Cognitive Models for Training Simulations

Heuvelink, A.

2009

Chapter 6

Cognitive Agent Capabilities

6.1 Introduction

In the previous chapters we modeled several cognitive capabilities in the form of components of cognitive agent models. We decided to follow a component-based approach to the modeling of cognitive agents to foster a structured, cost-effective agent-development method (Section 1.1.5). An important factor for making a component-based agent-development method cost-effective is the reuse of previously developed components. To support this, these components should be available in some repository that can be queried for relevant agent components. In addition, these existing components should be tagged with a proper description, so that they can be discovered for reuse.

In this chapter, we introduce a preliminary line of research that in the long run might support the tagging, thus querying, and thus reuse of cognitive agent components. The basic idea underlying the research is that cognitive agent components usually embed specific cognitive capabilities. From this it follows that it is useful to tag these components with a description of the cognitive agent capability they embed; advisable with a specification of the properties of that capability implementation. To support this we started the research presented here: the development of a Capability Description Framework (CaDeF) that can be used to analyze and describe the capabilities and accompanying properties of cognitive agent models.

Chapter overview

This chapter embeds a single paper (Section 6.2) in which we introduce our preliminary ideas for a Capability Description Framework (CaDeF). CaDeF has two functions, it 1)

defines a method for describing cognitive agent capabilities, and 2) provides definitions of generic, basic capabilities. The method proposes that capabilities are described by defining the properties of the entities that make up the capability, and that there exist three types of such entities: *means*, *processes on means*, and *control of processes on means*. For each of these entities *functional*, *system*, and *dynamic* properties can be defined. We demonstrate the use of this method by defining two well-known, generic cognitive agent capabilities. In addition, we demonstrate its use by describing specific implementations of these two capabilities in the existing cognitive agent BOA (Section 3.3).

Additional Remark

In the presented paper we use a semi-formal notation that combines aspects of set theory and predicate logic to clarify the ideas underlying CaDeF. This notation is only temporary; the ideas could have been formalized in another way. In order to develop a full formal framework additional research is required, at which time it also has to be decided whether an algebraic or logical approach is more suited.

Research Paper

6.2 CaDeF: Towards a Method for Describing Cognitive Agent Capabilities

Abstract

Reusing existing (parts of) cognitive agent models requires that it is specified which cognitive capabilities and properties each embeds. This is especially the case because capabilities can be realized in multiple ways, giving rise to capability variants. In this paper we introduce CaDeF, a framework to describe cognitive agent capabilities and properties. A capability is defined by the properties of its means, processes, and control of processes; a capability variant by specifying additional properties. We demonstrate the Capability Description Framework by defining two well-known, generic cognitive agent capabilities. Its applicability is shown by describing the implementation of (variants of) these capabilities in an existing agent.

This section is an unpublished paper:
Heuvelink, A., Mioch, T., and Doesburg, W. A. van. CaDeF: Towards a Method for Describing Cognitive Agent Capabilities.

6.2.1 Introduction

Our research concerns the development of cognitive agent models that, when implemented in software, can replace human role-players in training simulations and thus cut back the expenses of training. To ensure the cognitive agents do this, the costs of developing them should not be too high. This entails that it should not be so that for every new domain, task, or even scenario a new agent model has to be developed from scratch. Therefore, our research takes a component-based approach to the modeling of cognitive agents.

A prerequisite of a component-based design approach for cognitive agents is that one must be able to 1) design an agent model by specifying its capabilities and for each capability the specific properties for the task at hand, and 2) find existing components that support this specification. Unfortunately, there does not exist consensus on an approach to catalog cognitive agent capabilities and their properties, nor on the level at which these should be described. The focus of the research described in this paper is the development of a framework that can be used to analyze and describe the capabilities and accompanying properties of cognitive agent models. With our work we aim to support future component-based, cognitive agent modeling approaches that require structured, meaningful descriptions of components.

We start this paper with a short discussion on decomposing cognition, which is followed by an introduction of the ideas underlying the proposed Capability Description Framework (CaDeF). First, we elaborate on CaDeF's capability definition. Then, we explain this definition further by means of two example capabilities. Next, we illustrate the applicability of CaDeF to analyze and describe capabilities of existing cognitive agents. We conclude this paper with a short discussion about the proposed framework and future research plans.

