Integration of Behavioural Requirements Specification within Knowledge Engineering

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, niek}@cpsc.ucalgary.ca

[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. It is shown how specification of behavioural requirements from informal to formal can be integrated within knowledge engineering. The integration of requirements specification has addressed, in particular: the integration of requirements acquisition and specification with ontology acquisition and specification, the relations between requirements specifications and specifications of task models and problem solving methods, and the relation of requirements specification to verification.

1 Introduction

Requirements Engineering (RE) addresses the development and validation of methods for eliciting, representing, analysing, and confirming system requirements and with methods for transforming requirements into more formal specifications for design and implementation. Requirements Engineering is one of the early but important phases in the software development life cycle and numerous studies have revealed the misidentification of requirements as one of the most significant sources of customer dissatisfaction with delivered systems [10], [22], [28]. However, it is a difficult process, as it involves the elicitation, analysis and documentation of knowledge from multiple stakeholders of the system. There is an increased need to involve the users at this stage of the development life-cycle [8], 29]. It is recognised that the users are the experts in their work and a thorough understanding of the requirements is achieved only by promoting effective communication with them during the requirements engineering process [3]. It is also argued that an effective requirements definition requires involvement and mutual control of the process by all players, and that a good partnership between users and designers enables a high quality of the system being developed [19].

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. The process of requirements engineering within software development is an iterative process, in which a sharp borderline between defining requirements and constructing the system design is not always easy to draw. When an effective stakeholder-developer communication link is in place, on the basis of a (partially) constructed design description of the system, additional information may be elicited from the stakeholders (i.e., domain experts, users, system customers, managers), and more detailed requirements and scenarios can be developed which refer to this design description. Requirements 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.

The interleaving of the process of requirements engineering and the process of design is emphasised in current research in the area of AI & Design (e.g., [16], [17]), in which it is put forward that realistic design processes include both the manipulation of requirement specifications and the manipulation of design object specifications, resulting in a detailed description of a design object and a good understanding of the requirements. This perspective on design, applied in particular to the design of knowledge-intensive software, is employed throughout the paper. This is in contrast with the tradition in software engineering to separate the activity of manipulating software requirements from the 'design of software', the actual construction of the system design [4], [20], [25], [26].

Principled model-based methodologies for knowledge engineering, such as DESIRE (cf. [6], [7]), CommonKADS (cf. [27]) or MIKE (cf. [1]), the emphasis is on specification of the (conceptual) model of the system being developed and not on specification of required behaviour properties of a system to be developed. A transparent distinction between specification of the structure of a system (or task or problem solving method) and its (behavioural) properties is not made. For example, in the AI and Design community a specification of the *structure* of a design object is often distinguished from a specification of *function* or *behaviour*; e.g., [16], [17]. 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 [2], [9], [14], [15]. Such properties can be used as a basis for requirement specifications. In this paper it is shown how specification of behavioural requirements from informal to formal can be integrated within knowledge engineering.

From the basic ingredients in knowledge engineering methodologies the following are especially relevant to the integration of requirements specification: knowledge level approaches to *problem solving methods* (e.g., [14]), *ontologies* (e.g., [23]) and *verification* (e.g., [9]). It has to be defined how requirements specification relates to these basic ingredients. Therefore, integration of requirements specification within a principled knowledge engineering methodology has to address, in particular:

- integration of requirements acquisition and specification with ontology acquisition and specification
- relations between requirements specifications and specifications of task models with tasks at different levels of (process) abstraction, or problem solving methods
- relation of requirements specification to verification

These aspects are addressed in this paper. The different forms of representation of requirements and scenarios are presented in Section 2, for reasons of presentation illustrated by a simple example. In Section 3 refinement of requirements related to different proces abstraction levels (e.g., as in task or task/method hierarchies) is addressed. Section 4 briefly summarizes the relations between requirements and scenarios. Section 5 concludes the paper with a discussion.

