A Language for the Execution of Graded BDI Agents

Ana Casali[1], Lluís Godo[2] and Carles Sierra[2]

[1] Depto. de Sistemas e Informática
Facultad de Cs. Exactas, Ingeniería y Agrimensura
Universidad Nacional de Rosario
Av Pellegrini 250, 2000 Rosario, Argentina.

[2] Institut d'Investigació en Intel·ligència Artificial (IIIA) - CSIC
Campus Universitat Autònoma de Barcelona s/n
08193 Bellaterra, Catalunya, España.

Abstract. In this paper we present a calculus for the execution of Multi-context system with its corresponding semantics. This calculus is general enough to support different kinds of MCSs and particularly, we show how a graded BDI agent can be mapped to the language proposed. The graded BDI agent model is based on multi-context systems and is able to deal with graded mental attitudes. Taking advantage of the calculus presented we give operational semantics to this agent model.

1 Introduction

We are interested in the specification and deployment of multi-agent systems, and particularly we focus on the execution of intentional model of agents.

We consider that making the BDI architecture more flexible will allow us to design and develop agents potentially capable to have a better performance in uncertain and dynamic environments. Along this research line we have proposed a general model for Graded BDI Agents, specifying an architecture able to deal with the environment uncertainty and with graded mental attitudes. The logical framework of this model has been presented in [3] and how this model can be used to specify an architecture for a Travel Assistant Agent can be seen in [4].

The graded BDI model of agents (g-BDI) is based on multi-context systems. These systems are basically deductive machines. In this work we want to introduce another specification that allows us to define the operational semantics of this agent model. The semantics for a g-BDI model of agent will describe how a valid agent model is interpreted as sequences of computational steps. Operational semantics may define an abstract machine and give meaning to phrases by describing the transitions they induce on states of the machine. Alternatively, with different process calculus, operational semantics can be defined via syntactic transformations on phrases of the language itself. For our purpose we decided to follow this second approach.

Since the g-BDI agent model is formalized using multi-context systems (MCS), we first introduce a specific ambient calculus, which we call Multi-context Calculus (MCC), with its corresponding semantics. The calculus presented is general enough to support the execution of different kinds of MCSs and particularly, we show how a graded BDI agent can be mapped to it.

This paper is organized as follows: in Section 2, we introduce the graded BDI model of agent based in multi-context systems. Section 3 outlines some process calculus related to multiagent systems. In Section 4, we present the Multi-context calculus (MCC) and in next Section 5, we give this calculus semantics. The mapping from g-BDI agents to MCC is presented in Section 6 and finally, in Section 7 some conclusions are outlined.

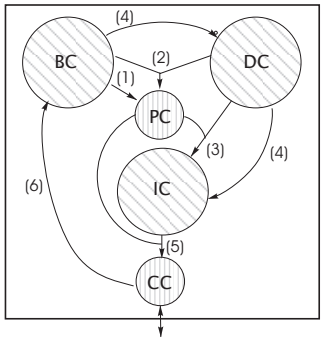
2 Graded BDI agent model

The graded BDI model of agent (g-BDI) allows to specify agent architectures able to deal with the environment uncertainty and with graded mental attitudes. In this sense, belief degrees represent to what extent the agent believes a formula is true. Degrees of positive or negative desire allow the agent to set different levels of preference or rejection respectively. Intention degrees give also a preference measure but, in this case, modeling the cost/benefit trade off of reaching an agent's goal. Thus, a higher intention degree towards a goal means that the benefit of reaching it is high, or the cost is low. Then, Agents having different kinds of behavior can be modeled on the basis of the representation and interaction of these three attitudes.

The specification of the g-BDI agent model is based on Multi-context systems (MCS). Multi-context systems were introduced by Giunchiglia et.al. [7] to allow different formal (logic) components to be defined and interrelated. The MCS specification contains two basic components: units or contexts and bridge rules, which channel the propagation of consequences among theories. Thus, a MCS is defined as a group of interconnected units: $\langle \{C_i\}_{i\in I}, \Delta_{br} \rangle$, where each context $C_i \in \{C_i\}_{i\in I}$ is the tuple $C_i = \langle L_i, A_i, \Delta_i \rangle$ where L_i, A_i and Δ_i are the language, axioms, and inference rules respectively. When a theory $T_i \subseteq L_i$ is associated with each unit, the specification of a particular MCS is complete. The deduction mechanism of these systems is based on two kinds of inference rules, internal rules Δ_i, and bridge rules Δ_{br}, which allow to embed formulae into a context whenever the conditions of the bridge rule are satisfied.

In the g-BDI agent model, we have *mental* contexts to represent beliefs (BC), desires (DC) and intentions (IC). We also consider two *functional* contexts: for Planning (PC) and Communication (CC). Thus, the g-BDI agent model is defined as the MCS: $A_g = (\{BC, DC, IC, PC, CC\}, \Delta_{br})$.

The overall behavior of the system will depend of the logic representation of each intentional notion in the different contexts and the bridge rules. Figure 1 illustrates the g-BDI agent proposed in [3] and shows one of the bridge rules included in the agent model.



Bridge rule (3)

$$\frac{DC : (D^{+}\varphi, d), PC : plan(\varphi, \alpha, P, A, c, r)}{IC : (I\varphi, f(d, c, r))}$$

Fig. 1. Multi-context model of a graded BDI agent and a bridge rule example.

