of Section 4.4 are assumptions on the first abstraction level. In case of a simulated world, they can be proven from properties of the next abstraction level. In this way, the verification of the external world can be separated from the verification of the other parts of the system. The compositionality of the system is exploited to reduce the complexity of the verification process.

5.3 Verification of the Main Property and of the Phase Properties

The proof tree of S0 uses the above properties. The proof comes down to chaining the phases. A roman number underneath a leaf in a proof tree means that this leaf property is proven in the tree with that number.



In this section the proof tree of property S1 is given and discussed. The other trees are not given.



The proof is done by induction. The base is W18. For the step, assume the agent was at p0 before and is not anymore now. Then the agent must be somewhere else (W17), so there has been a goto-action (W26) and thus there must have been certain observations earlier (S10). One of these observations is that the screen is gone, and this must have been a fact earlier (W21). Contradiction, for screenrise hasn't happened yet.

6 Basic Assumptions

In this section some of the more fundamental assumptions are discussed. These assumptions are basic in the sense that they have consequences at many places in the verification process. The first set of basic assumptions is about the time structure, discussed in Section 6.1. The second set of basic assumptions is about the co-ordination of the generation and execution of successive actions, discussed in Section 6.2.

6.1 The Time Structure

Different variants can be considered for time structures. The first choice is that we use incomplete (three-valued) states (instead of the more commonly used complete (two-valued) states), to be able to model the absence of information in an agent. The second choice we made is for linear temporal models and not for branching time models. The main reason for this choice is that linear structures are easier to handle, from a technical perspective. However, it is quite well possible that our approach in principle can be worked out for branching time models as well. The third choice we made is for a variant of dense time temporal models (e.g., based on the non-negative rational or real numbers) and not for discrete time models (e.g., based on the natural numbers). Discrete time structures would have the advantage that all processes can be modelled from step to step, using the next operator, and induction over time steps can be performed in proofs. An important disadvantage, however, is that all events have to be projected onto the discrete time scale, which would entail a more synchronous approach than desired, and/or a rather elaborate bookkeeping of all events at the specific discrete time points. An advantage of a dense time approach is that asynchronous events can be modelled in a more natural manner. A disadvantage may be that the possibility to use the next operator is lost, and induction over time steps cannot be used. Because the domain in our case study in principle has asynchronous events, we made the choice for dense time. However, we impose an additional constraint on the temporal models that allow us to do some forms of induction over courses of events. The constraint imposed is meant to exclude pathological models in which on a finite time interval, an infinite number of changes is possible. Roughly spoken, at each time point we require that the state of a component X is always persistent over an interval, in the following sense:

$$
\begin{aligned}
&\forall \mathcal{M} \in \mathrm{Traces}(S)\ \forall\, t, t' > t : \\
&\qquad \mathrm{state}(\mathcal{M}, t, X) \models \phi \ \wedge\ \mathrm{state}(\mathcal{M}, t', X) \not\models \phi \\
&\qquad \Rightarrow \\
&\exists\, t1, t2, t3:\ t \le t1 < t2 < t3 \le t' \quad \wedge \\
&\forall\, t'' : [\, t1 \le t'' < t2 \ \Rightarrow\ \mathrm{state}(\mathcal{M}, t'', X) \models \phi \ \wedge \\
&\qquad t2 \le t'' \le t3 \ \Rightarrow\ \mathrm{state}(\mathcal{M}, t'', X) \not\models \phi \ ]
\end{aligned}
$$

This assumption states that if a change occurs, then a time point exists such that before this time point there is an interval in which the situation before the change persists, and an interval after this time point in which the new situation persists for a while. In (Barringer, Kuiper, and Pnueli, 1985), for the case of temporal logic with two-valued states, the assumption is called finite variability. In our case, due to the incomplete states a variant of this assumption has to be imposed as well:

$$
\begin{aligned}
&\forall \mathcal{M} \in \mathrm{Traces}(S)\ \forall\, t, t' > t : \\
&\qquad \mathrm{state}(\mathcal{M}, t, X) \not\models \phi \ \wedge\ \mathrm{state}(\mathcal{M}, t', X) \models \phi \\
&\qquad \Rightarrow \\
&\exists\, t1, t2, t3:\ t \le t1 < t2 < t3 \le t' \quad \wedge \\
&\forall\, t'' : \quad [\, t1 \le t'' < t2 \ \Rightarrow\ \mathrm{state}(\mathcal{M}, t'', X) \not\models \phi \ \wedge \\
&\qquad t2 \le t'' \le t3 \ \Rightarrow\ \mathrm{state}(\mathcal{M}, t'', X) \models \phi \ ]
\end{aligned}
$$

