Design of Collaborative Information Agents

Catholijn Jonker*, Matthias Klusch**, Jan Treur*

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

**German Research Center for Artificial Intelligence
Stuhlsatzenhausweg 3, 66123 Saarbrücken, Germany
Email: klusch@dfki.de URL: http://www.dfki.de/~klusch

Abstract. Effective development of nontrivial systems of collaborative information agents requires that an in-depth analysis is made resulting in (1) specification of requirements at different levels of the system, (2) specification of design structures, and (3) a systematic verification. To support a widespread use of intelligent information agents for the Internet, the challenges are (1) to identify and classify a variety of instances of the different types of reusable (requirement, design and proof) patterns, (2) build libraries of them, and (3) provide corresponding easy-to-use plug-in information agent components to the common user. In a simplified example it is shown which types of reusable requirements patterns, design patterns, and proof patterns can be exploited, and how these patterns relate to each other.

1 Introduction

The domain of collaborative information agents (cf. [27]) imposes specific requirements on the functionality and behaviour of the agents and their interaction. Various applications have been developed and are being developed; e.g., [7], [8], [24], [25], [26], [31], [33]. Usually it is required that the information agents cooperate with each other and the users in a coordinated manner. Although, especially for the simpler applications, it is tempting to just focus on the programming of these agents, for more sophisticated information agent applications it is essential to design them on a higher conceptual level. A principled design process not only involves conceptual design specifications but also requirements specifications (e.g., [11], [28], [32]) and verification e.g., [30]).

In the first place, in order to develop a system with appropriate properties, such a design process includes the analysis of *requirements* on the functionality or behaviour of the overall (multi-agent) system consisting of the information agents. Moreover, requirements (to be) imposed on the individual information agents and the relationship of these requirements to dynamic properties of the overall (multi-agent) system are important. Requirements can be specified in a conceptual and precise manner. In

addition or instead of direct requirement formulations, often *scenario's* are used as a means to specify required behaviour.

During further design, in relation to these requirements *design structures* are used which are specified in a conceptual and precise manner. Both requirements and design structures can take the form of reusable patterns, which are maintained in a library.

A *verification* process, after a system has been designed can demonstrate that the designed system actually will show the required behaviour. In verification formalised behavioural requirements and a formalised conceptual design play a main role. Also for verification proofs reusable patterns can be specified.

The methodological approach discussed in this paper assumes two specification languages: a language to specify (behavioural) *requirements* and *scenarios* for (systems of) information agents, and a language to specify *design descriptions.* Each of these languages fulfills its own purpose. A language to specify a (multi-agent) system architecture needs features different from a language to express properties of such a system. Therefore, in principle the two languages are different. The distinction between these specification languages follows the distinction made in the AI and Design community (cf. [20]) between the *structure* of a design object on the one hand, and *function* or *behaviour* on the other hand. For both languages informal, semi-formal and formal variants are assumed, to facilitate the step from informal to formal. Formal models specified in the two languages can be related in a formal manner: it is formally defined when a design description satisfies a requirement or scenario specification, and this formal relation is used to verify that the design description fulfills the requirements and scenarios.

Reusability can be supported for all of the above aspects. Reusable requirement and scenario patterns can be used to identify and classify the type of properties required for the overall system and for each of the information agents. Reusable design patterns can be used to identify and classify a system design description. Reusable verification proof structures can be used to establish, for example, that a design description indeed fulfills certain requirements

For reasons of presentation we will use a *simple example domain* for illustration: the design of an information agent which collaborates with multiple information providers to keep its human users informed about information available on the Web within their scope of interests, both with (pull) or without (push) explicit requests from the user. Representations of requirements and scenarios for this example domain are discussed in section 2. In section 3 design structures are discussed. Verification is the topic of section 4 while section 5 concludes with a brief discussion.

2 Specification of Requirements and Scenarios

In Requirements Engineering (cf. [11], [13], [14], [15], [22], [28], [29], [32])the role of scenarios, in addition to requirements, has gained more importance; e.g., see [17], [34]. Traditionally, scenarios or use cases are examples of interaction patterns between the users and a system; they are often used during the requirement elicitation, being regarded as effective ways of communicating with the stakeholders (i.e., domain experts, users, system customers, managers, and developers). In the specific case of

designing collaborative information agents, requirements and scenarios of the following types can be specified:

- for a *multi-agent system as a whole* with respect to users and environment, abstracting from the specific agents in the system
- for the *interaction patterns between specific (groups of) agents* within the system, and
- for *individual agents* within the system.

During a development process, starting from behavioural requirements (and scenarios) for the system as a whole with respect to users and environment, by requirement refinement, requirements for interaction patterns and behavioural properties of the agents themselves can be identified. In this way refinement of the requirements and scenarios imposed on an overall system leads to the identification of agents within the system, and their properties. Such an approach actually makes part of the heuristics of the design process explicit (e.g., the design choice for which agents to distinguish, and with which properties). One of the underlying assumptions is that such a design method will lead to designs that are more transparent, better maintainable, and can be (partially) reused more easily within other designs.

Different representations can be used to express the same requirement or scenario, varying from informal to formal. Representations can be made, for example, in a graphical representation language, or a natural language, or in combinations of these languages, as is done in UML's use cases (cf. [18], [21]). Scenarios, for instance, can be represented using a format that supports branching points in the process, or in a language that only takes linear structures into account.

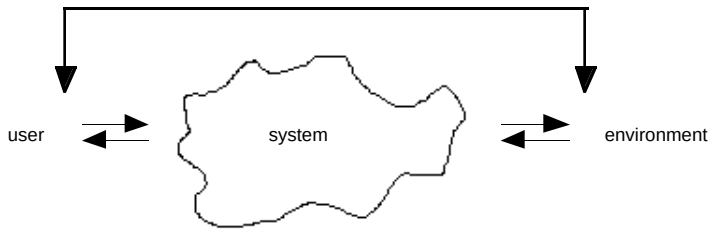

Fig. 1. Overall system requirement

2.1 Requirements for the overall system

At the most abstract level, the requirements on the behaviour of a system with respect to a user can be specified, abstracting from the form (architecture) the system has or will get. Because system behaviour may depend on its further environment, often also a reference to this environment is made. These overall system requirements are expressed as temporal relationships in terms of *user output and input* and *environment*

output and input at certain points in time, and make no reference to system structures; see Fig. 1. As an example, the following requirement for the overall system is shown.

