Requirements Analysis of an Agent's Reasoning Capability

Tibor Bosse[1], Catholijn M. Jonker[2], and Jan Treur[1]

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

Abstract. The aim of requirements analysis for an agent that is to be designed is to identify what characteristic capabilities the agent should have. One of the characteristics usually expected for intelligent agents is the capability of reasoning. This paper shows how a requirements analysis of an agent's reasoning capability can be made. Reasoning processes may involve dynamically introduced or retracted assumptions: 'reasoning by assumption'. It is shown for this type of reasoning how relevant dynamic properties at different levels of aggregation can be identified as requirements that characterize the reasoning capability. A software agent has been built that performs this type of reasoning. The dynamic properties have been expressed using the temporal trace language TTL and can and have been checked automatically for sample traces.

1 Introduction

Requirements analysis addresses the identification and specification of the functionality expected for the system to be developed, abstracting from the manner in which this functionality is realized in a design and implementation of this system; e.g. [1-3]. Recently, requirements analysis for concurrent systems and agent systems has been addressed in particular, for example, in [4, 5]. An agent-oriented view on requirements analysis can benefit from the more specific assumptions on structures and capabilities expected for agents, compared to software components in general. To obtain these benefits, a dedicated agent-oriented requirements analysis process can be performed that takes into account specific agent-related structures and capabilities. For example, for a number of often-occurring agent capabilities, a requirements analysis can be made and documented that is reusable in future agent-oriented software engineering processes. In the process of building agent systems, software engineering principles and techniques, such as scenario and requirements specification, verification and validation, can be supported by the reusable results of such a requirements analysis.

In this paper the results are presented of a requirements analysis of an agent's reasoning capability. Since reasoning can take different forms, intelligent agents may sometimes require nontrivial reasoning capabilities. The more simple forms of reasoning amount to determining the deductive closure of a logical theory (a knowledge base), given a set of input facts. Requirements for such reasoning processes can be specified in the form of a functional relation between input and output states, abstracting from the time it takes to perform the reasoning e.g. [6]. Properties of such a functional relation can be related to properties of a knowledge base used to realize the functionality, which provides possibilities for verification and validation of this knowledge e.g. [7]. However, more sophisticated reasoning capabilities can better be considered as involving a process over time; especially for nontrivial reasoning patterns when the temporal aspects play an important role in their semantics cf. [8, 9]. Therefore, within an agent-oriented software engineering approach to an agent's reasoning capability, requirements specification has to address the dynamic properties of a reasoning process.

This paper shows how such a requirements analysis of the dynamics of an agent's reasoning capability can be made. The approach makes use of a semantic formalization of reasoning processes by *traces* consisting of sequences of reasoning states over time, following the semantic formalization introduced in [8]. Reasoning processes as performed by humans may involve dynamically introduced or retracted assumptions: a pattern used as a case study in this paper, called 'reasoning by assumption'. For requirements acquisition, it is to be shown for this type of reasoning which relevant dynamic properties can be identified that characterize the reasoning pattern.

For the requirements analysis of an agent's capability to perform this type of reasoning, a methodology has been used that comprises the following steps:

- First, a number of scenarios of practical human reasoning processes considered as 'reasoning by assumption' have been analysed and specified to identify requirements that are characteristic for this reasoning pattern. Required dynamic properties at different levels of aggregation (or grain size) have been identified. These characterizing properties have been formalized using the temporal trace language TTL, thus permitting automated support of analysis.
- The specified dynamic properties at the lowest aggregation level are in an executable format; they specify reasoning steps. Using a variant of Executable Temporal Logic [10] and a dedicated software environment for simulation that has been developed [11], these executable properties were used to generate abstract simulation traces. Such traces can be used to provide system designers with a concrete idea of the intended flow of events over time, without having to actually implement the system.
- Next, logical relationships have been determined between dynamic properties at different aggregation levels, in such a way that the dynamic properties at one aggregation level together imply those at a higher aggregation level. Such logical relationships constitute a formal theory of the interdependencies of the different requirements.

- Finally, verification of the requirements has been performed. Supported by software tools, the dynamic properties at different levels have been checked against three different types of traces: (1) human traces, (2) simulation traces, and (3) prototype traces. As for (1), a number of reasoning puzzles were used to acquire scenarios of further practical human reasoning processes that intuitively fit the pattern of reasoning by assumption [12]. The properties were then automatically checked against the formalized scenarios of these human traces. Concerning (2), the (higher-level) dynamic properties were checked against the traces that resulted from the simulation mentioned above, and confirmed, which validates the identified logical relationships between the dynamic properties at different aggregation levels. Finally, as for (3), a design of an existing software agent performing reasoning by assumption [13] was analysed. This agent was designed using the component-based design method DESIRE [14]. Using the DESIRE execution environment, for this agent a number of reasoning traces were generated. For these traces, all identified dynamic properties (also the executable ones) were also checked and found to be confirmed.

