Knowledge-Based Systems 21 (2008) 764–785

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Matching cognitive characteristics of actors and tasks in information systems engineering

S.J. Overbeek [a,*], P. van Bommel [b], H.A. (Erik) Proper [b]

[a] e-office B.V., Duwboot 20, 3991 CD Houten, The Netherlands
[b] Institute for Computing and Information Sciences, Radboud University Nijmegen, Toernooiveld 1, 6525 ED Nijmegen, The Netherlands

ARTICLE INFO

Article history:
Received 21 November 2007
Accepted 28 March 2008
Available online 4 April 2008

Keywords:
Cognitive characteristics
Fuzzy match assessments
Information systems
Matchmaking
Task allocation

ABSTRACT

In daily practice, discrepancies may exist in the suitability match of actors and the tasks that have been allocated to them. A formal framework for cognitive matchmaking and a prototype implementation are introduced as a possible solution to improve the fit between actors and tasks. A case study has been conducted to clarify how the proposed cognitive matchmaking approach can be utilized in information systems engineering. The inductive-hypothetical research strategy has been applied as an overall research approach. A separate iteration of the strategy has been applied within the case study.

1. Introduction

The importance of an actor's (i.e. a human or a computer) abilities to acquire, apply and test knowledge increases due to, e.g. growing product complexity, the move toward globalization, the emergence of virtual communities and organizations, and the increase in focus on customer orientation [1]. When the pressure to acquire, apply and test more knowledge increases, actors struggle to manage their basic cognitive functions like, e.g. the willpower to fulfill a task or maintaining awareness of the requirements to fulfill a task. These cognitive functions are also referred to as *volition* and *sentience*, respectively, in cognitive literature [2,3]. A knowledge intensive task is a task for which acquisition, application or testing of knowledge is necessary in order to successfully fulfill the task. The framework discussed in this paper matches cognitive characteristics supplied by actors and the cognitive characteristics required to fulfill tasks. This may achieve a better fit between actors and tasks. Difficulties to control basic cognitive functions influences practice and potentially threatens the success of task fulfillment [4].

Research in cognitive psychology has demonstrated that individual knowledge processing is negatively influenced when experiencing an overload of knowledge that needs to be processed. For example, a burden of knowledge processing events may cause actors to underestimate the rate of events [5] and to be overconfident [6]. Within the enterprise, the benefits of cognitive matchmaking can be found in at least four areas:

Business process reengineering. BPR consists of computer-aided design of processes and automatic generation of process models to improve customer service [7]. The design and creation of processes and process models may be improved if the business process modeler knows beforehand which available actors best fit the tasks that need to be fulfilled as part of a newly designed business process.

Information systems engineering. Information systems engineering (ISE) is related with the conceptualization, design, development and implementation of information systems to support business functions [8]. Cognitive matchmaking can be utilized to support in allocating tasks to actors that are involved in every ISE phase.

Multi-agent systems. Multi-agent systems incorporate several software agents that may work together to assist humans in performing their tasks [9]. One way of providing assistance is to match tasks with human actors to understand which tasks fit best with which human actors.

Workflow management. The primary task of a workflow management system is to enact case-driven business processes by joining several perspectives [10]. One of these perspectives is the *task* perspective. This perspective describes the elementary operations performed by actors while executing a task for a specific case. An example of a case is a tax declaration. Integration of cognitive matchmaking in a workflow management system may prescribe

* Corresponding author. Fax: +3 184 758 5561.
E-mail addresses: Sietse.Overbeek@e-office.com (S.J. Overbeek), P.vanBommel@cs.ru.nl (P. van Bommel), E.Proper@cs.ru.nl (H.A. (Erik) Proper).

0950-7051/$ - see front matter © 2008 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2008.03.032

which available actors fit best with the tasks that are part of a case. This may improve the allocation of tasks to actors while enacting a business process.

The research reported in this paper is specifically concerned with the *matching* of cognitive characteristics required to fulfill a certain task instance with the cognitive characteristics actually *possessed* by an actor. There are three main goals of this paper. First, a formal theoretical framework of cognitive matchmaking is presented, which includes mathematical functions to calculate the match of an actor and a task. Second, the prototype implementation of this framework is discussed. In this prototype, the functions of the framework have been implemented resulting in a Web-based cognitive matchmaking application. Lastly, the results of a conducted explorative case study are discussed which show how the system can be utilized in information systems engineering. The main concepts illustrated in Fig. 1 provide a first overview of the aspects that are taken into consideration. Fig. 1 shows that an actor type and a task type can be instantiated by an actor, respectively, a task instance. This means that an actor can be classified as a certain type and a task instance can be classified as a certain type. While fulfilling a task instance, an actor supplies cognitive characteristics. Fig. 1 shows that the task instance that is fulfilled by the actor demands cognitive characteristics for successful fulfillment. The match of supply and demand can then be studied. In [11], we have already provided a preliminary discussion about several types of *knowledge intensive tasks*, each characterized by their characteristics. This earlier work about the characterization of task types is used as input for the formal theoretical framework presented here. The results from our initial work about cognitive matchmaking discussed in [12] are also integrated and extended in this paper.

The paper is structured as follows. Several cognitive settings of actors are discussed in Section 3 to be able to characterize the different actors fulfilling a task instance. A framework of cognitive matchmaking is introduced in Section 5 to be able to compute a match of the supply of certain cognitive characteristics by an actor and the demanded cognitive characteristics to fulfill a task instance. The theory is materialized throughout this section in a running example by matching an actor type from the theory of Section 3 with a task type from theory as discussed in Section 4. The framework includes functions that calculate a match based on numerical values. An extension of the theory that is able to present linguistic match values is discussed in Section 6. However, this extension has not yet been incorporated in the prototype of a cognitive matchmaker system, which is elaborated in Section 7. The cognitive matchmaking framework and the prototype are evaluated in Section 8 by means of a case study in information systems engineering. Section 9 briefly compares our models with other approaches in the field and outlines the benefits of our approach compared to others. Section 10 concludes this paper and gives an overview of future research plans.

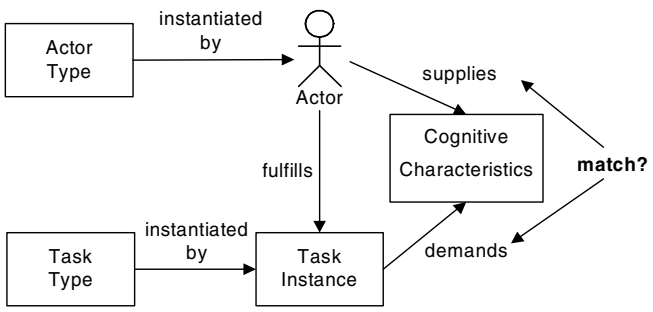

Fig. 1. Main concepts of cognitive matchmaking.

2. Research approach

The overall research strategy that has been applied is based on the inductive-hypothetical research strategy (see e.g. [13–16]). This research strategy consists of the following phases:

1. Initiation, in which empirical knowledge of the problem domain is elicited.
2. Abstraction, in which the elicited empirical knowledge is applied in a descriptive conceptual model.
3. Theory formulation, in which the descriptive conceptual model is made prescriptive.
4. Implementation, in which the prescriptive conceptual model is empirically tested.
5. Evaluation, a comparison of the elicited empirical knowledge (1) with the prescriptive empirical model (4).

The application of the strategy for this study results in the following steps:

1. Observation of and reasoning about the issues underlying the initiation of this study (Sections 1 and 9).
2. Abstraction of the results of phase 1 to our general cognitive model of actor types and task types (Sections 3 and 4 and in [11]).
3. Development of a cognitive matchmaking framework including the fuzzy match extension (Sections 5 and 6).
4. Implementation of the cognitive matchmaking framework.
   4.1. Prototype of a cognitive matchmaker system (Section 7).
   4.2. Case study in information systems engineering (Section 8).
5. Overall research evaluation (Section 10).

3. Cognitive actor settings

Before elaborating on matching cognitive characteristics possessed by an actor with the cognitive characteristics required when fulfilling a task instance, a characterization of possible actor types and task types is needed.

3.1. Actor types

Actor types may draw from a pool of basic cognitive characteristics an actor might possess, such as sentience, volition and causability. An actor type does not need to have all of these characteristics, and some have more than others. It is assumed that each of these characteristics can be isolated from the others, and so should be treated as distinct. Cruse [17] and Dowty [18] utilized syntactic tests to isolate characteristics from each other. Therefore, a characteristic has been correlated with a syntactic environment which admits one characteristic but not the other. The following characteristics can be distinguished that can be utilized to generate a framework for cognitive settings of possible different actor types:

- The *volition* characteristic is concerned with an actor's willpower to fulfill some knowledge intensive task instance. For instance, a skilled software developer may have more willpower to implement an intelligent search algorithm than implementing source code to access a database.
- *Sentience* expresses that an actor has much awareness of required knowledge to fulfill some task instance. When a project manager creates a project plan he may have all the necessary knowledge to create such a plan. This may be due to earlier planning experiences of the project manager or by education.

- The *causability* characteristic expresses that an actor has the ability to exert an influence on state changes of knowledge involved during fulfillment of a task instance. Suppose that a business consultant facilitates a brainstorm session in which he or she writes models on a whiteboard. In this case, the consultant causes knowledge that is implicitly present in his or her head to be made explicit on the white board [19]. This means that the consultant *causes* knowledge to change from one type to another. The concept of causability is further explored in Section 3.2.3.
- During fulfillment of certain knowledge intensive task instances an actor should be able to improve its own cognitive abilities. This is indicated by the *improvability* characteristic. For instance, a manager may have recently completed a course about cybernetics. During his work he or she successfully applies several principles that the manager has learned in the course. Participating in the course may thus have improved his or her cognitive abilities.
- The *independency* characteristic is necessary to be able to determine if an actor is able to fulfill a task instance on his own or not. An example is a journalist who may successfully write a news article without having to collaborate with others.

Having determined possible cognitive characteristics an actor may have it is now appropriate to distinguish several actor types. The combination of an actor type with the cognitive characteristics belonging to a type forms a *cognitive actor setting*. This characterization is shown in Table 1. The five distinguished actor types are based on a classification of knowledge worker types [20] and on linguistic literature [2]. The knowledge worker classification is more practically oriented than the ideas found in the linguistic literature. Practical as well as theoretical ideas now intermingle when developing a framework of cognitive actor settings. Now the set of actor types can be represented as:

$$\{\texttt{experiencer}, \texttt{collaborator}, \texttt{expert}, \texttt{integrator}, \texttt{transactor}\} \subseteq \mathscr{AT} \tag{1}$$

The set of cognitive characteristics can be represented as:

$$\{\texttt{volition}, \texttt{sentience}, \texttt{causability}, \texttt{improvability}, \texttt{independency}\} \subseteq \mathscr{CC} \tag{2}$$

An important remark to make here is that the possible actor types as well as the possible cognitive characteristics are not limited to five actor types and five cognitive characteristics. However, in this paper we restrict ourselves to the cognitive actor settings above. The actor types as shown in Table 1 can now be introduced:

The experiencer. The *experiencer* actor type has the sentience characteristic only. An experiencer is thus only aware of knowledge requirements to fulfill some task instance. Consider for example the following sentence: *John thoroughly reads an article about balanced scorecards before joining a meeting about balanced scorecards*. This indicates that John, as an experiencer, probably understands that reading an article about balanced scorecards is enough to successfully prepare himself for a meeting about that topic.

Table 1
Cognitive actor settings characterized

$\mathscr{AT}$	$\mathscr{CC}$				
	Volition	Sentience	Causability	Improvability	Independency
Experiencer	–	×	–	–	–
Collaborator	×	–	×	×	–
Expert	×	×	×	×	×
Integrator	×	–	×	–	–
Transactor	×	×	–	–	×

The collaborator. This actor type possesses the volition, causability, and improvability characteristics. A collaborator has the ability to exert an influence on state changes of knowledge involved during fulfillment of a task instance. During fulfillment of a knowledge intensive task instance a collaborator is also able to improve its own cognitive abilities. However, a collaborator does not have complete awareness of all required knowledge to fulfill a task instance and requires others to fulfill a task instance. Consider the following example: *John works at a hospital and requires knowledge about a patient's history. Therefore, he acquires the most recent patient log from a colleague*. This indicates that John, as a collaborator, understands that in order to acquire knowledge about a patient's history he must collaborate with another actor. After that John is able to update the patient's log with recent changes.

The expert. An expert possesses all characteristics depicted in Table 1. Suppose that John is an assistant professor working at a university and he would like to solve a difficult mathematical problem when developing a theory. He then uses his own knowledge about mathematics to solve the problem. John is also able to combine and modify his own knowledge while solving the problem and he can also learn from that.

The integrator. An integrator is able to fulfill a knowledge intensive task instance by working together and is able to initiate state changes of knowledge involved during task instance fulfillment. An integrator primarily wishes to acquire and apply knowledge of the highest possible quality. An engineer contributing to the construction of a flood barrier is an example of an integrator.

The transactor. Volition, sentience, and independency are the characteristics belonging to the transactor actor type. A transactor can fulfill a task instance without collaborating with others and is not required to cause modifications in the knowledge acquired and applied during task fulfillment. A customer support employee working at a software company is an example of a transactor.

A specific instantiation of an actor type is expressed by $\mathsf{AType} : \mathscr{AC} \rightarrow \mathscr{AT}$, where $\mathscr{AC}$ is a set of *actor instances* that can be classified by a specific type. The example $\mathsf{AType}(a) = \texttt{experiencer}$ for instance expresses that an actor $a \in \mathscr{AT}$ can be classified as an experiencer. We can view task instances that are fulfilled by a specific actor as a function $\mathsf{Fulfillment} : \mathscr{AC} \rightarrow \wp(\mathscr{TI})$. Here, $\mathscr{TI}$ is a set of task instances which are fulfilled by an actor. An actor $a \in \mathscr{AC}$ that fulfills a task instance $i \in \mathscr{TI}$ can be expressed as $\mathsf{Fulfillment}(a) = \{i\}$. A specific instantiation of a task type is expressed by $\mathsf{TType} : \mathscr{TI} \rightarrow \mathscr{TT}$, where $\mathscr{TT}$ is a set of *task types* that can be instantiated by a specific task instance. The expression $\mathsf{TType}(i) = \texttt{acquisition}$ can be used to assert that a task instance i is characterized as an acquisition task.

Now that a characterization of different actor types has been introduced (resulting in several cognitive actor settings), the different cognitive characteristics mentioned in Table 1 need to be explored.