Example 1: ***System requirement pattern***

if at any point in time the environment generates output ... to the system
and earlier the user generated output ... to the system,
then some time later the user will receive input ... from the system

In the context of our example domain we can state the following *informal* system requirements and a related scenario for the overall system with respect to users and the environment. The first of these requirements expresses pull behaviour of the system, and the second push behaviour.

Example 2: *Informal requirement and scenario*

1. Global system requirements

GR1 informal

If the user requests information for a scope of interest and at the WWW information within that scope is available, then this information is offered to the user.

GR2 informal:

The user is kept informed of new information on the World Wide Web which is within the user's scope of interest.

2. Global scenario (GS1 informal):

- *user generates a scope of interest*
- *user is waiting*
- *new information within the user's scope of interest becomes available on the World Wide Web*
- *the user receives results for his/her scope of interest*

Requirements and scenarios can be reformulated to more structured and precise forms. During such an analysis process the relevant concepts (*domain ontology*) can be identified and how they relate to *input* or *output*, for example of agents, or, as in this case to input and output of user and environment. For nontrivial behavioural requirements a *temporal structure* has to be reflected in the representation. This entails that terms such as 'at any point in time', 'at an earlier point in time', 'after', 'before', 'since', 'until', and 'next' are used to clarify the temporal relationships between different fragments in the requirement. An example of a *structured semi-formal* reformulation of the informally specified overall system requirement GR2 given above is as follows.

Example 3: ***Structured semi-formal system requirement***

Global system requirement (GR2 semi-formal):

At any point in time,
if *at an earlier point in time*
user output : *a scope of interest,* ***and***
since then
not user output : *retraction of this scope of interest****, and***
just now
World Wide Web output: *new information within this scope*
then just after now
user input: *new information within this scope*
?

The scenario GS1 can be formalized by utilizing ontological concepts (input and output) and sequence of events as a temporal trace as part of a formal temporal model. Regarding the formalization of requirement GR1 the following (sorted first order logic) formal ontology elements can be used.

Example 4: ***Ontological elements and relations***

ontology element:	*explanation:*
SCOPE	a sort for the scopes of users' interests
USER	a sort for the names of different users
INFO_ELEMENT	a sort for the information delivered by the agent
result_for_scope	a binary relation on INFO_ELEMENT and SCOPE
input:	
is_interested_in	a binary relation on USER,and SCOPE
output:	
result_for_user	a ternary relation on INFO_ELEMENT, USER and SCOPE

?

In addition, the temporal structure has to be expressed in a formal manner. For to obtain a more sophisticated formalisation of requirements of the domain we can use different variants of temporal logic (cf. [1]) depending on the type of properties to be expressed. For example, linear or branching time temporal logic are appropriate to specify various agent (system) behavioural properties. Examples of formal requirement specification languages based on such variants of temporal logic are described in [9], [10], [13], [14], [15], [16], [19], [30]. However, for information agents, it might be necessary to specify adaptive properties such as 'exercise improves skill' for which we have to explicitly express a comparison between different histories. This requires a form of temporal logic language which is more expressive than those allowing to model at each time point only one history. An example of

such a more expressive formal language in which different histories can be compared was introduced in [23]; it is defined as follows.

Definition 1: ***Temporal Requirement Language TRL***

The semantics of TRL are based on *compositional information states* which evolve over time.

1. An *information state* I of a (part of a) system S (e.g., the overall system, or an input or output interface of an agent) is an assignment of truth values {true, false, unknown} to the set of ground atoms describing the information within S.
2. The set of all possible information states of S is denoted by *IS*(S).
3. A *trace* $\mathcal{I}$ of S is a sequence of information states $(I^t)_{t \in \mathbf{N}}$ in *IS*(S). Given a trace $\mathcal{I}$ of S, the information state of the input interface of an agent A at time point t is denoted by $state_S(\mathcal{I}, t, input(A))$. Analogously, $state_S(\mathcal{I}, t, output(A))$, denotes the information state of the output interface of agent A at time point t within system S.
4. The information states can be related to statements via the formally defined satisfaction relation |=, comparable to the Holds-predicate in the situation calculus. Behavioural properties can be formulated in a formal manner, using quantifiers over time and the usual logical connectives such as not, &, ⇒.

The requirement and scenario informally described above in Example 2 can formally be specified in TRL as given in the following example.

Example 5: ***Formal requirement and scenario***

Global system requirements

GR1 formal:

$\forall\ \mathcal{I}, t$
$[\ state_S(\mathcal{I}, t, output(U)) \models is\_interested\_in(U:USER, S:SCOPE)\ \&$
$state_S(\mathcal{I}, t, output(Web)) \models result\_for\_scope(I:INFO\_ELEMENT, S:SCOPE)\ ]$
$\Rightarrow\ \exists t' > t:$
$state_S(\mathcal{M}, t', input(U)) \models result\_for\_user(I:INFO\_ELEMENT, U:USER, S:SCOPE)$

GR2 formal:

$\forall \mathcal{I}, t1, t2>t1$
$state_S(\mathcal{I}, t1, output(U)) \models is\_interested\_in(U:USER, S:SCOPE)\ \&$
$state_S(\mathcal{I}, t2, output(Web)) \models result\_for\_scope(I:INFO\_ELEMENT, S:SCOPE)\ \&$
$\forall t'\ [\ t1 < t' < t2 \Rightarrow$
$[\ not\ state_S(IT, t', output(Web)) \models result\_for\_scope(I:INFO\_ELEMENT, S:SCOPE)\ \&$
$not\ state_S(\mathcal{I}, t', output(U)) \models not\ is\_interested\_in(U:USER, S:SCOPE)\ ]$
$\Rightarrow \exists t3 > t2$
$state_S(IT, t3, input(U)) \models result\_for\_user(I:INFO\_ELEMENT, U:USER, S:SCOPE)]$

Global scenario (GS1 formal):

$state_S(\mathcal{I}, 1, output(U)) \models is\_interested\_in(U:USER, S:SCOPE)$

state$_S$(ω, 3, output(Web)) |= result_for_scope(I:INFO_ELEMENT, S:SCOPE)
state$_S$(ω, 4, input(U)) |= result_for_user(I:INFO_ELEMENT, U:USER, S:SCOPE)

GR1 addresses the case that information relating to a scope is already present (pull), whereas GR2 addresses the case that the information becomes available later (push). Note that in contrast to situation calculus, an infix notation is used for the |=-predicate and an explicit reference is made to a trace. This allows for specification of adaptive properties by comparison of different histories.

The formal scenario representation (GS1 formal) relates to the second formal requirement representation expressed above. Note that point at time point 2 nothing happens, which corresponds to the waiting of the user, of course in another (but similar) scenario the waiting could take more time.

So far the requirements and scenarios have been formulated for the system as a whole with respect to the users and the Web considered as the given environment. They express the desired behaviour from a global perspective, and only refer to input and output of users and the environment. Otherwise no assumptions were made on the design of the multi-agent system; in particular, no specific agents were assumed. However, one can identify more elementary units of behaviour by refining these requirements and scenarios (*behavioural refinement*); of course, which units of behaviour are chosen is a specific design decision.

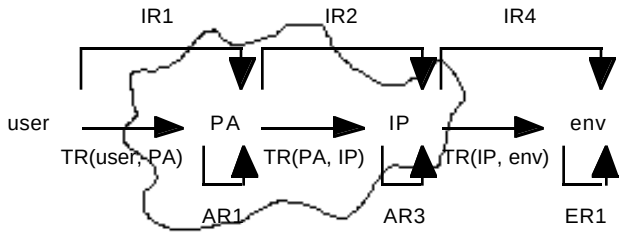

Fig. 2. Refinement of overall system requirements into interaction (IR), transfer (TR), agent (AR), and environment (ER) requirements.

In relation with the refinements for our example domain the design decision is made to identify at least two types of agents:

- *Personal Assistant agents*, that are in direct contact with users, and
- *Information Provider agents*, that only handle unpersonalized needs for information and are in contact with (parts of) the environment, i.e., the Web.

Fig. 2 summarizes the different types of requirements between the user, personal assistant agent (PA), information provider agents (IP), and the Web considered as to be the environment. Each of these types is discussed informally in subsequent sections.

2.2 Requirements for interaction patterns

The interaction between units of behaviour can be specified in terms of temporal relationships between their *output*. A typical example interaction pattern between two agents A and B is as follows.

Example 6: *Agent interaction pattern*

At any point in time
if *A generates as output ....*
*[**and*** in the past *at the output of A it was generated ...*
and in the past *at the output of B it was generated ...]*
then some time later *B generates as output ....*

This example can be generalised to interaction between an arbitrary number of agents. As an illustration, in the following examples the requirements for the interactions between the different agents of our example domain are discussed (see also fig.2).

2.2.1 Interaction between user and Personal Assistant

For our example, it can be postulated that on the basis of specific user outputs concerning their interest, an unpersonalized scope of interest is identified by the Personal Assistant agent (PA): interaction requirement IR2 below. Interaction requirement IR1 describes an interaction between three agents: user, PA and IP. It expresses that if the user puts forward a request scope, and earlier an IP has generated information for the PA within this scope, then this will be generated for the user.

Example 7: *Interaction requirement between user and PA*

User-PA interaction requirements
(IR1 informal):
At any point in time
if *a user has generated on its output a personal scope of interest,*
and *earlier an Information Provider generated as output for the Personal Assistant information within that scope which was not retracted in the meantime*
then *the Personal Assistant will generate this information on its output for the user*
(IR2 informal):
At any point in time
a. ***if*** *a user has generated on its output a personal scope of interest,*
then *the Personal Assistant will generate on its output an unpersonal scope based on this scope for a set of Information Providers*
b. ***if*** *a user has generated on its output a personal interest scope retraction,*
and *no other users have generated the same scope without having it retracted,*
then *the Personal Assistant will generate on its output an unpersonal scope retraction based on this scope.*

Note that by IR1 & 2 new ontology elements are created that need not be part of the ontologies of a user or system environment input or output (and are also not meant to be part of these ontologies).

2.2.2 Interaction between Personal Assistant and Information Provider

The requirements IR1 and IR2 are imposed on the interaction between users and personal assistants whereas the following requirements are valid for any interaction between assistant and provider agents. For example, the requirement IR4 expresses that a personal assistant will receive information within a specified scope from a provider agent as soon as it is available.

Example_8: ***Interaction requiremenst between assistant and provider agent***

Interaction requirement (IR3 informal)

At any point in time
if *a Personal Assistant has generated on its output an unpersonal scope for a set of Information Providers*
then *every Information Provider from this set generates a related scope for its environment*

Interaction requirement (IR4 informal)

At any point in time
if *an Information Provider generated on its output new information for a Personal Assistant,* ***and*** *the new information matches an unretracted scope that the Personal Assistant received* earlier *on its input from a user,*
then *this Personal Assistant will generate the new information on its output for this user.*

2.2.3 Interaction between information provider and environment

The following requirements specify the required interaction requirement pattern between information provider agents and the environment:

Example_9: ***Interaction requirement between Information Provider and environment***

Interaction requirement (IR5 informal):

At any point in time
if *an Information Provider has generated a scope on its output for its environment*

and later *information becomes newly available in its environment that matches this scope,*
and *this scope was not retracted before the new information became available,*
then *the environment will generate this information as output for the information Provider*

Interaction requirement (IR6 informal):
At any point in time
if *the environment of an Information Provider generates on its output new information,*
and *the new information matches an unretracted unpersonal scope that the Information Provider received* earlier *on its input from a Personal Assistant,*
then *this Information Provider will generate the new information at its output for the Personal Assistant.*

2.3 Transfer requirements

Any successful collaboration in our domain requires a secure, reliable transfer of information between different components of the system (e.g., communication between agents). This can be specified as a temporal relationship between *output* of one agent or component and *input* of another one. The following transfer requirement pattern between components A and B expresses this.