In Section 2 the dynamic perspective on reasoning is discussed in further detail, focussed on the pattern ‘reasoning by assumption’. Section 3 addresses some details of the language used. Section 4 presents a number of requirements in the form of dynamic properties identified for patterns of reasoning by assumption. Section 5 discusses logical relationships between dynamic properties at different aggregation levels. In Section 6, it is discussed in which respects verification has been performed. In Section 7, the contribution of the research presented in the paper is briefly discussed.

2 The Dynamics of Reasoning

Analysis of reasoning processes has been addressed from different areas and angles, for example, Cognitive Science, Philosophy and Logic, and AI. For reasoning processes in natural contexts, which are usually not restricted to simple deduction, dynamic aspects play an important role and have to be taken into account, such as dynamic focussing by posing goals for the reasoning, or making (additional) assumptions during the reasoning, thus using a dynamic set of premises within the reasoning process. Also, dynamically initiated additional observations or tests to verify assumptions may be part of a reasoning process. Decisions made during the process, for example, on which reasoning goal to pursue, or which assumptions to make, are an inherent part of such a reasoning process. Such reasoning processes or their outcomes cannot be understood, justified or explained without taking into account these dynamic aspects.

The approach to the semantic formalization of the dynamics of reasoning exploited here is based on the concepts pf reasoning state, transitions and traces.

Reasoning state. A reasoning state formalizes an intermediate state of a reasoning process. The set of all reasoning states is denoted by RS.

Transition of reasoning states. A transition of reasoning states or reasoning step is an element < S, S' > of RS x RS. A *reasoning transition relation* is a set of these transitions, or a relation on RS x RS that can be used to specify the allowed transitions.

Reasoning trace. Reasoning dynamics or reasoning behaviour is the result of successive transitions from one reasoning state to another. A time-indexed sequence of reasoning states is constructed over a given timeframe (e.g. the natural numbers). Reasoning traces are sequences of reasoning states such that each pair of successive reasoning states in such a trace forms an allowed transition. A trace formalizes one specific line of reasoning. A set of reasoning traces is a declarative description of the semantics of the behaviour of a reasoning process; each reasoning trace can be seen as one of the alternatives for the behaviour. In Section 3, a language is introduced in which it is possible to express dynamic properties of reasoning traces.

The specific reasoning pattern used in this paper to illustrate the approach is 'reasoning by assumption'. This type of reasoning often occurs in practical reasoning; for example, in everyday reasoning, diagnostic reasoning based on causal knowledge, and reasoning based on natural deduction. An example of everyday reasoning by assumption is 'Suppose I do not take my umbrella with me. Then, if it starts raining at 5 pm, I will get wet, which I don't want. Therefore I'd better take my umbrella with me'. An example of diagnostic reasoning by assumption in the context of a car that won't start is: 'Suppose the battery is empty, then the lights won't work. But if I try, the lights turn out to work. Therefore the battery is not empty.' Examples of reasoning by assumption in natural deduction are as follows. Method of indirect proof: 'If I assume A, then I can derive a contradiction. Therefore I can derive not A.'. Reasoning by cases: 'If I assume A, I can derive C. If I assume B, I can also derive C. Therefore I can derive C from A or B.'.

Notice that in all of these examples, first a reasoning state is entered in which some fact is *assumed*. Next (possibly after some intermediate steps), a reasoning state is entered where *consequences* of this assumption have been *predicted*. Finally, a reasoning state is entered in which an *evaluation* has taken place; possibly in the next state the assumption is retracted, and conclusions of the whole process are added. In Section 3 and 4, this pattern is to be characterized by requirements.

3 Dynamic Properties

To specify properties on the dynamics of reasoning, the temporal trace language TTL used in [5] is adopted. This is a language in the family of languages to which situation calculus [15], event calculus [16] and fluent calculus [17] also belong.

Ontology. An ontology is a specification (in order-sorted logic) of a vocabulary. For the example reasoning pattern 'reasoning by assumption' the state ontology includes binary relations such as assumed, rejected, on sorts INFO_ELEMENT x SIGN and the relation prediction_for on INFO_ELEMENT x SIGN x INFO_ELEMENT x SIGN. Table 1 contains all the relations that will be used in this paper, as well as their explanation. The sort INFO_ELEMENT includes specific domain statements such as car_starts, lights_burn, battery_empty, sparking_plugs_problem. The sort SIGN consists of the elements pos and neg.

Table 1. State ontology for the pattern 'reasoning by assumption'