3.2. Definitions of cognitive characteristics

3.2.1. Volition

An actor has the *volition* characteristic, if an actor has a certain willpower to fulfill some knowledge intensive task instance. It can be said that an actor has a motivation to fulfill a task instance. It is important to note that for each of the cognitive characteristics an actor might have, an actor may possess it at a certain level. The level on which an actor has willpower to fulfill a task is incorporated in the volition characteristic. The introduction of a motivation function is necessary to determine an actor's motivation while fulfilling a task instance:

$$\mathsf{Motivation} : \mathscr{AS} \rightarrow (\mathscr{AC} \times \mathscr{TI} \rightarrow \mathscr{MO}) \tag{3}$$

The set $\mathscr{AS}$ contains *actor states*. An actor state is necessary because an actor's motivation might change over time. For example, an actor might be strongly motivated in one state, while an actor might be weakly motivated in another state. To formally define an actor state, the actor identity function is required:

$$\mathsf{Identity} : \mathscr{AS} \rightarrow \mathscr{ID} \tag{4}$$

The set $\mathscr{ID}$ contains *actor identities*. If i is the identity of an actor, then $\mathscr{AS}_i$ is used to denote the set of actor states of i:

$$\mathscr{AS}_i \triangleq \{t | \mathsf{Identity}(t) = i\} \tag{5}$$

When an actor experiences knowledge, then this will lead to a change in both the actor's knowledge and mood, or, in a state change. In this paper, we restrict ourselves to state changes caused by experiencing knowledge. We do not consider other state changes. For example, forgetting knowledge may be seen as a special change of state. Assume $\{\mathtt{weak}, \mathtt{moderate}, \mathtt{neutral}, \mathtt{strong}\} \subseteq \mathscr{M}\mathscr{O}$. The set $\mathscr{M}\mathscr{O}$ includes possible motivation types of an actor. An actor a in a state $t \in \mathscr{AS}$ with a volition characteristic may be weakly, moderately, neutrally or strongly motivated. If an actor a in state t is strongly motivated to fulfill task instance i it can be denoted as: $\mathsf{Motivation}_t(a, i) = \mathtt{strong}$. Note that the motivation function can not be expressed as $\mathsf{Motivation}(t, a, i)$ because of the placed parentheses in the signature of the function. This has been done to indicate that an actor's motivation is coupled with its state. Because of these parentheses, it can be noticed that the *domain* of the motivation function consists of the set of actor states. The *range* of the motivation function consists of a nested total function including the set of actor instances, the set of task instances and the set of motivation types. The volition characteristic can now be modeled as follows. An actor $a \in \mathscr{AC}$ has the volition characteristic, denoted as $\mathsf{Volition}(a)$, if that actor has a state $t \in \mathscr{AS}$ in which that actor has one of the four motivation types for some task instance to be fulfilled:

$$\exists_{t \in \mathscr{AS}} \exists_{i \in \mathsf{Fulfillment}(a)} [\mathsf{Motivation}_t(a, i) \in \{\mathtt{weak}, \mathtt{moderate}, \mathtt{neutral}, \mathtt{strong}\}] \tag{6}$$

3.2.2. Sentience

An actor has the *sentience* characteristic, if that actor has significant awareness of required knowledge to fulfill some task instance. In [11] a function has been introduced to understand to what extent a *knowledge asset* (as part of the set $\mathscr{KA}$) is applicable for a task, i.e. has a useful effect for completing the task:

$$\mathsf{Applicable} : \mathscr{TI} \times \mathscr{KA} \rightarrow [0, 1] \tag{7}$$

These *assets* are tradeable forms of knowledge, i.e. knowledge that is exchangeable between actors. 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.

So, $\mathsf{Applicable}(i, k) > 0$ expresses that knowledge asset k is somehow applicable for a task instance i. Another function denotes the need for knowledge of an actor during fulfillment of a task instance [11]:

$$\mathsf{Need} : \mathscr{AS} \rightarrow (\wp(\mathscr{KA}) \times \mathscr{KA} \rightarrow [0, 1]) \tag{8}$$

The expression $\mathsf{Need}_t(S, k)$ is interpreted as the residual need for knowledge k of an actor in state t after the set $\mathscr{S}$ has been presented to an actor, where $t \in \mathscr{AS}$, $k \in \mathscr{KA}$ and $\mathscr{S} \subseteq \mathscr{KA}$. The set $\mathscr{S}$ can be interpreted as the personal knowledge of an actor (also called a knowledge profile). At this point the sentience characteristic can be modeled:

$$\exists_{i \in \mathsf{Fulfillment}(a)} \exists_{k \in \mathscr{KA}} \exists_{S \subseteq \mathscr{KA}} [\mathsf{Applicable}(i, k) > 0 \wedge \mathsf{Need}(\mathscr{S}, k) \geqslant 0] \tag{9}$$

In other words, an actor $a \in \mathscr{AC}$ has the sentience characteristic, denoted as $\mathsf{Sentience}(a)$, if that actor fulfills some task instance and if there exists a knowledge asset $k \in \mathscr{KA}$ that is applicable in a task instance and already possessed by actor a (i.e. part of that actor's knowledge profile $S \subseteq \mathscr{KA}$) or otherwise required by actor a. The actor's state has been omitted because it is not of particular relevance in the sentience characteristic.

3.2.3. Causability

An actor has the *causability* characteristic, if an actor has the ability to exert an influence on changes of the knowledge *type* involved during fulfillment of a task instance. The level of this influence is dependent of to what extent an actor masters this characteristic. Four knowledge types are distinguished [19,21]:

- Implicit & concealed knowledge: e.g. competencies or expertise of a worker unknown to the organization.
- Explicit & concealed knowledge: e.g. valuable insights concealed in available data collections (to be discovered by data mining).
- Implicit & revealed knowledge: e.g. known expertise of a worker which can be appealed to.
- Explicit & revealed knowledge: e.g. best-practice documentation, knowledge bases, scientific papers, etcetera.

Implicit knowledge comprises knowledge which is implicitly present in people's heads, such as skills which are difficult to make explicit [19]. Implicit knowledge is closely related to what is generally experienced as intuition. *Explicit knowledge* comprises knowledge which can be expressed in terms of facts, rules, specifications or textual descriptions.

Besides discerning implicit and explicit knowledge, another relevant distinction can be made. Sometimes knowledge is present while one is not aware of that knowledge. This varies from hidden skills of workers (for an individual or for the organization) to knowledge which is hidden in undiscovered patterns in data collections (the basis for data mining). This results in *revealed* and *concealed* knowledge. To be specific, it can be said that an actor having the causability characteristic can change knowledge from one type to another type. This can be modeled as a function:

$$\ltimes : \mathscr{AC} \rightarrow (\mathscr{KA} \times \mathscr{KT} \rightarrow \mathscr{KT}) \tag{10}$$

The set $\mathscr{KT}$ comprises the possible knowledge types. The four discussed knowledge types can formally be depicted as:

$$\{\mathtt{implicit-concealed}, \mathtt{implicit-revealed}, \mathtt{explicit-concealed}, \mathtt{explicit-revealed}\} \subseteq \mathscr{KT} \tag{11}$$

The knowledge type of a specific knowledge item k can easily be found by using the function $\mathsf{KType} : \mathscr{KA} \rightarrow \mathscr{KT}$. For example, $\mathsf{KType}(k) = s$ expresses that knowledge $k \in \mathscr{KA}$ is of the type $s \in \mathscr{KT}$. When knowledge asset k of type s is changed to another type by actor $a \in \mathscr{AC}$, this type change is denoted as: $\ltimes_a(k, s)$. When applying the infix notation this would result in: $\ltimes_a(k, s) \equiv k \ltimes_a s$. At this point the causability characteristic can be modeled:

$$\exists_{i \in \mathsf{Fulfillment}(a)} \exists_{k \in \mathscr{KA}} \exists_{s \in \mathscr{KT}} [\mathsf{Applicable}(i, k) > 0 \wedge k \ltimes_a s] \tag{12}$$

In other words, an actor $a \in \mathscr{AC}$ has the causability characteristic, denoted as $\mathsf{Causability}(a)$, if that actor fulfills some task instance and if there exists a knowledge asset $k \in \mathscr{KA}$ of some type $s \in \mathscr{KT}$ that is changed to some other type $k \ltimes_a s \in \mathscr{KT}$ by actor a.

3.2.4. Improvability

An actor has the *improvability* characteristic, if that actor is able to improve its own cognitive capabilities while fulfilling some task

instance. An actor may have a certain level to improve its own capabilities, ranging from, e.g. a low level to a high level. First, it is necessary to introduce a $\rightarrow^*$ operator that expresses an actor's state change after fulfilling some task instance:

$$\rightarrow^* : \mathscr{AS} \times \mathscr{TI} \rightarrow \mathscr{AS} \tag{13}$$

Thus, an actor state t changes to state $t \rightarrow^* i$ after experiencing knowledge during the fulfillment of task instance i. However, to construct the improvability characteristic the actual improvement of cognitive characteristics should also be tackled. The following actor characteristics function can be utilized to solve this issue:

$$\mathsf{AChar} : \mathscr{AS} \rightarrow (\mathscr{AC} \rightarrow \wp(\mathscr{CC})) \tag{14}$$

This function specifies which cognitive characteristics as part of the set $\mathscr{CC}$ belong to a certain actor instance (that is classified by an actor type). The set $\mathscr{AS}$ contains *actor states*. An actor state is necessary here because the characterization of an actor might change over time. An actor $a \in \mathscr{AC}$ possessing cognitive characteristics included in a set of cognitive characteristics C while in state $t \in \mathscr{AS}$ can be denoted as: $\mathsf{AChar}_t(a) = C$. The improvability characteristic can be modeled subsequently:

$$\exists_{i \in \mathsf{Fulfillment}(a)} \exists_{t \in \mathscr{AS}} [\mathsf{AChar}_t(a) \subseteq \mathsf{AChar}_{t \rightarrow^* i}(a)] \tag{15}$$

An actor $a \in \mathscr{AC}$ has the improvability characteristic, denoted as $\mathsf{Improvability}(a)$, if the set of cognitive characteristics $\mathsf{AChar}_t(a) = C$ can be complemented with additional characteristics after fulfilling some task instance while being in some state $t \in \mathscr{AS}$.

3.2.5. Independency

An actor has the *independency* characteristic, if that actor is able to fulfill some task instance on its own. If an actor is fully able to fulfill a task instance on its own, then it can be said that an actor has the characteristic at a (very) high level and vice versa. A fulfiller function is necessary to reason specifically about actors that are fulfilling some task instance:

$$\mathsf{Fulfiller} : \mathscr{TI} \rightarrow \wp(\mathscr{AC}) \tag{16}$$

If it is necessary to determine *fulfillers* of a task instance i, the fulfiller function returns actors responsible for the fulfillment of some task. The independency characteristic can be modeled as follows:

$$\exists_{i \in \mathscr{TI}} [\mathsf{Fulfiller}(i) = \{a\}] \tag{17}$$

An actor a has the independency characteristic, denoted as $\mathsf{Independency(a)}$, if for task instance i the only fulfiller is actor a.

In order to have a graphical representation of the discussed definitions throughout Section 3, an Object-Role Modeling (ORM) model is presented in Fig. 2. In such a model, ovals represent object types (which are counterparts of classes), whereas boxes represent relations between object types. For more details on Object-Role Modeling, see e.g. [22].

4. Task types

Now that several actor types have been described together with the cognitive characteristics that can be supplied by actors that instantiate these types it is necessary to focus on the knowledge

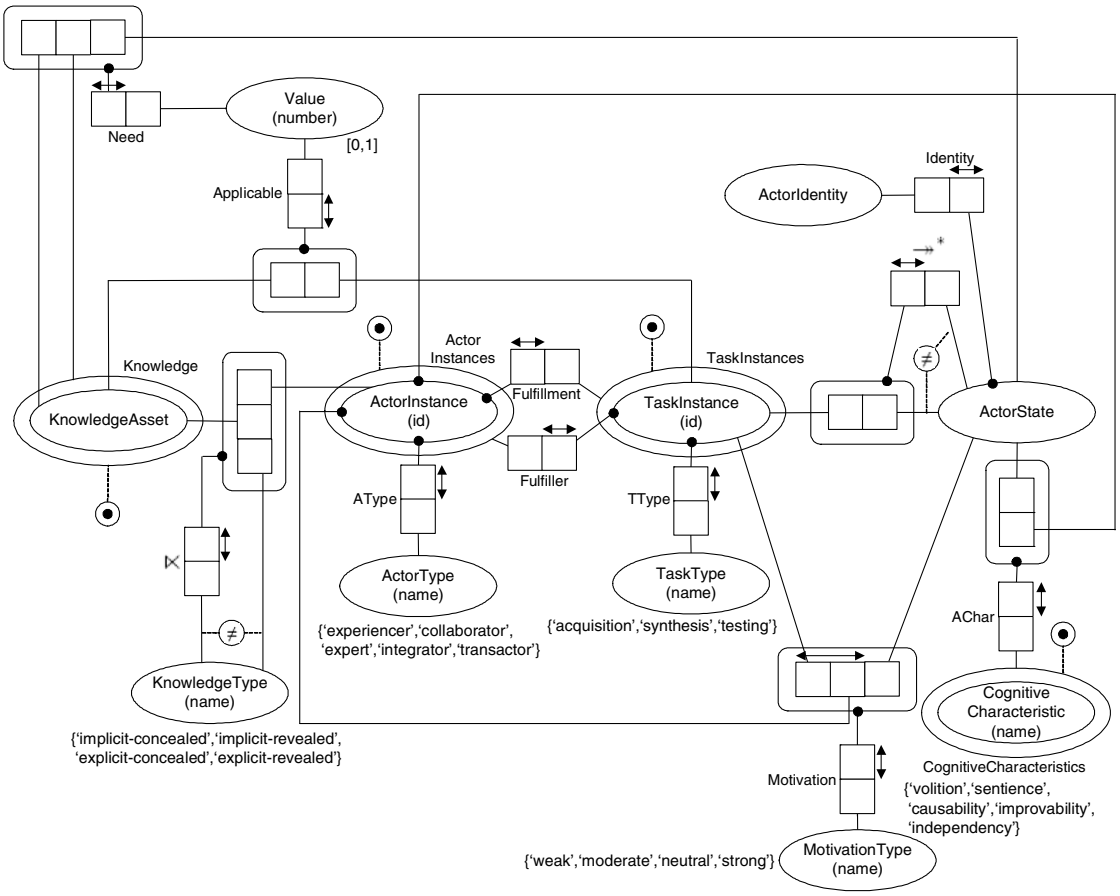

Fig. 2. Object-Role Modeling (ORM) model of cognitive actor settings.

intensive task types. A knowledge intensive task is a task for which acquisition, application or testing of knowledge is necessary in order to successfully fulfill the task. The following types are distinguished: The *acquisition* task type, the *synthesis* task type and the *testing* task type. As is elaborated in earlier work, possible knowledge intensive tasks that can be fulfilled can be abstracted to a pattern of three types [11]:

(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 following cognitive characteristics characterize these knowledge intensive task types:

- The *satisfaction* characteristic is related with a need for knowledge during a task's fulfillment and the eventual disappearance of that need. Suppose that a salesman requires insight in future developments of a certain market. Therefore, the salesman asks a colleague to provide a forecast of these developments. After interpreting the forecast, the salesman's need for this knowledge may have substantially decreased.
- *Relevance* is concerned with whether or not knowledge acquired is deemed appropriate during the fulfillment of a task. This is the case if, e.g. the salesman is not able to acquire the necessary knowledge by interpreting the aforementioned market forecast.
- The *applicability* characteristic expresses to what extent knowledge is applicable in a task. For instance, a requirements engineer interviews a customer to acquire certain requirements for an information system to be build. After the interview has been conducted the engineer has acquired a lot of knowledge about the customer's organization but not about requirements for the future system. In this example, the acquired knowledge is not very applicable for the task at hand.
- When knowledge is applied it should meet its requirements. This is indicated by the *correctness* characteristic. For instance, when a software developer writes code it should meet the requirements to be able to compile the code and to achieve a system that is working correctly.
- The *faultiness* characteristic is necessary to be able to determine whether or not applied knowledge contains flaws. For example, a software tester should be able to find bugs in software.
- To correct already applied knowledge containing flaws, the *rectification* characteristic can be determined. This may be the case when a software developer fixes a bug found by a software tester.