2 Representation of Requirements and Scenarios

In the approach presented in this paper, the processes of requirements engineering and system development are integrated by a careful specification of the co-operation between the two. The manipulation process of a set of requirements and scenarios, and the manipulation process of a design object description (i.e., a description of the system) are intertwined in the following way: first the set of requirements and scenarios is made as precise as possible. This requires multiple interaction with and among the stakeholders. Based on that set a possible (partial) description is made of the system. The description of the system is used not only to validate the understanding of the current set of requirements and scenarios, but also to elicit additional information from the stakeholders. This leads to more requirements and scenarios and to more detailed requirements and scenarios. The process continues, alternating between manipulating a set of requirements and scenarios, and manipulating a description of a system. Adequate representations of requirements and scenarios are required for each part of the overall process, and, therefore, the relations between the different representation forms of the same requirement or scenario need to be carefully documented.

One of the underlying assumptions on the approach presented in this paper is that a compositional design method will lead to designs that are transparent, maintainable, and can be (partially) reused within other designs. The construction of a compositional design description of the system that properly respects the requirements and scenarios entails making choices between possible solutions and possible system configurations. Such choices can be made during the manipulation of the set of requirements and scenarios, but also during the manipulation of the design object description. Each choice corresponds to an abstraction level. For each component of the system design further requirements and scenarios are necessary to ensure that the combined system satisfies the overall system requirements and scenarios. The different abstraction levels in requirements are reflected as levels of process abstraction in the design description during the manipulation of the compositional design description.

Different representations of requirements and scenarios are discussed in Sections 2.1 to 2.3. The use of process abstraction levels is explained further in Section 3. An overview of the relations between representations of requirements and scenarios, and different levels of process abstraction is presented in Section 4.

In Requirements Engineering the role of scenarios, in addition to requirements, has gained more importance, both in academia and industry practice [13], [30]. Scenarios or use cases are examples of interaction sessions between the users and the system [24], [30]; they are often used during the requirement engineering, being regarded as effective ways of communicating with the stakeholders (i.e., domain experts, users, system customers, managers, and developers). The initial scenarios can serve to verify (i.e., check the validity in a formal manner) the requirements specification and (later) the system prototypes. Evaluating the prototypes helps detecting misunderstandings between the domain experts and system designers if, for example, the system designers made the wrong abstractions based on the initial scenarios. In our approach requirements and scenarios both are explicitly represented, and play a role of equal importance. Having them both 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.

To enable effective ways of communicating with the stakeholders, requirements and scenarios are to be represented in a well-structured and easy to understand manner and precise 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 [25]. Informally represented requirements and scenarios are often best understood by the stakeholders (although also approaches exist using formal representations of requirements in early stages as well [11]). 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, for example in the context of verification and the detection of inconsistencies. Furthermore, the process of formalising the representations contributes to disambiguation of requirements and scenarios (in contact with stakeholders). 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 and relations between them are used: informal or structured semi-formal representations (obtained during the process of formalisation) in contact with stakeholders and designers of the system, and related 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. Examples are given below. When manipulating requirements and scenarios, different activities can be distinguished (see Fig. 1):

- requirements and scenarios are elicited from stakeholders, checked for ambiguities and inconsistencies, reformulated in a more precise or more structured form, and represented in different forms (informal, semi-formal, and formal) to suit different purposes (communication with stakeholders, verification of a design description)
- they are refined across process abstraction levels (which is addressed in Section 3).

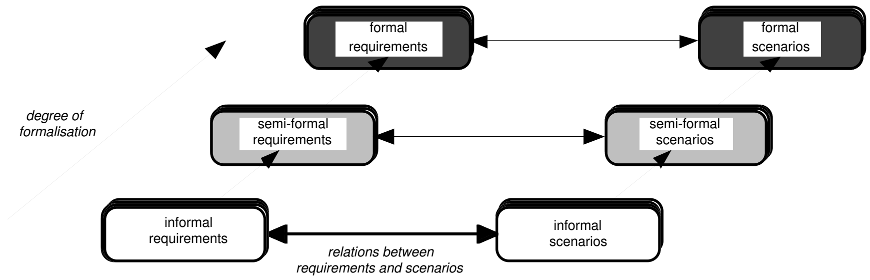


Fig. 1. Representations from informal to formal

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. 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. A simple example of a requirement R1 on a system to control a chemical process is the following:

Requirement R1
For situations that the temperature and pressure are high the system shall give a red alert and turn the heater off.