Internal concepts:	
initial_assumption(A:INFO_ELEMENT, S:SIGN)	The agent beliefs that it is most plausible to assume (A,S). Therefore, this is the agent's default assumption. For example, if it is most likely that the battery is empty, this is indicated by initial_assumption(battery_empty, pos).
assumed(A:INFO_ELEMENT, S:SIGN)	The agent currently assumes (A,S).
prediction_for(A:INFO_ELEMENT, S1:SIGN, B:INFO_ELEMENT, S2:SIGN)	The agent predicts that if (B,S2) is true, then (A,S1) should also be true.
rejected(A:INFO_ELEMENT, S:SIGN)	The agent has rejected the assumption (A,S).
alternative_for(A:INFO_ELEMENT, S1:SIGN, B:INFO_ELEMENT, S2:SIGN)	The agent beliefs that (A,S1) is a good alternative assumption in case (B,S2) is rejected.
Input and output concepts:	
To_be_observed(A:INFO_ELEMENT)	The agent starts observing whether A is true.
observation_result(A:INFO_ELEMENT, S:SIGN)	If S is pos, then the agent observes that A is true. If S is neg, then the agent observes that A is false.
External concepts:	
domain_implies(A:INFO_ELEMENT, S1:SIGN, B:INFO_ELEMENT, S2:SIGN)	Under normal circumstances, (A,S1) leads to (B,S2). For example, an empty battery usually implies that the lights do not work.
holds_in_world(A:INFO_ELEMENT, S:SIGN)	If S is pos, then A is true in the world. If S is neg, then A is false.

Reasoning state. A (reasoning) 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). A part of the description of an example reasoning state S is:

assumed(battery_empty, pos)	: true
prediction_for(lights_ burn, neg, battery_empty, pos)	: true
observation_result(lights_burn, pos)	: true
rejected(battery_empty, pos)	: false

The standard satisfaction relation |== between states and state properties is used: S |== p means that state property p holds in state S. For example, in the reasoning state S above it holds that S |== assumed(battery_empty, pos).

Reasoning trace. To describe dynamics, explicit reference is made to time in a formal manner. A fixed timeframe T is assumed that is linearly ordered. Depending on the application, for example, 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). A trace γ over an ontology Ont and timeframe T is a mapping $\gamma : T \rightarrow$ STATES(Ont), i.e. a sequence of reasoning states γ_t ($t \in T$) in STATES(Ont). Please note that in each trace, the current world state is included.

Expressing dynamic properties. States of a trace can be related to state properties via the formally defined satisfaction relation |== between states and formulae. Comparable to the approach in situation calculus, the sorted predicate logic temporal trace language TTL is built on atoms such as state(γ , t) |== p, referring to traces, time and state properties. This expression denotes that state property p is true in the state of trace γ at time point t. Here |== is a predicate symbol in the language (in infix notation), comparable to the Holds-predicate in situation calculus. Temporal formulae are built using the usual logical connectives and quantification (for example, over traces, time and state properties). The set TFOR(Ont) is the set of all temporal formulae that only make use of ontology Ont. We allow additional language elements as abbreviations of formulae of the temporal trace language. The fact that this language is formal allows for precise specification of dynamic properties. Moreover, editors can and actually have been developed to support specification of properties. Specified properties can be checked automatically against example traces to find out whether they hold.

Simulation. A simpler temporal language has been used to specify simulation models. This temporal language, the LEADSTO language [11], offers the possibility of modelling direct temporal dependencies between two state properties in successive states. This executable format is defined as follows. Let α and β be state properties of the form 'conjunction of atoms or negations of atoms', and e, f, g, h non-negative real numbers. In the LEADSTO language $\alpha \twoheadrightarrow_{e, f, g, h} \beta$, means:

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

For a precise definition of the LEADSTO format, see [11]. A specification of dynamic properties in LEADSTO format has as advantages that it is executable and that it can easily be depicted graphically.

4 Dynamic Properties as Characterizing Requirements

Careful analysis of the informal reasoning patterns discussed in Section 2 led to the identification of dynamic properties that can serve as requirements for the capability of reasoning by assumption. In this section, a number of the most relevant of those properties are presented in both an informal and formal way. The dynamic properties identified are at three different levels of aggregation:

- **Local properties** address the step-by-step reasoning process of the agent. They represent specific transitions between states of the process: *reasoning steps*. These properties are represented in *executable* format, which means that they can be used to generate simulation traces.
- **Global properties** address the *overall* reasoning behaviour of the agent, not the step-by-step reasoning process of the agent. Some examples of global properties are presented, regarding matters as termination, correct reasoning and result production.
- **Intermediate properties** are properties at an intermediate level of aggregation, which are used for the analysis of global properties (see also Section 5).

A number of local properties are given in Section 4.1. It will be shown how they can be used in order to generate simulation traces. Next, Section 4.2 provides some global properties and Section 4.3 some intermediate properties.

4.1 Local Dynamic Properties

At the lowest level of aggregation, a number of dynamic properties have been identified for the process of reasoning by assumption. These local properties are given below (both in an informal and in formal LEADSTO notation):

LP1 (Assumption Initialization)
The first local property LP1 expresses that a first assumption is made. Here, note that initial_assumption is an agent-specific predicate, which can be varied for different cases. Formalization:
initial_assumption(A, S) $\twoheadrightarrow_{0,0,1,1}$ assumed(A, S)

LP2 (Prediction Effectiveness)
Local property LP2 expresses that, for each assumption that is made, all relevant predictions are generated.
Formalization:
assumed(A, S1) and domain_implies(A, S1, P, S2) $\twoheadrightarrow_{0,0,1,1}$ prediction_for(P, S2, A, S1)

LP3 (Observation Initiation Effectiveness)
Local property LP3 expresses that all predictions made will be observed. Formalization:
prediction_for(P, S1, A, S2) $\twoheadrightarrow_{0,0,1,1}$ to_be_observed(P)