6.2.2 Related Work

Since the advent of (cognitive) agents it has been a recurring topic which capabilities a cognitive agent can, and should, possess (Franklin and Graesser, 1997). One reason for this discussion is that when it is understood which capabilities are critical for the modeling of behavior in agents, cognitive architectures can be built that support these capabilities.

A basic assumption underlying cognitive architectures as well as component-based cognitive agent designs is that human cognition is modular. Much research within psychology, artificial intelligence (AI), and neuroscience subscribes to this modularity of mind (Bryson, 2005). Fodor (1983) makes a distinction between *horizontal* vs. *verti-*

cal modules that both can be used to decompose cognition. *Horizontal* decompositions are those which identify processes which underlie all of cognition, such as memory, attention, perception, and judgment. Such a decomposition is embedded in cognitive architectures. *Vertical* decompositions identify particular skills, such as mathematics and language, that each have their own characteristic processes of memory, attention, etc. AI research that develops agents for specific goals, e.g., online auctions, is likely to subscribe to the vertical decomposition when developing agent components.

The literature shows that a wide variety of agent capability classifications are possible: some are inspired by cognitive theories and capture generic, 'horizontal' capabilities (Langley et al., 2006; Gluck and Pew, 2005), others stem from practical design choices and capture more specific, 'vertical' capabilities (Fineberg, 1995; Padgham and Lambrix, 2005). The fact that agent models and models of cognition are engineered makes that they can vary widely on their level of description and the roles their internal components play. However, it can be useful to reuse any of these model parts; it should therefore be possible to label each of them with a proper description of the capability variants they embed.

6.2.3 Approach

This paper proposes a Capability Description Framework for cognitive agent models. With CaDeF we aim to be able to describe the entire range of (parts of) models that can be considered to embed a capability: from abstract, generic models to task specific, instantiated ones. In order to bring structure to this wide amount of possible capability descriptions we adopt the following approach: 1) we describe the generic abstract cognitive capabilities that can be found in cognitive agents; 2) we develop a method to extend these generic descriptions to capture specific capability variants; 3) we investigate how (variants of) capabilities relate to each other.

At this moment we take as the basis for the generic cognitive capabilities to be described by CaDeF the work of Langley et al. (2006) that examines the capabilities that a cognitive architecture can support. They divide these capabilities into nine main areas, see Table 6.1.

Generic capability descriptions should be extendable to capture specific capability variants. We subscribe to the existence of two variant types: horizontal variants are formed by amending a generic description to capture new capability properties that hold for the entire agent; vertical variants are specifications of a generic capability whose added properties only hold for a task-specific module.

In order to model a new agent using CaDeF or to describe an existing agent, a functional analysis of the cognitive agent should be done to identify the capabilities of that

Table 6.1: Capabilities of Cognitive Agents, from Langley et al. (2006)

1.	Recognition and Categorization
2.	Decision-Making and Choice
3.	Perception and Situation Assessment
4.	Prediction and Monitoring
5.	Problem Solving and Planning
6.	Reasoning and Belief Maintenance
7.	Execution and Action
8.	Interaction and Communication
9.	Remembering, Reflection and Learning

agent. When a certain part of an agent plays an identifiable functional role (e.g. choosing between various goals) that part embeds a specific agent capability. In order to describe that capability not all the details of the part need to be recorded as properties of the capability, but only those aspects that are typical for its functioning. CaDeF functions as a means to record the outcomes of such a functional analysis.

Capability Definition

We consider a capability as a part of an agent that, by itself, can generate meaningful behavior. Because our final concern is the computational modeling of behavior, it is proposed that for defining a capability one needs to define resources (i.e., capability-means), processes acting upon these resources (i.e., capability-processes) and a logic that controls the behavior of that process (i.e., a capability-control), semi-formal denoted by:

`A capability is a collection of Means, Processes operating on these Means, and a Control entity:`

$$Capability = (Means, Processes, Control)$$

As mentioned in the previous section, capability entities are identified based on their functionality: the way in which specific entities of an agent are viewed depends on the role they fulfill within a capability. For example, in the next section we explain how some entities that have the role of means within one capability fulfill the role of processes in another. By describing capabilities in this functional way, the great variety found in level of detail of the entities of capabilities does not pose any problem for its description.

We propose to define the generic capabilities, whose descriptions are part of CaDeF, by denoting the properties that must hold for their Means, the ones that must hold for the Processes, and the ones that must hold for the Control.