A requirement is a *general* statement about the (required) behaviour of the system to be designed. This statement is required to hold for *every* instance of behaviour of the system. In contrast to this, a scenario is a description of a behaviour instance (e.g., to be read as an instance of a system trace the system has to show, given the user behaviour in the scenario). An example of an informal representation of a scenario is:

Scenario S1
The temperature and pressure are high.
A red alert is generated and the heater is turned off.

Note that this scenario describes one of the behaviour instances for which requirement R1 holds.

2.2 Structured semi-formal representations

Both requirements and scenarios can be reformulated to more structured and precise forms.

Requirements. To check requirements for ambiguities and inconsistencies, an analysis that seeks to identify the parts of a given requirement formulation that refer to the input and output of the system 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 the output of the system. 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 and/or scenarios. For example, the following requirement may be found as a result of such an analysis:

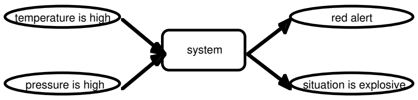
> *Requirement R1.1:*
> *at any point in time*
> *if the system received input that the temperature is high and the pressure is high*
> *then the system shall generate as output a red alert and an indication that the situation is explosive, and after the user gives an input that it has to be resolved, the system gives output that the heater is turned off*

A reformulation can lead to structured requirements in a semi-formal form that provide more detail, for example R1 can be reformulated to R1.1, but also to two parts:

> *Requirement R1a.1:*
> *at any point in time*
> *if the system received **input** that the temperature is high and the pressure is high*
> *then the system shall generate as **output** a red alert and an indication that the situation is explosive*

> *Requirement R1b.1:*
> *at any point in time*
> *if the system provided as **output** an indication that the situation is explosive and after this the user gave an **input** that it has to be resolved,*
> *then the system shall generate **output** that the heater is turned off*

Requirement R1a.1 can also be represented graphically, for example, by (here each of the pairs of arrows means that both arrows of the pair occur at the same time):



As a specific case, also requirements referring only to input or only to output can be encountered. For requirements formulated in such a structured manner the following classification can be made:

- requirements on input only, independent of output (*input requirements*),
- requirements on output only, independent of input (*output requirements*), and
- requirements relating output to input

The latter type of requirements can be categorised as:

- output is dependent on input (input-output-dependency): *function or behaviour requirement*,
- input is dependent on output (output-input-dependency): *environmental requirement or assumption*

When stating properties of the environment (which includes users) of the system (output-input-dependency), the term 'requirement' is avoided and the term 'assumption' is used: the environment is not within the scope of the software development; it cannot be 'tuned' to exhibit particular properties. As such, only assumptions can be made on its behaviour and properties. The term 'requirements' is used for those parts of the system that are within the scope of designable parts of the system.

In addition, requirements can be categorised according to the kind of properties they refer to: static requirements, or requirements. For nontrivial dynamic 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', 'next' are used to clarify the temporal relationships between different fragments in the requirement.

The input and output terms used in the structured reformulations form the basis of an ontology of input and output concepts. Construction of this ontology takes place during the reformulation of requirements: acquisition of a (domain or task or method) ontology is integrated within requirements engineering (requirements engineering contributes at least to part of the ontology acquisition). For the requirements engineering process it is very useful to construct an ontology of input and output concepts. For example, in R1b.1 the concepts indicated below in bold can be acquired.

Requirement R1b.1:
at any point in time
if the system provided as output an indication that the ***situation is explosive****,*
and after this the user gave an input that it has ***to be resolved****,*
then the system shall generate output that the ***heater is turned off***

This ontology later facilitates the formalisation of requirements and scenarios, as the input and output concepts are already defined.

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 and task/method) *ontologies* for input and output information

- *classify* the requirement according to the categories above
- 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’, ‘next’.

