Specification of Behavioural Requirements within Compositional Multi-Agent System Design

Daniela E. Herlea[1], Catholijn M. Jonker[2], Jan Treur[2], Niek J.E. Wijngaards[1,2]

[1] University of Calgary, Software Engineering Research Network
2500 University Drive NW, Calgary, Alberta T2N 1N4, Canada
Email: danah@cpsc.ucalgary.ca
URL: http://www.cpsc.ucalgary/~danah

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

Abstract. In this paper it is shown how informal and formal specification of behavioural requirements and scenarios for agents and multi-agent systems can be integrated within multi-agent system design. In particular, it is addressed how a compositional perspective both on design descriptions and specification of behavioural requirements can be exploited. The approach has been applied in a case study: the development of a mediating information agent. It is shown that compositional verification benefits from the integration of requirements engineering within the design process.

1 Introduction

Agent systems are among the most complex of systems to develop (cf. [20], [23]). The autonomy in the behaviour of the agents contributes inherently to this complexity. The tasks performed by the individual agents can be simple or complex in itself, but the agents' autonomy makes the emergent behaviour of the complete multi-agent system both hard to design and hard to verify. Nevertheless, in many applications it is required that the agents cooperate with each other and the users in a in some sense coordinated manner. Often it is essential to analyse requirements on the behaviour of the overall multi-agent system in relation to behavioural properties of the individual agents, in order to develop a system with the right properties.

Within multi-agent system development, the emphasis is often on specification of the system architecture that is designed, and on the implementation of this design. If requirements are considered, they are kept implicit or informal. In principle, the required behavioural properties play a heuristic role: the system design is made up in such a manner that the system behaviour does what is needed, although it is not explicitly specified what that means.

Requirements Engineering (cf. [5], [18], [21]) addresses the development and validation of methods for eliciting, representing, analysing, and confirming system requirements. Requirements express intended properties of the system, and scenarios specify use-cases of the intended system (i.e., examples of intended user interaction traces), usually employed to clarify requirements. Requirements and scenarios can be expressed in various degrees of formality, ranging from unstructured informal representations (usually during initial requirements acquisition) to more structured semi-formal representations and formal representations.

Requirements can be specified for a multi-agent system as a whole, but also for agents within a multi-agent system, and for components within agents. Starting from behavioural requirements for the *system as a whole*, by requirement refinement behavioural properties of *agents* can be identified, and, in a further step, for *components* within an agent. Such an approach fits quite well in compositional multi-agent system design, for example, as discussed in [3], and actually makes part of the heuristics of the design process explicit. One of the underlying assumptions is that such a compositional design method will lead to designs that are more transparent, better maintainable, and can be (partially) reused more easily within other designs. The process of requirements refinement defines the different *process abstraction levels* in more detail. On the basis of refinement of the requirements (and scenarios) for the entire system, system components are identified: agents, users and world components. For each of these components of the system, a specific sub-set of the refined requirements and scenarios is imposed to ensure that the system as a whole satisfies the overall requirements and scenarios. Also further refinement of the requirements and scenarios imposed on an agent leads to the identification of components within the agent, and their properties. The different refinement levels in requirements and scenarios are related to levels of process abstraction in the compositional design description being designed.

Within a compositional *verification* process, after a system has been designed, formalised behavioural requirements play a main role; cf. [17]. A verification process for an existing design often has a high complexity in two respects. On the one hand, the formal formulations of the properties at the different process abstraction levels have to be found. If no explicit requirements engineering has been performed, this can be very hard indeed, as the search space for requirement formulations is often not small and verification is only useful with respect to the appropriate requirements, and the properties and assumptions on which they depend. On the other hand, proofs have to be found. If, as part of the design process, requirements have been (formally) specified as well, these can be used as a starting point for a verification process, thus reducing the complexity of verification, by eliminating the search space for the requirement formulations at different process abstraction levels. If no requirements have been specified during the design process, during verification a form of reverse engineering has to be performed to obtain the (required) properties at the different process abstraction levels afterwards.

The methodological approach proposed results in the use of two (compositional) specification languages:

- a language to specify *design descriptions*
- a language to specify (behavioural) *requirements* and *scenarios*

Within the compositional multi-agent system development method DESIRE (cf. [3]; for a real-world case study see [2]), the first of these languages is already available; the second is currently being added. 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. [13], [14]) 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 needed, 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.

In this paper it is shown how specification of behavioural requirements and scenarios from informal to formal can be integrated within multi-agent system design, in particular for a compositional design method with an underlying formal conceptual model for design descriptions: DESIRE. The approach has been applied in a case study: the development of a mediating information agent.

The example domain for the case study is the development of a multi-agent system that keeps its human users informed with respect to their interests and the rapidly changing available information on the World Wide Web. The task of the multi-agent system is to inform each of its users on information available (e.g., papers) on the World Wide Web that is within their scope of interest. The sources of information are the World Wide Web, but also information providing agents that operate on the World Wide Web, for example, agents related to Web sites of research groups, which announce new papers included in their web-site.

Different representations of requirements and scenarios from informal via semi-formal to formal are discussed in Section 2. The use of requirements and scenarios refinement across process abstraction levels is explained further in Section 3. The integration of the verification process and Requirements Engineering is the topic of Section 4. Section 5 concludes the paper with a discussion.

