Characterizing Knowledge Intensive Tasks indicating Cognitive Requirements;
Scenarios in Methods for Specific Tasks

S.J. Overbeek[1], P. van Bommel[2], H.A. (Erik) Proper[2], and D.B.B. Rijsenbrij[2]

[1] e-office B.V., Duwboot 20, 3991 CD Houten, The Netherlands, EU
Sietse.Overbeek@e-office.com
[2] Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands, EU
{P.vanBommel, E.Proper, D.Rijsenbrij}@cs.ru.nl

Abstract. Methods for specific tasks can among others be identified in conceptual modeling of information systems and requirements engineering in software development. Such methods dictate a specific way of working by describing necessary knowledge intensive tasks to fulfill while applying the method. An actor may experience difficulties when trying to fulfill tasks as part of a method application, related to the cognitive abilities required to fulfill a certain task versus the specific cognitive abilities possessed by the actor. This paper specifically focusses on the cognitive abilities required to fulfill a knowledge intensive task while applying a method for specific tasks. This is based on a categorization and characterization of knowledge intensive tasks and on scenarios in conceptual modeling of information systems and requirements engineering.

1 Introduction

Methods for specific tasks contain a *way of working*, which is the strategy determining the manner how the method should be applied. This includes the necessary knowledge intensive tasks to fulfill when using a method in a certain context. When fulfilling a certain task, an actor that is applying a method may experience difficulties during a task's fulfillment. Independent of other reasons that may contribute to the existence of those difficulties, the research reported in this paper is concerned with the *cognitive* abilities necessary to execute a certain task while applying a method, as is shown in figure 1. As is described by Meiran [6] and Schraagen et al. [8], research in task analysis has a cognitive basis in psychological research. Analyzing task fulfillment from a cognitive viewpoint may yield knowledge underlying an actor's task performance. The research reported in this paper is part of an ongoing research effort to better understand cognitive settings of actors that are applying a method for specific tasks versus the cognitive abilities required to fulfill a typical task. As part of this ongoing research, it is also our wish to provide automated support to assist

Please use the following format when citing this chapter:

Overbeek, S. J., van Bommel, P., Proper, H. A. (Erik), Rijsenbrij, D. B. B., 2007, in IFIP International Federation for Information Processing, Volume 244, Situational Method Engineering: Fundamentals and Experiences, eds. Ralyté, J., Brinkkemper, S., Henderson-Sellers B., (Boston Springer), pp. 100-114.

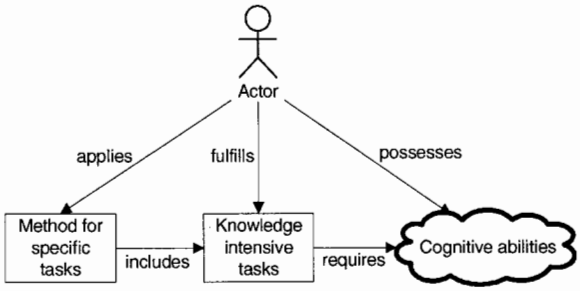

Fig. 1. Cognitive abilities during task fulfillment in a method.

an actor (characterized by a certain cognitive setting) in fulfilling a certain task (characterized by the cognitive abilities required to fulfill it). This automated support should be able to guide an actor that is applying a method through task fulfillment if his cognitive setting may cause difficulties in fulfilling a task.

To better understand knowledge intensive tasks and the nature of it, basic definitions are discussed in section 2.1. Then, the distinguished tasks are classified by their properties indicating an actor's requirements from a cognitive point of view. These properties are further elaborated in sections 2.2 and 2.3 and materialized in methods for specific tasks within conceptual modeling of information systems and requirements engineering (see sections 3 and 4). This leads up to two scenarios in which required cognitive abilities are denoted while fulfilling tasks in conceptual modeling and requirements engineering. Section 5 briefly compares our model with other approaches in the field and outlines benefits of our approach compared to others. Section 6 concludes this paper.

2 Categorizing and Characterizing Knowledge Intensive Tasks

Exploring the fundamentals of knowledge intensive tasks is necessary to gain a better understanding of that what we would like to categorize and characterize. The following subsections provide definitions and a cognition-based characterization of knowledge intensive tasks.

2.1 Basic Definitions

As the notion *knowledge intensive task* suggests, knowledge is very important and also emphatically present during an actor's fulfillment of a knowledge intensive task. It is relevant to mention that, according to Liang [4], knowledge can be regarded as 'wrapped' in information, whilst information is 'carried' by data (expressions in a symbol language). To be able to reason about those tasks on a conceptual level, a general categorization of knowledge intensive tasks is

suggested. For this categorization a parallel with the *inductive-hypothetical research strategy* mentioned in e.g. [9] has been made. This research strategy consists of five phases, which are:

1. Initiation, in which empirical knowledge of the problem domain is elicited.
2a. Abstraction, in which the elicited empirical knowledge is applied in a descriptive conceptual model.
2b. Theory formulation, in which the descriptive conceptual model is made prescriptive.
3a. Implementation, in which the prescriptive conceptual model is empirically tested.
3b. Evaluation, a comparison of the elicited empirical knowledge (1) with the prescriptive empirical model (3a).