These cognitive characteristics indicate which cognitive characteristics are at least demanded to fulfill a task instance of a certain type. Table 2 shows how the task types are characterized. An acquisition task, for instance, demands the satisfaction and relevance characteristics. Thus, an actor should be satisfied after fulfilling an acquisition task and the acquired knowledge should also be relevant enough to fulfill the task. The formal definitions of these characteristics have already been discussed in [11] and will therefore not be repeated here. Now the set of task types can be represented as:

$$\{\mathtt{acquisition}, \mathtt{synthesis}, \mathtt{testing}\} \subseteq \mathscr{TT} \tag{18}$$

The set of cognitive characteristics can be represented as:

$$\{\mathtt{satisfaction}, \mathtt{relevance}, \mathtt{applicability}, \mathtt{correctness}, \mathtt{faultiness}, \mathtt{rectification}\} \subseteq \mathscr{CC} \tag{19}$$

An important remark to make here is that the possible task types as well as the possible cognitive characteristics are not limited to three task types and six cognitive characteristics. However, in this paper we restrict ourselves to the three defined task types together with the characteristics. Next, a framework for cognitive matchmaking can be elaborated. The defined actor types and task types can now be utilized in the framework of the system together with the cognitive characteristics.

5. Framework for cognitive matchmaking

In this section, a framework for cognitive matchmaking is introduced that is able to compute a match between cognitive characteristics required for a specific task type and cognitive characteristics that are provided by a specific actor type. As a running example, we use the matchmaking framework to match the cognitive characteristics offered by the *transactor* actor type with the required cognitive characteristics of a *synthesis* task. Fig. 3 shows the architecture of the system on a conceptual level, which is translated into the formalisms throughout this section. In Section 3.2.4, a function $\mathsf{AChar}_j(a) = C$ indicated the cognitive characteristics that characterized an actor instance of a certain type, where j is a task type belonging to the set of task types $\mathscr{TT}$, a is an actor instance belonging to the set of actor instances $\mathscr{AC}$ and C is a set of cognitive characteristics that is a subset of or equal to $\mathscr{CC}$. Recall from Section 3.1 that the corresponding actor type can be found by using the *actor type* function: $\mathsf{AType}(a) = j$. With this in mind, a *supply* function can be modeled that returns a value expressing to what extent an actor type offers a certain cognitive characteristic:

$$\mathsf{Supply} : \mathscr{AT} \rightarrow (\mathscr{CC} \rightarrow \mathscr{CRN}) \tag{20}$$

The expression $\mathsf{Supply}_{\mathtt{transactor}}(\mathtt{s}) = 10$ shows that an actor characterized by the transactor type offers the sentience characteristic and is at least capable to perform this characteristic at level 10. Note that the word 'sentience' has been abbreviated to the letter 's'. For readability reasons we will continue to use this abbreviation for the remaining example expressions. The resulting value '10' is part of a characteristic rank domain $\mathscr{CRN}$ which contains integer values within the range [0,10]. The hard values as part of a domain of values can be found using the following function:

Table 2
Knowledge intensive task types characterized

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

$$\text{Numerical} : \wp(\mathscr{RN}) \rightarrow \mathbb{R} \tag{21}$$

Here, the set $\mathscr{RN}$ contains rank values and $\mathscr{CRN} \subseteq \mathscr{RN}$. Formally, the characteristic rank domain includes the following hard values:

$$\text{Numerical}(\mathscr{CRN}) = [0, 10]$$

A value of 0 means that an actor is not able to offer a certain characteristic, a value of 5 means that an actor is able to offer a characteristic at an average level and a value of 10 means that an actor is able to offer a characteristic at the highest level. So, in the case of the example, the transactor is able to offer the sentience characteristic at the highest level.

It is possible here to introduce a characteristic rank set containing linguistic (soft) values instead of a characteristic rank set that contains numerical (hard) values. A linguistic value differs from a numerical value in that its values are not numbers but words or sentences in some language. The resulting values of the examples reported, however, are mapped on a domain containing hard values only. In the case of the match example above, this would mean that we are able to reason that the transactor is able to offer the sentience characteristic at, e.g. a *very high* level. These capabilities have not been added to the framework yet because successful development of the match algorithm itself has been the main concern so far. However, an exploration of how *fuzzy assessments* can be used to indicate a certain capability level can be found in Section 6. It is planned to integrate this in the next version of the cognitive matchmaking framework and the prototype.

Besides modeling a supply function, a demand function is needed that returns a value expressing to what extent a cognitive characteristic is *required* for a certain task type:

$$\text{Demand} : \mathscr{TT} \rightarrow (\mathscr{CC} \rightarrow \mathscr{CRN}) \tag{22}$$

The expression $\text{Demand}_{\texttt{synthesis}}(\texttt{s}) = 10$ indicates that a sentience characteristic is required at the highest level in order to fulfill a task of the synthesis type. The supply and demand functions can now be used together to compute the characteristic match.

5.1. Characteristic match

In this section, a characteristic match function is defined to compare the resulting values from the supply and demand functions. This comparison should provide insight in the way supply and demand of cognitive characteristics are matched. In order to model a *characteristic match* function, an actor type as well as a task type are required as input, together with a cognitive characteristic from the set $\mathscr{CC}$ of cognitive characteristics:

$$\text{CharMatch} : \mathscr{AT} \times \mathscr{TT} \rightarrow (\mathscr{CC} \rightarrow \mathscr{MRN}) \tag{23}$$

As can be seen in Fig. 3, the characteristic match function returns a value from the match rank domain, where $\mathscr{MRN} \subseteq \mathscr{RN}$. The match rank domain includes the following values: $\text{Numerical}(\mathscr{MRN}) = [0, 10]$.

To compute the actual characteristic match value, a *proximity* function is necessary to be able to define the characteristic match function. This proximity function should compute the proximity of the level an actor offers a certain cognitive characteristic related to the level that is required in order to fulfill a task of a certain type. The values that should be used as input for the proximity function are part of the characteristic rank domain. The resulting proximity value is then a value that is part of the match rank domain:

$$\text{Proximity} : \mathscr{CRN} \times \mathscr{CRN} \rightarrow \mathscr{MRN} \tag{24}$$

A normalization function can be introduced that calculates the numerical proximity of supply and demand when a cognitive characteristic is concerned:

$$\text{Normalize} : \mathbb{R} \rightarrow [0, 1] \tag{25}$$

The normalization function can be defined by using the supply and demand functions and two additional constants $\texttt{min}$ and $\texttt{max}$:

$$\text{Normalize}(\text{Supply}_i(c) - \text{Demand}_j(c)) \triangleq \frac{\text{Supply}_i(c) - \text{Demand}_j(c) + \texttt{max} - \texttt{min}}{2 \cdot (\texttt{max} - \texttt{min})} \tag{26}$$

Here, i is an actor type of the set $\mathscr{AT}$, j is a task type of the set $\mathscr{TT}$ and c is a cognitive characteristic of the set $\mathscr{CC}$. The values of the constants $\texttt{min}$ and $\texttt{max}$ can be determined by interpreting the minimum and the maximum value of the characteristic rank domain. So, in the case of the running example $\texttt{min} = \texttt{0}$ and $\texttt{max} = \texttt{10}$. The minimum value that can be returned by the normalization function is 0. This occurs if there is absolutely no supply (i.e. an incapable actor is concerned) but there is a maximum demand of a certain cognitive characteristic in order to fulfill a task of a certain type. This situation is depicted below:

$$\text{Normalize}(0 - 10) = \frac{0 - 10 + \texttt{max} - \texttt{min}}{2 \cdot (\texttt{max} - \texttt{min})} = 0$$

The normalization function returns 1 in case of an overqualified actor that is capable to perform a cognitive characteristic at the highest level whilst the characteristic is not demanded at all:

$$\text{Normalize}(10 - 0) = \frac{10 - 0 + \texttt{max} - \texttt{min}}{2 \cdot (\texttt{max} - \texttt{min})} = 1$$

This means that the normalization function normalizes the proximity of supply and demand between 0 and 1. Using the normalization function, the proximity function can now be defined as follows:

$$\text{Proximity}(\text{Supply}_i(c), \text{Demand}_j(c)) \triangleq \text{Normalize}(\text{Supply}_i(c) - \text{Demand}_j(c)) \tag{27}$$

Regarding the running example the proximity function as defined above results in:

$$\text{Proximity}(10, 10) = \text{Normalize}(10 - 10) = 0.5$$

Now with the introduction of a proximity function the characteristic match can be defined by computing the proximity of supply and demand in the context of a given characteristic:

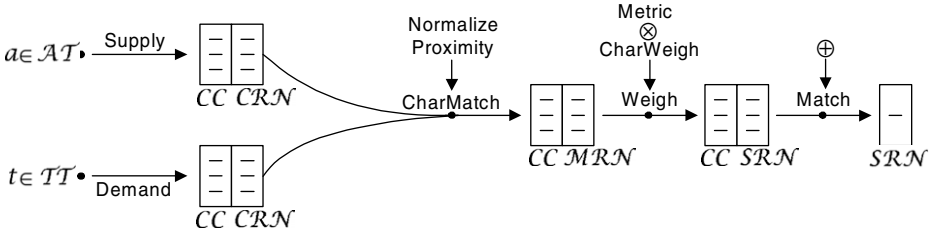

Fig. 3. Framework for cognitive matchmaking.

$$\mathsf{CharMatch}(i,j) \triangleq \lambda_{c\in\mathscr{CC}} \cdot \mathsf{Proximity}(\mathsf{Supply}_i(c), \mathsf{Demand}_j(c)) \quad (28)$$

Recall from Section 5 that an actor of the transactor type is able to perform the sentience characteristic at level 10, which equals the level to what extent a sentience characteristic should be mastered for a synthesis task type. In the case of our example the characteristic match results in:

$$\begin{aligned}&\mathsf{CharMatch}(\mathtt{transactor},\mathtt{synthesis}) =\\ &\mathsf{Proximity}(\mathsf{Supply}_{\mathtt{transactor}}(\mathtt{s}), \mathsf{Demand}_{\mathtt{synthesis}}(\mathtt{s})) =\\ &\mathsf{Proximity}(10,10) = 0.5\end{aligned}$$

This example shows that if an actor characterized as a transactor masters a sentience characteristic at level 10 and if it is also *needed* to master the sentience characteristic at level 10 to fulfill a task instance of the synthesis type, the eventual *proximity value* is 0.5. However, this proximity value is only related to the supply and demand of one specific cognitive characteristic. To compute a total match of the required cognitive characteristics in a task type and the characteristics offered, a *weighed suitability match* is introduced in the following section.

5.2. Weighed suitability

The cognitive matchmaking framework is completed by introducing a weighed suitability match, as is shown in the rightmost part of Fig. 3. The underlying match function has the following signature:

$$\mathsf{Match} : \mathscr{AT} \times \mathscr{TT} \to \mathscr{SRN} \quad (29)$$

This function returns a value from the suitability rank domain, where $\mathscr{SRN} \subseteq \mathscr{RN}$. The suitability rank domain includes the following values:

$$\mathsf{Numerical}(\mathscr{SRN}) = [0,10]$$

This means that an actor of a certain type can have suitability levels ranging from 0 to 10. To determine the suitability of the transactor fulfilling the synthesis task, the calculated proximity of supply and demand of a cognitive characteristic $c \in \mathscr{CC}$ can be weighed:

$$\mathsf{Weigh} : (\mathscr{CC} \to \mathscr{MRN}) \to (\mathscr{CC} \to \mathscr{SRN}) \quad (30)$$

To define the weigh function several other functions are necessary, though. As can be seen in Fig. 3, the weigh function uses the input from the characteristic match function and returns a value from the suitability rank domain as output. To construct the weigh function, a function is needed that has a match rank metric (i.e. the proximity value) as its input and a suitability rank metric as its output:

$$\mathsf{Metric} : \mathscr{MRN} \to \mathscr{SRN} \quad (31)$$

For instance, $\mathsf{Metric}(0.5) = 0.5$ shows that the value 0.5, which is the proximity value, equals the value 0.5 which is a suitability rank metric. A characteristic weigh function is needed to actually weigh the importance of a certain cognitive characteristic to fulfill a task of a certain type:

$$\mathsf{CharWeigh} : \mathscr{CC} \to \mathscr{SRN} \quad (32)$$

So, $\mathsf{CharWeigh}(\mathtt{s}) = 1.5$ means that a weigh factor of 1.5 is given to indicate the importance of mastering the sentience cognitive characteristic (for a certain task). Finally, the $\otimes$ operator is also needed to define a definite weigh function:

$$\otimes : \mathscr{SRN} \times \mathscr{SRN} \to \mathscr{SRN} \quad (33)$$

The $\otimes$ operator is necessary to multiply the metric value with the characteristic weigh value. If the values mentioned above are multiplied this results in $0.5 \otimes 1.5 = 0.75$. In case the underlying ranking domain contains real members, the $\otimes$ operator is the normal multiplier. Otherwise, a separate $\otimes$ operator is used. The weigh function can now be defined as:

$$\mathsf{Weigh}(c, \mathsf{CharMatch}(i,j)) \triangleq \lambda_{c\in\mathscr{CC}} \cdot \mathsf{Metric}(\mathsf{CharMatch}(i,j)) \otimes \mathsf{CharWeigh}(c) \quad (34)$$

Here, $c \in \mathscr{CC}, i \in \mathscr{AT}$ and $j \in \mathscr{TT}$. Continuing the running example, we would like to calculate the suitability of the transactor that is fulfilling a task instance of the synthesis type. Considering the sentience characteristic only, this can be computed as follows:

$$\begin{aligned}&\mathsf{Weigh}(\mathtt{s}, \mathsf{CharMatch}(\mathtt{transactor},\mathtt{synthesis})) =\\ &\mathsf{Metric}(0.5) \otimes \mathsf{CharWeigh}(\mathtt{s}) =\\ &0.5 \otimes 1.5 = 0.75\end{aligned}$$

In order to calculate the suitability match of the transactor actor type related to the synthesis task type of our example, it is mandatory to determine the cognitive characteristics supplied by the actor and demanded by the task. The transactor actor type supplies the *volition*, *sentience*, and *independency* characteristics as is shown in Table 1. The synthesis task type can be characterized by the *applicability* and *correctness* characteristics as is shown in Table 2.

In the case of the running example (i.e. only when the transactor actor type and the synthesis task type are concerned) the set $\mathscr{CC}$ contains the following characteristics:

$$\{\mathtt{volition}, \mathtt{sentience}, \mathtt{independency}, \mathtt{applicability}, \mathtt{correctness}\} \subseteq \mathscr{CC}$$

For all these properties a weigh value needs to be determined using the functions mentioned throughout Section 5. This is necessary to compute a final *suitability match* resulting in one suitability rank value. The calculations leading to weighed characteristic matches are elaborated in the Tables 3 and 4.

The actual characteristic weigh values (for every cognitive characteristic as part of the set $\mathscr{CC}$) denoted in Table 4 are: 2, 1.5, 0.5, 3 and 3. Note that these characteristic weigh values always summate to one and the same total value. In the case of our example the characteristic weigh values summate to 10. Thus, no matter how the weigh values are divided across the cognitive characteristics, they should always summate to a total of 10.