2 Informal and Formal Representation

In Requirements Engineering the role of scenarios, in addition to requirements, has gained more importance; e.g., see [11], [22]. Scenarios or use cases are examples of interaction sessions between the users and a system [22]; 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). Scenarios, for example, are also employed to formalise interactions among components within the system. Having both requirements and scenarios in a requirements engineering process provides the possibility of mutual comparison: the requirements can be verified against the scenarios, and the scenarios can be verified against the requirements. By this mutual verification process, ambiguities and inconsistencies within and between the existing requirements or scenarios may be identified, but also the lack of requirements or scenarios: scenarios may be identified for which no requirements were formulated yet, and requirements may be identified for which no scenarios were formulated yet.

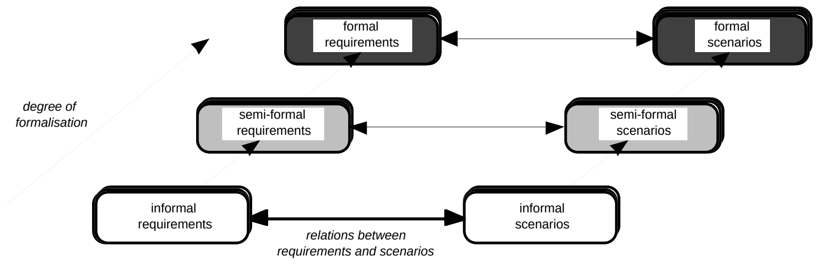

Fig. 1. Representations from informal to formal.

As stated above, requirements and scenarios are seen as effective ways of communicating with the stakeholders. This can only be true if requirements and scenarios are represented in a well-structured and easy to understand manner and are precise enough and detailed enough to support the development process of the system. Unfortunately, no standard language exists for the representation of requirements and scenarios. Formats of varying degrees of formality are used in different approaches. Informally represented requirements and scenarios are often best understood by the stakeholders (although approaches exist using formal representations of requirements in early stages as well: [9]). Therefore, continual participation of stakeholders in the process is possible. A drawback is that the informal descriptions are less appropriate when they are used as input to actually construct a system design. On the other hand, an advantage of using formal descriptions is that they can be manipulated automatically in a mathematical way, which enables verification and the detection of inconsistencies. Furthermore, the process of formalising the representations can be used as a way to disambiguate requirements and scenarios. At the same time however, a formal representation is less appropriate as a communication means with the stakeholders. Therefore, in our approach in the overall development process, different representations are used: informal and/or structured semi-formal representations (obtained during the process of formalisation) resulting from cooperation between stakeholders and designers of the system, and formal representations to be used by the designers during the construction of the design.

Independent of the measure of formality, each requirement and each scenario can be represented in a number of different ways, and/or using different representation languages, e.g., informally by way of use cases in UML. Examples are given below.

2.1 Informal Representations

Different informal representations can be used to express the same requirement or scenario. 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. [12], [15]). 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.

For the example application, first a list of nine, rather imprecisely formulated initial requirements was elicited. As an example, the elicited requirement on 'keeping aware' is shown below.

Example of an informal initial top level requirement:

L0.R1 The user needs to be kept 'aware' of relevant new information on the World Wide Web.

Requirement L0.R1 is based on the information elicited from the interview with the stakeholder. The following scenario was elicited from the stakeholder as well:

L0.Sc1

1. user generates an **awareness scope** : AS1
2. user is waiting
3. **new information** is made available on the World Wide Web
4. user receives **results for awareness scope** AS1: ASR1

The requirement L0.R1 was analysed and reformulated into a more precise requirement. In the (reformulated) scenarios and requirements, terminology is identified, relevant for the construction of domain ontologies (words in bold-face are part of the domain ontologies being acquired).

Example of a reformulation of a requirement at top level:

L0.R1.1 The user will be notified of **new information** (on the World Wide Web) on an **awareness scope**
after the user has expressed the **awareness scope** and
just after this **new information** becomes available on the World Wide Web,
unless the user has retracted the awareness scope (**awareness scope retraction**).

2.2 Structured Semi-formal Representations

Both requirements and scenarios can be reformulated to more structured and precise forms. To check requirements for ambiguities and inconsistencies, an analysis that seeks to identify the parts of a given requirement formulation that refer to input and to output of a process is useful. Such an analysis often provokes a reformulation of the requirement into a more structured form, in which the input and output references are made explicitly visible in the structure of the formulation. Moreover, during such an analysis process the concepts that relate to input can be identified and distinguished from the concepts that relate to output: *acquisition of a (domain) ontology* (cf. [19]) is integrated within requirements engineering. Possibly the requirement splits in a natural manner into two or more simpler requirements. This often leads to a number of new (representations of) requirements.

The ontology later facilitates the formalisation of requirements and scenarios, as the input and output concepts are already defined, at least at a semi-formal level. 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.