LP4 (Observation Result Effectiveness)
Local property LP4 expresses that, if an observation is made, the appropriate observation result will be received. Formalization:
to_be_observed(P) and holds_in_world(P, S) $\twoheadrightarrow_{0,0,1,1}$ observation_result(P, S)

LP5 (Evaluation Effectiveness)
Local property LP5 expresses that, if an assumption was made and a related prediction is falsified by an observational result, then the assumption is rejected. Formalization:
assumed(A, S1) and prediction_for(P, S2, A, S1) and observation_result(P, S3) and S2≠S3 $\twoheadrightarrow_{0,0,1,1}$ rejected(A, S1)

LP6 (Assumption Effectiveness)
Local property LP6 expresses that, if an assumption is rejected and there is still an alternative assumption available, this will be assumed. Formalization:
assumed(A, S1) and rejected(A, S1) and alternative_for(B, S2, A, S1) and not rejected(B, S2) $\twoheadrightarrow_{0,0,1,1}$ assumed(B, S2)

LP7 (Assumption Persistence)
Local property LP7 expresses that assumptions persist as long as they are not rejected. Formalization:
assumed(A, S) and not rejected(A, S) $\twoheadrightarrow_{0,0,1,1}$ assumed(A, S)

LP8 (Rejection Persistence)
Local property LP8 expresses that rejections persist. Formalization:
rejected(A, S) $\twoheadrightarrow_{0,0,1,1}$ rejected(A, S)

LP9 (Observation Result Persistence)
Local property LP9 expresses that observation results persist. Formalization:
observation_result(P, S) $\twoheadrightarrow_{0,0,1,1}$ observation_result(P, S)

Using the software environment that is described in [11], these local dynamic properties can be used to generate simulation traces. Using such traces, the requirements engineers and system designers obtain a concrete idea of the intended flow of events over time. A number of simulation traces have been created for several domains. An example simulation trace in the domain of car diagnosis is depicted in Fig. 1. Here, time is on the horizontal axis, and the state properties on the vertical axis. A dark box on top of the line indicates that the state property is true during that time period, and a lighter box below the line indicates that the state property is false. This figure shows the characteristic cyclic process of reasoning by assumption: making assumptions, predictions and observations for assumptions, then rejecting assumptions and creating new assumptions. As can be seen in Fig. 1, it is first observed that the car does not start. On the basis of this observation, an initial assumption is made that this is due to an empty battery. However, if this assumption turns out to be impossible (because the lights are working), this assumption is rejected. Instead, a second assumption is made (there is a sparking plugs problem), which turns out to be correct.

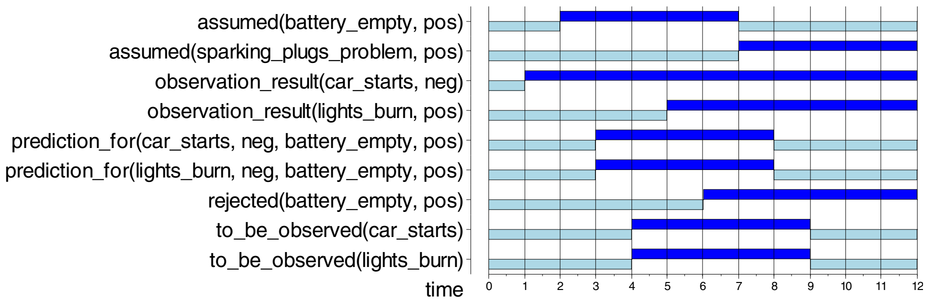

Fig. 1. Example simulation trace

4.2 Global Dynamic Properties

At the highest level of aggregation, a number of dynamic properties have been identified for the overall reasoning process. These global properties are given below (both in an informal and in formal TTL notation). Note that, in each formula, γ stands for a trace.

GP1 (Reasoning Termination)
Eventually there is a time point at which the reasoning terminates.
$\exists t{:}T \;\; \text{termination}(\gamma, t)$

Here termination(γ, t) is defined as follows:
$\forall t'{:}T \;\; t' \geq t \Rightarrow \text{state}(\gamma, t) = \text{state}(\gamma, t')$.

GP2 (Correctness of Rejection)
Everything that has been rejected does not hold in the world situation.
$\forall t{:}T \; \forall A{:}\text{INFO\_ELEMENT} \; \forall S{:}\text{SIGN}$
  $\text{state}(\gamma,t) \;|\!==\; \text{rejected}(A,S) \;\; \Rightarrow$
    $\text{state}(\gamma,t) \;|\!=\!/\!=\; \text{holds\_in\_world}(A,S)$

GP3 (At least one not Rejected Assumption)
If the reasoning has terminated, then there is at least one assumption that has been evaluated and not rejected.

```
∀t:T  termination(γ, t)
  ⇒ [ ∃ A: INFO_ELEMENT, ∃ S: SIGN
    state(γ, t) |== assumed(A, S)  ∧  state(γ, t) |=/= rejected(A, S) ]
```