Following the research approach, possible knowledge intensive tasks that can be fulfilled can be abstracted to a pattern of three types:

1. `Acquisition` tasks, which are related with the *acquisition* of knowledge. This can be illustrated by a student reading a book in order to prepare himself for an exam.
2. `Synthesis` tasks, which are related with the actual utilization of the acquired knowledge. An example is a student who utilizes knowledge (acquired by reading a book) while performing an exam.
3. `Testing` tasks, which are related with the identification and application of knowledge in practice inducing an improvement of the specific knowledge applied. E.g. a student who failed an exam studies a teacher's feedback on his exam. Then a re-examination attempt follows to improve his previously acquired and utilized knowledge.

The execution of an acquisition task can be compared to going through an *initiation* phase of the inductive-hypothetical research strategy to acquire knowledge and to understand the problem domain well enough so that the acquired knowledge can be abstracted to conceptual models as a next step. The *abstraction* and *theory formulation* phases of the aforementioned research strategy can be compared to the nature of a synthesis task, viz. applying elicited knowledge into a descriptive and a prescriptive conceptual model. The nature of an *implementation* phase and an *evaluation* phase is comparable to what is conducted in a testing task, namely to gain feedback by testing earlier elicited and applied knowledge. In the research strategy this can be translated to testing the prescriptive conceptual model and further the comparison of the elicited knowledge from the initiation phase with the prescriptive empirical model from the implementation phase. Now the set of tasks can be represented as:

$$\mathcal{TA} \triangleq \{\texttt{acquisition}, \texttt{synthesis}, \texttt{testing}\} \tag{1}$$

A specific instantiation of such a task is expressed by $\mathsf{Task} : \mathcal{TI} \rightarrow \mathcal{TA}$, where $\mathcal{TI}$ is a set of *task instances* which are fulfilled by an actor. Given a task instance i of a task $\mathsf{Task}(i)$, we can view the actor that is specifically fulfilling a task instance as a function $\mathsf{Fulfillment} : \mathcal{AC} \rightarrow \mathcal{TI}$. Here, $\mathcal{TI}$ is a set of task instances which are fulfilled by an actor (which is part of a set of actors $\mathcal{AC}$).

2.2 Characterization of Knowledge Intensive Tasks

The following properties are going to be discussed to characterize knowledge intensive tasks:

- The property of *satisfaction* is related with a need for knowledge during a task's fulfillment and the eventual disappearance of that need.
- *Relevance* is concerned with whether or not knowledge acquired is deemed appropriate during the fulfillment of a task.
- The *applicability* property expresses to what extent knowledge is applicable in a task.
- When knowledge is applied it should meet its requirements. This is indicated by the *correctness* property.
- The *faultiness* property is necessary to be able to determine whether or not applied knowledge contains flaws.
- To correct already applied knowledge containing flaws, the *rectification* property can be determined.

Formally, the set of task properties can be represented as:

$$\mathcal{CP} \triangleq \{\texttt{satisfaction}, \texttt{relevance}, \texttt{applicability}, \texttt{correctness}, \texttt{faultiness}, \texttt{rectification}\} \quad (2)$$

The properties shown in table 1 are globally discussed independent from each

Table 1. Characterization of knowledge intensive tasks by their properties

	$\mathcal{CP}$					
$\mathcal{TA}$	Satisfaction	Relevance	Applicability	Correctness	Faultiness	Rectification
Acquisition	×	×	–	–	–	–
Synthesis	–	–	×	×	–	–
Testing	×	–	×	–	×	×

other in the following sections. We understand that there may be other properties requiring specific cognitive abilities when fulfilling knowledge intensive tasks, but in this paper we will limit ourselves to the mutually independent properties mentioned above. The function $\mathsf{Characterization} : \mathcal{TA} \rightarrow \wp(\mathcal{CP})$ specifies which properties belong to a certain task. So following from table 1 an actor fulfilling e.g. an acquisition task should have the cognitive abilities to adhere to the *satisfaction* as well as the *relevance* property.

2.3 Definitions of Knowledge Intensive Task Properties

Before materializing the six task properties of table 1 in methods for specific tasks, the properties themselves are elaborated in this section.

Satisfaction The first property that is discussed is the property of *satisfaction*. A task has a *satisfaction* property, if a need for certain knowledge is present during task fulfillment and that need is indulged if the required knowledge is acquired. The need for knowledge is influenced by what an actor already has received in the past. This can be modeled as a function:

$$\mathsf{Need} : \mathcal{AS} \rightarrow (\wp(\mathcal{KA}) \rightarrow \mathcal{KA} \mapsto [0,1]) \tag{3}$$

The set $\mathcal{AS}$ contains *actor states*. The introduction of an actor state is necessary to understand how an actor's need for knowledge changes over time. The set $\mathcal{KA}$ represents the *knowledge assets* an actor may receive. These *assets* are tradeable forms of knowledge, i.e. knowledge which actors can exchange with each other. This may include knowledge obtained by viewing a Web site or a document or by conversing with a colleague. When an instructor explains a learner how to drive a car for instance, the explanation may contain valuable knowledge assets for the learner. $\mathsf{Need}_t(\mathcal{S}, k)$ is interpreted as the residual need for a knowledge asset k of an actor in state t after the set $\mathcal{S}$ has been presented to an actor, where $t \in \mathcal{AS}$, $k \in \mathcal{KA}$ and $\mathcal{S} \subseteq \mathcal{KA}$. The set $\mathcal{S}$ can be interpreted as the personal knowledge of an actor (also called a knowledge profile). When an actor a in state t experiences a knowledge asset k, then this actor will end up in a new state denoted as $t \ltimes k$:

$$\ltimes : \mathcal{AS} \times \mathcal{KA} \rightarrow \mathcal{AS} \tag{4}$$

No more knowledge is required by an actor if his need for knowledge deteriorates after experiencing the required knowledge, which is denoted by $\mathsf{Need}_{t \ltimes k}(\mathcal{S}, k) = 0$. Note that $\mathsf{Need}_{t \ltimes k}(\mathcal{S}, k) \equiv \mathsf{Need}(t \ltimes k, \mathcal{S}, k)$. However, it is not always necessary to include an actor's state for some of the task properties discussed and can, therefore, be omitted if desired.