For the means M of generic capability Cap, properties Prop_{M} must hold:

$$Means\_of\_Capability\_has\_Properties(M, Cap, Prop_M)$$

For the process P of generic capability Cap, properties Prop_{P} must hold:

$$Process\_of\_Capability\_has\_Properties(P, Cap, Prop_P)$$

For the control C of generic capability Cap, properties Prop_{C} must hold:

$$Control\_of\_Capability\_has\_Properties(C, Cap, Prop_C)$$

Additionally, we propose three types of properties: functional properties, system properties, and dynamic properties.

Properties Prop_{X} is a collection of functional properties FP_{X}, system properties SP_{X}, and dynamic properties DP_{X}:

$$Prop_X = (FP_X, SP_X, DP_X)$$

Functional Properties describe properties the capability entity has that are relevant to its role in the capability. In CaDeF we abstract away from what properties exactly mean: we assume that it is possible to define this in a separate ontology of cognitive agent terms. In CaDeF we denote properties simply as ordered attribute value pairs: $\langle A, V \rangle$.

System Properties describe inherent properties of capability entities, for example their representation. In generic capability descriptions the values of these properties are often not specified, because they are implementation specific. The reason to include them in the capability description is because these properties will be instantiated in specific variants, in which case they are relevant to know with respect to reuse.

Dynamic Properties also describe inherent properties of capability entities, but their specific value is not given by the developer, but by the current, dynamic, circumstances. As such, the value of dynamic properties always depend on the other capability entities.

A generic capability can be specialized. Specialization of a generic capability can be done by adding new properties that hold for any task the agent might perform (i.e. horizontal specialization). Additionally specialization can be done by adding properties that only hold for certain skills or tasks of the agent (i.e. vertical specialization)

Specific capability $\text{Cap}_{\text{specific}}$ is a specification of the Generic capability $\text{Cap}_{\text{generic}}$:

$$Specific\_Variant\_Of(Cap_{specific}, Cap_{generic})$$

6.2.4 Capability Cases

In this section we describe the generic capabilities *reasoning* and *decision-making* using CaDeF. This illustrates how CaDeF defines such generic capabilities by specifying the properties that must hold for each of the three entity types introduced. At the end of this section we discuss how certain levels of capabilities can relate to each other, causing the confusion between capabilities and properties of agents as often encountered, e.g., in Langley et al. (2006).

Reasoning

We consider reasoning "a central cognitive activity that lets an agent augment its knowledge state" (Langley et al., 2006).

In the following, we propose a definition of the generic capability *reasoning* using CaDeF by specifying the properties we believe each reasoning capability minimally needs to have. Capitals denote variables (for readability often informed variable names are chosen).

For the means Knowledge of generic capability reasoning, properties $\texttt{Prop}_{\texttt{Knowledge}}$ must hold:

$Means\_of\_Capability\_has\_Properties(Knowledge, reasoning, Prop_{Knowledge})$

$Prop_{Knowledge} = (FP_{Knowledge}, SP_{Knowledge}, DP_{Knowledge})$

Functional properties $\texttt{FP}_{\texttt{Knowledge}}$ are that Knowledge is declarative and accessible:

$FP_{Knowledge} = (\langle declarative, yes\rangle, \langle accessible, yes\rangle)$

Declarativeness means that an agent in principle can access the Knowledge's content, while accessible denotes that the agent can assess its content at the current moment.

System properties $\texttt{SP}_{\texttt{Knowledge}}$ are that Knowledge has representation R:

$SP_{Knowledge} = (\langle representation, R\rangle)$

Dynamic properties $\texttt{DP}_{\texttt{Knowledge}}$ are not defined:

$DP_{Knowledge} = \emptyset$

With the given definitions we denote the minimal properties the entities that function as means for reasoning have to have. They might have additional properties, e.g. certainty, representing an agent's confidence about the knowledge.