Scenarios. For scenarios, a structured semi-formal representation is obtained by performing the following:

- 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 scenario S1 shown earlier is reformulated into a structured semi-formal representation S1.1:

Scenario S1.1
- input: *temperature is high, pressure is high*
- output: *red alert, situation is explosive*
- input: *to be resolved*
- output: *heater is turned off*

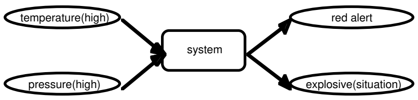
Notice that from this scenario, which covers both requirements given above, it is not clear whether or not always an input *to be resolved* leads to the heater being turned off, independent of what preceded this input, or whether this should only happen when the history actually was as described in the first two lines of the scenario. If the second part of the scenario is meant to be history independent, this second part is better specified as a separate scenario. However, we assume that in this example at least the previous output of the system *situation is explosive* on which the user reacts is a condition for the second part of the scenario (as also expressed in the requirements above). These considerations lead to the splitting of scenario S1.1 into the following two (temporally) independent scenarios S1a.1 and S1b.1:

Scenario S1a.1
- input: *temperature is high, pressure is high*
- output: *red alert, situation is explosive*

Scenario S1b.1
- output: *situation is explosive*
- input: *to be resolved*
- output: *heater is turned off*

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 [7] and [9]. To obtain formal representations of requirements, the input and output ontologies have to be chosen as formal ontologies. In the example this can be done, for example by formalising a conceptual relation of the form A is B, with as meaning that the object A has property B, in a predicate form: B(A); for example ‘the situation is explosive’ is formalised by explosive(situation), where situation is an object and explosive a predicate. This format can be used within an appropriate subset or extension of predicate logic. For example, requirement R1a.1 can also be represented formally in combined symbolic and graphical form by the following:



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. 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 $\mathrm{state}_D(\mathcal{M}, t, \mathrm{input}(C))$, where C is either D or a sub-component of D. Analogously, $\mathrm{state}_D(\mathcal{M}, t, \mathrm{output}(C))$, 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 $\models$. Behavioural properties can be formulated in a formal manner, using quantfiers over time and the usual logical connectives such as not, &, $\Rightarrow$. An alternative formal representation of temporal properties (using modal and temporal operators) within Temporal Multi-Epistemic Logic can be found in [12]. For example, requirement R1b.1 can be represented formally by:

Requirement R1b.2:

$$\begin{array}{lll}
\forall \mathcal{M}, t & [\ \mathrm{state}_S(\mathcal{M}, t, \mathrm{input}(S)) \models \mathrm{to_be_resolved}\ \ \& \\
\quad \exists t' < t & \mathrm{state}_S(\mathcal{M}, t', \mathrm{output}(S)) \models \mathrm{explosive(situation)} \Rightarrow \\
\quad \exists t'' > t & \mathrm{state}_S(\mathcal{M}, t, \mathrm{output}(S)) \models \mathrm{turn_off(heater)}\ ]
\end{array}$$

In this formalisation of R1b.1 the word “after” is represented by indicating that the time point t at which to_be_resolved appeared on the input is greater than some time point t’ at which the system reported that the situation is explosive on its output.

Scenario S1.1 can be represented formally by the temporal model that is defined as follows:

Scenario S1.2:

$\text{state}_S(\mathcal{M}, 1, \text{input}(S))$	$\models$	high(temperature)
$\text{state}_S(\mathcal{M}, 1, \text{input}(S))$	$\models$	high(pressure)
$\text{state}_S(\mathcal{M}, 2, \text{output}(S))$	$\models$	explosive(situation)
$\text{state}_S(\mathcal{M}, 2, \text{output}(S))$	$\models$	red_alert
$\text{state}_S(\mathcal{M}, 3, \text{input}(S))$	$\models$	to_be_resolved
$\text{state}_S(\mathcal{M}, 4, \text{output}(S))$	$\models$	turn_off(heater)

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*

This results in a temporal formula F for a requirement and in a temporal model $\mathcal{M}$ for a scenario.

Checking a temporal formula, which formally represents a requirement, against a temporal model, 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 R1b.2 is indeed true for the model S1.2: the explosive situation occurred at time point 2 in the scenario, at time point 3 (which is later than 2) the system received input to_be_resolved, and at time point 4 (again later than 3), the system has as output turn_off(heater).