In order to represent and reason about graded notions of beliefs, desires and intentions, we use a modal many-valued approach [6] where uncertainty reasoning is dealt with by defining suitable modal theories over suitable many-valued logics. For instance, let us consider a Belief context where belief degrees are to be modeled as probabilities. Then, for each classical formula φ, we consider a modal formula $B\varphi$ which is interpreted as "φ is probable". This modal formula $B\varphi$ is then a *fuzzy* formula which may be more or less true, depending on the probability of φ. In particular, we can take as truth-value of $B\varphi$ precisely the probability of φ. Moreover, using a many-valued logic, we can express the governing axioms of probability theory as logical axioms involving modal formulae. Then, the many-valued logic machinery can be used to reason about the modal formulae $B\varphi$, which faithfully respect the uncertainty model chosen to represent the degrees of belief. To set up an adequate axiomatization for our belief context logic we need to combine axioms for the crisp formulae, axioms of Łukasiewicz logic for modal formulae, and additional axioms for B-modal formulae according to the probabilistic semantics of the B operator. The same many-valued logic approach is used to represent and reason under graded attitudes in the other mental contexts. The formalization of the adequate logics for the different contexts is described in [3].

3 Process Calculus

The process calculus approach has been used to cope with formal aspects of multi-agent interactions. As for example, we present some of these calculus below.

The π-calculus is a process calculus developed by Milner et al. [8] as a continuation of the body of work on CCS (Calculus of Communicating Systems) [9]. The aim of the π-calculus is to be able to describe concurrent computations whose configuration may change during the computation. The Ambient Calculus due to Cardelli et al. [2] was developed as a way to describe the movement of processes (agents) and devices, including movement through boundaries (administrative domains). It can be seen as an extension of the π-calculus and it

is presented in more detail in next Subsection 3.1. The Lightweight Coordination Calculus (LCC) [10] can be also considered as a variant of the π-calculus with asynchronous semantics to coordinate processes that may individually be in different environments. LCC was designed specifically to formalize agent protocols for coordination and it is suitable to express interactions within multi-agent systems without any central control. It also makes it possible to verify the protocols using automated means, e.g. model checking [13]. Walton in [11] presents a language based in CCS [9] to specify agent protocols in a flexible manner during the interaction of agents. Then, in [12] he proposes a Multi-agent Protocol (MAP) oriented to agent dialogues. These protocols allow to separate agent dialogue from a specific agent reasoning technology. Ambient LCC [14] is a language based on process algebra concepts that combines the notions of LCC and Ambient calculus. It was specially designed to support the execution of electronic institutions, an organization model for Multi-Agent Systems.

In order to give a g-BDI model of agent semantics, we take advantage of process calculus. As in Ambient LCC we combine Ambient calculus with some LCC elements but in this case, for dealing with the execution of intentional agents. We focus on the work about Ambient Calculus [2] to capture the notion of bounded ambient and we take into account some elements of LCC syntax [10] to represent the state components (e.g. terms).

3.1 Mobile Ambient Calculus

Ambient calculus was developed as a way to express mobile computation [2]. The inspiration behind Ambient calculus is the observation that many aspects of mobility involve administrative considerations. For example, the authorization to enter or exit a domain, and the permission to execute code in a particular domain. These issues were principally motivated by the needs of mobile devices. However, they are very similar to the issues faced by agents in an open environment. The Ambient calculus addresses this problem by defining an ambient (informally) as a "bounded space where computation happens". The existence of a boundary determines what is inside and outside the ambient. Process mobility is represented as crossing of boundaries and security is represented as the ability or inability to cross them. In turn, interaction between processes is by shared locations within a common boundary. Ambients can also be nested, leading to a determined hierarchy. An ambient is also something that can be moved. For example, to represent a computer or agent moving from one place to another.

More precisely, each ambient has a name, a collection of local processes that run directly within the ambient, and a collection of sub-ambients. The syntactic categories are processes and capabilities. A process is analogous to an individual agent. A process may be placed inside an ambient, may be replicated, and may be composed in parallel with another process, which means that the processes execute together. In Ambient calculus, $n[P]$ denotes an ambient named n containing the process P. The syntax of Ambient calculus is shown in Table 1.

In general, an ambient exhibits a tree structure induced by the nesting of ambient brackets. Each node of this tree structure may contain a collection of

P, Q, R	::=	0	Inactivity
		$(\nu n)\,.P$	Restriction
		$P \mid Q$	Parallel Composition
		$M\,[P]$	Ambient
		$!P$	Replication
		$M.P$	Capability Action
M	::=	n	Name
		in M	can enter into M
		out M	can exit out of M
		open M	can open M
		ϵ	null
		$M.M'$	composite

Table 1. Syntax of Ambient calculus

(non-ambient) processes running in parallel, in addition to subambients. We say that these processes are running in the ambient, in contrast to the ones running in subambients. The general shape of an ambient is, therefore:

$$n\,[P_1 \mid \cdots \mid P_k \mid m_1\,[...] \mid \cdots \mid m_r\,[...]]$$

One of the relevant characteristics of the Ambient calculus is the definition of capabilities M for processes, which are described by actions. These capabilities permit things to happen within ambients. Especially, this calculus presents some actions related to crossing or opening ambient boundaries. Thus, different capabilities are defined for: entering an ambient (*in* m), exiting an ambient (*out* m) and opening an ambient (*open* m). For further information on the formal definition of Ambient calculus the reader is referred to [2].