An actor's *input* and *output* of knowledge are also considered as important concepts as part of the task properties. Input and output of knowledge assets can be represented as:

$$\mathsf{In}, \mathsf{Out} : \mathcal{AS} \rightarrow (\mathcal{AC} \rightarrow \wp(\mathcal{KA})) \tag{5}$$

Now that an indicator of the need for knowledge and the notation for input and output of knowledge have been explained, the satisfaction property can be assembled:

$$\mathsf{Satisfaction} : \mathsf{Need}_t(\mathcal{S}, k) > 0 \wedge k \in \mathsf{In}_t(a) \Rightarrow \mathsf{Need}_{t \ltimes k}(\mathcal{S}, k) = 0 \tag{6}$$

The satisfaction property includes an actor having a need for knowledge asset k while experiencing state t. To be able to adhere to the satisfaction property, such an actor receives knowledge asset k while in state t. When the actor is in a succeeding state $t \ltimes k$ the need for that specific knowledge asset k deteriorates indicating his specific needs have been satisfied. So if an actor still requires, say, knowledge assets k_1 and k_2 to complete a task, that actor should continue to gather knowledge until $\mathsf{Need}(\mathcal{S}, k_1) = 0$ and $\mathsf{Need}(\mathcal{S}, k_2) = 0$. An acquisition task as well as a testing task have this property. Both tasks require knowledge input, meaning that an actor is satisfied if the required knowledge has been obtained.

Relevance A task has a *relevance* property if, during fulfillment of a task, the knowledge acquired is indeed needed by an actor. To acquire relevant knowledge, an actor should experience a need for the knowledge to be acquired and an actor's knowledge profile should not already contain the knowledge to be acquired:

$$\mathsf{Relevance} : k \in \mathsf{In}(a) \Leftrightarrow \mathsf{Need}(\mathcal{S}, k) > 0 \wedge k \notin \mathcal{S} \tag{7}$$

To make sure that an actor solely acquires relevant knowledge, the relevance property should be adhered to when executing an acquisition task.

Applicability A task has an *applicability* property if knowledge is applied during task fulfillment and that applied knowledge has a useful effect on successfully completing the task. To understand to what extent knowledge is applicable for a task, i.e. has a useful effect for completing the task, the following function is necessary:

$$\mathsf{Applicable} : \mathcal{TI} \times \mathcal{KA} \mapsto [0, 1] \tag{8}$$

If a knowledge asset k is not applicable at all for a task instance i the function equals 0: $\mathsf{Applicable}(i, k) = 0$. If a knowledge asset k is most applicable for a task, the function equals 1. An actor adheres to the *applicability* property only if a certain knowledge asset k is applicable during a task instance:

$$\mathsf{Applicability} : k \in \mathsf{Out}(a) \Leftrightarrow \mathsf{Applicable}(i, k) > 0 \wedge k \in \mathcal{S} \tag{9}$$

The applicability property is not relevant for an acquisition task, because knowledge is not applied in such a task.

Correctness A task has a *correctness* property when the knowledge that is applied is useful for the specific task and the applied knowledge meets its requirements. To be able to determine whether or not applied knowledge is correct it should thus meet its *requirements*. The following function is therefore introduced:

$$\mathsf{Requirement} \subseteq \mathcal{KA} \times \wp(\mathcal{RQ}) \tag{10}$$

Suppose that a knowledge asset k should meet two requirements r_1 and r_2 which are part of a set of requirements $\mathcal{R}$. Then if knowledge k is applied and indeed meets its requirements this is indicated by $(k, \{r_1, r_2\}) \in \mathsf{Requirement}$. The correctness property can now be conceived as follows:

$$\mathsf{Correctness} : \mathsf{Applicable}(i, k) > 0 \wedge k \in \mathsf{Out}(a) \Leftrightarrow (k, \mathcal{R}) \in \mathsf{Requirement} \wedge k \in \mathcal{S} \tag{11}$$

Faultiness A *faultiness* property is part of a task if it is necessary to indicate if certain knowledge that has been obtained by an actor is not meeting its requirements:

$$\mathsf{Faultiness} : \mathsf{In}(a) = \mathcal{K} \wedge (k, \mathcal{R}) \notin \mathsf{Requirement} \wedge k \in \mathcal{K} \Rightarrow \mathsf{Out}(a) = \{k\} \tag{12}$$

Suppose that an actor a obtains a knowledge set $\mathcal{K}$. If an actor a observes that a knowledge asset $k \in \mathcal{K}$ does not meets its requirements this specific asset is returned as output to indicate that it is faulty.