For the informally specified requirement L0.R1.1, for example, the following reformulation steps can be made:

> At any point in time
> The user will receive on its input **results for awareness scope** , i.e., **new information** on an **awareness scope**
> after the user has generated on its output the **awareness scope** and
> just after this **new information** becomes available as output of the World Wide Web ,
> unless by this time the user has generated on its output an **awareness scope retraction**.

> At any point in time,
> if at an earlier point in time the user has generated on its output an **awareness scope,** and
> since then the user has not generated on its output an **awareness scope retraction** referring to this **awareness scope**, and
> just before **new information** within this **awareness scope** becomes available as output of the World Wide Web ,
> then the user will receive on its input this **new information** within the **awareness scope** .

Based on these reformulation steps the following semi-formal structured requirement can be specified:

```
L0.R1.2   At any point in time,
          if
            at an earlier point in time
                 user output :                an awareness scope, and
            since then
                 not user output :            retraction of this awareness scope, and
            just before
                 World Wide Web output:       new information within this awareness scope
          then
                 user input:                  new information within this awareness scope
```

In summary, to obtain a structured semi-formal representation of a *requirement*, the following is to be performed:

- explicitly distinguish *input and output* concepts in the requirement formulation,
- define (domain) *ontologies* for the input and output information,
- make the *temporal structure* of the statement explicit using words like, 'at any point in time', 'at an earlier point in time', 'after', 'before', 'since', 'until', and 'next'.

For *scenarios*, a structured semi-formal representation is obtained by:

- explicitly distinguish *input and output* concepts in the scenario description,
- define (domain) *ontologies* for the input and output information,
- represent the temporal structure described implicitly in the sequence of events.

The interplay between requirements elicitation and analysis and scenario elicitation and analysis plays an important role. To be more specific, it is identified which requirements and scenarios relate to each other; for example, L0.R1.2 relates to L0.Sc1.2. If it is identified that for a requirement no related scenario is available yet (isolated requirement), then a new scenario can be acquired.

L0.Sc1.2

1.	user output:	**awareness scope**
2.	user is waiting	
3.	World Wide Web output:	**new information**
4.	user input:	**results for awareness scope**

2.3 Formal Representations

A formalisation of a scenario can be made by using formal ontologies for the input and output, and by formalising the sequence of events as a temporal trace. Thus a formal temporal model is obtained, for example as defined in [4] and [17]. Of course other formal languages can be chosen as well as long as they allow the formalisation of temporal dependencies that can occur within behavioural requirements without having to make further design choices first.

To obtain formal representations of requirements, the input and output ontologies have to be chosen as formal ontologies. The domain ontologies acquired during the reformulation process for the example application were formalised; part of the domain ontologies related to the focus on requirements and scenarios is shown below:

ontology element:	*explanation:*
SCOPE	a sort for the search scopes and awareness scopes
USER	a sort for the names of different users
PERSISTENCE_TYPE	a sort to distinguish between persistent and incidental scopes
INFO_ELEMENT	a sort for the result information
result_for_scope	a binary relation on INFO_ELEMENT and SCOPE
persistent, incidental	objects of sort PERSISTENCE_TYPE corresponding to the difference in persistence between an awareness scope and a search scope
input:	
is_interested_in	a ternary relation on USER, SCOPE and PERSISTENCE_TYPE
output:	
result_for_user	a ternary relation on INFO_ELEMENT, USER and SCOPE

In addition, the temporal structure, if present in a semi-formal representation, has to be expressed in a formal manner. Using the formal ontologies, and a formalisation of the temporal structure, a mathematical language is obtained to formulate formal requirement representations. The semantics are based on compositional information states which evolve over time. An *information state* M of a component D is an assignment of truth values {true, false, unknown} to the set of ground atoms that play a role within D. The compositional structure of D is reflected in the structure of the information state. A formal definition can be found in [4] and [17]. The set of all possible information states of D is denoted by IS(D). A *trace* $\mathcal{M}$ of a component D is a sequence of information states $(M^t)_{t \in \mathbf{N}}$ in IS(D). Given a trace $\mathcal{M}$ of component D, the information state of the input interface of component C at time point t of the component D is denoted by $\text{state}_D(\mathcal{M}, t, \text{input}(C))$, where C is either D or a sub-component of D. Analogously, $\text{state}_D(\mathcal{M}, t, \text{output}(C))$, denotes the information state of the output interface of component C at time point t of the component D. These formalised information states can be related to statements via the formally defined satisfaction relation |=. Behavioural properties can be formulated in a formal manner, using quantifiers over time and the usual logical connectives such as not, &, ⇒. An alternative formal representation of temporal properties (using modal and temporal operators) within Temporal Multi-Epistemic Logic can be found in [10].

Examples of formal representations of top level requirements:
L0.R1.2 is formalised by L0.R1.3: The first part of this requirement addresses the case that information relating to an awareness scope is already present, whereas the second part addresses the case that the information becomes available later.