After these considerations, we find that the notion of ambient is also appropriate to represent contexts in Multi-context systems. Contexts encapsulate the local aspects of particular logical deductions in a global system and bridge rules enable to represent the interaction or compatibility between them. Then, each unit can be mapped to an adequate ambient having a state and a process running in it. Moreover, bridge rules may be also represented by special ambients whose mobile processes may be in charge of the inter-context deduction.

4 Multi-context Calculus

Multi-context systems (MCS) are specifications of deductive machines that modify the internal states of the different contexts through the context inner deductions and bridge rules [5]. In order to translate these MCS specifications into computable languages, we propose a Multi-context calculus (MCC) based on Ambient calculus. The notion of ambient allows us to encapsulate the states and processes of the different contexts and bridge rules in the MCS. The possibility of structuring ambients hierarchically enables us to represent complex contexts where different components may be represented by different ambients.

We also take advantage of the process mobility addressed in Ambient calculus to represent the process attached to a bridge rule. This process is meant to supervise a number of context ambients to verify if particular formulae are satisfied and if that is the case, to add a formula in another context ambient. Thus, this process will be getting in and out of the different ambients. Our definition of the actions for entering and exiting an ambient (i.e. *in* C and *out* C) is slightly different from the one used in Ambient calculus. In Ambient calculus a process gets into or out of an ambient C with the ambient enclosing it. In MCC calculus we want the process to move alone, then we redefine these capabilities as follows:

$$m\,[in\ n.P \parallel n\,[Q]] \ \rightarrow\ m\,[n\,[P \parallel Q]]$$

$$m\,[n\,[out\ n.P \parallel R]] \ \rightarrow\ m\,[P \parallel n\,[R]]$$

Furthermore, for defining our calculus we use some elements of the LCC [10] as the concept of structure terms to constitute the ambient state. In LCC terms are used to specify constraints that restrict the interchange of messages and to represent some postconditions after the message sending. In our calculus, the ambient state formulae determine the results of the execution of the context ambient process (inner-context deduction) and also can trigger some bridge rule processes (inter-context deduction).

The conceptual background of MCC is the global multi-context ambient structure MCS, having an identifier and a *Clause* inside it. This clause may generate a set of clauses (ambients) for representing contexts and bridge rules. A context ambient has an identifier, a state and the context process being executed in it. Moreover, a context ambient may have other context ambients inside it composing a nested structure. Besides, a bridge rule ambient has an internal state to record substitutions and a special process representing the inter-context deduction, attached to it. The ambient structure for such a MCS where the brackets are displayed as boxes, is illustrated in Figure 2.

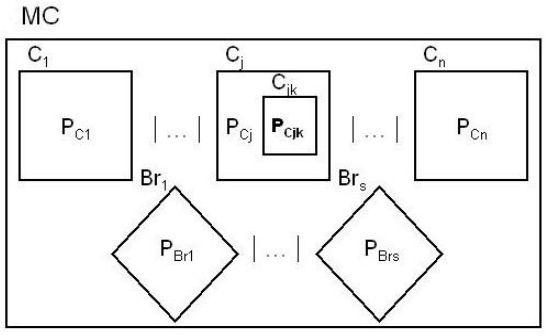


Fig. 2. The general ambient structure for a MCS

The definition of the MCC syntax is shown in Table 2. In the following items we describe the principal syntax categories in the definition.

$$
\begin{array}{lll}
MCS & ::= Id_{MC}\ [Clause] & (1)\\
Clause & ::= (Clause_c \parallel Clause) \mid (Clause_{br} \parallel Clause) & (2)\\
Clause_c & ::= C\ [P_c \parallel Clause_c] \parallel Clause_c \mid \epsilon & (3)\\
Clause_{br} & ::= Br\ [P_{br}] & (4)\\
C & ::= c(Id_c,\ S_c) & (5)\\
Br & ::= br(Id_{br},\ S_{br}) & (6)\\
S_c & ::= \{Term\} & (7)\\
S_{br} & ::= L & (8)\\
L & ::= \langle U \rangle & (9)\\
U & ::= \langle V \rightarrow Term \rangle & (10)\\
V & ::= variable & (11)\\
 & & \\
P_c & ::= Clause_c \mid \vdash_c \mid P_c \cdot P_c \mid P_c\ or\ P_c \mid & \\
 & \quad if\ Term\ then\ P_c\ else\ P_c \mid Action & (12)\\
 & & \\
Action & ::= in\ C \mid out\ C \mid get^*(Term, L) \mid getS(L_1, ..., L_n, L) \mid & \\
 & \quad add^*(L, Term) \mid remove(C, Term) \mid \epsilon & (13)\\
 & & \\
P_{br} & ::= Clause_{br} \mid (spy(Br, C_1, \varphi_1, L_1) \parallel spy(Br, C_2, \varphi_2, L_2) \parallel \ldots & \\
 & \quad \parallel spy(Br, C_n, \varphi_n, L_n)) \cdot put^*(Br, C_k, \varphi_k, L_1, ..., L_n) & (14)
\end{array}
$$

$$
spy(Br, C, Term, L) ::= out\ Br \cdot in\, C \cdot get^*(Term, L) \cdot out\ C \cdot in\ Br \quad (15)
$$

$$
\begin{array}{ll}
put^*(Br, C, Term, L_1, ..., L_n) ::= & out\ Br \cdot in\ C \cdot getS(L_1, ..., L_n, L) \cdot \\
 & add^*(L, Term) \cdot\ revise(C) \cdot\ out\ C \cdot in\ Br \quad (16)
\end{array}
$$

Table 2. Syntax of Multi-context calculus (MCC)

Multicontext System (MCS): is defined by an ambient structure where the global ambient identifier is Id_{MC} and $Clause$ will result in the ambients and processes inside it (see (1) in Table 2). $Clause$ leads us to a set of two type of clauses: $Clause_c$ and $Clause_{br}$ (2). $Clause_c$ generates a context ambient structure (possibly nested) with a context process P_c running in each ambient C (3). Respectively, $Clause_{br}$ becomes a bridge rule ambient Br (4) where a P_{br} process is being executed. In this way, we define a global ambient where different processes (P_c and P_{br} types) are running in parallel. As the processes are being executed in different ambients there is no possible interaction (e.g. concurrency problems) between them.