Rectification A task has a *rectification* property if it is part of the task to locate erroneously applied knowledge and then to rectify and return that knowledge so that it does meet its requirements. If an actor receives a knowledge asset k_1 and that knowledge does not meet its requirements $\mathcal{R}$ i.e. the knowledge is wrongly applied, then the actor broadcasts knowledge asset k_2 which does meet the requirements instead. This improvement process by an actor is denoted as *rectification*:

$$\text{Rectification}: \text{In}(a) = \{k_1\} \wedge (k_1, \mathcal{R}) \notin \text{Requirement} \Rightarrow \text{Out}(a) = \{k_2\} \wedge (k_2, \mathcal{R}) \in \text{Requirement} \wedge k_1 \preceq k_2 \quad (13)$$

The notation $k_1 \preceq k_2$ is verbalized as *the knowledge in* k_1 *is contained within* k_2 and is modeled by the function:

$$\preceq : \mathcal{KA} \rightarrow \mathcal{KA} \quad (14)$$

In terms of an actor's need for knowledge, the knowledge containment relation is defined as:

$$k_1 \preceq k_2 \equiv k_1 \preceq_{\text{Need}} k_2 \equiv \text{Need}(\{k_2\}, k_1) = 0 \quad (15)$$

Here, $k_1 \preceq_{\text{Need}} k_2$ represents the knowledge containment relation in the context of the knowledge need represented by 'Need'. In the notation of the rectification property we have omitted Need and denoted knowledge containment as $\preceq$. It is also possible that a certain knowledge asset is contained within more than one knowledge asset. Therefore the $+$ operator concatenates knowledge assets:

$$+ : \mathcal{KA} \times \mathcal{KA} \rightarrow \mathcal{KA} \quad (16)$$

The concatenation of e.g. knowledge assets k_2 and k_3 is therefore shown as $k_2 + k_3$. The function $k_1 \preceq (k_2 + k_3)$ expresses that the knowledge in k_1 is contained within k_2 and k_3.

In order to have a graphical representation of the discussed definitions, an object-role model (ORM) is presented in figure 2. For details on object-role models, see e.g. [2]. Thus far we have focussed on a theory about knowledge intensive tasks and their properties. In the next section a scenario in conceptual modeling of information systems is introduced to illustrate the theory in the context of a method for specific tasks.

3 Cognitive Requirements in Conceptual Modeling Tasks

The discussed theoretical model comes to life when it is illustrated by a practical situation in the process of conceptual modeling. An example of a method for conceptual modeling of information systems is object-role modeling (ORM). ORM is a fact oriented method and makes use of natural language statements by examining them in terms of elementary facts. ORM has a specific *way of working* which makes it a suitable method to study the cognitive requirements needed to fulfill possible knowledge intensive tasks while applying the method. Halpin [2] shows that the way of working in ORM is called the *Conceptual Schema Design Procedure* (CSDP), consisting of seven steps:

1. Transform familiar information examples into elementary facts, and apply quality checks.

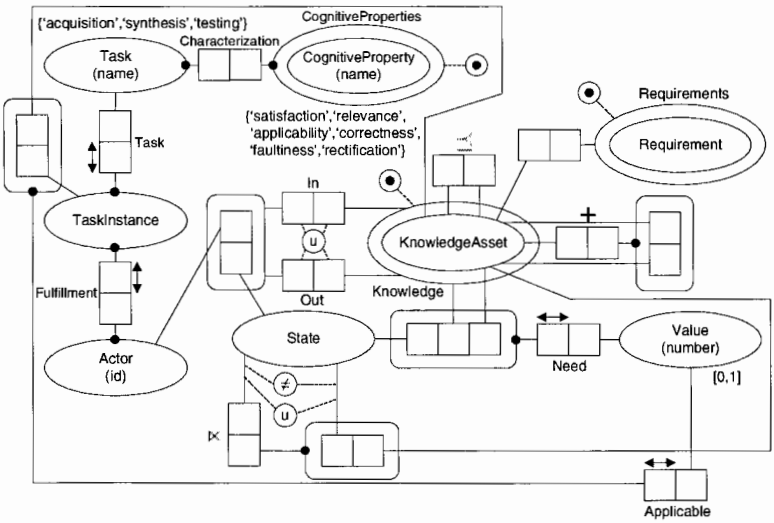

Fig. 2. Object-role model of knowledge intensive task properties.

2. Draw the fact types, and apply a population check.
3. Check for entity types that should be combined, and note any arithmetic derivations.
4. Add uniqueness constraints, and check clarity of fact types.
5. Add mandatory role constraints, and check for logical derivations.
6. Add value, set comparison and sub-typing constraints.
7. Add other constraints and perform final checks.

To let the theoretical model as discussed in section 2 materialize in a practical ORM modeling situation, suppose that a certain actor a who is acting as an *ORM modeler* wishes to create a conceptual model of an information system. Therefore, the ORM modeler walks through the seven steps as mentioned above. In this section we will focus on *step one* only, because the first step is already complex enough to illustrate our theory in the ORM method.

When initiating step one, an ORM modeler fulfills several knowledge intensive tasks. To understand how our theory materializes in an ORM method, a fragment of an information system's intended functionality is considered. One function of the information system to be modeled is to provide insight in a user's own knowledge profile. A partial screen mockup of an information system which should eventually include such functionality is shown in figure 3. The partial mockup shown is part of an application called `DEXAR` (Discovery and eXchange of Revealed knowledge) which is also currently under development as part of our research [7]. `DEXAR` is an application that assists the user in discovering and retrieving knowledge by implementing a question and answer mechanism with the user. The knowledge assets retrieved by the user are then stored in a (searchable) profile as can be seen in figure 3.