$\text{Processes}_{\text{Cap}}$ are entities that operate on the $\text{Means}_{\text{Cap}}$. We propose the following definition for the processes of the generic reasoning capability:

For the process Rule of generic capability reasoning, properties $\text{Prop}_{\text{Rule}}$ must hold:

Process_of_Capability_has_Properties(Rule, reasoning, $\text{Prop}_{\text{Rule}}$)

$Prop_{Rule} = (FP_{Rule}, SP_{Rule}, DP_{Rule})$

Functional properties FP_{Rule} are that Rule is task specific:

$FP_{Rule} = (\langle task\_specific, yes \rangle)$

System properties SP_{Rule} are that Rule has representation R, and determinism D:

$SP_{Rule} = (\langle representation, R \rangle, \langle deterministic, D \rangle)$

Dynamic properties DP_{Rule} are that Rule has executability E:

$DP_{Rule} = (\langle executability, E \rangle)$

The executability of a process is determined at runtime and depends on the current means.

It is possible to identify many additional process properties that can be used to define specific variants, e.g., the cognitive costs of processes, their utility, or their required input. These optional properties can play an important role for the control of reasoning, as is described at the end of this section.

$\text{Control}_{\text{Cap}}$ in an entity that determines which of the $\text{Processes}_{\text{Cap}}$ may become active. We propose the following definition for the generic control of reasoning:

For the control C of generic capability reasoning, properties Prop_{C} must hold:

Control_of_Capability_has_Properties(C, reasoning, Prop_C)

$Prop_C = (FP_C, SP_C, DP_C)$

Functional properties FP_{C} are that C is not task specific and that it always considers the processes' executability:

$FP_C = (\langle task\_specific, no \rangle, \langle considers\_executability, yes \rangle)$

That the control is task unspecific does not mean that for a specific task a specific type of control might not be more suited than another. But usually it is assumed that a single structure controls all reasoning, independent of the task.

System properties SP_{C} are that C has representation R, and loop type L:

$SP_C = (\langle representation, R \rangle, \langle type\_of\_loop, L \rangle)$

The type of loop can be either *single* or *multiple*, dependent on whether the control process can only execute once while executed the reasoning capability, or several times.

Dynamic properties DP_C are that C activates N active reasoning processes:

$DP_C = (\langle number\_of\_active\_reasoning\_processes, N \rangle)$

An additional property of control might be that it takes a certain constraint into account to determine which process to execute. For example, there can be a limited number of knowledge rules that may fire, or a maximal amount of processing costs these rules may have. To facilitate this the control may have as additional property that it take properties of processes into account besides their executability. At the end of this section we discuss such a specific control variant.

Decision-Making

We consider decision-making as "the ability to select among alternatives" (Langley et al., 2006). Here we describe our CaDeF definition of decision-making succinctly, in the next section we illustrate it further by means of an example.

For the means Alternative of generic capability decision-making, properties $Prop_{Alt}$ must hold:

$Means\_of\_Capability\_has\_Properties(Alternative, decision\text{-}making, (FP_{Alt}, SP_{Alt}, DP_{Alt}))$

Functional properties FP_{Alt} are that an Alternative is declarative and accessible, and has an additional Aspect A:

$FP_{Alt} = (\langle declarativeness, true \rangle, \langle accessibility, true \rangle, \langle Aspect, A \rangle)$

System properties SP_{Alt} are that an Alternative has representation R:

$SP_{Alt} = (\langle representation, R \rangle)$

Dynamic properties DP_{Alt} are that an Alternative has an evaluation score E:

$DP_{Alt} = (\langle evaluation\_score, E \rangle)$

Each alternative must have an additional Aspect A because based on this aspect the capability determines an evaluation score for it on which it basis its final decision.

Three different decision-making processes are identified; they specify three necessary steps within decision-making. The processes are labeled *Determination-of-Options (DO)*, *Evaluation-of-Options (EO)*, and *Selection-of-Options (SO)*. None of these processes has any required dynamic properties:

$DP_{DO} = DP_{FO} = DP_{SO} = \emptyset$

However, the processes differ in their functional and system properties. The first process *Determination-of-Options* is a structure that determines which of the Alternatives are currently actual options by taking a constraint C into account. For example, if a decision has to be made about which goal to attend to, this step might determine which goals are currently active and could therefore be actually attended to.

Functional properties FP_{DO} are that DO is task specific and considers a constraint:

$$FP_{DO} = (\langle task\_specific, yes\rangle, \langle considers\_constraint, yes\rangle)$$

System properties SP_{DO} are that DO has representation R and takes constraint C into account:

$$SP_{DO} = (\langle representation, R\rangle, \langle takes\_into\_account\_constraint, C\rangle)$$

The second process *Evaluation-of-Options* is a structure that determines evaluation scores E for the Alternatives based on one or more aspects A of them. This structure can embed an *injective* or a *non injective* function, which means that an option always receives a unique or possibly shared score respectively. For example, each active goal might receive a relevancy value based on their expected output.