Example 10: ***Transfer requirement between system components***

Transfer requirement between components A and B (TR(A, B) informal):
At any point in time
if *A generates information for B at its output,*
then some time later *B will receive this information on its input.*

2.4 Requirements for an individual information agent

Temporal relationships between an individual agent's *own input and output* define its behaviour. A typical pattern for an agent A is as follows:

if A receives as input ....
[and in the past as input A received ...
and in the past as output A generated ...]
then some time later A generates as output ....

Please note that such behaviour patterns describe requirements which are of a lower process abstraction level compared to the overall system requirements. For example,

the following requirements can be imposed on individual personal assistant and information provider agents.

Example 11: ***Assistant (PA) and provider (IP) agent requirements***

Personal assistant agent behaviour requirement (AR1 informal):
At any point in time
if *an incoming scope of interest of a user is received,*
then some time later *the PA will communicate an unpersonal scope based on that user scope to a set of Information Provider agents.*

Personal Assistant agent behaviour requirement (AR2 informal):
At any point in time
if *a PA receives new information on its input ,*
then some time later ***for each*** *user which communicated earlier a scope matching that information the PA will generate on its output the related scope result for that user, unless the PA received a corresponding scope retraction from this user before it received the new information.*

Information provider agent behaviour requirement (AR3 informal):
At any point in time
if *an incoming scope of interest is received by an IP,*
then some time later *the IP will generate this scope as output for its environment*

Information Provider agent behaviour requirement (AR4 informal):
At any point in time
if *an IP receives new information on its input ,*
then some time later ***for each*** *PA which communicated earlier a scope matching that information the IP will generate on its output the related scope result for that PA,* ***unless*** *the IP received a corresponding scope retraction from this PA before it received the new information.*

2.5 Requirements for the environment

Assumptions on the environment can be formulated as temporal relationships between the *environment's input and output*, according to the same patterns as for agents. In our example, it is assumed that the environment of the overall system (the Web), is partitioned according to different information provider agents each of which have information about their own (part of the) environment. An example of an environment requirement is the following.

Example 11: ***Environment requirements***

Environment requirement (ER1 informal):

At any point in time
if *the environment of an Information Provider has received a scope on its input*
and now *or* later *information is or becomes newly available in this environment that matches this scope,* ***and*** *this scope was not retracted before the new information became available,*
then some time later *the environment will generate this information as output for the Information Provider*

2.6 Requirements for components within an information agent

It is possible to define more elementary units of behaviour within an agent by refining some of the overall requirements imposed on the agent. Whether this is desirable or appears to be too complex to do for an agent certainly depends on whether or not its requirements are sufficiently elementary to serve as a starting point for a transparent design. For example, a possible behavioural refinement of the requirements imposed on personal assistants implies the internall use of user profiles.

Example 12: ***Personal Assistant agent (PA) component requirements***

Component requirement (CR1 informal):
The PA maintains a profile of its users that satisfies the following:
At any point in time
a. ***if*** *a user scope is received on PA's input,* ***then*** *it is added to the profile.*
b. ***if*** *a scope retraction is received on PA's input,* ***then*** *the corresponding user scope is removed from the profile .*

Component requirement (CR2 informal):
At any point in time
if *the PA receives new information on its input ,*
then for each *user profile matching that information the PA will generate on its output the related scope result for that user,*

3 Specification of Agent and Multiagent System Design Structures

In accordance to the specified requirements of agent behaviour and the environment in our example domain we can create now a design structure for individual agents and the multiagent system as a whole. Figure 3 depicts such a design structure. In this example, the multiagent system consists of two users, one personal assistant, two information providers, and the Web. An example of a specification language for such kind of design descriptions is given in DESIRE (cf. [5] for the principles behind DESIRE, and [3] for a case study). A personal assistant communicates with human agents (its users) and information provider agents which in turn communicate with the assistants and interact with their environment.

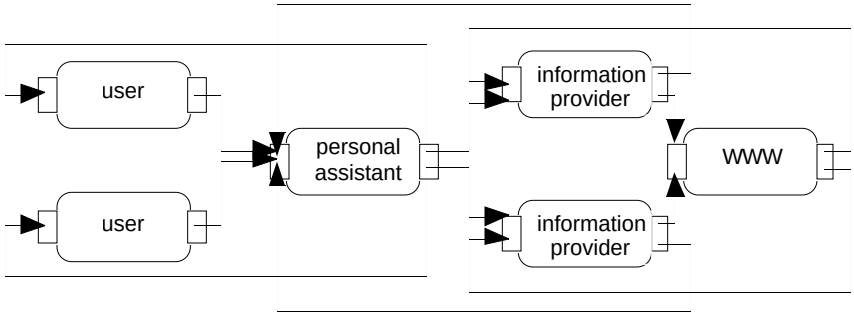

Fig. 3. Global multi-agent system design description in DESIRE.

Based on the requirements on the personal assistant agent as given above we can make the following design decisions in that we identify at least three sub-components of the agent:

1. a component to maintain the user profiles (called Maintenance of Agent Information),
2. a component capable of matching information with scopes (Proposal Determination, which takes place within Cooperation Management), and
3. a component that handles communication with other agents (called Agent Interaction Management).