Part of the modeling task is to clarify the meaning of the functionality intended. Conversations between a domain expert and the ORM modeler are therefore needed to clarify the required functionality and to let the ORM mod-

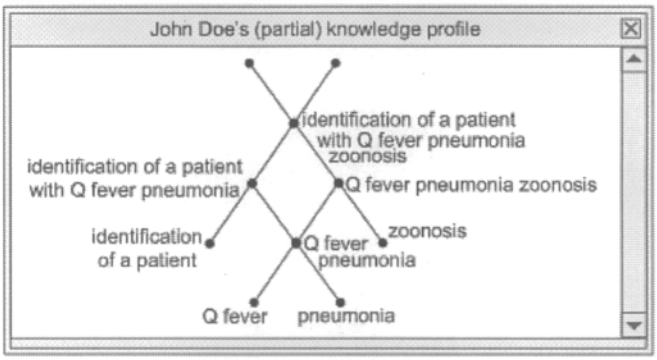

Fig. 3. Showing a (partial) knowledge profile.

eler interpret the example mockup correctly. Discussions with a domain expert are part of an acquisition task instance *acquire information examples* denoted by i_1, thus $\mathsf{Task}(i_1) = \texttt{acquisition}$. Furthermore we can say that, with respect to the partial DEXAR functionality, the ORM modeler responsible for acquiring the information examples has a *need* for those information examples. An information example can be interpreted as information that is presented to the modeler, i.e. graphical information, information on forms, tabularly information, etc. The need for knowledge k concerning an information example is formally expressed as $\mathsf{Need}(\mathcal{S}, k) > 0$, where $\mathcal{S}$ is the personal knowledge profile of the ORM modeler in this case. During fulfillment of task instance i_1 several knowledge assets can be discerned which can be of importance:

k_1 The knowledge profile of a user should be displayed as a lattice.
k_2 The user may browse through the lattice to learn about previously acquired knowledge and to gain insight in his own profile as a whole.
k_3 A lattice should consist of index expressions.

When executing the acquisition task instance i_1, the modeler needs to satisfy the *satisfaction* property, denoted as: $\forall_{n\in\{1,2,3\}}[\mathsf{Need}_t(\mathcal{S}, k_n) > 0 \wedge k \in \mathsf{In}_t(a) \Rightarrow \mathsf{Need}_{t\times k_n}(\mathcal{S}, k_n) = 0]$. In order to acquire knowledge that is not irrelevant, the modeler should satisfy the *relevance* property as follows: $\forall_{n\in\{1,2,3\}}[k_n \in \mathsf{In}(a) \Leftrightarrow \mathsf{Need}(\mathcal{S}, k_n) > 0 \wedge k_n \notin \mathcal{S}]$.

The knowledge gathered thus far is to be stated in terms of *elementary facts* as step one of the ORM method dictates. Basically, an elementary fact asserts that a particular object has a property, or that one or more objects participate in a relationship, where that relationship cannot be expressed as a conjunction of simpler (or shorter) facts. For example, to say that ORM is a modeling language and C++ is a programming language is to assert two elementary facts. Task instance i_1 is now followed by a second task instance i_2. Task instance i_2 is concerned with the creation of elementary facts based on the acquired knowledge k_1, k_2 and k_3 thus far. So this task instance can be referred to as *create elementary facts* and can be classified as a synthesis task.

The ORM modeler applies the knowledge acquired to generate four different elementary facts:

k_4 User has KnowledgeProfile displayed as Lattice
k_5 User browses through Lattice
k_6 Lattice contains Knowledge
k_7 Lattice consists of IndexExpressions

The *applicability* property now determines if the elementary facts are applicable for task instance i_2: $\forall_{n\in\{4,5,6,7\}}[\mathsf{Out}(a) = \{k_n\} \Leftrightarrow \mathsf{Applicable}(i_2, k_n) > 0 \wedge k_n \in \mathcal{S}]$. Once applied, the *correctness* property determines if the knowledge applied meets the requirements: $\forall_{n\in\{4,5,6,7\}}[\mathsf{Applicable}(i_2, k_n) > 0 \wedge k_n \in \mathsf{Out}(a) \Leftrightarrow (k_n, \mathcal{R}) \in \mathsf{Requirement} \wedge k_n \in \mathcal{S}]$. The set R contains the requirements for correctly conceiving elementary facts in ORM. Two possible requirements $r_1, r_2 \in \mathcal{R}$ can be:

r_1 The first letter of object types should be capitalized.
r_2 Each elementary fact should assert a binary relationship between two object types.

Knowledge asset k_4 does not meet requirement r_2, however, because three instead of two objects are part of k_4. In this case the correctness property fails: $(k_4, \{r_2\}) \notin \mathsf{Requirement}$ and the modeler should first alter elementary fact k_4.