5.3. Suitability match

The results of the weighed characteristic matches, which are denoted in the rightmost column of Table 4, have to be summated to generate a single *suitability match* value. To summate these values a $\oplus$ operator is required:

$$\oplus : \mathscr{SRN} \times \mathscr{SRN} \to \mathscr{SRN} \quad (35)$$

Now the final match function can be defined using the aforementioned functions:

$$\mathsf{Match}(i,j) \triangleq \bigoplus_{c\in\mathscr{CC}} \mathsf{Weigh}(c, \mathsf{CharMatch}(i,j)) \quad (36)$$

In the match function $i \in \mathscr{AT}, c \in \mathscr{CC}$ and $j \in \mathscr{TT}$. For the running example this means that the suitability match value of the transactor fulfilling a task instance of the synthesis type is computed as follows:

$$\begin{aligned}&\mathsf{Match}(\mathtt{transactor},\mathtt{synthesis}) = 1 \oplus 0.75 \oplus 0.35 \oplus 1.35 \oplus 1.2\\ &= 4.65\end{aligned}$$

As a result of the suitability match it can be concluded that the suitability of an actor characterized by the transactor type fulfilling a task instance of the synthesis type is 4.65. Remember that the lowest suitability value is 0 and the highest suitability value that can be reached is 10. The lowest value is reached if the supply of every characteristic is 0 and the demand of every characteristic is 10. The highest value is reached in the case of complete overqualification, i.e. if the supply of every characteristic is 10 and the demand of every characteristic is 0. At this point a decision can be

Table 3
Example calculations for characteristic matches

Item	Characteristic	Characteristic Match
a.	volition	CharMatch(transactor,synthesis) = Proximity(10,10) = 0.5
b.	sentience	CharMatch(transactor,synthesis) = Proximity(10,10) = 0.5
c.	independency	CharMatch(transactor,synthesis) = Proximity(10,6) = 0.7
d.	applicability	CharMatch(transactor,synthesis) = Proximity(7,8) = 0.45
e.	correctness	CharMatch(transactor,synthesis) = Proximity(6,8) = 0.4

Table 4
Example calculations for weighed characteristic matches

Item	Weighed Characteristic Match
a.	Weigh(volition,0.5) = Metric(0.5) ⊗ CharWeigh(volition) = 0.5 ⊗ 2 = 1
b.	Weigh(sentience,0.5) = Metric(0.5) ⊗ CharWeigh(sentience) = 0.5 ⊗ 1.5 = 0.75
c.	Weigh(indepedency,0.7) = Metric(0.7) ⊗ CharWeigh(independency) = 0.7 ⊗ 0.5 = 0.35
d.	Weigh(applicability,0.45) = Metric(0.45) ⊗ CharWeigh(applicability) = 0.45 ⊗ 3 = 1.35
e.	Weigh(correctness,0.4) = Metric(0.4) ⊗ CharWeigh(correctness) = 0.4 ⊗ 3 = 1.2

made whether or not the actor is suitable enough to fulfill this specific task or if another actor is present that is more suitable, i.e. has a better suitability match value. The suitability of an actor to fulfill a certain task is best if the resulting suitability value is 5. Underqualification as well as overqualification are both considered undesirable.

A certainty function can now be introduced to make sure how certain it is that an actor is suitable to fulfill a task:

$$\mu : \mathbb{R} \to [0,1] \tag{37}$$

A linear certainty function can be defined as follows:

$$\mu(u) \triangleq \begin{cases} \frac{2}{\mathtt{min}+\mathtt{max}} \cdot u & \mathtt{min} \leqslant \mathtt{u} \leqslant \frac{\mathtt{max}+\mathtt{max}}{2} \\ \frac{-2}{\mathtt{min}+\mathtt{max}} \cdot u + 2 & \frac{\mathtt{min}+\mathtt{max}}{2} \leqslant u \leqslant \mathtt{max} \end{cases} \tag{38}$$

For the running example, where $\mathtt{min} = 0$ and $\mathtt{max} = 10$, the following expression shows that the certainty that the transactor is suitable to fulfill the synthesis task is 0.93:

$$\mu(4.65) = \frac{2}{0+10} \cdot 4.65 = 0.93$$

This can be interpreted as being 93% sure that the transactor is suitable enough to fulfill the synthesis task. It might be a good choice to let the transactor fulfill the synthesis task, unless an available actor characterized by another type provides a better match.

Throughout Section 5 definitions have been discussed along with their corresponding examples. Table 5 provides an overview of the definitions and the examples. In order to also have a graphical representation of the discussed definitions throughout Section 5, another ORM model is presented in Fig. 4. All formalisms mentioned up till now are visualized by means of the ORM models of Figs. 2 and 4.

6. Fuzzy match assessments

The variables used while calculating with the functions of the cognitive matchmaker system were non-fuzzy. This means that the variables that were used to compute the eventual suitability match comprised numerical (hard) values. Zadeh's fuzzy logic research (see e.g. [23]) can be utilized to reason about fuzzy match assessments. A linguistic variable differs from a numerical variable in that its values are not numbers but words or sentences in some language. For example, the linguistic variable *length* might take *very small*, *small*, *average*, *tall*, or *very tall* as its values. Throughout the running example of Section 5, the 'numerical' function returned the hard values as part of a domain of values. In case of the characteristic rank domain the numerical function returned the values as part of that domain: $\mathsf{Numerical}(\mathscr{CRN}) = [0,10]$. These values could then be used to determine to what extent a characteristic was supplied by an actor, respectively, demanded by a task. However, it can be more meaningful to express the level of the supply or demand of a characteristic by using a linguistic value. For example, the transactor offers the sentience characteristic at level 10. From a linguistic point of view, this numerical value can be expressed by the value *very high*. Furthermore, such linguistic variables can be used to compute match rank values and eventually suitability rank values. In this section we will elaborate on this concept and it is shown how fuzzy assessments can be used in the cognitive matchmaker system.

First, the linguistic values of a ranking domain $\mathscr{RN}$ can be determined by the following function:

$$\mathsf{Linguistic} : \wp(\mathscr{RN}) \to \wp(\mathscr{LV}) \tag{39}$$

Here, the set $\mathscr{LV}$ contains linguistic values. Recall that the cognitive matchmaking framework incorporates three different ranking domains: the characteristic rank domain, the match rank domain and the suitability rank domain. The expression $\mathsf{Linguistic}(\mathscr{CRN}) \supseteq \{\mathtt{very-low}, \mathtt{low}, \mathtt{medium}, \mathtt{high}, \mathtt{very-high}\}$ shows that the characteristic rank variable takes *very low*, *low*, *medium*, *high* and *very high* as its values. To understand the use of fuzzy assessments in cognitive matchmaking, a membership function needs to be introduced.

The main distinction between fuzzy variables and non-fuzzy variables lies in this membership function. In case of fuzzy variables, the assignment of a value to a variable has a membership degree which expresses to what extent a variable has a certain value. The membership functions related to the linguistic characteristic rank values are illustrated in Fig. 5. Note that $\mathtt{min} = 0$ and $\mathtt{max} = 10$. The membership function can be used to understand how certain it is that the level on which a characteristic is supplied

Table 5
Definitions of the cognitive matchmaking framework with examples

Function	Example
$\mathsf{Supply} : \mathscr{AT} \to (\mathscr{CC} \to \mathscr{CRN})$	$\mathsf{Supply}_{\mathtt{transactor}}(\mathtt{s}) = 10$
$\mathsf{Numerical} : \wp(\mathscr{RN}) \to \mathbb{R}$	$\mathsf{Numerical}(\mathit{CRN}) = [0,10]$
$\mathsf{Demand} : \mathscr{TT} \to (\mathscr{CC} \to \mathscr{CRN})$	$\mathsf{Demand}_{\mathtt{synthesis}}(\mathtt{s}) = 10$
$\mathsf{CharMatch} : \mathscr{AT} \times \mathscr{TT} \to (\mathscr{CC} \to \mathscr{MRN})$	$\mathsf{CharMatch}(\mathtt{transactor}, \mathtt{synthesis}) = \mathsf{Proximity}(\mathsf{Supply}_{\mathtt{transactor}}(\mathtt{s}), \mathsf{Demand}_{\mathtt{synthesis}}(\mathtt{s})) = \mathsf{Proximity}(10,10) = 0.5$
$\mathsf{Proximity} : \mathscr{CRN} \times \mathscr{CRN} \to \mathscr{MRN}$	$\mathsf{Proximity}(10,10) = \mathsf{Normalize}(10-10) = 0.5$
$\mathsf{Normalize} : \mathbb{R} \to [0,1]$	$\mathsf{Normalize}(10-10) = \frac{10-10+\mathtt{max}-\mathtt{min}}{2\cdot(\mathtt{max}-\mathtt{min})} = 0.5$
$\mathsf{Match} : \mathscr{AT} \times \mathscr{TT} \to \mathscr{SRN}$	$\mathsf{Match}(\mathtt{transactor}, \mathtt{synthesis}) = 1 \oplus 0.75 \oplus 0.35 \oplus 1.35 \oplus 1.2 = 4.65$
$\mathsf{Weigh} : (\mathscr{CC} \to \mathscr{MRN}) \to (\mathscr{CC} \to \mathscr{SRN})$	$\mathsf{Weigh}(\mathtt{s}, \mathsf{CharMatch}(\mathtt{transactor}, \mathtt{synthesis})) = \mathsf{Metric}(0.5) \otimes \mathsf{CharWeigh}(\mathtt{s}) = 0.5 \otimes 1.5 = 0.75$
$\mathsf{Metric} : \mathscr{MRN} \to \mathscr{SRN}$	$\mathsf{Metric}(0.5) = 0.5$
$\mathsf{CharWeigh} : \mathscr{CC} \to \mathscr{SRN}$	$\mathsf{CharWeigh}(\mathtt{s}) = 1.5$
$\otimes : \mathscr{SRN} \times \mathscr{SRN} \to \mathscr{SRN}$	$0.5 \otimes 1.5 = 0.75$
$\oplus : \mathscr{SRN} \times \mathscr{SRN} \to \mathscr{SRN}$	$1 \oplus 0.75 \oplus 0.35 \oplus 1.35 \oplus 1.2 = 4.65$
$\mu : \mathbb{R} \to [0,1]$	$\mu(4.65) = 0.93$

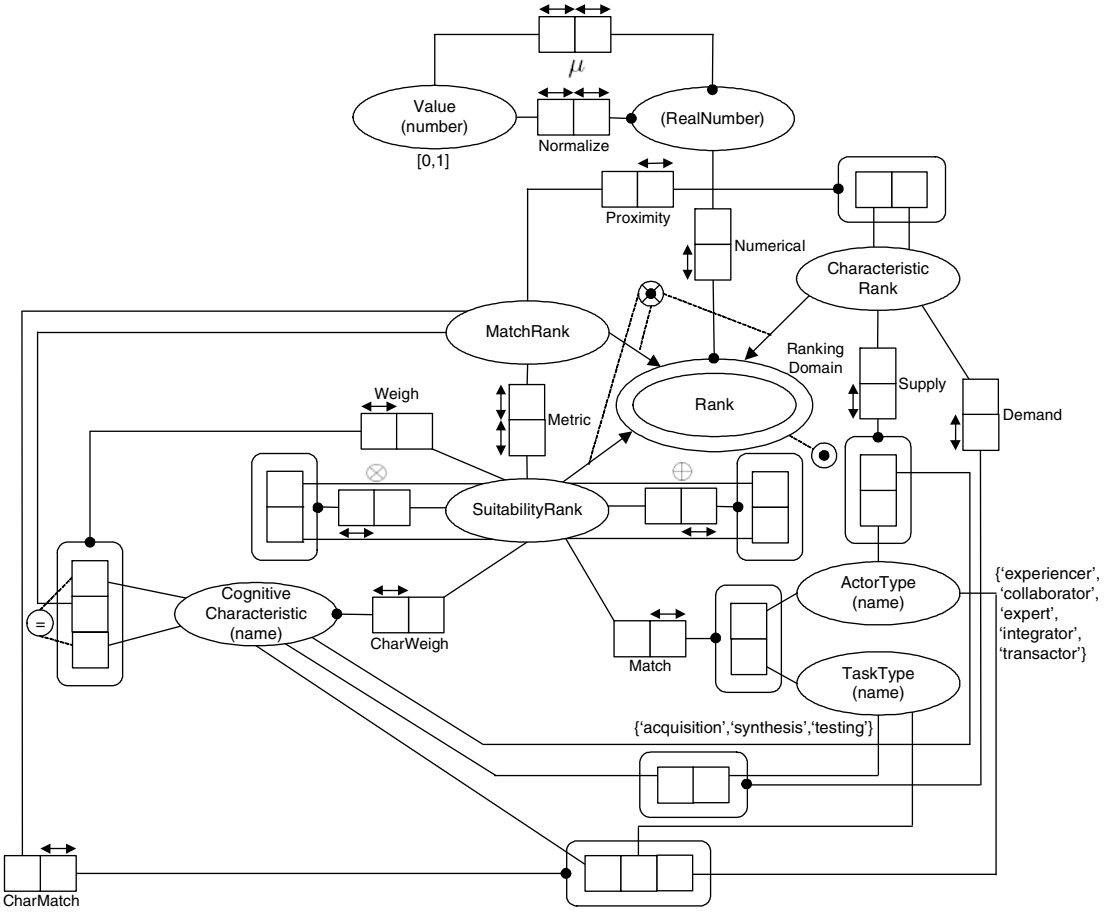

Fig. 4. Object-Role Modeling (ORM) model of the cognitive matchmaking framework.

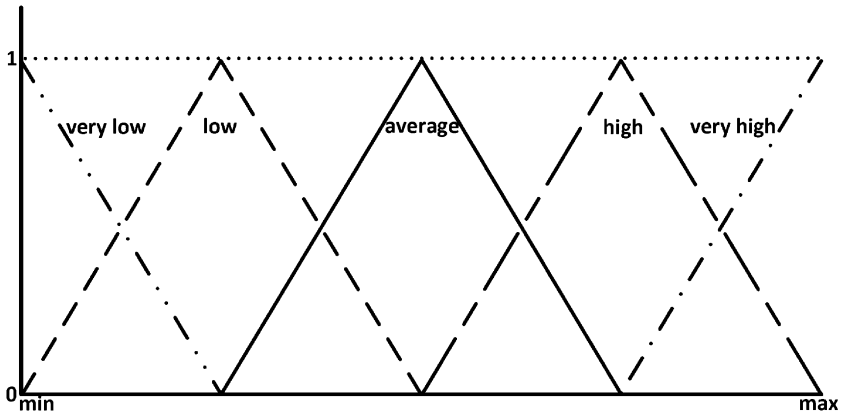

Fig. 5. Membership functions for the values of the linguistic variable 'characteristic rank'.

or demanded will be assessed by one of the available linguistic values. However, a probability function needs to be defined based on the membership function to actually calculate certainties. First, the membership function can be modeled as follows:

$$\mathsf{M} : \mathscr{LV} \rightarrow (\mathbb{R} \rightarrow [0,1]) \tag{40}$$

The expression $\mathsf{M}_{\mathtt{average}}(5) = 1$ can be interpreted as being 100% sure that supply or demand of a cognitive characteristic at level 5 is interpreted as the supply or demand at an 'average' level. In other words, the membership degree for the linguistic value 'average' is 1% or 100%. If we would like to know how certain it is that a cognitive characteristic is supplied or demanded on a certain level, a probability function is required. Assume that such a probability function has the same signature as function (40) above:

$$\mathsf{P} : \mathscr{LV} \rightarrow (\mathbb{R} \rightarrow [0,1]) \tag{41}$$

The probability function can then be defined as follows:

$$\mathsf{P}_v(u) \triangleq \int_0^u \mathsf{M}_v(u)du \tag{42}$$

Before we can compute $\mathsf{P}(5 = \mathtt{average})$ we must define the membership function for the linguistic value 'average':

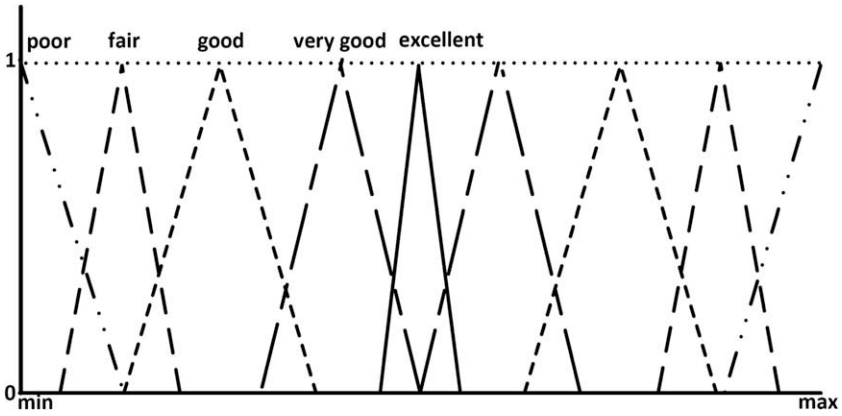

Fig. 6. Membership functions for the values of the linguistic variable 'match rank'.