L0.R1.3:
$\forall \mathcal{M}, t$
  [$\text{state}_S(\mathcal{M}, t, \text{output}(U))$ |= is_interested_in(U:USER, S:SCOPE, persistent) &
    $\text{state}_S(\mathcal{M}, t, \text{output}(WWW))$ |= result_for_scope(I:INFO_ELEMENT, S:SCOPE)]
  $\Rightarrow \exists t' > t$
    $\text{state}_S(\mathcal{M}, t', \text{input}(U))$ |= result_for_user(I:INFO_ELEMENT, U:USER, S:SCOPE)

&

$\forall \mathcal{M}, t1, t2 > t1$
    $\text{state}_S(\mathcal{M}, t1, \text{output}(U))$ |= is_interested_in(U:USER, S:SCOPE, persistent) &
    $\text{state}_S(\mathcal{M}, t2, \text{output}(WWW))$ |= result_for_scope(I:INFO_ELEMENT, S:SCOPE) &
    $\forall t'$ [$t1 < t' < t2$ $\Rightarrow$
    [not $\text{state}_S(\mathcal{M}, t', \text{output}(WWW))$ |= result_for_scope(I:INFO_ELEMENT, S:SCOPE) &
      not $\text{state}_S(\mathcal{M}, t', \text{output}(U))$ |= not is_interested_in(U:USER, S:SCOPE, persistent)]
  $\Rightarrow \exists t3 > t2$
    $\text{state}_S(\mathcal{M}, t3, \text{input}(U))$ |= result_for_user(I:INFO_ELEMENT, U:USER, S:SCOPE)

Example of a formal representation of a top level scenario
The following formal scenario representation 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.

L0.Sc1.3:
    $\text{state}_S(\mathcal{M}, 1, \text{output}(U))$ |= is_interested_in(U:USER, S:SCOPE, persistent)
    $\text{state}_S(\mathcal{M}, 3, \text{output}(WWW))$ |= result_for_scope(I:INFO_ELEMENT, S:SCOPE)
    $\text{state}_S(\mathcal{M}, 4, \text{input}(U))$ |= result_for_user(I:INFO_ELEMENT, U:USER, S:SCOPE)

To summarise, formalisation of a requirement or scenario on the basis of a structured semi-formal representation is achieved by:

- choosing *formal ontologies* for the input and output information
- formalisation of the *temporal structure* in a formal mathematical language

Checking a temporal formula F, which formally represents a requirement, against a temporal model $\mathcal{M}$, formally representing a scenario, means that formal verification of requirements against scenarios can be done by model checking. A formal representation $\mathcal{M}$ of a scenario S and a formal representation F of a requirement are compatible if the temporal formula is true in the model. For example, the temporal formula L0.R1.3 is indeed true in scenario L0.Sc1.3: the result was available in the world at time point 4 in the scenario (after the user generated the persistent interest on its output at time point 1), at time point 5 (which is later than 4) the user has the information on its input.

3 Requirements at Different Process Abstraction Levels

In this section three levels of abstraction are discussed: requirements for the system as a whole, requirements for an agent within the system, and requirements for components within an agent. Example requirements at different levels of process abstraction for the example domain are used as illustration.

3.1 Requirements for the Multi-agent System as a Whole

First, the requirements for the multi-agent system as a whole, including interaction with users are considered. The requirements and scenarios in the previous sections are formulated with respect to the users and the World Wide Web, which is considered as the given environment. Otherwise no assumptions were made on the design of the multi-agent system that is to support the users. For example, no specific agents were assumed as yet. The requirements and scenarios as presented in Section 2 express the desired behaviour from a global perspective, and only refer to input and output of users and the environment (the World Wide Web). By refining these requirements and scenarios, more elementary units of behaviour can be identified (*behavioural refinement*); which units of behaviour are chosen is a specific design decision. For example, it can be postulated that on the basis of specific user outputs concerning its interest, an unpersonalized scope of interest is identified:

L0.R2 For each **search scope** of a user, an unpersonal **incidental need for information** on the scope of the search scope is generated.

L0.R3
a. For each **awareness scope** of a user, an unpersonal **persistent need for information** on the scope of the awareness scope is generated.
b. For each **awareness scope retraction** of a user, an unpersonal **persistent need for information retraction** on the scope of the awareness scope is generated.

Available new information is to be presented to those users interested in that information:

L0.R4 If **new information** is available , then each user with an **awareness scope** that has not been retracted by that user and that matches that information will receive that information.

L0.R5 If there is a **persistent need for information** that has not been retracted and information becomes newly available on the World Wide Web that matches this persistent need, then this information is identified as **new information**.

Note that new ontology elements are created that need not be part of the ontologies of a user input or output and are not meant to be part of these ontologies. In relation with the refinements L0.R2 to L0.R5 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. Requirements L0.R2 and L0.R3 are imposed on the Personal Assistants, requirements L0.R4 and L0.R5 are imposed on the co-ordinated behaviour of both types of agents. The interfaces of the Personal Assistants and Information Providers will occur in semi-formal reformulations of the above requirements.

A global design of the multi-agent system is described in Fig. 2, in which two users, one Personal Assistant, two Information Providers, and the World Wide Web are depicted. The Personal Assistant has to co-operate with human agents (its users), and the Information Provider agents. The task of the Personal Assistant is to inform each of its users on information available (e.g., papers) on the World Wide Web that is within their scope of interest. Information on available papers is communicated to the Personal Assistant by Information Providers. The Personal Assistant is also capable itself of searching for information on the World Wide Web.