Context ambient: this ambient has a context process running in it. The context ambient C is defined as $c(Id_c, S_c)$ where Id_c is its identifier and S_c its state (5). In turn the state S_c is a set of $Terms$ of an adequate language $\mathcal{L}^c$

(e.g. Prolog formulae) that represents the valid formulae in the context (7). In many cases it may be useful to use a nested structure of context ambients. As for example, to represent a complex context where its language or deduction system are built using different layers. In a nested structure of ambients we can deal with this complexity defining different ambients for each layer. In the MCC syntax it is possible to represent a context ambient structure. From $Clause_c$ we can generate parallel context ambients (at the same level of hierarchy) or embedded context ambients, by using the rewriting rule (3).

Context process: consists of a deductive operator $\vdash_c$ corresponding to the context logical deduction. The P_c may be composed using the basic operators: sequential processing (.), deterministic choice (*or*) and the classical conditional $if\ then\ else$. Furthermore, rewriting P_c as $Clause_c$ the recursion of processes is allowed. Then, different kinds of programs may be represented by P_c (12).

Bridge rule ambient: this ambient has a special process P_{br} running in it (4). These ambients are defined as $br(Id_{br}, S_{br})$, having an identifier Id_{br} and a state S_{br} (6). The state for a Br ambient is a kind of substitution memory L composed by the substitution lists returned by the P_{br} process (8).

Bridge rule process: this process is a key characteristic in the MCC and represents the inter-context deduction process of a certain bridge rule (14). Each P_{br} is composed by a finite set of parallel $spy(br, C, Term, L)$ processes followed by a $put^*(Br, C, Term, L_1, ..., L_n)$ process. In the following items we describe in some detail these important components:

- $spy(Br, C, Term, L)$ process (15) gets out of the Br ambient and gets into the C ambient. In this ambient it retrieves in L all the substitution lists that result of unifying $Term$ with formulae in the context state. This task is done by the process $get^*(Term, L)$, which is the heart of the *spy* process. Then, it returns to the Br ambient.

- $put^*(Br, C, Term, L_1, ..., L_n)$ process (16) is executed after all the lists of substitutions L_1,...,L_n have been extracted by the different processes $spy(Br, C_i, Term_i, L_i)$, $i = 1, ..., n$. This process gets out of the Br ambient, comes into the C ambient and using the $getS(L_1, ..., L_n, L)$ process, retrieves in L all the substitutions compatible with the lists of substitutions L_1,...,L_n. Then, using the $add^*(L, Term)$ process, adds all the instances of $Term$ applying the resulting substitutions in L. In order to maintain the consistency in the ambient state, as the $add^*(L, Term)$ process may introduce new formulae in it, a $revise(C)$ process is needed.

- $revise(C)$ process is defined according to a suitable revision method chosen to keep the ambient state consistent. If we want to revise using time considerations as for example, allowing in the state to retain the

more recent formulae respect to the conflicting ones, the insertion time t of a formulae in an ambient state, must be included in the calculus. In our case that means that the context ambient state S_c may be redefined as $S_c =:: \{(Term, t)\}$, where the parameter t will be only used by the revise process. Since in some revision processes we may need to remove formulae from the state, we include the $remove(C, Term)$ as a possible action.

5 Operational Semantics

One of the purposes of defining the MCC is to provide the Multi-context computational model with a clean and unambiguous semantics, allowing to be interpreted in a consistent way. There are different methods for giving semantics to a process calculus as for example, defining structural congruence between processes and reduction relations [2], or using rewriting rules for the clause expansion [10]. We have chosen the *natural semantics* to provide operational semantics for the MCC. This formalism is so called because the evaluation rules are in some way similar to natural deduction and it has been used to specify the semantics of Multi-Agent Protocols (MAP) [12]. In natural deduction we define relations between the initial and final states of program fragments. Thus, we found it suitable for our case since the different processes may change the ambient states. A program fragment in our model is either a context process P_c or a bridge rule process P_{br}.

We define the evaluation rules for the different processes. The general form of these rules is: $M, a \ \diamond P \Rightarrow \quad M'$, where M is the MCS at the start of the evaluation, a is the ambient (C or Br type) where the procedure P is executed and M' is the final global system.