$$\mathsf{M}_{\mathtt{average}}(u) \triangleq \begin{cases} \frac{2}{5}\cdot u - 1 & 2\frac{1}{2} \leqslant u \leqslant 5 \\ -\frac{2}{5}\cdot u + 3 & 5 \leqslant u \leqslant 7\frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{43}$$

The expression $\mathsf{P}_{\mathtt{average}}(5) = 1$ indicates that we are approximately 100% certain that the supply or demand of a cognitive characteristic will be assessed as 'average' when it is supplied or demanded at level 5. For example, the expert supplies the independency characteristic at level 5. It is now approximately 100% certain that the expert is able to offer this characteristic at an average level. The definitions of the remaining membership functions are presumed to be:

$$\mathsf{M}_{\mathtt{very-low}}(u) \triangleq \begin{cases} -\frac{2}{5}\cdot u + 1 & 0 \leqslant u \leqslant 2\frac{1}{2} \\ 0 & \text{otherwise} \end{cases} \tag{44}$$

$$\mathsf{M}_{\mathtt{low}}(u) \triangleq \begin{cases} \frac{2}{5}\cdot u & 0 \leqslant u \leqslant 2\frac{1}{2} \\ -\frac{2}{5}\cdot u + 2 & 2\frac{1}{2} \leqslant u \leqslant 5 \\ 0 & \text{otherwise} \end{cases} \tag{45}$$

$$\mathsf{M}_{\mathtt{high}}(u) \triangleq \begin{cases} \frac{2}{5}\cdot u - 2 & 5 \leqslant u \leqslant 7\frac{1}{2} \\ -\frac{2}{5}\cdot u + 4 & 7\frac{1}{2} \leqslant u \leqslant 10 \\ 0 & \text{otherwise} \end{cases} \tag{46}$$

$$\mathsf{M}_{\mathtt{very-high}}(u) \triangleq \begin{cases} \frac{2}{5}\cdot u - 3 & 7\frac{1}{2} \leqslant u \leqslant 10 \\ 0 & \text{otherwise} \end{cases} \tag{47}$$

When analyzing the expert and the synthesis task, it is now trivial to verify that:

- The certainty that the expert's supply of the causability characteristic is indeed interpreted as 'high': $\mathsf{P}_{\mathtt{high}}(7) = 0.80$.
- The certainty that the expert's supply of the causability characteristic is indeed interpreted as 'average': $\mathsf{P}_{\mathtt{average}}(7) = 0.20$.
- The certainty that the demand of the synthesis task with respect to the causability characteristic is indeed interpreted as 'average': $\mathsf{P}_{\mathtt{average}}(4) = 0.60$.
- The certainty that the demand of the synthesis task with respect to the causability characteristic is indeed interpreted as 'low': $\mathsf{P}_{\mathtt{low}}(4) = 0.40$.

Now that fuzzy assessments can be made concerning the supply and demand of cognitive characteristics, it is logical to explore the possibility of fuzzy assessments for match results. Table 3 for instance shows the characteristic match results for the transactor related with a synthesis task. Clarification of the linguistic values related to the match rank variable is necessary to assess the characteristic matches in a fuzzy way. The expression $\mathsf{Linguistic}(\mathit{MRN}) \supseteq \{\mathtt{poor}, \mathtt{fair}, \mathtt{good}, \mathtt{very-good}, \mathtt{excellent}\}$ shows that the match rank variable takes *poor*, *fair*, *good*, *very good* and *excellent* as its values. The corresponding membership functions are shown in Fig. 6. Note that $\mathtt{min} = 0$ and $\mathtt{max} = 1$ in Fig. 6, because the characteristic match results are never greater than 1. When a match result of 0.5 is achieved, the certainty is 100% that one assesses that match as 'excellent'. The more a match result diverges from this value towards 0 or 1, the worse a match result is assessed. If the match result lies between 0 and 0.5 an actor is underqualified for the task. If the match result lies between 0.5 and 1 an actor is overqualified for the task. This causes the membership functions to be symmetrical. The membership functions for the values of the match rank variable are presumed to be:

$$\mathsf{M}_{\mathtt{poor}}(u) \triangleq \begin{cases} -8\cdot u + 1 & 0 \leqslant u \leqslant \frac{1}{8} \\ 0 & \frac{1}{8} \leqslant u \leqslant \frac{7}{8} \\ 8\cdot u - 7 & \frac{7}{8} \leqslant u \leqslant 1 \end{cases} \tag{48}$$

$$\mathsf{M}_{\mathtt{fair}}(u) \triangleq \begin{cases} 0 & 0 \leqslant u \leqslant \frac{1}{20} \\ 13\frac{1}{3}\cdot u - \frac{2}{3} & \frac{1}{20} \leqslant u \leqslant \frac{1}{8} \\ -13\frac{1}{3}\cdot u + 2\frac{2}{3} & \frac{1}{8} \leqslant u \leqslant \frac{1}{5} \\ 0 & \frac{1}{5} \leqslant u \leqslant \frac{4}{5} \\ 13\frac{1}{3}\cdot u - 10\frac{2}{3} & \frac{4}{5} \leqslant u \leqslant \frac{7}{8} \\ -13\frac{1}{3}\cdot u + 12\frac{2}{3} & \frac{7}{8} \leqslant u \leqslant \frac{19}{20} \\ 0 & \frac{19}{20} \leqslant u \leqslant 1 \end{cases} \tag{49}$$

$$\mathsf{M}_{\mathtt{good}}(u) \triangleq \begin{cases} 0 & 0 \leqslant u \leqslant \frac{1}{8} \\ 8\cdot u - 1 & \frac{1}{8} \leqslant u \leqslant \frac{1}{4} \\ -8\cdot u + 3 & \frac{1}{4} \leqslant u \leqslant \frac{3}{8} \\ 0 & \frac{3}{8} \leqslant u \leqslant \frac{5}{8} \\ 8\cdot u - 5 & \frac{5}{8} \leqslant u \leqslant \frac{3}{4} \\ -8\cdot u + 7 & \frac{3}{4} \leqslant u \leqslant \frac{7}{8} \\ 0 & \frac{7}{8} \leqslant u \leqslant 1 \end{cases} \tag{50}$$

$$\mathsf{M}_{\mathtt{very-good}}(u) \triangleq \begin{cases} 0 & 0 \leqslant u \leqslant \frac{3}{10} \\ 10\cdot u - 3 & \frac{3}{10} \leqslant u \leqslant \frac{2}{5} \\ -10\cdot u + 5 & \frac{2}{5} \leqslant u \leqslant \frac{1}{2} \\ 10\cdot u - 5 & \frac{1}{2} \leqslant u \leqslant \frac{3}{5} \\ -10\cdot u + 7 & \frac{3}{5} \leqslant u \leqslant \frac{7}{10} \\ 0 & \frac{7}{10} \leqslant u \leqslant 1 \end{cases} \tag{51}$$

$$\mathsf{M}_{\mathtt{excellent}}(u) \triangleq \begin{cases} 0 & 0 \leqslant u \leqslant \frac{9}{20} \\ 20\cdot u - 9 & \frac{9}{20} \leqslant u \leqslant \frac{1}{2} \\ -20\cdot u + 11 & \frac{1}{2} \leqslant u \leqslant \frac{11}{20} \\ 0 & \frac{11}{20} \leqslant u \leqslant 1 \end{cases} \tag{52}$$

An example of a fuzzy assessment for a characteristic match result can be given as follows. Table 3 shows that the applicability characteristic match is 0.45 for the transactor/synthesis task com-

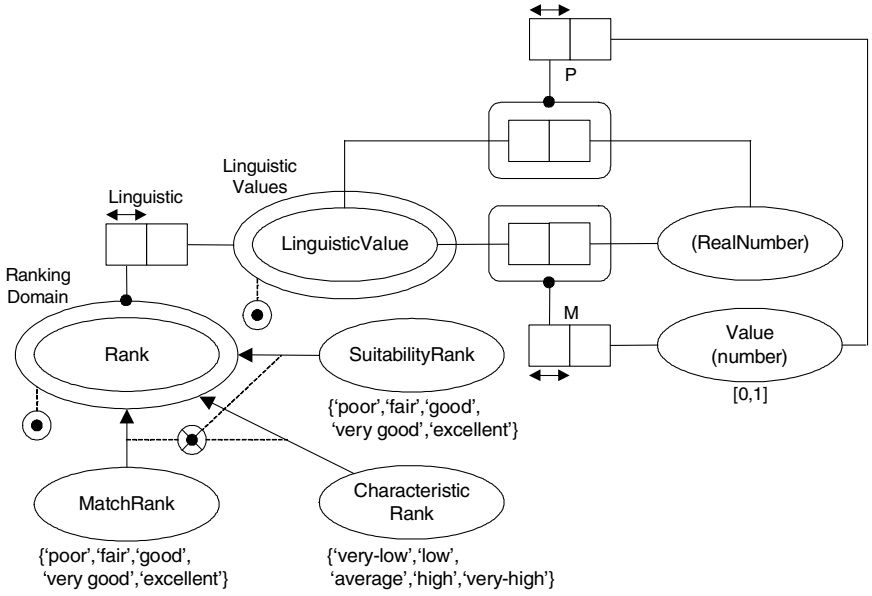

Fig. 7. Object-Role Modeling (ORM) model of the fuzzy match functions.

bination. The certainty that this applicability characteristic match is indeed interpreted as 'good' is $P_{good}(0.45) = 0$. The certainty that this applicability characteristic match is indeed interpreted as 'very good' is $P_{very-good}(0.45) = 0.5$. In other words, the certainty that someone interprets this match result as 'very good' is 50%.

Eventually, fuzzy assessments for suitability match results can also be provided after weighing the characteristic match results and the summation of the weighed match results. The same membership functions as pictured in Fig. 6 can be used. Unlike a characteristic match result, a suitability match result can vary over the values from 0 up to and including 10. This causes $\mathtt{min} = 0$ and $\mathtt{max} = 10$. The expression $\mathrm{Linguistic}(\mathscr{SRN}) = \{\mathtt{poor}, \mathtt{fair}, \mathtt{good}, \mathtt{very-good}, \mathtt{excellent}\}$ shows that the suitability rank variable also takes *poor*, *fair*, *good*, *very good* and *excellent* as its values. The definitions of the membership functions for the values of the linguistic variable 'suitability rank' are considered trivial. Recall that the suitability match of the transactor/ synthesis task combination shown in Section 5.3, equalled 4.65. The probability function can be used to determine the certainty that this suitability match is interpreted as a 'good' match. This is specified by the following expression: $P_{good}(4.65) = 0$. The certainty is 0% that one assesses the suitability of the transactor fulfilling the synthesis task as very good. The certainty that this suitability match is assessed as 'very good' can be calculated as follows: $P_{very-good}(4.65) = 0.35$. Thus, the certainty is 35% that one assesses the suitability of the transactor fulfilling the synthesis task as very good. Finally, the certainty is 30% that the result is assessed as excellent: $P_{excellent}(4.65) = 0.3$. The functions discussed in this section can be graphically supported by the ORM model of Fig. 7. Now that the framework and the fuzzy match extension of cognitive matchmaking have been discussed it is necessary to determine if a system can be build by means of a prototype implementation.

7. Prototype of the cognitive matchmaker system

The prototype of the cognitive matchmaker system has been designed as a Web application according to the three tier software architecture depicted in Fig. 8. The graphical user interface is based on the Microsoft.NET Framework 2.0 Web UI namespace that provides classes and interfaces to create user interface elements. The business layer includes the main components of the application. The most important one is the kernel, which is an implementation of the formal functions shown in Fig. 3 and in Table 5. Furthermore, the 'matching factory' instantiates all the objects involved when a suitability match should be calculated and enables the application to follow the flow of the matchmaking process as depicted in Fig. 3. The business layer also includes an implementation of the possible ranking domains that can include characteristic ranks, match ranks and suitability ranks. The data layer includes code to interact with connected databases. The architecture shows that it is possible to include a *project-specific* database as well as a database including *abstract* types and characteristics. This signifies that the prototype of the cognitive matchmaker system can compute matches between project-specific actor types and task types as well as between the abstract actor types and task types we have defined in our framework. Project-specific actor types and task types are types that can be defined to categorize all the actors and tasks that are part of a specific project. For instance, a person called 'John Doe' working on a software project can be categorized as a project-specific actor type 'developer' for instance meaning that he acts as a software developer in a specific software project. An information systems engineering method is often used during the enactment of a project. An example of such a method is the Microsoft Solutions Framework (MSF) (see e.g. [24]). Such methods often include pre-defined actor types and/or task types that can be instantiated when applying the method. Once these types are added to a project-specific database they can be used to determine matches. Section 8.1 includes the project-specific actor types and task types as part of the elaborated case study. Dependent of the choice the user of the cognitive matchmaker system makes, the system communicates with one of the available databases to calculate matches. The data layer is based on the Microsoft Enterprise Library 3.0 that already contains pre-defined chunks of source code for, e.g. data access and exception handling.

The user of the cognitive matchmaker system has to walk through six steps to let the system calculate a suitability match. In the first step, the user should select an actor type and a task type for which a suitability match should be calculated. Suppose that

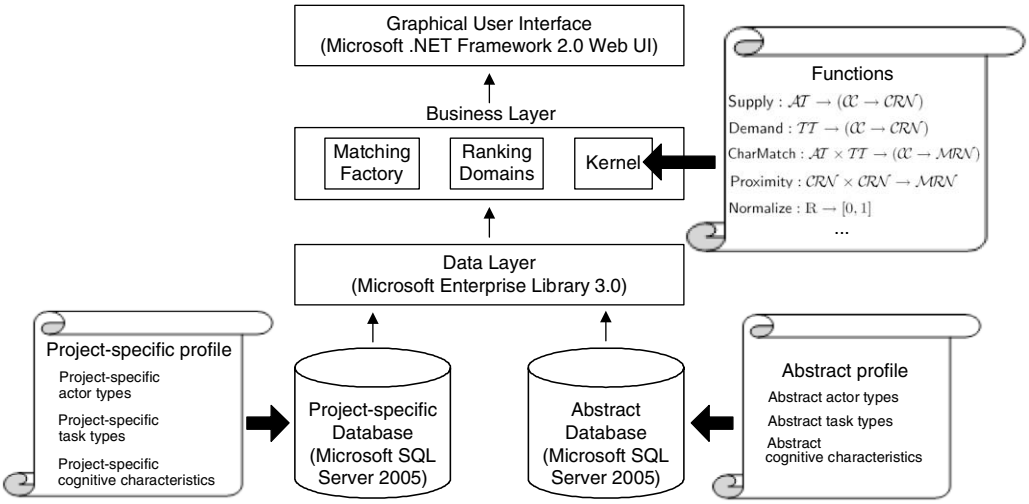

Fig. 8. Cognitive matchmaker system architecture.

the user selects the *transactor* actor type and the *synthesis* task type. This causes the application to generate a list of all the cognitive characteristics that have been used to characterize the transactor actor type and the synthesis task type. In the following step, the application displays on which level the expert supplies the involved characteristics and on which level the synthesis task demands the characteristics for successful fulfillment of the task. The next part shows the characteristic match results for all cognitive characteristics. This is shown in Fig. 9. The user can provide the weigh values for the cognitive characteristics by entering them for each characteristic involved in the next screen that is shown in Fig. 10. Fig. 11 shows the eventual suitability match result with the corresponding graph after calculating and summating the weighed characteristic matches. The resulting graph shows that in this case the suitability match of the expert fulfilling the synthesis task is 4.65. The certainty that the expert is able to fulfill the synthesis task is 93%. The source code of the prototype is based on the formal framework conceptually shown in Fig. 3. This is necessary to show that the framework can be implemented in a prototype application. For example, the code implementation of the suitability match function depicted in Section 5.3 is shown in Fig. 12. Recall that the match function has been defined as follows:

$$\mathrm{Match}(i,j) \triangleq \bigoplus_{c \in \mathscr{C}} \mathrm{Weigh}(c, \mathrm{CharMatch}(i,j)) \tag{53}$$