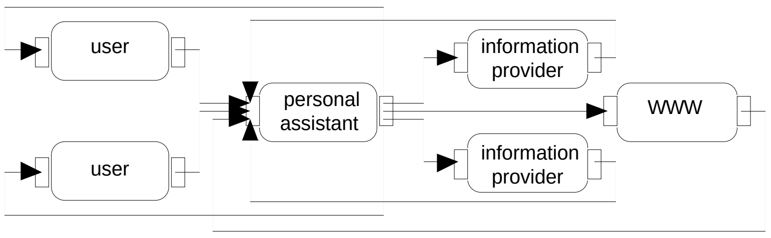

Fig. 2. Global multi-agent system description.

Given the design choice additional level L0 requirements can be formulated in terms of input and output of the agents and processes as identified:

L0.R6.2a /* Refinement of L0.R1.2 */
At any point in time
The user shall receive on its input **results for awareness scope**
after the user has generated on its output that **awareness scope** and
just after **new information** relevant for that awareness scope becomes available on the input of the Personal Assistant
and the Personal Assistant did not receive on its input an **awareness scope retraction** before the **new information** was received on the Personal Assistants input.

L0.R6.2b /* Refinement of L0.R1.2 */
At any point in time
The Personal Assistant will generate on its output **results for awareness scope** for a user after it has received on its input an **awareness scope** of that user and
just after **new information** relevant for that awareness scope becomes available on the output of the World Wide Web or on the output of one of the Information Providers
and the Personal Assistant did not receive on its input an **awareness scope retraction** before the **new information** was received on the Personal Assistants input.

L0.R2.2 At any point in time
If a user has generated on its output a **search scope** , then the Personal Assistant will generate on its output an unpersonal **incidental need for information** on the scope of the search scope.

L0.R3.2 At any point in time
a. If a user has generated on its output an **awareness scope**,
then the Personal Assistant will generate on its output an unpersonal **persistent need for information** on the scope of the awareness scope is generated.
b. If a user has generated on its output an **awareness scope retraction**,
and no other users have generated the same awareness scope without having it retracted, then the Personal Assistant will generate on its output an unpersonal **persistent need for information retraction** on the scope of the awareness scope is generated.

L0.R4.2 At any point in time
a. If the Personal Assistant receives **new information** on its input ,
then each user with an unretracted **awareness scope** matching that information will receive that information on its input.
b. If an Information Provider receives on its input information that it identifies as **new information**,
and the **new information** matches an unretracted **persistent need for information** that the Information Provider received on its input from the Personal Assistant,
then the Personal Assistant will receive this **new information** on its input.

L0.R5.2 At any point in time
If an Information Provider receives on its input a **persistent need for information** and later information becomes newly available on the World Wide Web that matches this persistent need, and this persistent need was not retracted before the new information became available,
then this information is identified as **new information** and generated on the output of the Information Provider.

In summary, for the current problem a design choice was made to design the system as a multi-agent system instead of as a single component system. The agents of the multi-agent system, the users, and the World Wide Web are all components of the system at the same level of process abstraction. For the top level, L0, the requirements pose constraints on more than one component of the multi-agent system. In other words the L0 requirements either specify the global purpose behaviour of the system (like L0.R1) or they specify the behaviour of the whole system in terms of relations between the behaviour of several of its components (L0.R2 to L0.R6). The latter are also requirements on the composition relation of the multi-agent system. Before the individual components can be designed, also requirements are to formulated on the behaviour of those individual agents and the component WWW.

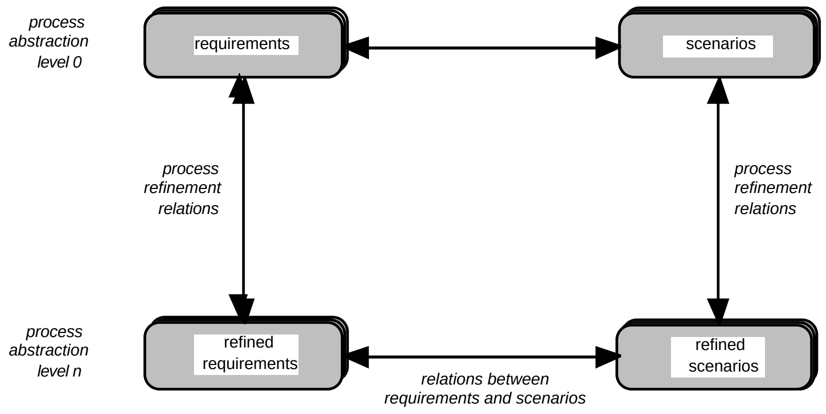
process abstraction level 0
requirements
scenarios
process refinement relations
process refinement relations
process abstraction level n
refined requirements
refined scenarios
relations between requirements and scenarios

Fig. 3. Process abstraction level refinements.

3.2 Requirements for an Agent within the System

Requirements formulated on the behaviour of those individual agents and the component WWW are requirements of the next process abstraction level (see Fig. 3), in this case level L1. Again these requirements can be formulated informally, semi-formally, or formally. For example, the following requirements can be imposed on the Personal Assistant agent (abbreviated as PA):

L1.P1 For each incoming **search scope** of a user, the PA shall initiate an incidental quest for useful information on that scope by generating on its output:
a. **communication** to Information Providers regarding an **incidental need for information** on the scope of the search scope.
b. **observation to be performed** on the World Wide Web regarding the scope of the search scope.