I- Evaluation rules for context processes: $M, C \ \diamond \ P_c \Rightarrow \ M, C'$

Since each context process P_c runs in a particular context C of M and its execution only changes its state, in the following evaluation rules we can omit the reference to M. As the context ambient C is defined as $c(Id_c, S_c)$, we represent as C' the modification of its ambient state i.e. $C' = c(Id_c, S'_c)$.

$$\frac{C \diamond P_{c1} \Rightarrow \ C' \quad C' \diamond P_{c2} \Rightarrow \ C''}{C \diamond P_{c1} \cdot P_{c2} \Rightarrow \ C''} \tag{1}$$

$$\frac{C \diamond P_{c1} \Rightarrow \ C'}{C \diamond P_{c1} \ or \ P_{c2} \Rightarrow \ C'} \tag{2}$$

$$\frac{C \diamond P_{c1} \Rightarrow \ C \quad C \diamond P_{c2} \Rightarrow \ C''}{C \diamond P_{c1} \ or \ P_{c2} \Rightarrow \ C''} \tag{3}$$

$$\frac{\begin{array}{c} C \vdash Term \\ C \diamond P_{c1} \Rightarrow\ C' \end{array}}{C \diamond\ if\ Term\ then\ P_{c1}\ else\ P_{c2} \Rightarrow\ C'} \tag{4}$$

$$\frac{\begin{array}{c} C \nvdash Term \\ C \diamond P_{c2} \Rightarrow\ C'' \end{array}}{C \diamond\ if\ Term\ then\ P_{c1}\ else\ P_{c2} \Rightarrow\ C''} \tag{5}$$

II- Evaluation rule for bridge rule process: $M, Br\ \diamond\ P_{br} \Rightarrow\ M'$

As the fundamental processes for the P_{br} definition are the processes $get^*(Term, L)$, $getS(L_1, ..., L_n, L)$ and $add^*(L, Term)$, to define their semantics it is enough to have the P_{br} semantics well defined. In some rules we use $\emptyset$ to denote that the result of the process execution is independent of the ambient where it is running.

$$\frac{\forall\ Term_i\ \left\{ \left\{ \begin{array}{l} C \vdash Term_i \\ unify(Term, Term_i)\ =\ U_i \end{array} \right\} \leftrightarrow member(U_i, L') \right\}}{\emptyset\ \diamond\ get^*(Term, L) \Rightarrow\ L = L'} \tag{6}$$

$$\frac{\begin{array}{l} \forall\ (U_1 \in L_1, \ldots, U_n \in L_n) \\ \{(unify^*(U_1, \ldots, U_n)\ =\ L^*) \leftrightarrow member(L^*, L')\} \end{array}}{\emptyset\ \diamond\ getS(L_1, \ldots, L_n, L) \Rightarrow\ L = L'} \tag{7}$$

$$\frac{\begin{array}{l} C = c(Id_c, S_c) \\ \forall L_i \{\ member(L_i, L) \leftrightarrow (Term\,[L_i] \in TermSet)\} \end{array}}{C \diamond\ add^*(L, Term) \Rightarrow\ c(Id_c, S_c \cup TermSet)} \tag{8}$$

Where $unify^*(U_1, \ldots, U_n)$ is a variant of the classical $unify$ function, where lists of substitutions $(U_1, \ldots, U_n)$ instead of formulae are unified. If $unify^*$ succeeds, its result is a list L^* of the unified substitutions.

6 Mapping a g-BDI Agent to the MCC

Given a g-BDI agent defined by its multi-context specification (see Section 2): $A_g = (\{BC, DC, IC, PC, CC\}, \Delta_{br})$ we want to map it into the MCC language. Thus, we need to define a mapping $\mathcal{F} : \{A_g\} \mapsto MCC$, which maps each g-BDI agent A_g with its multi-context components (contexts and bridge rules) to the MCC language. The general insights of the mapping $\mathcal{F}$ between these two formalisms are the following:

Global ambient: the multi-context agent A_g is mapped to a global ambient $\mathcal{A}_g$ in MCC:
$\mathcal{F}:\ A_g = (\{BC, DC, IC, PC, CC\}, \Delta_{br}) \ \mapsto\ \mathcal{A}_g\,[Clause]$

Context ambient: each context $C_i \in \{BC, DC, IC, PC, CC\}$ in the agent A_g, either mental or functional, is mapped to a suitable ambient structure (possibly nested) in MCC. Ambient calculus enables us to represent nested ambient structures as $C_i\,[P_{C_i} \parallel C_{i0}\,[P_{C_{i0}} \parallel [C_{i1} \parallel ...]]\ ]$, which is very useful in order to represent complex contexts. For example, different ambients may help to individualize different layers used in the context language definition and also in the deduction processes.
$\mathcal{F}:\ C_i = \langle \mathcal{L}_i, A_i, \Delta_i, T_i \rangle \ \mapsto\ c(C_i, S_{C_i})\,[P_{C_i} \parallel C_{i0}\,[P_{C_{i0}} \parallel [C_{i1} \parallel ...]]\ ]$

- **Language:** before setting the ambient state for a context C_i, we have to define the ambient language $\mathcal{AL}^{C_i}$. Since the languages of different mental contexts in the g-BDI agent model are built using different language layers, we create the corresponding ambient hierarchical structure where the inner an ambient is, a more basic language it has. The ambient state will be composed by formulae of the top level language (i.e., ambient). This structure allows us to differentiate the language layers in different ambient states, but using the mobility of processes we can access the different formulae in them.
  $\mathcal{F}: \mathcal{L}_i \ \mapsto\ \left\{\mathcal{AL}^{C_i}, \mathcal{AL}^{C_{i0}}, ..., \mathcal{AL}^{C_{ik}}\right\}$

- **Context ambient state:** the initial ambient state S_{C_i} is composed by the translation of the theory T_i formulae into the ambient language.
  $\mathcal{F}: T_i \ \mapsto\ S_{C_i} \subset \mathcal{AL}^{C_i}$