However, requirement R1b.2 would also be true in the following two scenarios. Scenario S2 is an example of a situation in which the system turns off the heater when this is not appropriate, scenario S3 is an example of a situation in which the system waits too long before it turns off the heater (which might lead to an explosion).

Scenario S2
The temperature and the pressure are high
The system generates a red alert and turns off the heater,
The temperature and the pressure are medium
The temperature is low and the pressure is medium
The system turns off the heater

Scenario S3
The temperature and the pressure are high
The system generates a red alert and turns off the heater,
The system increases the heater
The system increases the heater
An explosion occurs
The system turns off the heater

Furthermore, the requirement would also be true in a scenario in which the system waited with turning off the heater, maybe even first increasing the heat for some time. This last scenario is formalised as scenario S3.1:

Scenario S3.1:

$\text{state}_S(\mathcal{M}, 1, \text{input}(S))$	$\models$	high(temperature)
$\text{state}_S(\mathcal{M}, 1, \text{input}(S))$	$\models$	high(pressure)
$\text{state}_S(\mathcal{M}, 2, \text{output}(S))$	$\models$	explosive(situation)
$\text{state}_S(\mathcal{M}, 2, \text{output}(S))$	$\models$	red_alert
$\text{state}_S(\mathcal{M}, 3, \text{input}(S))$	$\models$	to_be_resolved
$\text{state}_S(\mathcal{M}, 4, \text{output}(S))$	$\models$	increase(heater)
$\text{state}_S(\mathcal{M}, 5, \text{output}(S))$	$\models$	increase(heater)
$\text{state}_S(\mathcal{M}, 6, \text{input}(S))$	$\models$	occurred(explosion)
$\text{state}_S(\mathcal{M}, 7, \text{output}(S))$	$\models$	turn_off(heater)

In other words, requirement R1b.2 leaves too many possibilities for the system's behaviour, and, being a formalisation of R1b.1, so do the requirements that form the reason for formulating R1b.1, i.e., R1a.1, and R1.1. During the requirement engineering process this has to be resolved in contact with the stakeholders. In this case, the semi-formal R1.1 and R1a.1, and the formal R1a.2 have to be reformulated: after a discussion with the stakeholders, R1.1 is reformulated into:

Requirement R1.2:
at any point in time
if the system received input that the temperature is high and the pressure is high
then at the next point in time the system shall generate as output a red alert and an indication that the situation is explosive, and at the next point in time after the user gives an input that it has to be resolved, the system gives output that the heater is turned off

Requirement R1b.1 is reformulated into:

Requirement R1b.3:
at any point in time
if the system provided as output
an indication that the ***situation is explosive****,*
and at the next time point after the user gave an input
that the situation has ***to be resolved****,*
then the system shall generate output
that the ***heater is turned off***

Based on these reformulations (that also affect the ontologies), the requirement engineers choose a different representation of R1b.2:

Requirement R1b.3:

$$\forall \mathcal{M}, t \quad [\quad \text{state}_S(\mathcal{M}, t, \text{input}(S)) \models \text{to\_be\_resolved}(\text{situation}) \;\& $$
$$\text{state}_S(\mathcal{M}, \text{prev}(t), \text{output}(S)) \models \text{explosive}(\text{situation}) \;\Rightarrow$$
$$\text{state}_S(\mathcal{M}, \text{succ}(t), \text{output}(S)) \models \text{turn\_off}(\text{heater}) \;]$$

Requirement R1b.3 is true in scenario S1.2 (let prev be the function: n -> n-1 and succ: n -> n+1), but not in the sketched unwanted scenarios like S3.1.