Functional properties FP_{EO} are that EO is task specific and considers an aspect:

$$FP_{EO} = (\langle task\_specific, yes\rangle, \langle considers\_aspect, yes\rangle)$$

System properties SP_{EO} are that EO has representation R, outputs evaluation type T, and bases its evaluation on Alternatives' aspect A:

$$SP_{EO} = (\langle representation, R\rangle, \langle evaluation\_type, T\rangle, \langle takes\_into\_account\_aspect, Aspect\rangle)$$

The third process *Selection-of-Options* is a structure that determines which of the Alternatives are selected and basis this on their evaluation score E taking a constraint into account. This is based on the assumption that decision-making always involves some kind of decision: selecting randomly is not considered decision-making. For example, this process may select the goal which has the highest relevancy.

Functional properties FP_{SO} are that SO is task specific, and considers a constraint and the evaluation scores of the Alternatives:

$$FP_{SO} = (\langle task\_specific, yes\rangle, \langle considers\_constraint, yes\rangle, \langle considers\_evaluation\_score, yes\rangle)$$

System properties SP_{SO} are that SO has representation R and takes constraint C into account:

$SP_{SO} = (\langle representation, R\rangle, \langle takes\_into\_account\_constraint, C\rangle)$

Control C of the decision-making capability determines which of the three decision-making processes may execute.

Functional properties FP_C are that C is not task specific, and that it has a fixed process order:

$FP_C = (\langle task\_specific, no\rangle, \langle fixed\_process\_order, true\rangle)$

Independent of the task, the three decision-making capability processes are executed in the same order, namely first *DO*, then *EO*, and last *SO*.

System properties SP_C are that C has representation R and operates in a single loop:

$SP_C = (\langle representation, R\rangle, \langle type\_of\_loop, single\rangle)$

To make a decision, the control loop just has to execute once.

There are no dynamic properties DP_C: $DP_C = \emptyset$

Combining Capabilities

Previously we introduced CaDeF's definition of the generic reasoning capability. We mentioned that in addition to the specified required properties additional properties might hold for specific capability variants, e.g., the control might take constraints into account for deciding which of the executable reasoning processes may fire. When there exist such a constraint, e.g., on the number of Rules that may fire, a selection has to be made among the executable ones. Interestingly, a possible selection process is the decision-making capability.

When the control of reasoning is based on decision-making, *the processes of reasoning are interpreted as means for decision-making*. For this, the properties of the reasoning processes are expanded with the properties of the decision-making means. This entails that reasoning processes should be declarative and accessible. In addition, they should have an additional (functional or dynamic) property besides their dynamic property 'executability' that can function as aspect to base the decision on, and they receive as new additional dynamic property an evaluation score.

The processes and control of decision-making are comprised within the reasoning control. This entails, e.g., that the latter now has the additional functional property that it takes a constraint, and a processes' aspect into account.

With this section we want to stress that different capabilities can be embedded in each other and that the properties of the capability levels depend on each other. For example, when reasoning embeds decision-making, the reasoning processes should be specified declaratively and have an additional property that can be used to base a decision on.

6.2.5 Applying CaDeF to a Pre-Existing Agent

In this section, CaDeF is used to describe two capabilities of a previously developed cognitive agent named BOA (Both and Heuvelink, 2007), which is implemented in ACT-R (Anderson and Lebiere, 1998). BOA is developed to perform the so-called tactical picture compilation task, which consists of gathering and integrating information from a screen about radar contacts and classifying these contacts. In addition to the information from the screen, BOA can send a helicopter to gain visual information about contacts. In the following, we describe the horizontal variant of the reasoning capability embedded in BOA and a vertical decision-making variant.

Reasoning

The major part of the task execution by BOA consists of reasoning about its beliefs using its domain knowledge in order to deduce new beliefs. For BOA, the $\text{Means}_{\text{reasoning}}$ are the beliefs the agent holds. These beliefs are implemented, and thus represented, in the declarative memory of ACT-R as *chunks*, and are therefore declarative. BOA is based on the belief framework developed in Heuvelink (2007), which causes all beliefs to have as additional properties a certainty level, a source label, and a time stamp.