- **Context ambient process:** the process P_{C_i} attached to a context ambient is derived from its logical deduction system. Thus, it is built from the context theory, axioms and inference rules.
  $\mathcal{F}: \langle A_i, \Delta_i, T_i \rangle \ \mapsto\ P_{C_i}$

  Essentially the P_{C_i} process is composed by the following sequential schema: $P_{C_i} \ ::=\ P^*_{A_i} \cdot P^*_{\Delta_i}$, where the $P^*_{A_i}$ process represents the generation of finitely-many instances of the different context axioms i.e. $P^*_{A_i} ::= P^*_{A_{i1}} \cdot ... \cdot P^*_{A_{in}}$, where the A_{ij}'s are axioms in A_i. Respectively, $P^*_{\Delta_i}$ is composed by processes in charge of generating the instances of the different inference rules. i.e. $P^*_{\Delta_i} ::= P^*_{\Delta_{i1}} \cdot ... \cdot P^*_{\Delta_{ik}}$, where the Δ_{ij}'s are rules in Δ_i. These processes are described in more detail for DC context in next Subsection 6.1.

Bridge rule ambient: each bridge rule Br_i is mapped to a suitable ambient Br_i having as internal state a list of possible substitutions L_i and a special process P_{Br_i}. The definition of both elements related to the Br_i ambient

(i.e. L_i and P_{Br_i}) depends on the premise and conclusion of the bridge rule that it represents:

$$\mathcal{F}: \; Br_i = \frac{C_1 : \varphi_1, \ldots, C_n : \varphi_n}{C_k : \varphi_k} \;\; \mapsto \;\; br(Br_i, L_i)\,[P_{Br_i}]$$

- **Internal state:** is the list L_i of n substitution lists, i.e.:
  $L_i = \langle \langle L_{i1} \rangle, \ldots, \langle L_{in} \rangle \rangle$ where each sublist $\langle L_{ij} \rangle$ will contain the resulting substitutions of unifying the formulae φ_j with formulae in the context C_j.

- **Bridge rule process:** the special process P_{Br} is created in MCC (see (12) in Table 2) to represent the bridge rule inference. For the bridge rule Br_i this process will add determined instances of the formula φ_k in an ambient C_k when the preconditions are satisfied.
  $P_{Br_i} ::= (spy(Br, C_1, \varphi_1, L_1) \parallel \ldots \parallel spy(Br, C_n, \varphi_2, L_n)) \cdot$
  $\cdot \; put^*(Br, C_k, \varphi_k, L_1, ..., L_n)$

The ambient structure in MCC for representing a g-BDI agent A_g is illustrated in Figure 3. Therefore, for each mental or functional context in the g-BDI

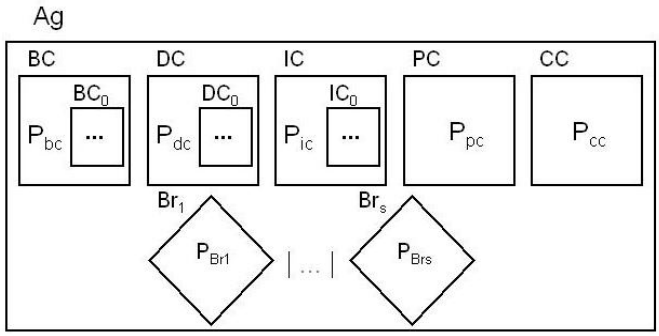


Fig. 3. The ambient structure for a g-BDI agent

agent specification, we can define the corresponding ambient structure in MCC. Since the planning and communication context are based in first order logic, the mapping is straightforward and both contexts can be easily passed to a corresponding ambient.

In the case of the mental contexts, since the logical framework is more complex, some details must be analyzed. As a matter of example, in the next subsection we describe the mapping $\mathcal{F}$ for the Desire Context. In a similar way, the ambients for the other mental contexts may be developed.

6.1 Mapping the Desire context (DC) into a Desire ambient

We next define a mapping $\mathcal{F}$ from a desire context DC to a suitable ambient structure in MCC

$$\mathcal{F} : DC = \langle \mathcal{L}_{DC}, A_{DC}, \Delta_{DC}, T_{DC} \rangle \ \mapsto \ c(DC, S_{DC})\,[P_{DC} \parallel ...]$$

For this, we start with a synthesized description of the components of the DC context: the language $\mathcal{L}_{DC}$, the axioms A_{DC}, the inference rules Δ_{DC} and a theory T_{DC}. A more complete description can be found in [3].

Language ($\mathcal{L}_{DC}$): it is defined over a (classical) propositional language $\mathcal{L}$ (generated from a finite set of propositional variables and connectives $\neg$ and $\rightarrow$) by introducing two (fuzzy) modal operators D^+ and D^-. As in other mental contexts, we use a (modal) many-valued logic to formalize reasoning about graded desires by means of the trick of interpreting the (positive and negative) degrees of desires over a (classical) proposition φ as the truth-degrees of the modal formulas $D^+\varphi$ and $D^-\varphi$ respectively. We choose Łukasiewicz logic, extended with rational truth-constants, as the underlying many-valued logic dealing with the many-valued modal formulas. The $\mathcal{L}_{DC}$ language is built therefore as follows:

- If $\varphi \in \mathcal{L}$ then $D^-\varphi, D^+\varphi \in \mathcal{L}_{DC}$
- If $r \in \mathbb{Q} \cap [0,1]$ then $\overline{r} \in \mathcal{L}_{DC}$
- If $\Phi, \Psi \in DC$ then $\Phi \rightarrow_L \Psi \in \mathcal{L}_{DC}$ and $\neg_L \Phi \in \mathcal{L}_{DC}$