3 Requirements Refinement and Process Abstraction Levels

The requirements engineering process considers the system as a whole, in interaction with its stakeholders. However, during a design process, often a form of structuring of the system is used: sub-processes are distinguished, for example in relation to development or selection of a task or task/method hierarchy. For the processes at the next lower process abstraction level, also requirements can be expressed. Thus a distinction is made between *stakeholder requirements* and *stakeholder scenarios* (for the top level of the system, elicited from stakeholders, such as users, customers) and *designer requirements* and *designer scenarios* (for the lower process abstraction levels, constructed by requirement engineers and designers). Designer requirements and scenarios are dependent on a description of the system. Requirements on properties of a sub-component of a system reside at a next lower level of process abstraction than the level of requirements on properties of the system itself; often sets of requirements at a lower level are chosen in such a way that they realise a next higher level requirement. This defines a process abstraction level refinement relation between requirements. These process abstraction refinement relationships can also be used to validate requirements: e.g., if the refinements of a requirement to the next lower process abstraction level all hold for a given system description, then the refined 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. In the example used above, for the structured semi-formal requirements two processes can be distinguished:

interpret process info
input information of type: temperature is high, pressure is high
output information of type: situation is explosive

generate actions
input information of type: situation is explosive
output information of type: red alert, heater is turned off

At the next lower abstraction level of these two processes the following requirements can be formulated, as a refinement of the requirements given earlier:

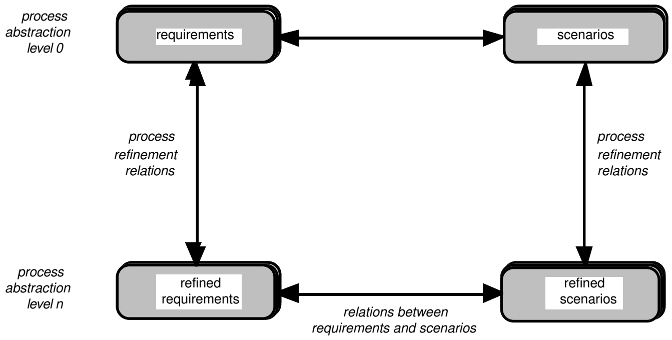


Fig. 2. Process abstraction level refinements

interpret process info

Requirement R1int.1:
at any point in time
*if the component received **input** that the temperature is high and the pressure is high*
*then the component shall generate as **output** an indication that the situation is explosive*

generate actions

Requirement R1acta.1:
at any point in time
*if the component received **input** that the situation is explosive ,*
*then the component shall generate as **output** a red alert*

Requirement R1actb.1:
at any point in time
*if the component received **input** that the situation is explosive,*
*and after this it received an **input** that it has to be resolved,*
*then the component shall generate **output** that the heater is turned off*

Furthermore, scenarios S1a.1 and S1b.1 given earlier can be refined to

Scenario S1inta.1
- system input: *temperature is high, pressure is high*
    - interpret process info input: *temperature is high, pressure is high*
    - interpret process info output: *situation is explosive*
    - generate actions input: *situation is explosive*
    - generate actions output: *red alert*
- system output: *situation is explosive, red alert*

Scenario S1intb.1

- system output: *situation is explosive*
- system input: *to be resolved*
    - generate actions input: *to be resolved*
    - generate actions output: *heater is turned off*
- system output: *heater is turned off*

4 Traceability Relations for Requirements and Scenarios

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. Due to the increase in system complexity nowadays, more complex requirements and scenarios result in documents that are more and more difficult to manage. The different activities in requirements engineering lead to an often large number of inter-related representations of requirements and scenarios.

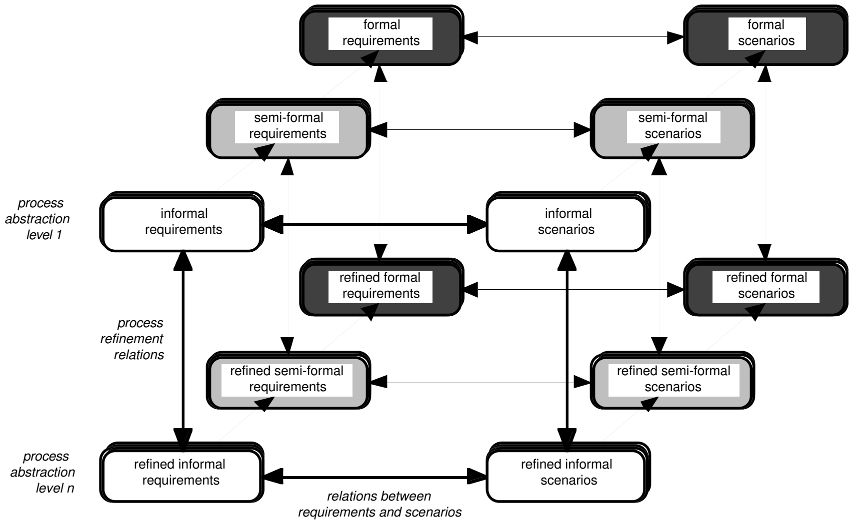


Fig. 3. Traceability relations

The explicit representation of these *traceability relations* is useful in keeping track of the connections; traceability relationships can be made explicit:

- among requirements at the same process abstraction level (Fig. 1),
- between requirements at different process abstraction levels (Fig. 2),

- among scenarios at the same process abstraction level (Fig. 1),
- between scenarios at different process abstraction levels (Fig. 2),
- between requirements and scenarios at the same process abstraction level (Figs 1, 2 and 3)
- among requirements at the same level of formality (Fig. 3)
- between requirements and scenarios at the same level of formality (Fig. 3).

These relationships are often adequately 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. Thus requirements and scenarios resulting from an extensive case-study have been placed in a hyperlinked structure [18]; see Fig. 3, which combines Figures 1 and 2.

5 Discussion

Requirements 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 to agent-based systems, the dynamics or behaviour of the system plays an important role in description of the successful operation of the system. Requirements 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 system satisfies them). Typical software requirements engineering practices are geared toward the development of a formal requirements specification.

The process of making requirements more precise is supported by using both semi-formal and formal representations for requirements. Part of this process is to relate concepts used in requirements to input and output of the system. Since requirement specifications need system-related concepts, it has been shown how the acquisition and specification of requirements goes hand in hand with the acquisition and specification of *ontologies*.

The formalisation of behaviour requirements has to address the semantics of the evolution of the system (input and output) states over time. In this paper the semantics of properties of compositional systems is based on the temporal semantics approach, which can be found in the development of a compositional verification method for knowledge-intensive systems; for diagnostic process models see [9]; for co-operative information gathering agents, see [21]; for negotiating agents, see [5]. By adopting the semantical approach underlying the compositional verification method, a direct integration of requirements engineering with the specification of properties of *problem solving methods* and their *verification* could easily be established.

For some example systems requirements and scenarios have been elicited, analysed, manipulated, and formalised. The lessons learned from these case studies are:

- 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.

- Categorisation of requirements on input, output and function or behaviour requirements, and distinguishing these from assumptions on the environment clarifies the overall picture.
- 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.

In current and future research, further integration of requirements engineering in the compositional development method for multi-agent systems, DESIRE and, in particular, in its software environment is addressed.

References

1. Angele, J., Fensel, D., Landes, D., and Studer, R., Developing Knowledge-based Systems with MIKE. *Journal of Automated Software Engineering*, 1998
2. Benjamins, R., Fensel, D., Straatman, R. (1996). Assumptions of problem-solving methods and their role in knowledge engineering. In: W. Wahlster (Ed.), *Proceedings of the12th European Conference on AI, ECAI'96*, John Wiley and Sons, pp. 408-412.
3. Beyer, H.R. and Holtzblatt, K. (1995). Apprenticing with the customer, *Communications of the ACM*, vol. 38(5), pp. 45-52.
4. Booch, G. (1991). *Object oriented design with applications*. Benjamins Cummins Publishing Company, Redwood City.
5. Brazier, F.M.T., Cornelissen, F., Gustavsson, R., Jonker, C.M., Lindeberg, O., Polak, B., and Treur, J. (1998). Compositional Design and Verification of a Multi-Agent System for One-to-Many Negotiation. In: *Proceedings of the Third International Conference on Multi-Agent Systems, ICMAS'98*. IEEE Computer Society Press, pp. 49-56.
6. Brazier, F.M.T., Jonker, C.M., and Treur, J. (1998). 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*, pp. 347-360.
7. Brazier, F.M.T., Treur, J., Wijngaards, N.J.E. and Willems, M. (1999). Temporal Semantics of Compositional Task Models and Problem Solving Methods. *Data and Knowledge Engineering*, vol. 29(1), 1999, pp. 17-42.
8. Clavadetscher, C. (1998). User involvement: key to success, *IEEE Software,* Requirements Engineering issue, March/April, pp. 30-33.
9. Cornelissen, F., Jonker, C.M., and Treur, J. (1997). Compositional verification of knowledge-based systems: a case study in diagnostic reasoning. In: E. Plaza, R. Benjamins (eds.), *Knowledge Acquisition, Modelling and Management, Proceedings of the 10th European Knowledge Acquisition Workshop, EKAW'97*, Lecture Notes in AI, vol. 1319, Springer Verlag, Berlin, pp. 65-80.
10. Davis, A. M. (1993). *Software requirements: Objects, Functions, and States*, Prentice Hall, New Jersey.
11. Dubois, E., Yu, E., Petit, M. (1998). From Early to Late Formal Requirements. In: Proc. IWSSD'98. IEEE Computer Society Press.
12. 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, 1999