In ACT-R, each chunk automatically receives an activation level that determines how available it is. ACT-R limits the accessibility of the means for reasoning: only the chunk with the highest activation that matches a retrieval request can be retrieved. However, for BOA it is required that certain types of belief can be retrieved independent of their activation. Because this is impossible to do within ACT-R, beliefs are not retrieved by ACT-R's retrieval buffer, but by functions in LISP, the language underlying ACT-R. As such, all of the agent's beliefs are always accessible, and the additional ACT-R property 'activation' is not used and thus not specified.

$Means\_of\_Capability\_has\_Properties(Belief, reasoning_{BOA},$
$(FP_{Belief}, SP_{Belief}, DP_{Belief})$

$FP_{Belief} = (\langle declarative, yes\rangle, \langle accessible, yes\rangle, \langle time, T\rangle, \langle source, S\rangle,$
$\langle certainty, C\rangle)$

$SP_{Belief} = (\langle representation, chunk\rangle)$

$DP_{Belief} = \emptyset$

The $\text{Processes}_{\text{reasoning}}$ operate on the beliefs that the agent holds. These processes are task-specific; for BOA, they constitute its procedural knowledge on how to reason about its beliefs using domain knowledge. In ACT-R, procedural knowledge is represented by production rules, which is also the case for BOA. ACT-R adds an utility value to its production rules, but this is not used within BOA.

$$Process\_of\_Capability\_has\_Properties(Rule, reasoning_{BOA}, (FP_{Rule}, SP_{Rule}, DP_{Rule})$$

$$FP_{Rule} = (\langle task\_specific, yes \rangle)$$

$$SP_{Rule} = (\langle representation, production\text{-}rule \rangle, \langle deterministic, true \rangle)$$

$$DP_{Rule} = (\langle executability, E \rangle)$$

The $\text{Control}_{\text{reasoning}}$ determines which rules operate on which beliefs. In ACT-R, the control of reasoning takes one clear constraint into account; a maximum of one production rule may fire at a time, which is therefore also the case for BOA.

To determine which process may execute, the antecedent of a production rule is matched with the contents of ACT-R's buffers to determine the rule's executability. In principle, it is possible that the antecedents of multiple rules match, in which case ACT-R uses a conflict-resolution mechanism to decide which rule may fire. This entails that ACT-R's reasoning control is a form of decision-making. However, BOA was implemented in such a way that maximally one production rule can be activated at a time, which means that the conflict-resolution mechanism is never activated.

$$Control\_of\_Capability\_has\_Properties(C, reasoning_{BOA}, (FP_C, SP_C, DP_C)$$

$$FP_C = (\langle task\_specific, no \rangle, \langle considers\_executability, yes \rangle)$$

$$SP_C = (\langle representation, Lisp \rangle, \langle type\_of\_loop, single \rangle)$$

$$DP_C = (\langle number\_of\_active\_reasoning\_processes, 1 \rangle)$$

By choosing ACT-R for BOA's implementation some of BOA's reasoning capability properties were fixed. For example, because ACT-R only allows one production rule to fire at a time, BOA's reasoning control has as dynamic property that the number of active reasoning processes is always 1.

The specific reasoning capability variant presented here is an example of a *horizontal* variant, as the specified properties hold for any reasoning task agent BOA performs.

Decision-Making

Part of the tactical picture compilation task consists of employing an available helicopter to gather additional visual information about radar contacts. To do this in a correct way

one important decision has to be made: where to send the helicopter to. The specific vertical decision-making capability that enables this is elaborated on here.

The $\text{Means}_{\text{decision-making}}$ are alternatives from which one or several have to be chosen. For BOA, these alternatives are the current positions of the contacts on the radar screen: to one of these locations the helicopter should be sent.

$$Means\_of\_Capability\_has\_Properties(Positions, decision\text{-}making_{Heli}, (FP_{Pos}, SP_{Pos}, DP_{Pos}))$$

$$FP_{Pos} = (\langle declarativeness, true\rangle, \langle accessibility, true\rangle, \langle time, current\rangle, \langle urgency, U\rangle, \langle informativeness, I\rangle, \langle resource\_economics, R\rangle)$$

$$SP_{Pos} = (\langle representation, chunk\rangle)$$

$$DP_{Pos} = (\langle evaluation\_score, Relevancy\rangle)$$

Three $\text{Processes}_{\text{decision-making}}$ are identified within any decision-making capability. Also for this variant it holds that these processes do not have any required dynamic properties.