In addition, some *assumptions on the domain* can be specified:

WP1 (Static World)
If something holds in the world, it will hold for all time.

```
∀t:T ∀A:INFO_ELEMENT ∀S:SIGN
  state(γ,t) |== holds_in_world(A,S)  ⇒
    [ ∀t':T ≥ t:T  state(γ,t') |== holds_in_world(A,S) ]
∀t:T ∀A:INFO_ELEMENT ∀S:SIGN
  state(γ,t) |=/= holds_in_world(A,S)  ⇒
    [ ∀t':T ≥ t:T  state(γ,t') |=/= holds_in_world(A,S) ]
```

WP2 (World Consistency)
If something holds in the world, then its complement does not hold.

```
∀t:T ∀A:INFO_ELEMENT ∀S1,S2:SIGN
  state(γ,t) |== holds_in_world(A,S1) ∧ S1 ≠ S2  ⇒
    state(γ,t) |=/= holds_in_world(A,S2)
```

DK1 (Domain Knowledge Correctness)
The domain-specific knowledge is correct in the world.

```
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2:SIGN
 state(γ,t) |== holds_in_world(A,S1) ∧ domain_implies(A,S1,B,S2)
   ⇒  state(γ,t) |== holds_in_world(B,S2)]
```

4.3 Intermediate Dynamic Properties

In the sections above, on the one hand, global properties for a reasoning process as a whole have been identified. On the other hand, at the lowest level of aggregation, local (executable) properties representing separate reasoning steps have been identified. It may be expected that any trace that satisfies the local properties automatically will satisfy the global properties (semantic entailment). As a form of verification, it can be proven that the local properties indeed imply the global properties. To construct a transparent proof, a number of *intermediate properties* have been identified. Examples of intermediate properties are property IP1 to IP7 shown below (both in an informal and in formal TTL notation).

IP1 (Proper Rejection Grounding)
If an assumption is rejected, then earlier on there was a prediction for it that did not match the corresponding observation result.

```
∀t:T ∀A:INFO_ELEMENT ∀S1:SIGN
  state(γ,t) |== rejected(A,S1) ⇒
    [∃t':T ≤ t:T ∃B:INFO_ELEMENT ∃S2,S3:SIGN
      state(γ,t') |== prediction_for(B,S2,A,S1)  ∧
      state(γ,t') |== observation_result(B,S3) ∧ S2 ≠ S3]
```

IP2 (Prediction-Observation Discrepancy implies Assumption Incorrectness)
If a prediction does not match the corresponding observation result, then the associated assumption does not hold in the world.

```
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2,S3:SIGN
  state(γ,t) |== prediction_for(B,S2,A,S1) ∧
  state(γ,t) |== observation_result(B,S3) ∧ S2 ≠ S3 ⇒
    state(γ,t) |=/= holds_in_world(A,S1)
```

IP3 (Observation Result Correctness)
Observation results obtained from the world indeed hold in the world.

```
∀t:T ∀A:INFO_ELEMENT ∀S:SIGN
  state(γ,t) |== observation_result(A,S)  ⇒
    state(γ,t) |== holds_in_world(A,S)
```

IP4 (Incorrect Prediction implies Incorrect Assumption 1)
If a prediction does not match the facts from the world, then the associated assumption does not hold either.

```
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2,S3:SIGN
  state(γ,t) |== prediction_for(B,S2,A,S1) ∧
  state(γ,t) |== holds_in_world(B,S3) ∧ S2 ≠ S3 ⇒
    state(γ,t) |=/= holds_in_world(A,S1)
```

IP5 (Observation Result Grounding)
If an observation has been obtained, then earlier on the corresponding fact held in the world.

```
∀t:T ∀A:INFO_ELEMENT ∀S:SIGN
  state(γ,t) |== observation_result(A,S)  ⇒
    [ ∃t':T ≤ t:T   state(γ,t') |== holds_in_world(A,S) ]
```

IP6 (Incorrect Prediction implies Incorrect Assumption 2)
If a prediction does not hold in the world, then the associated assumption does not hold either.

```
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2:SIGN
  state(γ,t) |== prediction_for(B,S2,A,S1) ∧
  state(γ,t) |=/= holds_in_world(B,S2) ⇒
    state(γ,t) |=/= holds_in_world(A,S1)
```

IP7 (Prediction Correctness)
If a prediction is made for an assumption that holds in the world, then the prediction also holds.

```
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2:SIGN
  state(γ,t) |== prediction_for(B,S2,A,S1) ∧
  state(γ,t) |== holds_in_world(A,S1) ⇒
    state(γ,t) |== holds_in_world(B,S2)
```

5 Relationships Between Dynamic Properties

A number of logical relationships have been the identified between properties at different aggregation levels. An overview of all identified logical relationships relevant for GP2 is depicted as an AND-tree in Fig. 2. Here the grey ovals indicate that the so-called *grounding* variant of the property is used. Grounding variants make a specification of local properties more complete by stating that there is no other means to produce certain behaviour. For example, the grounding variant of LP2 can be specified as follows (in TTL notation):

LP2G Prediction effectiveness groundedness
Each prediction is related (via domain knowledge) to an earlier made assumption.
∀t:T ∀A,B:INFO_ELEMENT ∀S1,S2:SIGN
 state(γ,t) |== prediction_for(B,S2,A,S1) ⇒
 [∃t':T ≤ t:T state(γ,t') |== assumed(A,S1) ∧
 domain_implies(A,S1,B,S2)]

This property expresses that predictions made always have to be preceded by a state in which the assumption was made, and the domain knowledge implies the prediction.