L1.P2 For each incoming **awareness scope** of a user, the PA shall initiate an persistent quest for useful information on that scope by :
a. only once generating on its output **communication** to Information Providers regarding a **persistent need for information** on the scope of the awareness scope.
b. repeatedly generating on its output an **observation to be performed** on the World Wide Web regarding the scope of the awareness scope,
until the PA receives on its input a corresponding **awareness scope retraction**.

L1.P3 At any point in time
If the PA receives **new information**on its input ,
then for each user with an **awareness scope** matching that information the PA will generate on its output an **awareness scope result** for that user,
unless the PA received a corresponding **awareness scope retraction** before it received the **new information**.

Similarly, requirements (or assumed requirements) can be formulated for the other components of the multi-agent system. Again it is possible to define more elementary units of behaviour by refining some of the requirements. Whether it is desirable to do so depends on whether or not the constructed requirements are considered sufficiently elementary to serve as a starting point for a transparent design of each of the components. For some of the components the required behaviour might still be too complex for this aim. In that case, the requirements of those components can be refined in terms of their behaviour. For example, a possible behavioural refinement of L1.P3 includes the following three requirements:

L1.P4 The PA maintains a profile of its users that satisfies the following:
a. contains the unretracted **awareness scopes** of that user.
b. An **awareness scope** is added to the profile if it is received on PA's input.
c. An **awareness scope** is removed from the profile if a corresponding **awareness scope retraction** is received on PA's input.

L1.P5 The PA is capable of matching **new information**received on its input with **awareness scopes** and determine the appropriateness of the information for the users that issued the **awareness scopes**.

L1.P6 The PA is capable of interpreting incoming communication information and generating communication information to other agents.

In relation to the behavioural refinements of requirements, a process refinement can be created if this is deemed appropriate. Furthermore, if a process refinement is deemed appropriate, then the choice still has to be made whether the current *process abstraction level is extended* with more processes (for example, whether the Personal Assistant agent is replaced by a number of agents at the same process abstraction level), or that an *additional process abstraction level* is created within one of the agents or processes (for example, composing the Personal Assistant of a number of sub-components). In this case, the latter alternative is chosen. For this choice a number of additional level L1 requirements have to be formulated to specify the behavioural properties of those sub-processes in their relation to each other (the behaviour of the sub-processes in relation to each other is global with respect to level L1), and a number of L2 requirements have to be formulated to specify the behavioural properties of the individual sub-processes (this behaviour is local with respect to level L2). Before considering the requirements of level L2, first the sub-components must be identified and their global level L1 behaviour analyzed.

Based on the requirements L1.P4 to L1.P6 a design decision is made to identify at least three sub-components for the Personal Assistant: a component to maintain the user profiles (called Maintenance of Agent Information), a component capable of matching information with awareness scopes (called Proposal Determination), and a component that handles communication with other agents (called Agent Interaction Management). Requirement L1.P4 is imposed on component Maintenance of Agent Information (abbreviated by MAI), requirement L1.P5 on component Proposal Determination (PD), and L1.P6 is imposed on component Agent Interaction Management (AIM). For the other requirements of the Personal Assistant similar behavioural refinements can be made, this results in the addition of a few other sub-components. The next level of process abstraction of the design of the Personal Assistant is described in Fig. 4. Apart from the already mentioned components, the Personal Assistant also consist of a component called World Interaction Management (abbreviated by WIM, used for observing the World Wide Web and interpreting the observation results), a component called Maintenance of World Information (abbreviated by MWI, used to maintain information on the World Wide Web), and a component called Own Process Control (abbreviated by OPC, used to determine the strategies, goals, and plans of the agent).

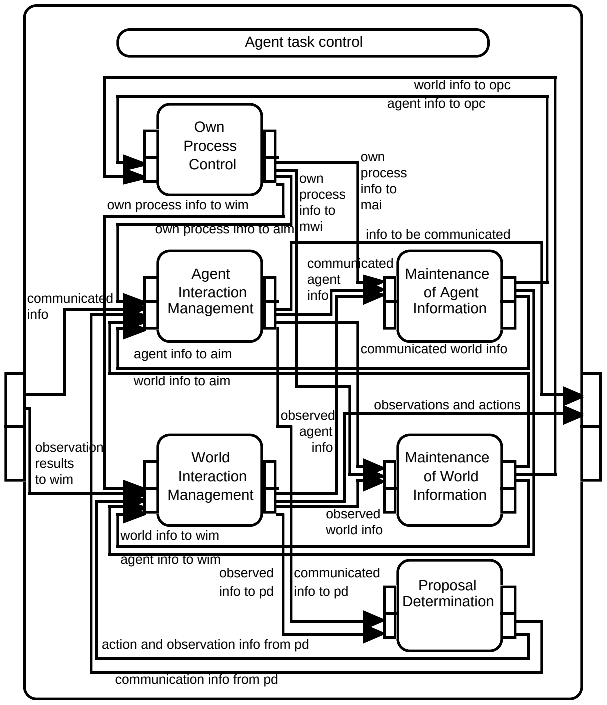

Fig. 4. Internal description of the Personal Assistant.