$$DP_{DO} = DP_{FO} = DP_{SO} = \emptyset$$

The first process, *Determination-of-Options*, determines which positions are eligible to send the helicopter to, which are positions of contacts that have not been visually identified before and not been identified as neutral or friendly.

$$FP_{DO} = (\langle task\_specific, yes\rangle, \langle considers\_constraint, yes\rangle)$$

$$SP_{DO} = (\langle representation, production\text{-}rule\rangle, \langle takes\_into\_account\_constraint, not\_identified\_visually\rangle, \langle takes\_into\_account\_constraint, no\_neutral\_id\rangle, \langle takes\_into\_account\_constraint, no\_friendly\_id\rangle)$$

The second process, *Evaluation-of-Options*, takes three aspects into account to calculate an evaluation score that represents how relevant it is to go to that location, namely a location's *urgency*, *informativeness*, and *resource economics*. The *urgency* of a location represents the time that is left to take precautionary actions in case the contact turns out to be hostile. Its *informativeness* is based on the number of contacts that can be identified visually from that location, as well as on their expected identity. The distance of a location towards the helicopter determines its *resource economics*.

$$FP_{EO} = (\langle task\_specific, yes\rangle, \langle considers\_aspect, yes\rangle)$$

$$SP_{EO} = (\langle representation, production\text{-}rule\rangle, \langle evaluation\_type, non\_injective\rangle, \langle takes\_into\_account\_aspect, urgency\rangle, \langle takes\_into\_account\_aspect, informativeness\rangle, \langle takes\_into\_account\_aspect, resourceeconomics\rangle)$$

The third process, *Selection-of-Options*, makes the actual selection and takes as constraint into account that no other location may have a higher score than the one selected.

$$FP_{SO} = (\langle task\_specific, yes\rangle, \langle considers\_constraint, yes\rangle, \langle considers\_evaluation\_score, yes\rangle)$$
$$SP_{SO} = (\langle representation, production\text{-}rule\rangle, \langle takes\_into\_account\_constraint, highest\_relevancy\rangle)$$

The $\text{Control}_{\text{decision-making}}$ simply executes the three processes in a single loop in the fixed order, by which it is decided where to send the helicopter to.

This specific decision-making capability variant presented here is an example of a *vertical* variant, as the specified properties do not hold for all, but only for a specific decision-making task of the agent.

6.2.6 Discussion and Conclusion

In this paper we presented a new approach for defining cognitive agent capabilities: by specifying the required properties for the capability entity types means, processes and control. With this approach it is possible to specify *generic* capabilities, i.e., processes that are considered horizontal modules by Fodor (1983), as well as *specific* variants, independent of whether these constitute a vertical or a horizontal module.

By means of examples we have shown how CaDeF can be applied to describe both conceptual and implemented capabilities. However, the reasoning of BOA is not very complex. In the future we will apply CaDeF's generic reasoning capability definition to other agents to evaluate whether it is also suitable to capture more complex reasoning variants. The treated examples do show that the framework offers room to vary the level of abstraction between entities, properties and thus capabilities. Entities can range from simple items to complex structures embedding processes and control themselves. This feature of the framework explicitly acknowledges the nature of agents and models of cognition where levels of description are not fixed and where capabilities come in many variants.

So, although CaDeF structures the description of cognitive agents through mandatory entities for describing capabilities, it provides enough flexibility to describe all types and variants of capabilities. Our aim is to start using these descriptions to catalog (components of) agents that possess these capabilities. Such a catalog is desired because it enables an agent designer to draw upon a pool of existing and implemented capabilities while designing a new agent. When this is possible, the development costs of new cognitive agent models are reduced, which in turn will lead to a more cost-effective construction of training simulations that embed cognitive agents.

In future work, the effectiveness of this approach will be further evaluated. In addition, its possible role in selecting an appropriate cognitive architecture for the development of an agent will be examined. Because different architectures provide more or less support for certain properties, our framework might aid to identify which architecture is most suited for a certain cognitive agent design.

Acknowledgments

This research has been supported by the research program 'Cognitive Modeling' (V524), funded by the Netherlands Defense Organization.