The design description shown in Fig. 4 includes these components. Moreover components 'Own Process Control' and 'Maintenance of World Information' are part of the design for internal coordination and maintenance of a world model, respectively. The design of the agent is based on a reusable design pattern for the weak agent notion (cf. [35]), called GAM (Generic Agent Model) (cf. [6]). A more extensive description of this design pattern can be found in [6]. At the highest abstraction level within the agent, a number of processes is distinguished (see Fig. 4).

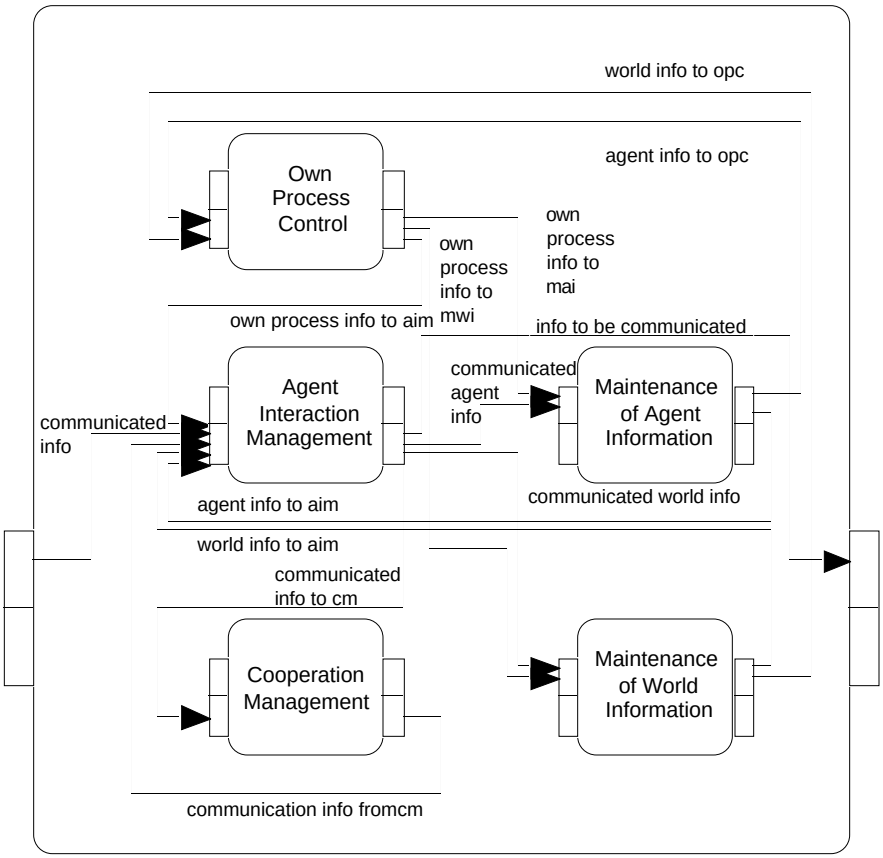

Fig. 4. Internal design description of the Personal Assistant.

First, a process that manages communication with other agents, modelled by the component agent interaction management. This component identifies and extracts the information from incoming communication and determines which other processes within the agent need this communicated information. Moreover, outgoing communication is prepared. Next, the agent needs to maintain (e.g., profile) information on the other agents with which it co-operates: maintenance of agent information. The component maintenance of world information is included to store the world information (e.g., information on attributes of information). The process own process control defines different characteristics of the agent and determines foci for behaviour. The component world interaction management within GAM models interaction with the world: initiating observations, receiving observation results, and execution of actions in the world. As the PA only communicates and has itself no direct interaction with the world, this component was left out for the example here. The component cooperation management models collaboration with other agents; in this example it was used to model the matching between information and scopes.

4 Verification

In general, verification can take place in different forms. For example:

- to verify scenarios against requirements
- to verify requirements on a system or agent against other requirements on the system or agent
- to verify requirements on a system against requirements of agents included in the system
- to verify a design structure against requirements

Each of these types of verification will be discussed briefly in subsequent sections.

4.1 Verification of requirements against scenarios

Having specified both, requirements and scenarios in a requirements engineering process provides the possibility of mutual comparison and verification: the requirements can be verified against the scenarios, and vice versa. By this, ambiguities and inconsistencies within and between the existing requirements or scenarios may be identified, but also missing requirements or scenarios.

Checking a temporal formula F, which formally represents a requirement, against a temporal model $\mathfrak{M}$, formally representing a scenario, means that formal verification of requirements against scenarios can be done by *model checking*. A formal representation $\mathfrak{M}$ of a scenario SC and a formal representation F of a requirement are compatible if the temporal formula is valid for the model. As an example, scenario GS1 can be verified against requirement GR2 to find out that the scenario indeed fulfills the requirement. It is possible to reformulate and/or refine requirements at the same (system or agent) level. It can be verified whether the reformulation implies the original requirement, or that they are even logically equivalent.

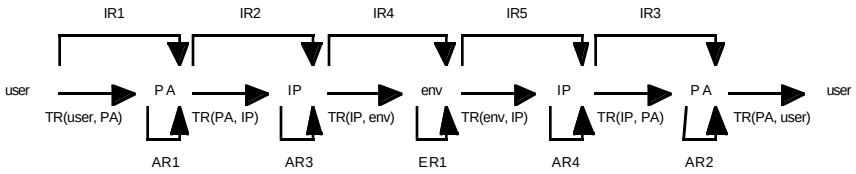

Fig. 5. Overview of different relations between requirements

4.2 Verification of system requirements against other system requirements

Consider the following example.