When altering k_4, the modeler fulfills a testing task instance i_3 denoted as *correct errors in elementary facts.* A testing task has four properties as can be viewed in table 1. The improvement process or 'quality checks' that are part of task instance i_3 should satisfy the four properties. The *faultiness* property of task instance i_3 stipulates that asset k_4 does not meet requirement r_2: $\mathsf{In}(a) = \mathcal{K} \wedge (k_4, \{r_2\}) \notin \mathsf{Requirement} \wedge k_4 \in \mathcal{K} \Rightarrow \mathsf{Out}(a) = \{k_4\}$. Now when fulfilling task instance i_3, the modeler desires at least one or perhaps more knowledge assets that do meet requirement r_2. To be able to meet the requirement, the modeler, currently in a state t, has a desire to split up knowledge asset k_4 into two new knowledge assets: $k_{4'}$ and $k_{4''}$. These assets should be part of the modeler's profile $\mathcal{S}$ at state $t \ltimes k_{4'} \ltimes k_{4''}$. Therefore the *satisfaction* property is part of the task: $\forall_{n\in\{4',4''\}}[\mathsf{Need}_t(\mathcal{S}, k_n) > 0 \wedge k_n \in \mathsf{In}_t(a) \Rightarrow \mathsf{Need}_{t\ltimes k_n}(\mathcal{S}, k_n) = 0]$. When the newly produced knowledge assets are applied during the task, they should be relevant enough to reach the task's goal. The *applicability* property is thus also part of the task: $\forall_{n\in\{4',4''\}}[k_n \in \mathsf{Out}(a) \Leftrightarrow \mathsf{Applicable}(i_3, k_n) > 0 \wedge k_n \in \mathcal{S}]$. Finally, the *rectification* property determines if requirement r_2 has been met by replacing k_4 with assets $k_{4'}$ and $k_{4''}$: $\mathsf{In}(a) = \{k_4\} \wedge (k_4, \{r_2\}) \notin \mathsf{Requirement} \Rightarrow \mathsf{Out}(a) = \{k_{4'}, k_{4''}\} \wedge (k_{4'}, \{r_2\}) \in \mathsf{Requirement} \wedge (k_{4''}, \{r_2\}) \in \mathsf{Requirement} \wedge k_4 \preceq (k_{4'} + k_{4''})$. Remember from section 2.3 that the knowledge containment relation can be determined by the $\preceq$ symbol and that concatenated knowledge assets are represented by the $+$ symbol. In the property above, the following knowledge containment relation is depicted: $k_4 \preceq (k_{4'} + k_{4''})$. This can be verbalized as: *the knowledge in k_4 is contained within the concatenation of $k_{4'}$ and $k_{4''}$*. The resulting facts are then displayed as follows after the completion of testing task instance i_3:

$k_{4'}$ User has KnowledgeProfile
$k_{4''}$ KnowledgeProfile is displayed as Lattice
k_5 User browses through Lattice

k_6 Lattice contains Knowledge
k_7 Lattice consists of IndexExpressions

The following section shows how the defined task properties can be situated in a requirements engineering scenario by focussing on the way of working of COLOR-X, which is an example of a requirements engineering method.

4 Cognitive Requirements in Requirements Engineering Tasks

In the previous section a scenario in conceptual modeling of information systems has been presented in which our theory came alive. We will now elaborate a scenario in the area of requirements engineering. Requirements engineering is an indication for the first phase of a software development process, in which the main objective is to correctly understand the needs of the system's customers or users: *What* is the system supposed to do. The process of understanding these needs or requirements, i.e. requirements engineering, can be defined as the systematic process of developing requirements through an iterative cooperative process of analyzing the problem, documenting the resulting observations in a variety of representation formats, and checking the accuracy of the understanding gained [5]. The Ph.D. thesis of Burg [1] illustrates the COLOR-X method for requirements engineering. The COLOR-X way of working covers requirements specification, verification and validation phases. In this section we will limit ourselves to how the knowledge intensive tasks of section 2.1 can be fulfilled in a *requirements specification* phase indicating the cognitive requirements for fulfilling those tasks. The process of requirements specification consists of mapping real-world phenomena as described in the requirements document onto basic concepts of a specification language, i.e. describing a certain problem in an as precise, concise, understandable and correct as possible manner. The COLOR-X method divides the requirements specification stage in two parts: a natural language approach and a scenario based approach. In this section we will limit ourselves to the natural language approach, which equals most how the ORM method specifies a conceptual model. The COLOR-X natural language approach for specifying requirements consists of four steps:

1. Select the words and sentences from the requirements document that are relevant for the COLOR-X models.
2. Break up complex sentences and / or combine several redundant or overlapping sentences into understandable ones (i.e. structured sentences).
3. Annotate additional syntactic and semantic information, retrieved from the lexicon, to the words selected from the requirements document.
4. Transform the structured sentences into formal specifications.

In this section, a possible acquisition task as part of step *one* is discussed. Furthermore a synthesis task and a testing task as part of step *two* are dealt with. Suppose that actor a is a *requirements modeler* and wishes to go through the requirements specification phase and therefore applies the COLOR-X method. Assume that the following snippet is part of the requirements document of the `DEXAR` application:

```
A partial knowledge profile should be represented by a lattice also
referred to as a power index expression. Such a lattice should be
constructed by using index expressions. A power index expression
contains all index expressions, including the empty index expression
and the most meaningful index expression. An example of an index
expression is '(identification of a patient) with (Q fever
pneumonia)'. Simply put, (power)index expressions are used by DEXAR
as a representation for a knowledge profile.
```

While walking through the first step as mentioned above, the requirements modeler selects the words and sentences from the requirements document snippet. This is part of an acquisition task instance *acquire words and sentences* denoted by i_1, thus $\mathsf{Task}(i_1) = \texttt{acquisition}$. The requirements modeler has a *need* for those words and sentences. The acquired words and sentences i.e. knowledge assets can be depicted as follows:

k_1 A partial knowledge profile is represented by a lattice.
k_2 A lattice equals a power index expression.
k_3 A power index expression contains all index expressions.
k_4 A power index expression includes the empty index expression and the most meaningful index expression.