Axioms and inference rules (A_{DC} and Δ_{DC}): to axiomatize the logical system DC we need to combine axioms of classical propositional calculus (CPC) for formulas of $\mathcal{L}$ with Łukasiewicz logic axioms for modal formulae, plus additional axioms characterizing the behavior of the modal operators D^+ and D^-:

- Axioms of CPC for formulas of $\mathcal{L}$
- Axioms of Łukasiewicz logic modal formulae.
- Axioms[1] for D^+ and D^- over Łukasiewciz logic:
  $Ax_1 : \ D^+(A \vee B) \equiv_L D^+A \wedge_L D^+B$
  $Ax_2 : \ D^-(A \vee B) \equiv_L D^-A \wedge_L D^-B$
- Rules are: modus ponens for $\rightarrow$ and $\rightarrow_L$ and introduction of D^+ and D^- for implications:
  Δ_1: from $A \rightarrow B$ derive $D^+B \rightarrow_L D^+A$ and
  Δ_2: from $A \rightarrow B$ derive $D^-B \rightarrow_L D^-A$.

Theory (T_{DC}): it consists of a set of formulas from $\mathcal{L}_{DC}$

Now we are ready to define the corresponding Desire Ambient $\mathcal{F}(DC)$ by describing its state language $\mathcal{AL}^{DC}$, its initial state S_{DC} and its process P_{DC}.

[1] These are the basic ones, one could consider additional axioms introducing some constraints between both modalities.

1. **State Language** $\mathcal{AL}^{DC}$
   Since the modal language $\mathcal{L}_{DC}$ for the desire context is built in two layers (one base propositional language $\mathcal{L}$ and the modal $\mathcal{L}_{DC}$), we define two ambients to represent these language layers. We define the ambient DC_0 (to represent language $\mathcal{L}$) inside the ambient DC (to represent language $\mathcal{L}_{DC}$), having the following ambient structure: $DC\ [P_{DC} \parallel DC_0]$
   This nested ambient structure enables us to deal in a proper way with the different fomulae in the two language layers. The language for the DC_0 ambient is the basic language used in the DC context for building the language $\mathcal{L}_{DC}$. As it is convenient for the definition of the deductive process P_{DC}, we consider that the formulas of this language are in Disjunctive Normal Form (DNF). So we have or DC_0 the mapping $\mathcal{F} : \mathcal{L} \mapsto \mathcal{AL}^{DNF}$ defined by
   - $\mathcal{F}(\psi) = \psi^{DNF}$

   The mapping from the language $\mathcal{L}_{DC}$ to the language $\mathcal{AL}^{DC}$ for the desire ambient, $\mathcal{F} : \mathcal{L}_{DC} \mapsto \mathcal{AL}^{DC}$ is then defined as follows:
   - $\mathcal{F}\ (D^+\varphi) = d^+(\mathcal{F}(\varphi))$
   - $\mathcal{F}\ (D^-\varphi) = d^-(\mathcal{F}(\varphi))$
   - $\mathcal{F}\ (\overline{r}) = r$
   - $\mathcal{F}\ (\neg_L \Phi) = neg(\mathcal{F}(\Phi))$
   - $\mathcal{F}\ (\Psi \rightarrow_L \Phi) = imp(\mathcal{F}(\Psi), \mathcal{F}(\Phi))$

2. **Initial state** S_{DC}
   The DC ambient state S_{DC} is composed by the translated formulae of the context theory: $S_{DC} = \{\mathcal{F}(\Phi) \mid \Phi \in T_{DC}\}$

3. **DC Process** P_{DC}
   We need to map the logical deduction of the desire context DC, composed by two different layers of axioms and inference rules, into the P_{DC} process. Actually, it can be shown that reasoning in the DC axiomatic system can be reduced to reasoning in plain Łukasiewicz logic from a big, but finite, theory which gathers suitable translations of instances of all the axioms and inference rules, and of the formulas of the context theory T_{DC}. We will consider deduction in Łukasiewicz logic as an encapsulated process $P_{\text{Ł}}$ without entering in its internals. This is possible since there exist theorem provers for this many-valued logic [1]. We describe next how to build such a theory in the context ambient which incorporates a finite set of instances of the axioms and inference rules that model the behavior of D^+ and D^-. The idea is that, since we have a language $\mathcal{L}$ built over a finite set of propositional variables, there are only a finitely-many different DNF formulas, so there are finitely-many instances of axioms and rules over these DNF formulas. Therefore the P_{DC} process will consist of two parts. The first one, involving four processes $P_V, P_{DNF}, P_{AX}, P_{\Delta}$, will add to the initial ambient state S_{DC} (context theory) the set of instances of the axioms and inference rules, changing the initial state into S'_{DC}:

$$c(DC, S_{DC}) \diamond\ (P_V(VSet) \cdot P_{DNF}(VSet) \cdot P_{AX} \cdot P_{\Delta}) \Rightarrow\ c(DC, S'_{DC})$$

Then, over the state S'_{DC}, the deduction over Łukasiewicz logic, represented by the process $P_{\text{Ł}}$ can be be applied. Thus, the P_{DC} process is defined as the following schema of sequential processes:

$$P_{DC} \ ::= \ P_V(VSet) \cdot P_{DNF}(VSet) \cdot P_{AX} \cdot P_{\Delta} \cdot P_{\text{Ł}}$$

In the following items we describe the four first processes:

- The $P_V(VSet)$ process extracts from S_{DC} the finite set of variables appearing in the formulas of T_{DC} and puts them in $VSet$.