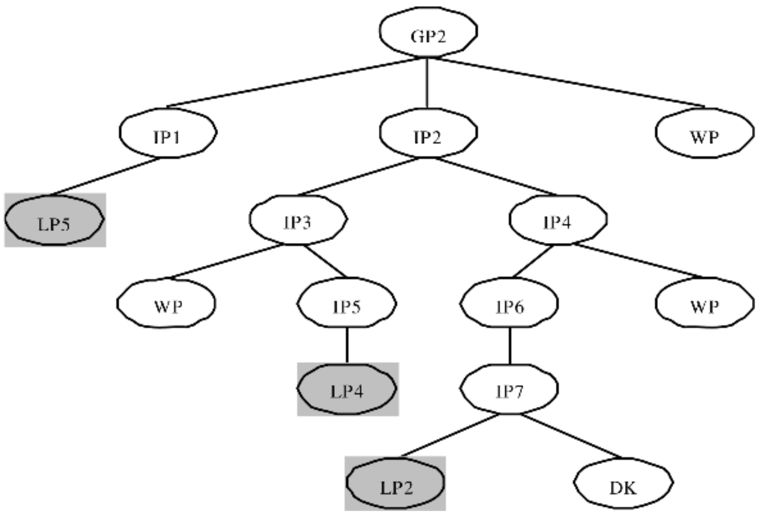

Fig. 2. AND-tree of dynamic properties

The relationships depicted in Fig. 2 should be interpreted as semantic entailment relationships. For example, the relationship at the highest level expresses that the implication IP1 & IP2 & WP1 => GP2 holds. A sketch of the proof for this implication is as follows.

Suppose IP1 holds. This means that, if an assumption is rejected at time t, then at a certain time point in the past (say t') there was a prediction for it that did not match the corresponding observation result. According to IP2, at the very same time point (t') the assumption for which the prediction was made did not hold in the world. Since the world is static (WP1), this assumption still does not hold at time point t. We may thus conclude that, if something is rejected at a certain time point, it does not hold in the world.

Logical relationships between dynamic properties can be very useful in the analysis of empirical reasoning processes. For example, if a given person makes an incorrect rejection (i.e. property GP2 is not satisfied by the reasoning trace), then by a refutation process it can be concluded that either property IP1, property IP2, or property WP1 fails (or a combination of them). If, after checking these properties, it turns out that IP1 does not hold, then this must be the case because LP5G does not hold. Thus, by this example refutation analysis, it can be concluded that the cause of the unsatisfactory reasoning process can be found in LP5G. For more information about the analysis of human reasoning processes, see [12].

6 Verification

In addition to the simulation software described in Section 4, a special tool has been developed that takes a formally specified property and a set of traces as input, and verifies whether the property holds for the traces.

Using this checker tool, dynamic properties (of all levels) can be checked automatically against traces, irrespective of who/what produced those traces: humans, simulators or an implemented (prototype) system. A large number of such checks have indeed been performed for several case studies in reasoning by assumption. Table 2 presents an overview of all combinations of checks and their results. A '+' indicates that all properties were satisfied for the traces, a '+/-' indicates that some of the properties were satisfied.

Table 2. Overview of the different verification results

	Human Traces (Taken from [12])	**Simulation Traces** (This paper)	**Prototype Traces** (Taken from [13])
Local Properties	+/-	+	+
Intermediate Properties	+/-	+	+
Global Properties	+/-	+	+

As can be seen in Table 2, three types of traces were considered. First, the dynamic properties have been checked against human traces in reasoning experiments. It turned out that some of the properties were satisfied by all human traces, whereas some other properties sometimes failed. This implies that some properties are indeed characteristic for the pattern 'reasoning by assumption', whereas some other properties can be used to discriminate between different approaches to the reasoning. For example, human reasoners sometimes skip a step; therefore LP2 does not always hold. More details of these checks can be found in [12].

Second, the dynamic properties have been checked against simulation traces such as the one presented in Section 4.1 of this paper. As shown in Table 2, all properties eventually were satisfied for all traces. Note that this was initially not the case: in some cases, small errors were made during the formalization of the properties. Checking the properties against simulation traces turned out to be useful to localize such errors and thereby debug the formal dynamic properties.

Finally, all dynamic properties have been verified against traces generated by a prototype of a software agent performing reasoning by assumption [13]. This agent was designed on the basis of the component-based design method DESIRE, cf. [14]. Also for these traces eventually all dynamic properties turned out to hold.