When executing the acquisition task instance above, the requirements modeler needs to satisfy the *satisfaction* property, denoted as: $\forall_{n\in\{1,2,3,4\}}[\mathsf{Need}_t(\mathcal{S}, k_n) > 0 \wedge k \in \mathsf{In}_t(a) \Rightarrow \mathsf{Need}_{t\ltimes k_n}(\mathcal{S}, k_n) = 0]$. In order to acquire knowledge that is not irrelevant, the modeler should satisfy the *relevance* property as follows: $\forall_{n\in\{1,2,3,4\}}[k_n \in \mathsf{In}(a) \Leftrightarrow \mathsf{Need}(\mathcal{S}, k_n) > 0 \wedge k_n \notin \mathcal{S}]$.

Step one can be seen as an intensive knowledge acquirement step, i.e. the requirements document is sifted for relevant words and sentences. It is not until step two of the requirements specification process as prescribed by COLOR-X that a synthesis task can be identified. Task instance i_1 is now followed by a task instance i_2. Task instance i_2 can be referred to as *create structured sentences* and is part of step two. Table 2 represents the knowledge assets following from task i_2. Knowledge assets k_5 up to and including k_9

Table 2. The created structured sentences

	Subject	Predicate	Direct object
k_5	A lattice	represents	a knowledge profile
k_6	A lattice	equals	a power index expression
k_7	A power index expression	contains	all index expressions
k_8	A power index expression	includes	the empty index expression
k_9	A power index expression	includes	the most meaningful index expression

are mostly similar with assets k_1 up to and including k_4, but the knowledge assets of table 2 include additional *grammatical* knowledge instead. The *applicability* property now determines if the structured sentences are applicable

for task i_2: $\forall_{n\in\{5,6,7,8,9\}}[\mathsf{Out}(a) = \{k_n\} \Leftrightarrow \mathsf{Applicable}(i_2, k_n) > 0 \wedge k_n \in \mathcal{S}]$. Once applied, the *correctness* property determines if the structured sentences meet the requirements: $\forall_{n\in\{5,6,7,8,9\}}[\mathsf{Applicable}(i_2, k_n) > 0 \wedge k_n \in \mathsf{Out}(a) \Leftrightarrow (k_n, \mathcal{R}) \in \mathsf{Requirement} \wedge k_n \in \mathcal{S}]$. The set R contains the requirements for correctly conceiving structured sentences in COLOR-X. Two possible requirements $r_1, r_2 \in \mathcal{R}$ can be:

r_1 Annotate a main sentence structure, i.e. the subject, predicate and direct object.
r_2 Annotate special grammatical elements, i.e. the adjectives, adverbs and nominal predicates.

Knowledge assets k_5 up to and including k_9 do not meet requirement r_2, however, because no special grammatical elements are shown in table 2. In this case the correctness property fails and the requirements modeler should first add special grammatical elements.

When altering k_5 up to and including k_9, the requirements modeler fulfills a testing task instance i_3 denoted as *correct omitted special grammatical elements*. The resulting special grammatical elements are displayed in table 3 after completing testing task instance i_3. Now the properties of task instance

Table 3. The created special grammatical elements

	Grammatical concept	Word	Category
$k_{5'}$	Adjective	Partial	Property
$k_{6'}$	Nominal predicate	A lattice is a power index expression	Specialization
$k_{7'}$	Adverb	All	Quantity
$k_{8'}$	Adjective	Empty	Property
$k_{9'}$	Adjective	Most meaningful	Property

i_3 should be analyzed to determine how they are satisfied. For asset k_5, the *faultiness* property stipulates that the asset does not meet requirement r_2: $\mathsf{In}(a) = \mathcal{K} \wedge (k_5, \{r_2\}) \notin \mathsf{Requirement} \wedge k_5 \in \mathcal{K} \Rightarrow \mathsf{Out}(a) = \{k_5\}$. To be able to meet the requirement, the modeler, currently in a state t, has a desire to create another knowledge asset $k_{5'}$ that includes special grammatical elements for the sentence included in k_5. The concatenation of k_5 and $k_{5'}$, i.e. $k_5{+}k_{5'}$ should meet both requirements r_1 and r_2. The concatenated asset should be part of the modeler's profile $\mathcal{S}$ at state $t \ltimes k_5{+}k_{5'}$. Therefore the *satisfaction* property for $k_5{+}k_{5'}$ results in: $\mathsf{Need}_t(\mathcal{S}, k_5 + k_{5'}) > 0 \wedge k_5 + k_{5'} \in \mathsf{In}_t(a) \Rightarrow \mathsf{Need}_{t\ltimes k_5+k_{5'}}(\mathcal{S}, k_5 + k_{5'}) = 0$. When the concatenated knowledge asset $k_5 + k_{5'}$ is applied during the task, it should be relevant enough to reach the task's goal. This is expressed by the *applicability* property: $k_5 + k_{5'} \in \mathsf{Out}(a) \Leftrightarrow \mathsf{Applicable}(i_3, k_5 + k_{5'}) > 0 \wedge k_5 + k_{5'} \in \mathcal{S}$. Finally, the *rectification* property determines if requirement r_2 has been met by creating asset $k_{5'}$ and concatenating it with k_5: $\mathsf{In}(a) = \{k_5\} \wedge (k_5, \{r_2\}) \notin \mathsf{Requirement} \Rightarrow \mathsf{Out}(a) = \{k_5 + k_{5'}\} \wedge (k_5 + k_{5'}, \{r_2\}) \in \mathsf{Requirement}$. Following the same approach as above, properties k_6 up to and including k_9 can be concatenated with the created grammatical elements. So k_6 should be concatenated

with $k_{6'}$ and so on. This completely satisfies the properties of task instance i_3 eventually.