13. Erdmann, M. and Studer, R. (1998). Use-Cases and Scenarios for Developing Knowledge-based Systems. In: J. Cuena (ed.), *Proceedings of the 15th IFIP World Computer Congress, WCC'98, Conference on Information Technology and Knowledge Systems, IT&KNOWS'98*, pp. 259-272.
14. Fensel, D. (1995). Assumptions and limitations of a problem solving method: a case study. In: B.R. Gaines, M.A. Musen (Eds.), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-based Systems Workshop, KAW'95*, Calgary: SRDG Publications, Department of Computer Science, University of Calgary.
15. Fensel, D., Benjamins, R. (1996) Assumptions in model-based diagnosis. In: B.R. Gaines, M.A. Musen (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop, KAW'96*, Calgary: SRDG Publications, Department of Computer Science, University of Calgary, pp. 5/1-5/18.
16. Gero, J.S., and Sudweeks, F., eds. (1996) *Artificial Intelligence in Design '96*, Kluwer Academic Publishers, Dordrecht.
17. Gero, J.S., and Sudweeks, F., eds. (1998) *Artificial Intelligence in Design '98*, Kluwer Academic Publishers, Dordrecht.
18. Herlea, D., Jonker, C.M., Treur, J. and Wijngaards, N.J.E. (1998). *A Case Study in Requirements Engineering*. Report, Vrije Universiteit Amsterdam, Department of Artificial Intelligence. URL: http://www.cs.vu.nl/~treur/pareqdoc.html
19. Holzblatt, K. and Beyer, K.R. (1995). Requirements gathering: the human factor, *Communications of the ACM*, vol. 38(5), pp. 31.
20. Jackson, M.A. (1975). *Principles of Program Design*, Academic Press.
21. Jonker, C.M. and Treur, J. (1998). 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
22. Kontonya, G., and Sommerville, I. (1998). *Requirements Engineering: Processes and Techniques*. John Wiley and Sons, New York.
23. Musen, M. (1998). 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*, pp. 17-20.
24. Potts, C., Takahashi, K. and Anton, A. (1994). Inquiry based requirements analysis, *IEEE Software*, 11(2), March.
25. Pressman, R.S. (1997). *Software Engineering: A practitioner's approach*. Fourth Edition, McGraw-Hill Series in Computer Science, McGraw-Hill Companies Inc., New York.
26. Sage, A.P., and Palmer, J.D. (1990). *Software Systems Engineering*. John Wiley and Sons, New York.
27. Schreiber, A.Th., Wielinga, B.J., Akkermans, J.M., Velde, W. van de, and Hoog, R. de (1994). CommonKADS: A comprehensive methodology for KBS development. In: *IEEE Expert*, 9(6).

28. Sommerville, I., and Sawyer P. (1997). *Requirements Engineering: a good practice guide*. John Wiley & Sons, Chicester, England.
29. The Standish Group, (1995) *The High Cost of Chaos*: http://www.standishgroup.com
30. Weidenhaupt, K., Pohl, M., Jarke, M. and Haumer, P. (1998). Scenarios in system development: current practice, *IEEE Software*, pp. 34-45, March/April.