To conclude, all automated checks described above have played an important role in the requirements analysis of reasoning capabilities of software agents, since they permitted the results of the requirements elicitation and specification phase to be formally verified and improved.

Note that, although the dynamic properties shown in the previous sections are mainly aimed at functional requirements, in principle the approach based on TTL allows one to verify non-functional requirements as well. Examples of non-functional requirements are efficiency, reliability and portability of the system [18]. Despite the fact that these types of requirements are generally difficult to formalize, some initial steps have been made towards their formalization in TTL [19]. In that paper, it is suggested that the efficiency of a system can be measured, for example, by counting the amount of components that need to be activated in order to be successful. This property is formalized in TTL as follows:

```
efficiency(γ:TRACE) ≡
  successfulness(γ) ∧
  ∃i :INTEGER component_activations(γ, i) ∧ i = shortest_path
```

Here, it is assumed that the length of the shortest path is known for the particular example being checked. To enable a definition of the number of activations of a component, first the activation of one component is defined, including its interval:

```
has_activation_interval(γ:TRACE, c:COMPONENT, tb:TIME, te:TIME) ≡
  tb < te ∧ state(γ,te) |≠ activated(c) ∧
  [∀t tb≤t<te ⇒ state(γ,t) |= activated(c)] ∧
  ∃t1<tb  [∀t2 t1≤t2<tb ⇒ state(γ,t2) |≠ activated(c)]
```

An example of a definition for a trace with one component activation is shown below.

```
component_activations(γ:TRACE, 1) ≡
  ∃c:COMPONENT, tb:TIME, te:TIME
  has_activation_interval(γ, c:COMPONENT, tb:TIME, te:TIME) ∧
  [∀c2:COMPONENT, tb2:TIME, te2:TIME
  [has_activation_interval(γ, c2:COMPONENT, tb2:TIME, te2:TIME) ⇒
    c = c2 ∧ tb = tb2 ∧ te = te2]]
```

Another way to describe efficiency is by considering the amount of computation time the approach needs to generate a solution.

7 Discussion

In the literature, software engineering aspects of reasoning capabilities of intelligent agents have not been addressed well. Some literature is available on formal semantics of the dynamics of non-monotonic reasoning processes; for an overview see [9]. However, these approaches focus on formal foundation and are far from the more practical software engineering aspects of actual agent system development.

In this paper, it is shown how, during an agent development process, a requirements analysis can be incorporated. The desired functionality of the agent's reasoning capabilities can be identified (for example, in cooperation with stakeholders), using temporal specifications of scenarios and requirements specified in the form of (required) traces and dynamic properties. This paper shows, for the example reasoning pattern 'reasoning by assumption', how relevant dynamic properties can be identified as requirements for the agent's reasoning behaviour, expressed in a temporal language, and verified and validated. Thus a set of requirements is obtained that is reusable in other agent development processes. The main reason for the reusability of these requirements is the fact that, within the presented dynamic properties, generic and domain-specific concepts can be treated separately (compositionality of knowledge). For example, in global property DK1, the domain-specific sort INFO_ELEMENT and the relation domain_implies can be filled in for any specific case. This allows the software engineer to reuse the presented requirements in any given domain, as long as it involves reasoning by assumption. In fact, the approach has already been applied to several different examples: whereas the main reasoning problem addressed in the current paper is about 'car diagnosis', other reasoning problems have been addressed in the past, among which the 'wise person's puzzle' [13] and the game of 'Mastermind' [12].

The language TTL used here allows for precise specification of the requirements for an agent's reasoning behaviour, covering both qualitative and quantitative aspects of states and their temporal relations. Moreover, software tools have been developed to (1) support specification of (executable) dynamic properties, and (2) automatically check specified dynamic properties against example traces to find out whether the properties hold for the traces. This provides a useful supporting software environment to evaluate reasoning scenarios both in terms of simulated and prototype traces (in the context of prototyping) and empirical traces (in the context of requirements elicitation and validation in co-operation with stakeholders). In the paper, it is shown how this software environment can be used to automatically check the dynamic properties during a requirements analysis process. Note that it is not claimed that TTL is the only language appropriate for this. For example, most of the properties encountered could as well have been expressed in a variant of linear time temporal logic. The language is only used as a vehicle; the contribution of the paper is in the method of application of requirements analysis to an agent's reasoning capability, and the reusable results obtained by that method.

For an elaborate description about the role that the current approach may take in requirements engineering, the reader is referred to [20]. In that paper, it is shown in detail how dynamic properties can be used to specify (both functional and non-functional) requirements of agent systems. Moreover, it is shown how these requirements may be refined and fulfilled according to the Generic Design Model (GDM) by Brazier *et al.* [21]. However, GDM is just one possible approach for Agent-Oriented Software Engineering. Recently, several other architectures have been proposed, for example, Tropos [22], KAOS [23] or GBRAM [24]. In future work, the possibilities may be explored to incorporate the approach based on dynamic properties presented here within such architectures. These possibilities are promising, especially for architectures that provide a specific language for formalization of requirements (KAOS for example uses a real-time temporal logic to specify requirements in terms of goals, constraints and objects).