These basic assumptions look rather natural; they exclude processes in which changes can occur in succession in arbitrarily small intervals. Moreover, continuous changes, such as, for example, in Newtonian mechanics, are excluded as well. For the case study in this paper, where only a small number of discrete events can change the world, the assumptions are reasonable. An advantage is that the assumptions imply that induction can be performed over a course of events: in a countable number of successive events the whole dense time frame is covered.

6.2 Co-ordination of Successive Actions

For a human agent interacting with a system, sometimes it is not clear whether an action that was initiated already was performed (invisible effects or invisible action initiation), or is still under execution; especially visitors of Web-sites

experience this often. As long as an initiated action is still under execution, initiating a next action may disturb the process to such an extent that nothing reasonable results (e.g., due to interference of the two actions). Moreover, as long as the effects of the previous action are not clear to the agent, it is doubtful whether an action that is chosen next is justified. In addition to the basic assumptions discussed in Section 6.1, a basic assumption is made to guarantee that the interaction between agent and external world is transparent. The agent is assumed to refrain from generating a new action as long as it has not noticed that the previous action has led to the expected (observable) changes in the world. Also, until the result of the action is observed by the agent, there is no change in the observations, which means that no events happen in the external world while the action is still being executed. In this system, the assumption pertains to two kinds of actions, namely eat- and goto-actions. These different actions lead to different observable outcomes. So, the assumption is formalised in two parts:

$$
\begin{aligned}
&\forall \mathcal{M} \in \mathrm{Traces(S)}\ \forall\ \mathrm{p, q} \in \mathrm{POSITION}\ \forall\ \mathrm{o} \in \mathrm{OBJECT}\\
&\quad \forall\ \mathrm{s} \in \mathrm{SIGN}\ \forall\ \mathrm{A} \neq \mathrm{goto(p)}\ \forall\ \mathrm{t}\ \forall\ \mathrm{t''} > \mathrm{t}:\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t}, \mathrm{output(M)}) \models \mathrm{to\_be\_performed(goto(p))}\ \wedge\\
&\exists\ \mathrm{t'} < \mathrm{t}\ \forall\ \mathrm{t'''}: [\mathrm{t'} \leq \mathrm{t'''} < \mathrm{t} \quad \Rightarrow\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{output(M)}) \not\models \mathrm{to\_be\_performed(goto(p))}]\ \wedge\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t''}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(self, p), pos)}\ \wedge\\
&\forall\ \mathrm{t'''}: [\mathrm{t} \leq \mathrm{t'''} < \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{input(M)}) \not\models\\
&\qquad\quad \mathrm{observation\_result(at\_position(self, p), pos)}]\\
&\qquad \Rightarrow\\
&\forall\ \mathrm{t'}: [\mathrm{t} < \mathrm{t'} \leq \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{output(M)}) \models \mathrm{to\_be\_performed(goto(p))} \wedge\\
&\qquad\quad \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{output(M)}) \not\models \mathrm{to\_be\_performed(A)}]\ \wedge\\
&[\exists\ \mathrm{t'''}: \mathrm{t} \leq \mathrm{t'''} < \mathrm{t''} \wedge \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(o, q), s)}\ \Rightarrow\\
&\forall\ \mathrm{t'}: (\mathrm{t} \leq \mathrm{t'} < \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(o, q), s))}]
\end{aligned}
$$