Given the above design choices, additional level L1 requirements can be formulated:

L1.P7 Incoming communication information received on the input of the PA is interpreted by AIM into **new information**, **awareness scopes**, **awareness scope retractions**, and **search scopes**.

a. AIM provides MAI with **awareness scopes** and **awareness scope retractions**.
b. AIM provides PD with **new information** and **search scopes**.
c. AIM provides MWI with **new information**.

L1.P8 PD provides AIM with **awareness scope results** and **search scope results** that are to be communicated to a user.

L1.P9 MAI provides PD with the current awareness scopes **and** search scopes**.**

3.3 Requirements for a Component within an Agent

The individual behavioural properties of the components distinguished at process abstraction level L1 are specified by requirements at abstraction level L2. A few examples of level L2 requirements are:

L2.PD1 PD matches the current **awareness scopes** with **new information** it receives; if a match is found, then PD generates a corresponding **awareness scope result**.

L2.AIM1 If AIM receives an **awareness scope result**, then AIM generates an **outgoing communication** containing that result for the relevant user.

4 Requirements Specification and Verification

In this section some of the methodological aspects are summarized and discussed. In particular, the relation between requirements specification and compositional verification is addressed.

4.1 Methodological Aspects

Within the proposed methodology, during a design process, in relation to behavioural refinements of requirements, a process refinement is created if this is deemed appropriate. A refinement of a behavioural requirement defines requirements on more elementary units of behaviour. Starting with behavioural requirements of the entire system, on the basis of requirement refinement it can be decided about the agents to use in the system, and in particular, which 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 refine the requirements for the agent further, depending on whether or not the requirements constructed for the agent are elementary enough to serve as a starting point for a transparent design of the agent itself. For some of the agents the required individual behaviour may still be too complex to lead to a transparent design. In that case, the behavioural requirements of those agents can be refined. This 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.

```
                    highest process abstraction level:
             required behavioural properties of entire system

                                / | \

                       next process abstraction level:
                       behavioural properties of agents

                       / | \    / | \    / | \

                       next process abstraction level:
               behavioural properties of components within agents

              / | \    / | \    / | \    / | \    / | \
                                  .
                             (and so on)
                                  .
```

Fig. 5. Behavioural properties at different process abstraction levels

This requirements refinement process can be iterated for some of the agent components, which, depending on the complexity of the agent, may lead to an arbitrary number of process abstraction levels within the agent (see Fig. 5). The result of this process is the identication of different process abstraction levels, from the level of the entire system, to the level of each of the agents separately, and further down to more specific processes within agents.

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.

4.2 Relation to Compositional Verification

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. [17]). 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 (cf. [17]).

Compositional verification as described in [17] takes the process abstraction levels used within the system and the related compositional 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. 5).

Primitive components (those components that are not composed of others) can be verified using more traditional verification methods. 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. In [16] a requirements engineering process model is described that can be used to support the requirements engineering process. Note that the idea of refinement is well-known in the area of (sequential) programs, e.g., [6]. The method of compositional requirements specification proposed here exploits a similar idea in the context of behavioural requirements.

5 Discussion

Requirements and scenarios describe the required properties of a system (this includes the functions of the system, structure of the system, static properties, and dynamic properties). In applications of agent systems, the dynamics or behaviour of the system plays an important role in description of the successful operation of the system. Requirements and scenarios specification has both to be informal or semi-formal (to be able to discuss them with stakeholders) and formal (to disambiguate and analyse them and establish whether or not a constructed model for a multi-agent system satisfies them).

As requirements and scenarios form the basis for communication among stakeholders (including the system developers), it is important to maintain a document in which the requirements and scenarios are organised and structured in a comprehensive way. This document is also important for maintenance of the system once it has been taken into operation. The different activities in requirements engineering lead to an often large number of inter-related representations of requirements and scenarios. The explicit representation of these *traceability relations* is useful in keeping track of the connections; traceability relationships can be made explicit (see Figs. 1 and 3): *reformulation* relations among *requirements* (resp., *scenarios*) at the same process abstraction level, *behavioural refinement* relations between *requirements* (resp., *scenarios*) at different process abstraction levels, and *satisfaction* relations between requirements and scenarios. For the case-study, these relationships have been specified using hyperlinks. This offers traceability; i.e., relating relevant requirements and scenarios as well as the possibility to 'jump' to definitions of relevant requirements and scenarios.

In summary, the main lessons learned from our case studies are:

- Integration of requirements and scenarios acquisition with *ontology acquisition* supports the conceptual and detailed design of the input and output interfaces of agents and agent components
- Refinement of requirements and scenarios in terms of requirements and scenarios on more elementary behaviours at the top level supports the identification of agents and the *allocation of behaviours to agents*
- Refinement of requirements and scenarios in terms of requirements and scenarios on more elementary behaviours within an agent supports the identification of agent components and the *allocation of functionality to agent components*
- A compositional approach to requirements and scenarios specification provides a basis for the *design rationale*: a documentation of the design choices made and the reasons for them
- A compositional approach to requirements and scenarios specification as an integral part of the multi-agent system design process strongly facilitates *compositional verification* of the designed system.