The code implementation obviously shows that the match function takes an actor type and a task type as input parameters and a suitability rank value as output parameter just like the formal match function. Then, for each cognitive characteristic involved in the process of computing the suitability match the results of the weighed characteristic match function are summated. This also corresponds with the definition of the suitability match function. Note that the weighed characteristic match function as well as the characteristic match functions are nested in the definition of the suitability match function. This can also be discovered in

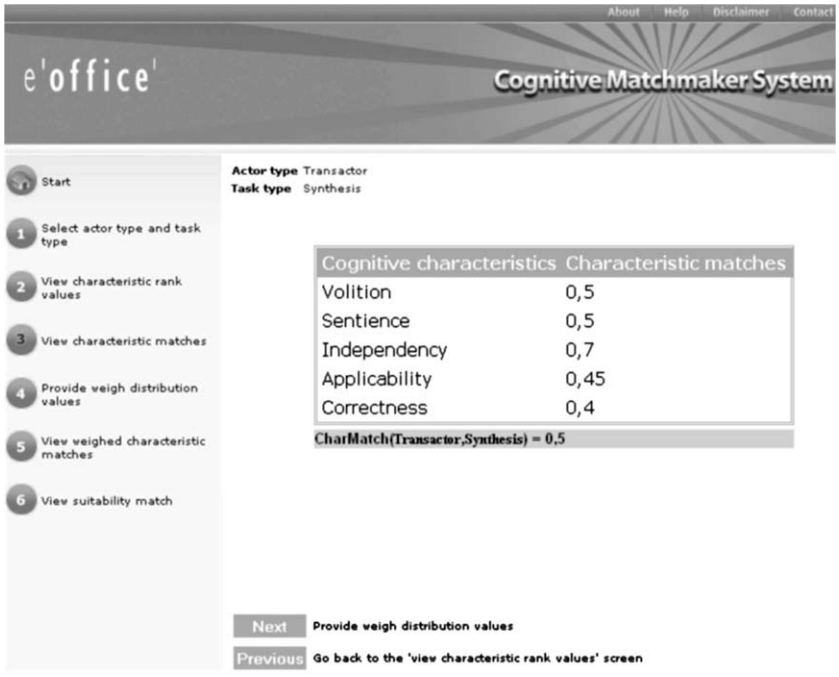

Fig. 9. Characteristic match screen.

Fig. 10. Weigh distribution values screen.

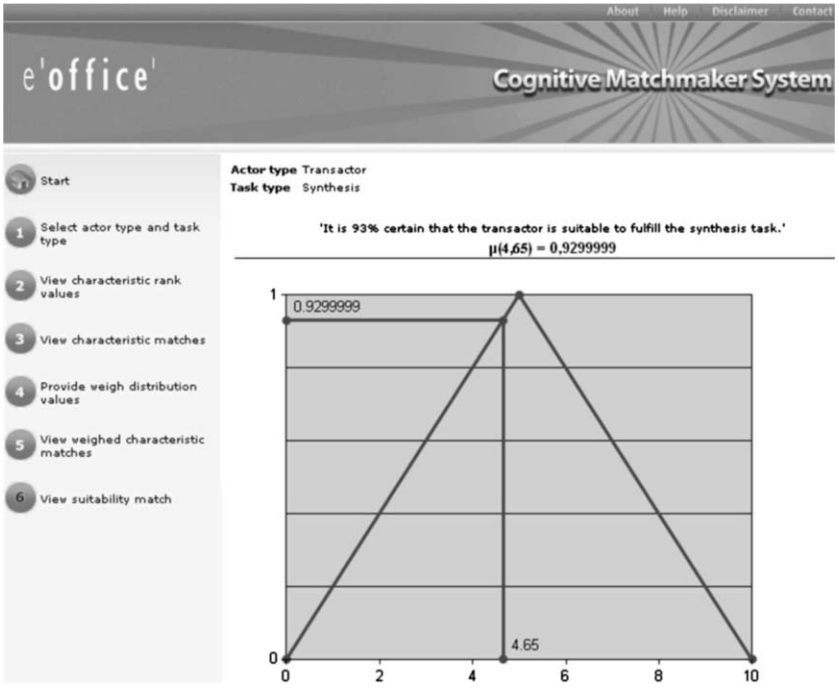

Fig. 11. Suitability match screen.

Fig. 12. The source code of the characteristic match function is depicted in Fig. 13. The source code of the weighed characteristic match function is depicted in Fig. 14. Up till now the framework, the fuzzy match extension and the prototype of a cognitive matchmaker system have been discussed. In the following section we will evaluate the cognitive matchmaking framework and the prototype of a cognitive matchmaker system. Therefore, a case study has been conducted in the area of information systems engineering (ISE). The case study clarifies the benefits of the system when utilized in ISE.

8. Case study in information systems engineering

The case study that has been conducted is related with a recently completed ISE project at 'e-office'. This is a company specialized in providing computer-aided support for human actors to support them in their office work. The ISE project has been concerned with the development of an 'Action Reporting Tool' (ART for short) for an international provider of banking and insurance services to personal, business and institutional customers. The action reporting tool is a Web application that can generate risk

```
public static SuitabilityRank Match(TaskType taskTypeObject, ActorType actorTypeObject) {

  SuitabilityRank SuitabilityRankObject = new SuitabilityRank();

  foreach (Characteristic CharacteristicObj in _matching.RetrieveCharacteristics()) {
    SuitabilityRankObject.RankValue += Weigh(CharacteristicObj,
    CharMatch(actorTypeObject, taskTypeObject,
    CharacteristicObj)).RankValue;
  }

  return SuitabilityRankObject;
}
```

Fig. 12. Source code of the suitability match function.

```
public static MatchRank CharMatch(ActorType actorTypeObject, TaskType taskTypeObject,
                                  Characteristic characteristicObject) {

  if (characteristicObject.RankActor == null)
  {
    characteristicObject.RankActor = Supply(actorTypeObject, characteristicObject);
  }

  if (characteristicObject.RankTask == null)
  {
    characteristicObject.RankTask = Demand(taskTypeObject, characteristicObject);
  }

  return Proximity(characteristicObject.RankActor, characteristicObject.RankTask);
}
```

Fig. 13. Source code of the characteristic match function.

```
public static SuitabilityRank Weigh(Characteristic characteristicObject,
                                    MatchRank matchRankObject) {

  SuitabilityRank SuitabilityRankObj = new SuitabilityRank();
  SuitabilityRankObj.RankValue = (Metric(matchRankObject).RankValue *
                                  CharWeigh(characteristicObject));

  return SuitabilityRankObj;
}
```

Fig. 14. Source code of the weighed characteristic match function.

reports for the user. This tool should assist risk management to better monitor and control insurance risks. This includes monitoring and controlling the risks themselves and also the actions of the actors involved in providing insurance services.

The case study has been conducted by applying a separate iteration of the inductive-hypothetical research strategy. This separate iteration is part of phase 4 of the overall research strategy mentioned in Section 2 and can be described as follows:

4.2. Case study in information systems engineering
- 4.2.1. Description of the project phases in which the ISE project has been divided (Section 8.1). The description includes project-specific actor types and task types and relations between them.
- 4.2.2. Abstraction of the results of phase 1 of the research strategy to our general model of actor types and task types mentioned in Sections 3 and 4 and in [11] (Section 8.2).
- 4.2.3. Formulation of how the cognitive matchmaker system can be utilized in every project phase related to the actor types and task types involved in the project (Section 8.3).
- 4.2.4. Analysis to identify the benefits if the cognitive matchmaker system had been applied in the studied ISE project (Section 8.4).
- 4.2.5. Evaluation by comparing phase 1 with phase 4 (Section 8.5).

8.1. Initiation

The ART project is based on the Microsoft Solutions Framework (MSF) information systems engineering method. The resulting tool is a Web application running on the Microsoft Office SharePoint Server 2007 platform (MOSS 2007 for short). Applications based

on this server platform are aimed to facilitate organizational collaboration, content management, business process management and access to information related with organizational goals and processes. The following project phases are determined as part of the ART project: the definition phase, the development phase, the acceptance phase and the implementation phase. During the *definition* phase requirements have been engineered by means of interviews with the future users of the tool. Interactive workshops have also been conducted involving multiple users. Proceeding from this requirements engineering process several use cases have been created to determine the interactions between the users and the tool. Possible screen mockups have then been created based on the use cases. The tool is developed in an iterative way during the *development* phase. The results after every iteration are tested before proceeding to the next iteration. The tool is tested integrally during the *acceptance* phase in conformity with a test plan. The acceptance test has been carried out by the banking and insurance service provider. Eventually, the final version of the tool is implemented at the banking and insurance provider during the *implementation* phase.

The actors participating in the project have been categorized into several project-specific actor types based on the MSF method. Despite the fact that MSF incorporates many more project-specific actor types, the following types were identified in the ART project. First, an *integrated program management (IPM) officer* can be identified. The IPM officer is the executive responsible for the overall organizational scheduling, planning and resource allocation. The *project manager* type is charged with planning and scheduling duties including developing project and iteration plans, monitoring and reporting status, identifying and managing issues to closure, and identifying and mitigating risk. The *product manager* insures that the project stays within budget and that the business case is realized. Besides these management-oriented actor types, several actor types can be identified that are more directly involved in the development of the tool. The *infrastructure architect* type, for instance, focuses on the deployment of both the physical and virtual servers and services which run on them. Furthermore, the *solution architect* is responsible for defining both the organizational structure of the application and the physical structure of its deployment. Finally, the *lead developer* and the *developer* actor types can be identified. The lead developer lends experience and skill and shows leadership by coaching fellow developers. Lead developers carry responsibility for source code reviews, design and testing. The developer is responsible for the bulk of the work building the product. The developer should suffer a minimum of communication overhead allowing for a maximum effort on construction of source code.

The project manager of the ART project has created plans for every phase that include breakdowns of the tasks to be fulfilled in every phase. Using this documentation a project-specific task type categorization can be described together with the fulfilled tasks. Analysis of the project documentation also reveals which project-specific actor type is related to a project-specific task type. In other words, it can be made explicit which project-specific actor is responsible to fulfill a project-specific task. It is also possible that more than one actor is related to a task. The results of this analysis are shown in Table 6.

8.2. Abstraction

When performing the second phase of the inductive-hypothetical research strategy, it is possible to abstract the project-specific actor types and task types. First, it is shown how the project-specific task types can be abstracted to the abstract task types mentioned in Section 4. Second, this section discusses how the project-specific actor types can be abstracted to the actor types mentioned in Section 3.1.

Table 6
Project-specific actor types and task types

Task instance	Project phase	Project-specific task type	Project-specific actor type
Conduct interview with stakeholder	Definition Development	Elicitation task	Project manager Product manager Solution architect
Conduct workshop with stakeholders	Definition Development	Elicitation task	Project manager Product manager Solution architect
Design use case	Definition Development	Design task	Solution architect Developer
Design mockup	Definition	Design task	Developer
Design risk report	Definition	Design task	Lead developer Developer
Write technical tool description	Definition Development	Documentation task	Developer
Write project initiation document	Definition	Documentation task	Project manager Product manager
Determine hardware requirements	Definition	Documentation task	Infrastructure architect
Write security plan	Definition Development	Documentation task	Infrastructure architect
Write project plan	Definition Development	Documentation task	Project manager
Attend project meeting	All phases	Meeting task	All actor types
Attend steering committee meeting	All phases	Meeting task	IPM officer
Set up MOSS 2007 environment	Development	Code development task	Infrastructure architect
Build custom Web part	Development	Code development task	Developer
Configure Web part	Development	Code development task	Developer
Create risk report	Development	Code development task	Lead developer Developer
Implement security for tool	Development	Code development task	Infrastructure architect
Commit partial system test	Acceptance	System test task	Lead developer Developer
Commit integral system test	Acceptance	System test task	Lead developer Developer
Deploy completed tool	Implementation	Deployment task	Lead developer Developer

Table 7
Actor type and task type abstraction

Project-specific actor type	Abstract task type	Abstract actor type
IPM officer	Acquisition	Collaborator
Project manager	Acquisition	Collaborator
	Synthesis	Transactor
Product manager	Acquisition	Collaborator
	Synthesis	Transactor
Infrastructure architect	Acquisition	Experiencer
	Synthesis	Expert
Solution architect	Acquisition	Collaborator or Experiencer
	Synthesis	Expert
Lead developer	Acquisition	Experiencer
	Synthesis	Expert or Integrator
	Testing	Collaborator
Developer	Acquisition	Experiencer
	Synthesis	Collaborator or Expert or Integrator
	Testing	Collaborator

The distinguished abstract task types are the acquisition task type, synthesis task type and the testing task type. The project-specific task types depicted in Table 6 can be abstracted to these task types as follows. The mentioned elicitation tasks are typical knowledge *acquisition* tasks. The actors executing an elicitation task acquire and memorize knowledge by means of interviews or workshops. Design tasks can be abstracted as *synthesis* tasks. In a design task, the actor applies already acquired knowledge when designing a use case, mockup or risk report. Documentation tasks can also be classified as synthesis tasks. The documentation tasks mentioned in Table 6 are related with the application of knowledge when writing a technical tool description, project initiation document, hardware requirements report, security plan and project plan. Next, meeting tasks are abstracted to acquisition tasks. During project meetings and steering committee meetings it is intended to acquire knowledge about, e.g. project planning, project status and the remaining budget. Code development tasks can be viewed as synthesis tasks. These tasks are necessary to build the action reporting tool itself. The build process consisted of setting up the programming environment, and the creation of Web parts and risk reports. Web parts are the visual components that are part of a Microsoft SharePoint application which include functionality, such as: listed announcements, a calendar, a discussion part, etc. *Testing* tasks are related with the project-specific system test tasks. In a testing task, earlier applied knowledge is thoroughly examined inducing an improvement of the specific knowledge applied. The partial and integral system tests are needed to identify and correct flaws in the action reporting tool. Finally, the deployment task can be abstracted to a synthesis task. Here, all relevant knowledge that is applied is related with a successful deployment of the system at the customer's location.

Recall that the distinguished abstract actor types are the collaborator, experiencer, expert, integrator and the transactor. Table 7 shows the project-specific actor types, the abstract task types that a project-specific actor type can fulfill and the abstract actor types. For instance, a project manager may be classified as a collaborator when fulfilling an acquisition task. However, if a project manager executes a synthesis task he may act differently and may be classified as a transactor instead. Note that some project-specific actors are not related with every abstract task type. For instance, a solution architect does not fulfill a testing task.

8.3. Theory formulation

The results of applying the third phase of the inductive-hypothetical research strategy are discussed throughout this section. We will show how the cognitive matchmaker system can be utilized in all four phases of the ART project.