Example 13: ***Verification of system/system requirements***

Assume a given transfer requirement TR(PA, user) from PA to user is proven to be satisfied. Then, requirement GR1 on the system level is implied by the interaction requirements IR1, IR2, IR3, IR4, and IR5; this yields the following proof pattern (see Fig. 5, upper part):

(pp1) IR1 & IR2 & IR3 & IR4 & IR5 & TR(PA, user) => GR1

4.3 Verification of system requirements against requirements on agents

Obviously, to prove that overall system requirements are fulfilled we need to prove the behavioural requirements of the agents involved. In particular, if it can be proven that the agent requirements are fulfilled, then also the interaction requirements hold. This requires that the appropriate transfer requirements (for agents communication) hold. In our example domain we can give the following example.

Example 13: ***Verification of system/agents requirements***

Personal Assistant agent requirement AR1 (cf. ex. 11) implies the interaction requirement IR1 (cf. Example 7) under the condition that the communication successfulness requirement TR(user, PA) (cf. Example 10) holds. This is expressed by the following proof pattern:

(pp2) AR1 => [TR(user, PA) => IR1]

A similar proof pattern shows that the other interaction requirements can be derived from the agent requirements, assuming communication successfulness:

(pp3) AR3 => [TR(PA, IP) => IR2]
ER1 => [TR(IP, env) => IR4]
AR4 => [TR(env, IP) => IR5]
AR2 => [TR(IP, PA) => IR3]

In terms of Fig. 3, this can be formulated as 'the upper steps are implied by the lower steps'. As a result, the combination of proof patterns (pp3) and (pp1) yields the following proof pattern:

(pp4) [TR(user, PA) & TR(PA, IP) & TR(IP, env) & TR(env, IP) & TR(IP, PA)
& AR1 & AR2 & AR3 & AR4 & ER1]
=> GR1

4.4 Verification of agent requirements against component requirements

Within an agent it is possible to determine logical relationships between requirements of the agent as a whole and requirements on components within the agent. As an example, both component requirements CR1 and CR2 (cf. Example 12) together imply the individual agent requirement AR2 (cf. Example 11), assuming the appropriate transfer requirements within the agent.

4.5 Verification of requirements specification against design specification

In the case that no refined requirements for agents or agent components exist, it can be directly verified that the given requirements are implied by the design description. This requires a (standard) embedding of the information involved in a design specification in the specification language for behavioural properties.

4.6 Compositional verification as a verification method

A refinement of a behavioural requirement during a design process defines requirements on more elementary units of behaviour. According to this, starting with behavioural requirements of the entire system, it can be decided about what kind of agents to use in the system, and in particular, which individual agent is meant to show which type of behaviour. In a next step, for each of the agents it can be decided whether it is desirable to further refine its requirements depending on whether or not they are elementary enough to serve as a starting point for a transparent design of the agent itself. This, in turn, can lead to the identification of requirements on more elementary units of behaviour within the agent and, in relation to this, to different components within the agent to perform this behaviour. This iterative requirements refinement process may yield an arbitrary number of high to lower process abstraction levels within the system and agents (see Fig. 6). Sets of requirements at a lower level can be chosen in such a way that they realise a higher level requirement, in the following sense:

given the *process composition* relation defined in the design description,

if the chosen *refinements* of a given requirement *are satisfied*,
then also the original *requirement is satisfied*.

This defines the logical aspect of a behavioural refinement relation between requirements. Based on this logical relation, refinement relationships can also be used to verify requirements: e.g., if the chosen refinements of a given requirement all hold for a given system description, then this requirement can be proven to hold for that system description. Similarly, scenarios can be refined to lower process abstraction levels by adding the interactions between the sub-processes. At each level of abstraction, requirements and scenarios employ the terminology defined in the ontology for that level. The methodological approach to the creation of different process abstraction levels in relation to requirements refinement has a natural connection to the process of *compositional verification* (cf. [23]).

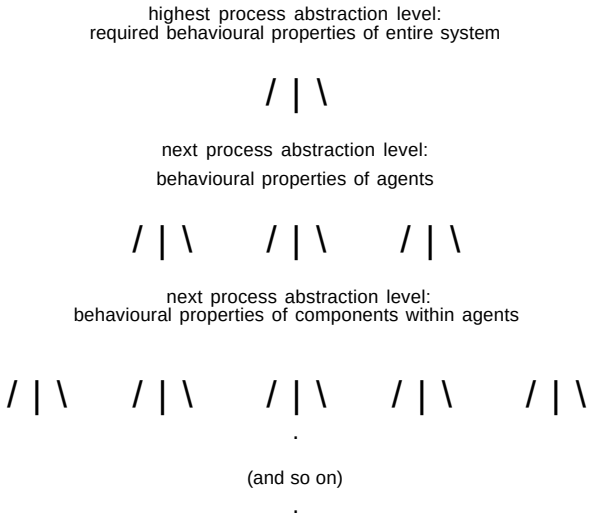

Fig. 6. Behavioural properties at different process abstraction levels

The purpose of verification is to prove that, under a certain set of assumptions, a system will adhere to a certain set of properties, expressed as requirements and scenarios. In order to prove that a system is behaving as required, not only a complete specification of the system is necessary, but also the set of requirements and scenarios to verify the system against. If this set is not available, the verification process is hampered to a great extent, because formulating sufficiently precise requirements (and scenarios) for an existing system is nontrivial. For the purpose of verification it has turned out useful to exploit compositionality.

Compositional verification as described in [23ositional structure of the system into account. The requirements and scenarios are formulated formally in terms of temporal semantics. During the verification process the requirements and scenarios of the system as a whole can be derived from properties of agents (one process

abstraction level lower) and these agent properties, in turn, can be derived from properties of the agent components (again one abstraction level lower), and so on (see Fig. 6).

Primitive components (those components that are not composed of others) can be verified using more traditional verification methods making use of the design description only. Verification of a (composed) component at a given process abstraction level is done using:

- *properties of the sub-components* it embeds
- a specification of the *process composition relation*
- *environmental properties* of the component (depending on the rest of the system, including the world).