- The process of achieving an understanding of a requirement involves a large number of *different formulations and representations*, gradually evolving from informal to semi-formal and formal.
- *Scenarios* and their formalisation are, compared to requirements, of equal importance.
- Keeping track on the various relations between different representations of requirements, between requirements and scenarios, and many others, is supported by *hyperlink specifications* within a requirements document.

As a result of these explorative studies it is proposed that a principled design method for multi-agent systems should include two specification languages:

- a language to specify *design descriptions*
- a language to specify (behavioural) *requirements* and *scenarios*

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 [13], [14] 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 communication with stakeholders.

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. This language 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 as proposed in this paper in the compositional development method for multi-agent systems, DESIRE and, in particular, in its software environment is addressed.

The requirements and scenarios for agent systems often have to address complex behavioural properties. In comparison, it is not clear how, for example, in UML more complex behavioural requirements can be specified; use cases are informal, and activity diagrams seem too design specific and cannot express the necessary more complex temporal dependencies relevant to both requirements and scenarios. In the development of UML different representations of behavioural requirements and scenarios was not an issue [12], [15]. In [7], [8] an approach is presented in which more complex behavioural properties can be specified. A difference to our work is that no compositionality is exploited in requirements specification and verification. In recent research in knowledge engineering, identification and formalisation of properties of knowledge-intensive systems is addressed, usually in the context of verification or competence assessment of complex tasks; e.g., [1]. Such properties can be used as a basis for requirement specifications, and may play a role within specific agent compomenents.

References

1. Benjamins, R., Fensel, D., and Straatman, R., Assumptions of problem-solving methods and their role in knowledge engineering. In: W. Wahlster (Ed.), Proceedings of the Twelfth European Conference on Artificial Intelligence, ECAI'96, John Wiley and Sons, 1996, pp. 408-412.
2. 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.
3. 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.
4. Brazier, F.M.T., Treur, J., Wijngaards, N.J.E. and Willems, M., Temporal semantics of compositional task models and problem solving methods. Data and Knowledge Engineering, vol. 29(1), 1999, pp. 17-42.
5. Davis, A. M., Software requirements: Objects, Functions, and States, Prentice Hall, New Jersey, 1993.
6. Dijkstra, E.W., A discipline of programming. Prentice Hall, 1976.
7. Dubois, E. (1998). ALBERT: a Formal Language and its supporting Tools for Requirements Engineering.
8. Dubois, E., Du Bois, P., and Zeippen, J.M., A Formal Requirements Engineering Method for Real-Time, Concvurrent, and Distributed Systems. In: Proceedings of the Real-Time Systems Conference, RTS’95, 1995.
9. Dubois, E., Yu, E., Petit, M., From Early to Late Formal Requirements. In: Proceedings IWSSD’98. IEEE Computer Society Press, 1998.
10. 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.), Pre-proc. of the Fifth International Workshop on Agent Theories, Architectures and Languages, ATAL'98, 1998, pp. 91-106. To appear in: J.P. Mueller, M.P. Singh, A.S. Rao (eds.), Intelligent Agents V. Lecture Notes in AI, Springer Verlag. In press, 1999
11. 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.
12. Eriksson, H. E., and Penker, M., UML Toolkit. Wiley Computer Publishing, John Wiley and Sons, Inc., New York, 1998.
13. Gero, J.S., and Sudweeks, F., (eds.), Artificial Intelligence in Design ’96, Kluwer Academic Publishers, Dordrecht, 1996.
14. Gero, J.S., and Sudweeks, F., (eds.), Artificial Intelligence in Design ’98, Kluwer Academic Publishers, Dordrecht, 1998.

15. Harmon, P., and Watson, M., Understanding UML, the Developer’s Guide. Morgan Kaufmann Publishers, San Francisco, 1998.
16. Herlea, D.E., Jonker, C.M., Treur, J., and Wijngaards, N.J.E. A Formal Knowledge Level Process Model of Requirements Engineering. In: Proceedings of the 12th International Conference on Industrial and Engineering Applications of AI and Expert Systems, IEA/AIE'99. Lecture Notes in AI, Springer Verlag, 1999, To appear.
17. 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
18. Kontonya, G., and Sommerville, I., Requirements Engineering: Processes and Techniques. John Wiley and Sons, New York, 1998.
19. Musen, M., Ontology Oriented Design and Programming: a New Kind of OO. 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. 17-20.
20. Nwana, H.S., and Ndumu, D.T., A Brief Introduction to Software Agent Technology. In Jennings, N.R., and Wooldridge, M. (eds.), Agent Technology: Foundations, Applications, and Markets. Springer Verlag, Berlin, 1998, pp. 29-47.
21. Sommerville, I., and Sawyer P., Requirements Engineering: a good practice guide. John Wiley & Sons, Chicester, England, 1997.
22. Weidenhaupt, K., Pohl, M., Jarke, M. and Haumer, P., Scenarios in system development: current practice, in IEEE Software, pp. 34-45, March/April, 1998.
23. 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.