8.3.1. Definition phase

Based on the results of Section 8.1 and 8.2 it is now possible to calculate the certainty that the actors involved in the definition phase of the ART project can successfully fulfill the tasks allocated to them. The results after calculating the cognitive matches are depicted in Table 8. Table 6 provides the project-specific task types and the project-specific actor types that are involved in the definition phase. The cognitive matchmaker system can be utilized to calculate the matches between the actor types and task types involved in the definition phase. For this purpose the abstraction of the project-specific task types and actor types shown in Table 7 must be used. The results of Table 8 can be explained as follows. The solution architect, for instance, acts as a collaborator when working on an acquisition task and acts as an expert when working on a synthesis task, respectively. In the definition phase, the solution architect conducts interviews and workshops, and attends project meetings. These tasks can be regarded as knowledge acquisition tasks. Five weigh values have to be provided by the user of the cognitive matchmaker system when calculating the suitability match of the collaborator fulfilling an acquisition task. The weigh values express the importance of the involved cognitive characteristics when the solution architect needs to fulfill an acquisition task in the definition phase. Because we were the users of the cognitive matchmaker system, we have provided the following weigh values for the volition, causability, improvability, satisfaction and relevance characteristics, respectively: 1.5, 2, 1.5, 3 and 2. At the moment, the weigh values have to be provided manually by the user. However, the next version of the prototype should include an algorithm that determines these weigh values dependent of

Table 8
Cognitive matchmaking in the definition phase

Project-specific actor type	Task type	Actor type	Weigh values	Suitability match	Certainty (%)
IPM officer	Acquisition	Collaborator	2, 1.5, 0.5, 3, 3	4.3	86
Project manager	Acquisition	Collaborator	2, 1, 1, 3, 3	4.4	88
	Synthesis	Transactor	1, 1.5, 1.5, 3, 3	4.85	97
Product manager	Acquisition	Collaborator	2, 1, 1, 3, 3	4.4	88
	Synthesis	Transactor	1, 1.5, 1.5, 3, 3	4.85	97
Infrastructure architect	Acquisition	Experiencer	4, 3, 3	3.5	70
	Synthesis	Expert	1.5, 1, 1, 1, 1.5, 2, 2	4.575	91.5
Solution architect	Acquisition	Collaborator	1.5, 2, 1.5, 3, 2	4.4	88
	Synthesis	Expert	1, 1.5, 1, 0.5, 1, 2, 3	4.8	96
Lead developer	Acquisition	Experiencer	4, 3, 3	3.5	70
	Synthesis	Expert	1, 2, 1, 1.5, 1, 1.5, 2	4.6	92
Developer	Acquisition	Experiencer	3, 4, 3	3.3	66
	Synthesis	Collaborator	2, 1.5, 2.5, 2, 2	4.4	88

how important a cognitive characteristic is in a certain combination of an actor type and a task type. The highest weigh value has been applied to the satisfaction characteristic. That the solution architect should supply the satisfaction characteristic is obviously very important when fulfilling an acquisition task. This is to make sure that the solution architect is pleased with the knowledge acquired and that no additional need for knowledge remains. The cognitive matchmaker system then sums up the resulting weighed characteristic matches resulting in a suitability match of 4.4. The certainty that the solution architect acting as a collaborator can successfully fulfill an acquisition task is: $\mu(4.4) = \frac{2}{0+10} \cdot 4.4 = 0.88$ or $0.88 \cdot 100\% = 88\%$. The solution architect acts as an expert when working on a synthesis task during the definition phase. These synthesis tasks are related with the design of use cases. The solution architect should be able to use his own knowledge about use cases to correctly design them. The architect should also be able to combine and modify his own knowledge while designing use cases and he should also be able to learn from that process. The expert actor type matches very well with the synthesis task in this case, because the result of the suitability match calculation is 4.8 and the result of the certainty function is 96%.

8.3.2. Development phase

Notice that the development phase includes a variety of project-specific task types when analyzing Table 6. Concretely, the development phase includes elicitation tasks, design tasks, documentation tasks, meeting tasks and code development tasks. The code development tasks cover the majority of the development phase of the ART project. The project-specific task types in the development phase can be abstracted to acquisition tasks and synthesis tasks. A project-specific actor type can also be classified as a certain abstract actor type dependent of the task at hand. Table 9 shows how the cognitive matchmaker system can be utilized in the development phase. We will consider the contributions of the developer in the development phase to explain the meaning of the contents of Table 9. Table 6 shows that building a custom Web part is a specific code development task for a developer. Recall that Web parts are the visual components that are part of a Microsoft SharePoint application which include functionality. Building a Web part is a typical synthesis task: technical knowledge about creating Web parts in the context of the ART project is put into practice by a developer. When working on such a synthesis task, the developer acts as an expert. Mostly individual knowledge about creating Web parts is used to fulfill the task and the developer is able to combine and modify this knowledge while working on the code development task. Lessons learned after fulfilling the task can be taken into account for future code development tasks. Seven cognitive characteristics are involved when matching the suitability of the expert type with the synthesis task: volition, sentience, causability, improvability, independency, applicability and correctness. In this case, the correctness characteristic weighs most according to the weigh values shown in Table 9. This means that when the developer applies knowledge it is important that this knowledge is useful for the specific task and the applied knowledge meets its requirements. The suitability match of 4.55 implies that the certainty of the developer acting as an expert can successfully fulfill a synthesis task is 91%.

8.3.3. Acceptance phase

The acceptance phase is less comprehensive than the definition and development phases. Table 6 reveals that the lead developer and the developer play the most important role in this phase, whilst the other project-specific actor types take part in the regular project meetings that are also part of the acceptance phase. The results of utilizing the matchmaker system in this phase are shown in Table 10. Testing the action reporting tool in its totality is one the testing tasks that is part of the acceptance phase. The lead developer and developer types are responsible for fulfillment of this task. Both the lead developer and the developer types act as collaborators in this testing task. Collaboration with other project members is necessary to fulfill the testing task, because neither the lead developer nor the developer possess all knowledge about the action reporting tool as a whole to complete the task.

8.3.4. Implementation phase

In the implementation phase the completed action reporting tool is deployed at the customer organization. The lead developer and developer types are involved in this deployment, whereas the other actors commit to participate in project meetings during this phase. Table 11 shows the results of the cognitive matches in the implementation phase. The lead developer and the developer synthesize relevant knowledge to successfully deploy the tool. Therefore, they primarily wish to apply knowledge of very high quality by working together and they act as integrators in the implementation phase.

8.4. Implementation

The results from the theory formulation phase are now utilized to describe how an ISE project can benefit from the cognitive matchmaker system. Three viewpoints are distinguished for an ISE project, namely: Design time, runtime and post-mortem viewpoints. From each of these viewpoints the applications of the system are described:

Design time. This viewpoint embraces the situation before the project is initiated (before the definition phase starts). First, the project-specific actor types and the project-specific task types need to be conceived. If this is done, there are two options to choose from: Use the project-specific actor and task profile as a starting point or the abstract profile including the abstract actor types and task types from our framework. The latter has been done in

Table 9
Cognitive matchmaking in the development phase

Project-specific actor type	Task type	Actor type	Weigh values	Suitability match	Certainty (%)
IPM officer	Acquisition	Collaborator	1.5, 1, 0.5, 3, 4	4.3	86
Project manager	Acquisition	Collaborator	2, 1, 1, 3, 3	4.4	88
	Synthesis	Transactor	2, 1, 1, 3, 3	4.75	95
Product manager	Acquisition	Collaborator	1, 1, 1, 3, 4	4.35	87
Infrastructure architect	Acquisition	Experiencer	2, 4, 4	3	60
	Synthesis	Expert	2, 1, 0.5, 1, 0.5, 2, 3	4.75	95
Solution architect	Acquisition	Collaborator	1, 1.5, 0.5, 3, 4	4.25	85
	Synthesis	Expert	1, 1, 0.5, 0.5, 1, 2, 4	4.725	94.5
Lead developer	Acquisition	Experiencer	2, 3, 5	2.9	58
	Synthesis	Expert	1, 0.5, 0.5, 0.5, 0.5, 3, 4	4.85	97
Developer	Acquisition	Experiencer	3, 4, 3	3.3	66
	Synthesis	Expert	2, 1, 0.5, 2, 0.5, 1, 3	4.55	91

Table 10
Cognitive matchmaking in the acceptance phase

Project-specific actor type	Task type	Actor type	Weigh values	Suitability match	Certainty (%)
IPM officer	Acquisition	Collaborator	2, 0.5, 0.5, 4, 3	4.25	85
Project manager	Acquisition	Collaborator	1, 0.5, 1.5, 4, 3	4.35	87
Product manager	Acquisition	Collaborator	1.5, 0.5, 1, 4, 3	4.3	86
Infrastructure architect	Acquisition	Experiencer	1.5, 5, 3.5	2.95	59
Solution architect	Acquisition	Experiencer	1, 6, 3	2.9	58
Lead developer	Acquisition	Experiencer	3.5, 3.5, 3	3.4	68
	Testing	Collaborator	1, 0.5, 0.5, 1.5, 1.5, 2, 3	3.975	79.5
Developer	Acquisition	Experiencer	2.5, 4, 3.5	3.15	63
	Testing	Collaborator	1, 0.5, 1.5, 1, 1.5, 3, 1.5	3.925	78.5

Table 11
Cognitive matchmaking in the implementation phase

Project-specific actor type	Task type	Actor type	Weigh values	Suitability match	Certainty (%)
IPM officer	Acquisition	Collaborator	2, 0.5, 0.5, 6, 1	4.05	81
Project manager	Acquisition	Collaborator	1.5, 0.5, 1, 5, 2	4.2	84
Product manager	Acquisition	Collaborator	2, 0.5, 0.5, 5, 2	4.15	83
Infrastructure architect	Acquisition	Experiencer	0.5, 6, 3.5	2.75	55
Solution architect	Acquisition	Experiencer	2, 4, 4	3	60
Lead developer	Acquisition	Experiencer	4, 3, 3	3.5	70
	Synthesis	Integrator	1, 0.5, 2.5, 6	5.55	89
Developer	Acquisition	Experiencer	3, 4, 3	3.3	66
	Synthesis	Integrator	1, 0.5, 3.5, 5	5.5	90

the case study as is elaborated in Section 8.2. When using a project-specific profile as input for the cognitive matchmaker system, a project-specific profile of actors and tasks should be generated. This has also been done in Section 8.1. If not already entered in the project-specific database as is shown in Fig. 8, the actor and task data should be provided as a next step. The person that needs to allocate tasks to actors, the project manager for instance, can now calculate the suitability matches. Based on these results he can allocate tasks to actors before starting the project.

Runtime. During the enactment of an ISE project, suitability matches can be recalculated if changes to task allocations are necessary. This may be the case if a different actor needs to work on a task than the one specified in the project plan. The cognitive matchmaker system can then be used again to recalculate the suitability match. New tasks may also be introduced during the project that need to be allocated to actors. This may entail the need to calculate additional suitability matches during project enactment. The cognitive matchmaker system can also be utilized to evaluate task allocations after every project phase. The suitability matches may be compared with the actual fulfillment of the tasks in a phase. An in-depth analysis may be necessary if there are striking differences between the suitability match and the results of task fulfillment by an actor.

Post-mortem. From a 'post-mortem' point of view, task allocations in the project as a whole can be analyzed. The suitability matches for every actor/ task combination in the ISE project may be compared to the actual results brought forward by the actors. Lessons learned should then be recorded for future projects. This may help to better decide which actor types are suitable to work with which types of tasks.

8.5. Case study evaluation

In this section, the results of the initiation phase are compared with the results of the implementation phase. The evaluation of the initiation phase is related to the three viewpoints of the implementation phase:

Design time. First, the choices leading to the project-specific actor types as shown in Table 6 have not been argued in the ART project documentation. Recall that the project-specific actor types originate from the Microsoft Solutions Framework ISE method. Entering the actor types that MSF distinguishes in the project-specific database of the cognitive matchmaker system enables a better argued decision of which actor types to use in a project. For instance, the system test tasks shown in Table 6 are performed by the (lead) developer actors. However, the MSF method also distinguishes the tester and test manager actor types. Including these actor types in the ART project may have improved the suitability matches related with the system test tasks. A difficulty is that the MSF method does not provide a clear description of the cognitive characteristics that characterize an actor type. The MSF method, however, provides a natural language description of each actor type included in the method. Proceeding from these descriptions the administrator of the cognitive matchmaker system should be able to characterize the project-specific actor types by adding cognitive characteristics to the project-specific database or by reusing characteristics.

Runtime. The results of the theory formulation phase included suitability matches for every actor/task combination differentiated to a specific project phase. At 'runtime', these suitability matches may be reviewed after every project phase. The lowest certainty percentages shown in Tables 8–11 deserve special attention to discover the reasons of the lowest match results. For instance, Table 8 shows that the developer acting as an experiencer has a certainty of 66% to successfully fulfill an acquisition task. When viewing Table 6 it can be interpreted that the acquisition task performed by the developer in the definition phase is related with the attendance of project meetings. This may be caused in the case if a meeting is not very relevant for a developer. For instance, when a large part of a certain meeting is about project management issues a developer may not have a satisfied feeling after the meeting. Letting developers attend the most relevant meetings may increase the suitability matches for these acquisition tasks. In the same way, the other calculated matches can be analyzed for every project phase.

Post-mortem. For instance, the testing tasks shown in Table 6 deserve attention when comparing the actual project results with the suitability matches. According to Table 6 the partial and integral system tests are conducted by the developer and lead

developer types. Table 10 shows that the certainty is 78.5% that the developer can successfully fulfill these testing tasks. The MSF method includes the tester actor type that may be more suitable to fulfill testing tasks in general. According to the MSF, a key goal for the tester is to find and report the significant bugs in the product by testing the product. Once a bug is found, it is also the tester's job to accurately communicate its impact and describe any solutions that could lessen its impact. The purpose of testing is to prove that known functions work correctly and to discover new product issues. Obviously, more bugs could have been found and solved after testing each iteration and the overall product by the tester actor type. In the current project situation, the developer has the responsibility for code development and testing as well. Usability issues also arose during the system test tasks. What can be seen in Table 6 is that the developer is also responsible for designing the mockups. The responsibility of the developer to design, develop as well as test the system may have contributed to the existence of some usability problems. The MSF advocates the addition of a user experience architect to the project to increase the usability of the tool. According to the MSF, the user experience architect is responsible for the form and function of the user interface, its aesthetics and the overall product usability. Recall that designing mockups is a synthesis task. The certainty that the developer can successfully fulfill a synthesis task in the definition phase is 88%. This is not low, but may further increase when the main focus of a developer is on developing code. So, for future projects it may be a good idea to introduce a tester and a user experience architect as well.

9. Discussion

Literature indicates that matchmaking in general (i.e. not specifically from a cognitive point of view) is possible in different ways. The research of [25] is related with matchmaking between a Web service provider on the one hand and a Web service requester on the other hand. Their matchmaking framework matches the supply of the service provider with the demand of the requester. A division of the concept of matchmaking is made in two categories:

(1) Syntactic matchmaking: which uses the structure or format of a task specification to match a requester with a provider to decide which service providers to recommend.
(2) Semantic matchmaking: which uses the meaning and informational content of the request to match it with the meaning of the offered services.

Relating this division with the matchmaking problem discussed in our study, it can be said that syntactic matchmaking determines a match dependent of the enacted task, which can be an acquisition task, synthesis task or a testing task in the case of our model. In the case of semantic matchmaking the meaning and informational content of the cognitive characteristics provided by an actor are related with the meaning and informational content of the task requirements. However, it is not self-evident to categorize our matchmaking framework in one of these two categories. Our framework is capable of more matchmaking functionality than syntactic matchmaking because of the introduced formalisms that not only take task specifications into account, but also actor specifications and cognitive characteristics. Semantic matchmaking however is a matchmaking category that requires an ontology to determine the informational semantics of that what is required and that what is supplied. The disadvantages of this type of matchmaking are related with the necessity of an ontology and the quite complex algorithms needed to compute an actual match result. In principle a different category such as 'cognitive matchmaking' based on numerical as well as fuzzy values is more suitable to classify our model. Advantages of our approach when compared with syntactic and semantic matchmaking as described above is that no ontology is needed to define a match and not only task specifications but also actor specifications are determined. The weighing of specific characteristics from a cognitive point of view then adds an additional dimension when determining a match.