This exploits the compositionality in the verification process. Given a set of environmental properties, the proof that a certain component adheres to a set of behavioural properties depends on the (assumed) properties of its sub-components, and the composition relation: properties of the interactions between those sub-components, and the manner in which they are controlled. The assumptions under which the component functions properly, are the properties to be proven for its sub-components. This implies that properties at different levels of process abstraction play their own role in the verification process. A condition to apply a compositional verification method is the availability of an explicit specification of how the system description at an abstraction level is composed from the descriptions at the adjacent lower abstraction level.

Compositionality in verification reduces the search space for the properties to be identified, and the proofs, and supports reuse of agents and components. Complexity in a compositional verification process is two-fold: both the identification of the appropriate properties at different levels of abstraction and finding proofs for these properties can be complex. If the properties already are identified as part of the requirements engineering process, this means that the complexity of part of the verification process is reduced: 'only' the complexity of finding the proofs remains. Our experience in a number of case studies is that having the right properties reduces much more than half of the work for verification: due to the compositionality, at each process abstraction level the search space for the proofs is relatively small.

If no explicit requirements engineering has been performed, finding these properties for the different process abstraction levels can be very hard indeed, as even for a given process abstraction level the search space for possible behavioural requirement formulations can be nontrivial. If as part of the design process requirements have been (formally) specified as well at different levels of process abstraction, these can be used as a useful starting point for a verification process; they provide a detailed map for the verification process and thus reduce the complexity by eliminating the search space for the requirement formulations at different process abstraction levels.

Integration of the requirements engineering process within the system design process leads to system designs that are more appropriate for verification than arbitrary architectures. Moreover, reuse is supported; for example, replacing one component by

another is possible without violating the overall requirements and scenarios, as long as the new component satisfies the same requirements and scenarios as the replaced component. Note that the idea of refinement is well-known in the area of (sequential) programs, e.g., [12]. The method of compositional requirements specification proposed here exploits a similar idea in the context of behavioural requirements.

5 Discussion

Effective development of nontrivial systems of collaborative information agents requires an in-depth analysis resulting in (1) specification of requirements at different levels of the system, (2) specification of design structures, and (3) a systematic verification. To support a widespread use of intelligent information agents for the Internet, the challenges are (1) to identify and classify a variety of instances of the different types of reusable (requirement, design and proof) patterns, (2) build libraries of them, and (3) provide corresponding easy-to-use plug-in information agent components to the common user. In this paper we have informally shown by the use of a simplified example, which types of reusable requirements patterns, design patterns, and proof patterns can be exploited, and how these patterns relate to each other.

In particular, it is proposed that two specification languages are used: a language to specify *design descriptions* such as DESIRE, and a language to specify (behavioural) *requirements* and *scenarios* such as TRL (cf. section 2.1). Each of these languages has its own chacteristics to fulfill its purpose. The distinction is similar to the one made in the AI and Design community [20], namely to distinguish between the structure of a design object on the one hand, and *function or behaviour* on the other hand. For both languages informal, semi-formal and formal variants have to be available to support the step from informal to formal, and, for example, to support a communication with stakeholders.

As said above, a formal specification language of the first type, and a semi-formal and graphical variant of this language, is already available in the compositional multi-agent system development method DESIRE, and is supported by the DESIRE software environment. A number of generic models or design patterens for agents are available; for example for weak agents [6], for cooperative agents able to work in projects [2], and BDI-agents [4]. However, this language focuses on design structures and was never meant to specify requirements or scenarios; a language to specify a (multi-agent) system architecture at a conceptual design level needs features different from a language to express properties of a system. In current research, further integration of the approach to requirements engineering is addressed.

Another main challenge in the design of (collaborative) information agents is the thoughtful design and deployment of standardized plug-in information agent components to the community. This would enable the common user of the Internet to build its own intelligent information agent thereby treating the Web as a common public good, in contrast to the current practice of searchbots and associated ranking warfare. The examples included in this paper are only a starting point for a more systematic exploration of this vision of composite plug-in information agents.

References

1. Benthem, J.F.A.K. van, The Logic of Time: A Model-Theoretic Investigation into the Varieties of Temporal Ontology and Temporal Discourse, Dordrecht: Reidel, 1983.
2. Brazier, F.M.T., Cornelissen, F., Jonker, C.M., and Treur, J., Compositional Specification of a Reusable Co-operative Agent Model International Journal of Cooperative Information Systems. In press, 2000.
3. Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N.R. and Treur, J. Formal specification of Multi-Agent Systems: a real World Case. In: Lesser, V. (ed.), Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95, MIT Press, Menlo Park, VS, 1995, pp. 25-32. Extended version in: International Journal of Cooperative Information Systems, M. Huhns, M. Singh, (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.
4. Brazier, F.M.T., Dunin-Keplicz, B.M., Treur, J., and Verbrugge, L.C., Modelling Internal Dynamic Behaviour of BDI agents. In: J.-J. Ch. Meyer and P.Y. Schobbes (eds.), Formal Models of Agents (Selected papers from final ModelAge Workshop). Lecture Notes in AI, vol. 1760, Springer Verlag, 1999, pp. 36-56.
5. Brazier, F.M.T., Jonker, C.M., and Treur, J., Principles of Compositional Multi-agent System Development. In: J. Cuena (ed.), Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98, 1998, pp. 347-360. To be published by IOS Press, 2000.
6. Brazier, F.M.T., Jonker, C.M., and Treur, J., Compositional Design and Reuse of a Generic Agent Model. Applied Artificial Intelligence Journal. In press, 2000.
7. Chavez, A., Maes, P., Kasbah: An Agent Marketplace for Buying and Selling goods. In: Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'96, The Practical Application Company Ltd, Blackpool, 1996, pp. 75-90.
8. Chavez, A., Dreilinger, D., Gutman, R., Maes, P., A Real-Life Experiment in Creating an Agent Market Place. In: Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'97, The Practical Application Company Ltd, Blackpool, 1997, pp. 159-178.
9. Dardenne, A., Lamsweerde, A. van, and Fickas, S., Goal-directed Requirements Acquisition. Science in Computer Programming, vol. 20, 1993, pp. 3-50.
10. Darimont, R., and Lamsweerde, A. van, Formal Refinement Patterns for Goal-Driven Requirements Elaboration. Proc. of the Fourth ACM Symposium on the Foundation of Software Engineering (FSE4), 1996, pp. 179-190.
11. Davis, A. M., Software requirements: Objects, Functions, and States, Prentice Hall, New Jersey, 1993.
12. Dijkstra, E.W., A discipline of programming. Prentice Hall, 1976.
13. Dubois, E. (1998). ALBERT: a Formal Language and its supporting Tools for Requirements Engineering.
14. 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.