$$
\begin{aligned}
&\forall \mathcal{M} \in \mathrm{Traces(S)}\ \forall\ \mathrm{p, q} \in \mathrm{POSITION}\ \forall\ \mathrm{o} \in \mathrm{OBJECT}\\
&\quad \forall\ \mathrm{s} \in \mathrm{SIGN}\ \forall\ \mathrm{A} \neq \mathrm{eat}\ \forall\ \mathrm{t}\ \forall\ \mathrm{t''} > \mathrm{t}:\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t}, \mathrm{output(M)}) \models \mathrm{to\_be\_performed(eat)}\ \wedge\\
&\exists\ \mathrm{t'} < \mathrm{t}\ \forall\ \mathrm{t'''}: [\mathrm{t'} \leq \mathrm{t'''} < \mathrm{t} \quad \Rightarrow\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{output(M)}) \not\models \mathrm{to\_be\_performed(eat)}]\ \wedge\\
&\qquad \mathrm{state}(\mathcal{M}, \mathrm{t''}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(food, p), neg)}\ \wedge\\
&\forall\ \mathrm{t'''}: [\mathrm{t} \leq \mathrm{t'''} < \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{input(M)}) \not\models\\
&\qquad\quad \mathrm{observation\_result(at\_position(food, p), neg)}]\\
&\qquad \Rightarrow\\
&\forall\ \mathrm{t'}: [\mathrm{t} < \mathrm{t'} \leq \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{output(M)}) \models \mathrm{to\_be\_performed(eat)} \wedge\\
&\qquad\quad \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{output(M)}) \not\models \mathrm{to\_be\_performed(A)}]\ \wedge\\
&[\exists\ \mathrm{t'''}: \mathrm{t} \leq \mathrm{t'''} < \mathrm{t''} \wedge \mathrm{state}(\mathcal{M}, \mathrm{t'''}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(o, q), s)}\ \Rightarrow\\
&\forall\ \mathrm{t'}: (\mathrm{t} \leq \mathrm{t'} < \mathrm{t''} \Rightarrow \mathrm{state}(\mathcal{M}, \mathrm{t'}, \mathrm{input(M)}) \models\\
&\qquad\quad \mathrm{observation\_result(at\_position(o, q), s))}]
\end{aligned}
$$

Whether or not this assumption holds, depends on the behaviour of the external world. Only if actions can be executed without being interrupted by events, it can be guaranteed that the system behaviour is as desired.

7 Proof Pragmatics

First, in Section 7.1, some specific issues are identified that result from the dynamics of the world, and make the verification process difficult. Later on, in Section 7.2 heuristics will be discussed that were identified to overcome these difficulties.

7.1 Aspects Entailed by Dynamic Worlds

A system with a dynamic world has *far more possible behaviours* (formalised by traces) than a comparable system with a static world. In the dynamic world, events happen that do not lie inside the range of influence of the system. The system can register these events, but it doesn't know what to expect. When the world would be static, all events happening were caused by and restricted to the system, and this would reduce the complexity of the verification process. Systems interacting with a dynamic world need more flexible knowledge, that can deal with any situation arising in the world. The behaviour of these systems is more diverse, and therefore harder to describe in properties. The system is reacting to situations occurring in the world. So, in properties there will often be conditions describing the situations for which the properties hold. This considerably complicates the formalization of the system behaviour.

As a natural consequence, the proving process also becomes more intricate. Using a property is only possible when its conditions are fulfilled. When proving that something holds at some time t, it is very well possible that the conditions of the property you want to use refer to a time point earlier than t. You are then forced to "prove backwards" that those conditions held earlier, and this can be a lot of work. Especially when the basic assumptions introduced in Section 6.2 about co-ordination of successive actions are not fulfilled, a lot of actions may be initiated in the past and still are due to have their effects. There is a danger that you are regularly doing retrospection in a branching time past, to see whether all possible pasts satisfy some conditions, which may lead to a *combinatorial explosion of histories*. This may seem strange, because usually the future is branching, uncertain and unpredictable as it is. But in a dynamic system, there can be many scenarios that have caused the current situation as well as many scenarios to follow the current point. One of the reasons to introduce the basic assumptions of Section 6.2 was this problem of exploding histories.