A lot of studies in the matchmaking field are especially related to recommender systems. The research of [26] for instance presents how opportunities for collaboration between actors can be determined by matching an actor's current context (as determined by the actor's work environment) with other actors that might have related interests or work. Their matchmaking framework consists of a 3-step process:

(1) Given the work environment an actor is currently working on, look for other users currently in similar environments.
(2) Within those found, look for documents in the environments that are similar to the document currently being worked on.
(3) Ask the document owner whether the documents found can be sent and furnish information on the user who will be receiving it.

Matches are made through keyword similarity calculation. Every document requires an associated keyword list and these lists are compared to determine similarity between the documents and find possible matches. A drawback of this approach is that a specific algorithm is required to create a list of keywords for every document as part of an actor's work environment. Vivacqua et al. [26] use the TFIDF (Term Frequency Inverse Document Frequency) algorithm to generate a keyword list for every document in order to determine a match value. Compared to our approach though, no additional algorithms are necessary to compute a match value because of the ranking mechanisms that are already implemented in the theory of Section 5. This might save some time while generating match values using the cognitive matchmaker system. Another aspect relevant for the discussion is that matchmaking based on document comparison computes a match value based on documents only. However, to what extent can collaboration opportunities between actors be determined when only their overlapping documents are taken into consideration? Thus, the relation between the matchmaking framework and the goals that should be reached by such a system is somewhat difficult to interpret. When looking at the matchmaking framework of Section 5 this is more straightforward. For short, a goal of using an implemented version of such a framework is to find actor/ task matches based on cognitive characteristics to diminish the cognitive load of an actor.

Literature indicates that cognitive matchmaking can be found in several areas of computer science. One of these initiatives is Cognitive Match Interface Design (COMIND) [27]. COMIND is the designing of system processes so that they proceed and interact with the user in a manner that parallels the flow of the user's own thought processes. It consists of several principles, such as: the user should be able to express his needs to the computer with constructs which mirror the user's own thought processes. Another principle is the readiness of a computer to solve problems of the user in his/her area of need. Also, the computer should sanction flexibility just like the mind. The mind is regarded as a versatile and flexible problem solver. The authors tried to apply these principles when designing a medical information system. Unfortunately, a method for interface design that incorporates COMIND is not introduced. Only the medical information system case is elaborated. Creation of a COMIND framework including the proposed cognitive principles for user interface design would have possibly enabled reuse of COMIND in different areas. The existence of our cognitive matchmaking framework does enable its specific application in many different areas.

Another interesting study is the cognitive matchmaking of students with e-learning system functionality [28]. A way of working is presented to design e-learning systems that better adapt to the cognitive characteristics of students. First, a taxonomy of learning styles is selected to classify the user. Next, techniques should be developed to introduce the adaptation into the system that fits the learning styles. The designed adaptation is then implemented on a computer. Finally, a selection of the technologies is made that are adequate for the adaptation. Besides this described way of working, a cognitive method or a system to match students and e-learning systems is not proposed. The mentioned concept of reflection can be very useful for our own work, though. Reflection is defined as the capability of a computational system to adjust itself to changing conditions. This can be seen on, e.g. http://maps.google.com. The process of adaptation is made stronger since it is possible to create specific code depending on the supplied characteristics of the user when using the system. Adding reflection to our system may take situational elements into account when determining a match, for instance. Concretely, the actual availability of actors during the studied ART project may be included when allocating tasks to actors.

Jaspers et al. [29] argue that early involvement of cognitive matchmaking in ISE may be of importance to design systems that fully support the user's work practices. From this perspective, cognitive matchmaking is used for requirements engineering to match system requirements with the user's task behavior. To understand the task behavior of future users of a clinical system, the think aloud method has been applied [29]. Think aloud is a method that requires subjects to talk aloud while performing a task. This stimulates understanding of the supplied cognitive characteristics when performing a task. Unfortunately, the method has only been utilized to design a user interface for a clinical information system. The study lacks a more abstract framework that can be reused to design interfaces in general that better match task behavior of its users. Task-analysis methods such as the think aloud method can be useful when refining our research. For instance, these methods may be very valuable to improve the way we have characterized the abstract actor types and task types based on cognitive characteristics. Jaspers et al. [29] also included a simplified model of the human cognitive system. Studying that model may further improve the way we interpret cognitive matchmaking processes.

Cognitive psychology includes important research related to actors solving a problem and the underlying cognitive processes. In their earlier work, Newell and Simon [30] proposed a theory which includes a set of cognitive processes or mechanisms that produce the behavior of an actor. According to the theory, the task instructions and previous experience in solving similar tasks contribute significantly to the determination of an actor's perception of how to fulfill a task. The task instructions not only define the task but also provide a specific representation that helps define ideas of how to fulfill a task. Building upon Newell and Simon's work, human factors research explored the influence of the nature of the task and the way an actor performs to fulfill the task. The ideas presented in this paper somehow relate with an important notion generated from Newell and Simon's work which is that of *cognitive fit* [31]. The basic model of cognitive fit views task fulfillment as the outcome of the relationship between the actor's perception of how to fulfill the task and the nature of the task, which are both characterized by the type of knowledge they emphasize. When the types of knowledge emphasized in the actor and task elements match, the actor can employ processes (and formulate a mental representation) that also emphasize the same type of knowledge. Cognitive fit exists because the cognitive processes used to complete the task match. This synergy results in superior task performance. Conversely, when a mismatch occurs between the perception of how to complete a task and the actual requirements to fulfill a task, cognitive fit will not result and task performance will deteriorate. A difference with our approach is that an actor can be classified as an experiencer, a collaborator, an expert, an integrator or a transactor based on that actor's current cognitive profile (i.e. the way an actor is able to perform the defined cognitive characteristics) instead of determining an actor's perception of how to complete a task. Elaborating an actor's perception related to the fulfillment of every task may be a time consuming process in practice and therefore an advantage of our approach may be that a match value can easily be determined once actors and tasks are classified by their types (by following the system's flow as presented in Fig. 3). However, an actor's cognitive capabilities may change over time (they may improve or deteriorate) and that may cause an actor to be classified as a different type in our model at different points in time.

10. Conclusions and future work

This paper describes how actors and tasks can be matched based on cognitive characteristics. First, several actor types and task types are categorized based on cognitive characteristics that can be supplied, respectively, demanded. Next, a framework for cognitive matchmaking is elaborated. This framework includes the functions to calculate the suitability of an actor to fulfill a task. Furthermore, an extension of this framework shows how numerical suitability match values can be translated to linguistic suitability match values. Proceeding from the framework the prototype implementation of a cognitive matchmaker system is demonstrated. An information systems engineering project provided the breeding ground for the case study in which possible benefits of the cognitive matchmaker system has been evaluated. The ISE project has been concerned with the development of an 'Action Reporting Tool' for an international provider of banking and insurance services. The action reporting tool is a software application that can generate risk reports for the user. The suitability matches of the tasks allocated to the actors in every project phase have been evaluated using the cognitive matchmaker system.

By utilizing the actor type and task type categorizations it has been possible to calculate matches with the functions of the framework. The implementation of these functions in a prototype shows that it is possible to build a working cognitive matchmaker system based on the framework. Several conclusions can be drawn when reviewing the results of the case study. It can be concluded that the cognitive matchmaker system can provide support for task allocation in an ISE project in at least three different ways: before project initiation (at design time), during project enactment (at runtime) and after the project has finished (post-mortem). At design time, the person that needs to allocate tasks to actors, the project manager for instance, can calculate the suitability matches. Based on these results he can allocate tasks to actors before starting the project. At runtime, suitability matches can be recalculated if changes to task allocations are necessary. The cognitive matchmaker system can also be utilized to evaluate task allocations after every project phase. The calculated suitability matches can be compared with actual task performance. From a post-mortem point of view, the suitability matches for every actor/task combination in the project can be compared to the actual results brought forward by the actors. Lessons learned may help to better decide which actor types are suitable to work with which types of tasks. In this case, the cognitive matchmaker system is related with information systems engineering. However, the system may be usable in other areas as we have already mentioned in the introduction, such as: multi-agent systems, workflow management and BPR.

Future work is concentrated on improving the theoretical framework as well as the prototype and further evaluation in case studies. At this moment, it is only possible to calculate a match based on one actor type and one task type. However, there are situations imaginable that multiple actors are working together to fulfill a set of tasks. If this is the case, it might be interesting to determine a match based on the total amount of actors and the total amount of tasks the actors are fulfilling as a group. The prototype of the cognitive matchmaker system can also be expanded with the capability to compute a suitability match based on fuzzy assessments. In other words, the formalisms of the fuzzy match part can be implemented in a second version of the prototype. Besides these additions, the future cognitive matchmaking framework and prototype may consider situational elements. This may include personal preferences of actors, personal goals of actors and availability of actors during a project. Suppose that an actor has a high match value when fulfilling a certain task but does not like to fulfill that task at all, then this may negatively influence the actor's task performance. The next version of the cognitive matchmaking framework and prototype should also take the concept of *actor contention* into account. This can be explained as follows. Assume that two actors receive the same best suitability for a task. Let one of these actors be mediocre at all required characteristics, while the other actor is really good at some and really bad at others. Somehow, the system should choose an actor to assign the task to. Finally, methods from cognitive science to better understand task behavior can be studied to improve the framework. Cognitive task analysis, protocol analysis and the 'think aloud' methods are most common to analyze in detail the way in which humans perform tasks, mostly in interaction with a prototype computer system. Naturally, more case studies need to be conducted (most likely in the aforementioned areas). This is necessary to evaluate and improve the theory and the prototype.

Acknowledgements

This study is partly supported by the Dutch agency for innovation and sustainable development (SenterNovem) under Grant SO07026170. We cordially thank Martijn Duiveman and Peter Spronk of e-office for implementing a major part of the prototype.

References

[1] S. Staab, R. Studer, H. Schnurr, Y. Sure, Knowledge processes and ontologies, IEEE Intelligent Systems 16 (1) (2001) 26–34.
[2] E. Kako, Thematic role properties of subjects and objects, Cognition 101 (1) (2006) 1–42.
[3] C. Weir, J. Nebeker, L. Bret, R. Campo, F. Drews, 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) (2007) 65–75.
[4] N. Meiran, Modeling cognitive control in task-switching, Psychological Research 63 (3–4) (2000) 234–249.
[5] R. Hertwig, G. Barron, E. Weber, I. Erev, The role of information sampling in risky choice, in: K. Fiedler, P. Juslin (Eds.), Information Sampling and Adaptive Cognition, Cambridge University Press, New York, NY, USA, 2006, pp. 72–91.
[6] D. Koehler, Explanation, imagination, and confidence in judgment, Psychological Bulletin 110 (3) (1991) 499–519.
[7] M. R-Moreno, D. Borrajo, A. Cesta, A. Oddi, Integrating planning and scheduling in workflow domains, Expert Systems with Applications 33 (2) (2007) 389–406.
[8] K. Joshi, S. Sarker, S. Sarker, Knowledge transfer within information systems development teams: examining the role of knowledge source attributes, Decision Support Systems 43 (2) (2007) 322–335.
[9] E. Shakshuki, O. Prabhu, I. Tomek, FCVW agent framework, Information and Software Technology 48 (6) (2006) 385–392.
[10] W. van der Aalst, A. ter Hofstede, Verification of workflow task structures: a Petri-net-based approach, Information Systems 25 (1) (2000) 43–69.
[11] S. Overbeek, P. van Bommel, H. Proper, D. Rijsenbrij, Characterizing knowledge intensive tasks indicating cognitive requirements – scenarios in methods for specific tasks, in: J. Ralyté, S. Brinkkemper, B. Henderson-Sellers (Eds.), Proceedings of the IFIP TC8/ WG8.1 Working Conference on Situational Method Engineering: Fundamentals and Experiences, vol. 244, Geneva, Switzerland, Springer, Boston, USA, 2007, pp. 100–114.
[12] S. Overbeek, P. van Bommel, H. Proper, D. Rijsenbrij, Matching cognitive characteristics of actors and tasks, in: R. Meersman, T. Zari (Eds.), On the Move to Meaningful Internet Systems 2007: DOA, CoopIS, ODBASE, GADA, and IS, Vilamoura, Portugal, November 25–30, 2007, Proceedings, Part I, vol. 4803 of Lecture Notes in Computer Science, Vilamoura, Portugal, EU, Springer, Berlin, Germany, EU, 2007, pp. 371–380.
[13] H. Sol, Simulation in information systems, Ph.D. thesis, University of Groningen, The Netherlands, EU, 1982.
[14] C. Churchman, The Design of Inquiring Systems: Basic Concepts of Systems and Organization, Basic Books, New York, NY, USA, 1971.
[15] G. de Vreede, N. Jones, R. Mgaya, Exploring the application and acceptance of group support systems in africa, Journal of Management Information Systems 15 (3) (1998) 197–234.
[16] Y. Wang, E. van de Kar, G. Meijer, Designing mobile solutions for mobile workers: Lessons learned from a case study, in: ICEC'05: Proceedings of the 7th International Conference on Electronic Commerce, Xi'an, China, ACM Press, New York, NY, USA, 2005, pp. 582–589.
[17] D. Cruse, Some thoughts on agentivity, Journal of Linguistics 9 (1973) 11–23.
[18] D. Dowty, Thematic proto-roles and argument selection, Language 67 (3) (1991) 547–619.
[19] I. Nonaka, H. Takeuchi, The Knowledge Creating Company, Oxford University Press, New York, NY, USA, 1995.
[20] T. Davenport, Thinking for a Living – How to Get Better Performances and Results From Knowledge Workers, Harvard Business School Press, Boston, MA, USA, 2005.
[21] S. Hoppenbrouwers, H. Proper, Knowledge discovery: De zoektocht naar verhulde en onthulde kennis, DB/Magazine 10 (7) (1999) 21–25. in Dutch.
[22] T. Halpin, Information Modeling and Relational Databases, From Conceptual Analysis to Logical Design, Morgan Kaufmann, San Mateo, CA, USA, 2001.
[23] L. Zadeh, Toward a generalized theory of uncertainty (GTU): an outline, Information Sciences 172 (1–2) (2005) 1–40.
[24] M. Turner, Microsoft Solutions Framework Essentials, Microsoft Press, Redmond, WA, USA, 2006.
[25] G. Shu, O. Rana, N. Avis, C. Dingfang, Ontology-based semantic matchmaking approach, Advances in Engineering Software 38 (1) (2007) 59–67.
[26] A. Vivacqua, M. Moreno, J. de Souza, Profiling and matchmaking strategies in support of opportunistic collaboration, in: R. Meersman, T. Zahir, D. Schmidt (Eds.), On The Move to Meaningful Internet Systems 2003: CoopIS, DOA, and ODBASE, Springer, Berlin, Germany, EU, 2003, pp. 162–177.
[27] R. Coll, J. Coll, Cognitive match interface design, a base concept for guiding the development of user friendly computer application packages, Journal of Medical Systems 13 (4) (1989) 227–235.
[28] M. Ruiz, M. Díaz, F. Soler, J. Pérez, Adaptation in current e-learning systems, Computer Standards & Interfaces 30 (1–2) (2008) 62–70.
[29] M. Jaspers, T. Steen, C. van den Bos, M. Geenen, The think aloud method: a guide to user interface design, International Journal of Medical Informatics 73 (11–12) (2004) 781–795.
[30] A. Newell, H. Simon, Human Problem Solving, Prentice Hall, Englewood Cliffs, NJ, USA, 1972.
[31] I. Vessey, Cognitive fit: a theory-based analysis of the graphs versus tables literature, Decision Sciences 22 (2) (1991) 219–240.