15. Dubois, E., Yu, E., Petit, M., From Early to Late Formal Requirements. In: Proceedings IWSSD'98. IEEE Computer Society Press, 1998.
16. Engelfriet, J., Jonker, C.M. and Treur, J., Compositional Verification of Multi-Agent Systems in Temporal Multi-Epistemic Logic. In: J.P. Mueller, M.P. Singh, A.S. Rao (eds.), Intelligent Agents V, Proc. of the Fifth International Workshop on Agent Theories, Architectures and Languages, ATAL'98. Lecture Notes in AI, vol. 1555, Springer Verlag, 1999, pp. 177-194. Extended version in Journal of Logic, Language and Information, to appear, 2000.
17. Erdmann, M. and Studer, R., Use-Cases and Scenarios for Developing Knowledge-based Systems. In: Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technologies and Knowledge Systems, IT&KNOWS (J. Cuena, ed.), 1998, pp. 259-272.
18. Eriksson, H. E., and Penker, M., UML Toolkit. Wiley Computer Publishing, John Wiley and Sons, Inc., New York, 1998.
19. Fisher, M., Wooldridge, M., On the Formal Specification and Verification of Multi-Agent Systems. International Journal of Cooperative Information Systems, M. Huhns, M. Singh, (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, vol. 6, 1997, pp. 67-94.
20. Gero, J.S., and Sudweeks, F., (eds.), Artificial Intelligence in Design '98, Kluwer Academic Publishers, Dordrecht, 1998.
21. Harmon, P., and Watson, M., Understanding UML, the Developer's Guide. Morgan Kaufmann Publishers, San Francisco, 1998.
22. 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, Proceedings of the 9th European Workshop on Modelling Autonomous in a Multi-Agent World, MAAMAW'99. Lecture Notes in AI, vol. 1647, Springer Verlag, Berlin, 1999, pp. 8-27.
23. Jonker, C.M. and Treur, J., Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness. In: W.P. de Roever, H. Langmaack, A. Pnueli (eds.), Proceedings of the International Workshop on Compositionality, COMPOS'97. Lecture Notes in Computer Science, vol. 1536, Springer Verlag, 1998, pp. 350-380
24. Jonker, C.M., Lam, R.A., and Treur, J., A Multi-Agent Architecture for an Intelligent Website in Insurance. In: M. Klusch, O. Shehory, and G. Weiss (eds.), Cooperative Information Agents III, Proceedings of the Third International Workshop on Cooperative Information Agents, CIA'99. Lecture Notes in Artificial Intelligence, vol. 1652. Springer Verlag, 1999, pp. 86-100.
25. Jonker, C.M., and Treur, J., Information Broker Agents in Intelligent Websites. In: I. Imam, Y. Kodratoff, A. El-Dessouki, and M. Ali (eds.), Multiple Approaches to Intelligent Systems (Proc. of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE'99). Lecture Notes in AI, vol. 1611, Springer Verlag, 1999, pp. 430-439.
26. Jonker, C.M., and Vollebregt, A.M.., ICEBERG: Exploiting Context in Information Brokering Agents. In: M. Klusch, L. Kerschberg, M. Papazoglou, and O. Shehory (eds.), Cooperative Information Agents IV, Proceedings of the Fourth International Workshop on Cooperative Information Agents, CIA 2000. Lecture Notes in Artificial Intelligence, Springer Verlag, 2000. This volume.

27. Klusch, M. (ed.), Intelligent Information Agents. Springer Verlag, 1999.
28. Kontonya, G., and Sommerville, I., Requirements Engineering: Processes and Techniques. John Wiley and Sons, New York, 1998.
29. Lamsweerde, A. van, Darimont, R., and Letier, E. (1998). Managing Conflicts in Goal-Driven Requirements Engineering. IEEE Transactions on Software Engineering, Special Issue on Managing Inconsistency in Software Engineering.
30. Manna, Z., and Pnueli, A., Temporal Verification of Reactive Systems: Safety. Springer Verlag, 1995.
31. Martin, D., Moran, D., Oohama, H., Cheyer, A., Information Brokering in an Agent Architecture. In: Proceedings of the Second International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'97, The Practical Application Company Ltd, Blackpool, 1997, pp. 467-486.
32. Sommerville, I., and Sawyer P., Requirements Engineering: a good practice guide. John Wiley & Sons, Chicester, England, 1997.
33. Tsvetovatyy, M., Gini, M., Toward a Virtual Marketplace: Architectures and Strategies. In: Proceedings of the First International Conference on the Practical Application of Intelligent Agents and Multi-Agent Technology, PAAM'96, The Practical Application Company Ltd, Blackpool, 1996, pp. 597-613.
34. Weidenhaupt, K., Pohl, M., Jarke, M. and Haumer, P., Scenarios in system development: current practice, in IEEE Software, pp. 34-45, March/April, 1998.
35. Wooldridge, M., and Jennings, N.R., Agent Theories, Architectures, and Languages: a survey. In: Wooldridge, M., and Jennings, N.R. (eds.) Intelligent Agents, Lecture Notes in Artificial Intelligence, vol. 890, Springer Verlag, Berlin, 1995, pp. 1-39.