References

1. Dardenne, A., Lamsweerde, A. van, and Fickas, S.: Goal-directed Requirements Acquisition. *Science in Computer Programming*, vol. 20 (1993) 3-50
2. Kontonya, G., and Sommerville, I.: *Requirements Engineering: Processes and Techniques*. John Wiley and Sons, New York (1998)

3. Sommerville, I., and Sawyer P.: *Requirements Engineering: a good practice guide*. John Wiley & Sons, Chicester, England (1997)
4. Dubois, E., Du Bois, P., and Zeippen, J.M.: A Formal Requirements Engineering Method for Real-Time, Concurrent, and Distributed Systems. In: *Proceedings of the Real-Time Systems Conference, RTS'95* (1995)
5. Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E.: 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*. LNAI, vol. 1647, Springer Verlag (1999) 8-27
6. Treur, J.: Semantic Formalisation of Interactive Reasoning Functionality. *International Journal of Intelligent Systems,* vol. 17 (2002) 645-686
7. Leemans, N.E.M., Treur, J., and Willems, M.: A Semantical Perspective on Verification of Knowledge. *Data and Knowledge Engineering,* vol. 40 (2002) 33-70
8. Engelfriet, J., and Treur, J.: Temporal Theories of Reasoning. *Journal of Applied Non-Classical Logics,* 5 (1995) 239-261
9. Meyer, J.-J., Ch., and Treur, J. (eds.): *Dynamics and Management of Reasoning Processes*. Series in Defeasible Reasoning and Uncertainty Management Systems (D. Gabbay, Ph. Smets, series eds.), Kluwer Acad. Publishers (2001)
10. Barringer, H., Fisher, M., Gabbay, D., Owens, R., and Reynolds, M.: *The Imperative Future: Principles of Executable Temporal Logic*, Research Studies Press Ltd. and John Wiley & Sons (1996)
11. Bosse, T., Jonker, C.M., Meij, L. van der, and Treur, J.: LEADSTO: a Language and Environment for Analysis of Dynamics by SimulaTiOn. In: Eymann, T. *et al.* (eds.), *Proceedings of the 3rd German Conference on Multi-Agent System Technologies, MATES'05*. Lecture Notes in AI, vol. 3550, Springer Verlag (2005) 165-178
12. Bosse, T., Jonker, C.M., and Treur, J.: Formalization and Analysis of Reasoning by Assumption. *Cognitive Science Journal*, vol. 30, issue 1 (2006) 147-180
13. Jonker, C.M., and Treur, J.: Modelling the Dynamics of Reasoning Processes: Reasoning by Assumption. *Cognitive Systems Research Journal,* vol. 4 (2003) 119-136
14. Brazier, F.M.T., Jonker, C.M., and Treur, J.: Principles of Component-Based Design of Intelligent Agents. *Data and Knowledge Engineering*, vol. 41 (2002) 1-28
15. Reiter, R.: Knowledge in Action: Logical Foundations for Specifying and Implementing Dynamical Systems. MIT Press (2001)
16. Kowalski, R., and Sergot, M.: A logic-based calculus of events. *New Generation Computing*, vol. 4 (1986) 67-95
17. Hölldobler, S., and Thielscher, M.: A new deductive approach to planning. *New Generation Computing*, vol. 8 (1990) 225-244
18. Davis, A.M.: *Software Requirements: Objects, Functions, and States*. Prentice Hall (1993)
19. Bosse, T., Hoogendoorn, M., and Treur, J.: Automated Evaluation of Coordination Approaches. In: *Proceedings of the Eighth International Conference on Coordination Models and Languages, Coordination'06*. Lecture Notes in Computer Science, vol. 4038. Springer Verlag (2006) 44-62
20. Bosse, T., Jonker, C.M., and Treur, J.: Analysis of Design Process Dynamics. In: R. Lopez de Mantaras, L. Saitta (eds.), *Proceedings of the 16th European Conference on Artificial Intelligence, ECAI'04*, IOS Press (2004) 293-297
21. Brazier F.M.T., Langen P.H.G. van, Treur J.: Strategic knowledge in design: a compositional approach. In: K. Hori (ed.), *Knowledge-Based Systems*. Special Issue on Strategic Knowledge and Concept Formation, vol. 11, issue 7-8 (1998) 405-416

22. Bresciani, P., Giorgini, P., Giunchiglia, F., Mylopoulos, J., and Perini, A.: Tropos: An Agent-Oriented Software Development Methodology. *Journal of Autonomous Agent and Multi-Agent Systems*, vol. 8 (2004) 203-236
23. Darimont, R., Delor, E., Massonet, P., and van Lamsweerde, A.: GRAIL/KAOS: An Environment for Goal-Driven Requirements Engineering, *Proc. ICSE'98 - 20th International Conference on Software Engineering*, Kyoto, vol. 2 (1998) 58-62
24. Antón, A.I.: Goal-based Requirements Analysis, *Proc. of the International Conference on Requirements Engineering (ICRE'96)*, IEEE Computer Soc. Press, Colorado Springs, Colorado, USA (1996) 136-144