7.2 The Proof Heuristics Identified

During the verification process the following heuristics were identified:

- identify a number of *phases* of the process and *milestones* reached after these phases

For the example system phases are identified in Section 4.5.

- *define each of the milestones* by pinpointing the moment of the essential change

This means identification of the statement that holds for the milestone and the moment that it started to hold (i.e., before which in a time interval it does not hold).

- design a form of *causal chains* from one milestone to another

It seems natural to reason in terms of causes and effects. Something (X) happens because of some reason, and the reason induces the effect. When there is nothing to cause X, then X won't happen. To find proofs, frequently, causal chains were traced back. Although in our current way of formalising using temporal models, causal relations cannot be expressed directly, it can be expressed that something (X) always happens before something else (Y), but this may not be the same as saying that X is the cause of Y. Nevertheless, the notion of causality can serve as a heuristic to find proofs that are expressed using temporal models. Whether or not an appropriate notion of causality could be formalised by other means remains open.

- use *forward or backward induction* over chains of events

A number of times induction turned out a valuable proof technique, not induction over time points, but induction over chains of events. Both induction to the future and induction to the past may be useful.

- *abstract from the history* by assuming all effects of past actions have been obtained

This can be done using the basic assumptions of Section 6.2.

- distinguish different *types of properties* according to the *compositional structure*

For example, properties were distinguished in (combined) system properties, (material) world properties, (mental) agent properties and interaction properties.

- distinguish a *separate initial phase*, and let time start after initialisation of the system

Initialisation of the system is a rather deviant type of process. If it is not separated then many properties get additional conditions that make them less transparent.

8 Interaction between Design and Verification Processes

A central aim of verification is to explicit the assumptions under which a system design is appropriate, with respect to the requirements imposed. Another aim is to discover flaws, in order to improve the design: the design process and the verification process may go hand in hand. A more specific interaction between design and verification processes occurs when a design is improved in order to support the verification process; e.g., to make it more transparent and more structured, and increase flexibility in the context of maintenance and reuse. A gain that is achieved on these aspects, however, may entail a loss on other aspects; e.g., the proofs may become larger if they are made more compositional, due to the explicit treatment of the interaction steps between components. A trade-off may occur between aspects related to the current system that is designed and future work involving maintenance and reuse. In previous experiences, in compositional verification of diagnostic reasoning, information gathering agents, and negotiating agents (described, respectively in (Cornelissen, Jonker and Treur, 1997; Jonker and Treur, 1998a; Brazier, Cornelissen, Gustavsson, Jonker, Lindeberg, Polak, and Treur, 1998)) at some points designs have been adapted to support transparency and reusability of verification.

In this respect, in the example discussed in this paper, a point of discussion is whether the interface of EW should represent the world state (choice 1), or should be restricted to actions and observation results only, thereby hiding the world state within EW (choice 2). For the example design as such, the direct information about the world state is not used (only in the form of observation results), so choice 2 would be possible. However, then the verification process would have to take into account the internal structure of EW. Choice 1 has been made, because then it is possible to abstract from the internal structure of EW, and thus it supports reuse of the verification proof, for different fillings of EW, either as the real world or as a simulated world.

9 Discussion

One of the central difficulties in verification of real world systems is the complexity of the verification process, both for humans and automated verifiers. Especially, for verification of dynamic properties of a system, the search space for properties and proofs is often enormous. Some approaches have been put forward to handle this complexity, one of which, compositional verification, structures proofs according to different process abstraction levels; cf. (Cornelissen, Jonker and Treur, 1997; Jonker and Treur, 1998a). Using this method, the compositional system structure plays a heuristic role in finding the properties and proofs; actually the search space is composed of a number of smaller subspaces. This case study of an agent acting in a dynamic world, described in this paper, aims at identification of more detailed proof structures and heuristics for multi-agent systems within a dynamic world. However, at least some of these structures and heuristics are expected to be more generally applicable, which will be a topic of future research.