Now that the theoretical part and possible applications of it in methods for specific tasks have been discussed, it is appropriate to compare our approach with other approaches in the field. The next section therefore deals with this matter.

5 Discussion

Literature indicates that characterizing tasks on a cognitive basis is possible in several different ways. The research of Weir et al. [10] includes a characterization of *information management tasks* by studying activities of workers in the primary care setting. This has resulted in an abstraction of several information management tasks during the research, such as: assignment tasks, determination tasks, organization tasks, etc. First, Weir et al. [10] show that they have analyzed tasks in primary clinical care and from that specialized analysis an abstraction has been made constituting a general categorization of tasks. Compared to our study, this is a bottom-up approach from analyzing tasks in a certain context to the eventual abstraction of tasks. We have analyzed tasks using a top-down approach by generalizing tasks based on parallels made with an inductive-hypothetical research approach before materializing the theory in methods for specific tasks. An advantage of our approach is that the theory is not stemming from a study in a specialized context and thus does not run the risk of being useful only in a certain context. Therefore, it is assumed that our theory is applicable in numerous contexts and can be adapted to that context if desired. For instance, sections 3 and 4 are an indication that this is possible.

Especially when methods for specific tasks are concerned, it is difficult to identify significant research related to matching an actor's cognitive abilities with the cognitive abilities required to perform a certain task. However, the research of Zhang et al. [11] shows that the *human-centered distributed information system design* methodology includes *user analysis* and *task analysis* as part of information system design. The methodology has a much broader focus than only dealing with the match / mismatch between a user's cognitive abilities and the cognitive abilities necessary to fulfill a specific task. An important function of task analysis in human-centered distributed information system design is to ensure that the system implementation includes only the necessary and sufficient task features that match user capacity and are required by the task. This contrasts with our research, because we do not wish to exclude the situations in which an actor / task combination does not match very well, but instead we would like to provide support for it in the future. We assume that instead of excluding the situations in which an actor / task combination does not match it is better to provide *support* for it, simply because it occurs often enough in everyday practice. An early attempt by e.g. Harris and Brightman [3] shows a preliminary attempt to couple potential automated support with cog-

nitive task fulfillment by academics. The proposed automated support however consists of existing tools only and suggestions for future, possibly better, tools are not made. Hence it seems that our longer term research goals, as mentioned in section 1, are worth pursuing.

6 Conclusion

This paper describes a categorization and characterization of knowledge intensive tasks, illustrated by definitions of task properties indicating cognitive requirements for task fulfillment. Proceeding from these definitions method application scenarios in conceptual modeling of information systems respectively requirements engineering show how the theory can be materialized.

References

1. J.F.M. Burg. *Linguistic Instruments in Requirements Engineering.* PhD thesis, Vrije Universiteit Amsterdam, The Netherlands, EU, 1997.
2. T. Halpin. *Information Modeling and Relational Databases, from Conceptual Analysis to Logical Design.* Morgan Kaufmann, San Mateo, CA, USA, 2001.
3. S.E. Harris and H.J. Brightman. Design implications of a task-driven approach to unstructured cognitive tasks in office work. *ACM Transactions on Information Systems*, 3(3):292–306, 1985.
4. T.Y. Liang. The basic entity model: A fundamental theoretical model of information and information processing. *Information Processing & Management*, 30(5):647–661, 1994.
5. P. Loucopoulos and V. Karakostas. *System Requirements Engineering.* McGraw-Hill Book Company Europe, Berkshire, UK, EU, 1995.
6. N. Meiran. Modeling cognitive control in task-switching. *Psychological Research*, 63(3–4):234–249, 2000.
7. S.J. Overbeek, P. van Bommel, H.A. Proper, and D.B.B. Rijsenbrij. Knowledge discovery and exchange – Towards a web-based application for discovery and exchange of revealed knowledge. In J. Filipe, J. Cordeiro, B. Encarnação, and V. Pedrosa, editors, *Proceedings of the Third International Conference on Web Information Systems and Technologies (WEBIST)*, pages 26–34. Barcelona, Spain, EU, INSTICC Press, Setúbal, Portugal, EU, 2007.
8. J. Schraagen, S. Chipman, and V. Shalin. *Cognitive Task Analyis.* Lawrence Erlbaum Associates, Mahway, NJ, USA, 2000.
9. H.G. Sol. *Simulation in Information Systems.* PhD thesis, University of Groningen, The Netherlands, EU, 1982.
10. C.R. Weir, J.J.R. Nebeker, L.H. Bret, R. Campo, F. Drews, and B. LeBar. A cognitive task analysis of information management strategies in a computerized provider order entry environment. *Journal of the American Medical Informatics Association*, 14(1):65–75, 2007.
11. J. Zhang, V.L. Patel, K.A. Johnson, J.W. Smith, and J. Malin. Designing human-centered distributed information systems. *IEEE Intelligent Systems*, 17(5):42–47, 2002.