The properties proven for this system clearly are domain-dependent. This is both a strength and a weakness: though the desired system behaviour can be precisely expressed using these properties, they are not immediately reusable for other systems. But for other agent systems in dynamic domains, many properties (for example, those related to actions and their effects) will be similar. These properties only need to be adjusted to a new domain. Also, there are classes of properties that are essential in verifying every multi-agent system. For example, the basic properties of primitive components and links have a general format. Also, by identifying heuristics to find and formulate the appropriate properties, the system-dependency can be alleviated. As an instance of this, the heuristic of causal chaining can be used to semi-automatically generate properties. Future research will take these issues into further consideration.

The results obtained in this case study can be a starting point for the development of an interactive verifier to verify dynamic properties in which it is possible to explicitly express proof heuristics of the type as identified.

References

Abadi, M. and Lamport, L., (1993). Composing Specifications, ACM Transactions on Programming Languages and Systems, Vol. 15, No. 1, p. 73-132.

Barringer, H., Kuiper, R., and Pnueli, A., (1986). A Really Abstract Concurrent Model and its Temporal Logic. In: Conference Record of the 15th ACM Symposium on Principles of Programming Languages, POPL'86, pp. 173-183.

Brazier, F.M.T., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., and Treur, J., (1998). Compositional Design and Verification of a Multi-Agent System for One-to-Many Negotiation. In: Proc. of the Third International Conference on Multi-Agent Systems, ICMAS-98, IEEE Computer Society Press, pp. 8, to appear.

Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N., and Treur, J., (1997). DESIRE: Modelling Multi-Agent Systems in a Compositional Formal Framework. International Journal of Cooperative Information Systems, vol. 6, Special Issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, (M. Huhns and M. Singh, eds.), pp. 67-94.

Brazier, F.M.T., Treur, J., Wijngaards, N.J.E., and Willems, M., (1996). Temporal semantics of complex reasoning tasks. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, pp. 15/1-15/17. Extended version to appear in Data and Knowledge Engineering (1998).

Cornelissen, F., Jonker, C.M., and Treur, J., (1997). Compositional verification of knowledge-based systems: a case study in diagnostic reasoning. In: E. Plaza, R. Benjamins (eds.), Knowledge Acquisition, Modelling and Management, Proceedings of the 10th EKAW'97, Lecture Notes in AI, vol. 1319, Springer Verlag, pp. 65-80.

Dams, D., Gerth, R., and Kelb, P. (1996). Practical Symbolic Model Checking of the full μ-calculus using Compositional Abstractions. Report, Eindhoven University of Technology, Department of Mathematics and Computer Science.

Fensel, D. (1995). Assumptions and limitatons of a problem solving method: a case study. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'95, Calgary: SRDG Publications, Department of Computer Science, University of Calgary.

Fensel, D., and Benjamins, R. (1996) Assumptions in model-based diagnosis. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, pp. 5/1-5/18.

Fensel, D., Schonegge, A., Groenboom, R., and Wielinga, B. (1996). Specification and verification of knowledge-based systems. In: B.R. Gaines, M.A. Musen (Eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, pp. 4/1-4/20.

Harmelen, F. van, and Teije, A. ten (1997). Validation and verification of diagnostic systems based on their conceptual model. In: J. Vanthienen, F. van Harmelen (Eds.), Proceedings of the Fourth European Symposium on the Validation and Verification of Knowledge-based Systems, EUROVAV'97, Katholieke Univerisiteit Leuven, pp. 117-128.

Hooman, J. (1994). Compositional Verification of a Distributed Real-Time Arbitration Protocol. Real-Time Systems, vol. 6, pp. 173-206.

Jonker, C.M., and Treur, J., (1998a). Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. In: W.P. de Roever, A. Pnueli et al. (eds.), Proceedings of the International Workshop on Compositionality, COMPOS'97, Springer Verlag.

Jonker, C.M., and Treur, J., (1998b). Agent-based simulation of reactive, pro-active and social animal behaviour. In: Proc. of the 11th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE98. Lecture Notes in AI, Springer Verlag, pp. 12.

Vauclair, J., (1996). Animal Cognition. Harvard Univerity Press, Cambridge, Massachusetts, 1996.

Wooldridge, M.J., and Jennings, N.R., (1995). Intelligent Agents: theory and practice. In: Knowledge Engineering Review, 10(2), pp. 115-152.