- The $P_{DNF}(VSet)$ process enters in the DC_0 ambient and through the $add^*_{DNF}(VSet)$ process creates and adds to S_{DC_0} the finite set of DNF formulae built upon the variables in $Vset$, i.e.:

$$P_{DNF}(VSet) ::= in\ DC_0 \cdot add^*_{DNF}(VSet) \cdot out\ DC_0$$

- The P_{AX} process is composed by all the processes derived from each context axiom. For this particular case of the DC ambient we have:

$$P_{Ax} ::= P_{Ax_1} \cdot P_{Ax_2}$$

  These processes represent respectively the axioms Ax_1 and Ax_2 (see below), for instance P_{Ax_1} is defined as:

$$P_{Ax_1} ::= in\ DC_0 \ \cdot \ get^*(dpair(x,y), L) \cdot out\ DC_0 \ \cdot \\ \cdot\ add^*(\mathcal{F}(D^+(x) \wedge D^+(y) \equiv D^+(x \vee y)), L)$$

  where the special component processes have the following meaning:
  - $get^*(dpair(x,y), L)$ stores in L all the pairs (x,y) satisfying the condition $dpair(x,y)$: $x, y \in \ S_{DC_0}$ and $x \neq y$;
  - $add^*(\mathcal{F}(D^+(x) \wedge D^+(y) \equiv D^+(x \vee y)), L)$, using the pairs $(x,y) \in L$ for substitution, instantiates and adds to the ambient state S_{DC} the formulae $\mathcal{F}(D^+(x) \wedge D^+(y) \equiv D^+(x \vee y))$.

  In a similar way the P_{Ax2} process represents the corresponding axiom for D^-.

- The P_{Δ} process is composed of the processes representing the instances of the different inference rules. For the DC ambient there are two processes representing the rules Δ_1 and Δ_2, hence

$$P_{\Delta} ::= P_{\Delta_1} \cdot P_{\Delta_2} \ , \qquad \text{with}$$

  $P_{\Delta_1} ::= in\ DC_0 \cdot get^*(\mathcal{F}(x \rightarrow y), L) \cdot out\ DC_0 \cdot add^*(\mathcal{F}(D^+(y) \rightarrow D^+(x)), L)$

  and similarly for P_{Δ_2}, where
  - $get^*(F(x \rightarrow y), L)$ gets the pairs (x,y) resulting from the unification of $F(x \rightarrow y)$ in DC_0 ambient;
  - $add^*(\mathcal{F}(D^+(y) \rightarrow D^+(x)), L)$ adds all the instances of the formula $\mathcal{F}(D^+(y) \rightarrow D^+(x))$ with pairs $(x,y) \in L$.

- The final process $P_{\text{Ł}}$ applies Łukasiewicz logic deduction $\vdash_{\text{Ł}}$ to the state S'_{DC} resulting from the previous processes, i.e. $P_{\text{Ł}} ::= \vdash_{\text{Ł}}$

7 Conclusions

In this work we have defined a MCC calculus for Multi-context systems (MCS) execution. The MCC proposed is based on Ambient calculus [2] and includes some elements of LCC [11]. The operational semantics for this language was given using Natural Semantics. We expect that MCC will be able to specify different kinds of MCSs. Particularly, we have shown how graded BDI agents can be mapped to this calculus. Through MCC we are given to this agent model computational meaning and in this way, we are getting closer to the development of an interpreter of the g-BDI agents.

References

1. Beavers G. Automated theorem proving for Łukasiewicz logics. *Studia Logica*, Volume 52, Number 2 pp. 183-195, 1993.
2. Cardelli L. and Gordon A. D.. Mobile Ambients. In Maurice Nivat, editor, *Foundations of Software Science and Computational Structures*, number 1378 in Lecture Notes in Computer Science, pp. 140155. Springer-Verlag, 1998.
3. Casali A., Godo L. and Sierra C. Graded BDI Models For Agent Architectures. Leite J. and Torroni P. (Eds.) *CLIMA V, Lecture Notes in Artificial Intelligence LNAI 3487*, pp. 126-143, Springer-Verlag, Berling Heidelberg, 2005.
4. Casali A., Godo L. and Sierra C. Modeling Travel Assistant Agents: a graded BDI Approach. IFIP, Volume 217, *Artificial Intelligence in Theory and Practice*, Ed. Max Bramer (ISBN 0-387-34654-6) (Boston: Springer), 415-424, 2006.
5. Ghidini C. and Giunchiglia F. Local Model Semantics, or Contextual Reasoning = Locality + Compatibility *Artificial Intelligence*,127(2):221-259, 2001.
6. Godo, L., Esteva, F. and Hajek, P. Reasoning about probabilities using fuzzy logic. *Neural Network World*, 10:811–824, 2000.
7. Giunchiglia F. and Serafini L. Multilanguage Hierarchical Logics (or: How we can do without modal logics) *Journal of Artificial Intelligence*, vol.65, pp. 29-70, 1994.
8. Milner R., Parrow J. and Walker D., A calculus of mobile processes, Parts 1-2. *Information and Computation*, 100(1), 1-77. 1992.
9. Milner R., *Communication and Comcurrency*. Prentice-Hall International, 1989.
10. Robertson D. Multi-Agent Coordination as Distributed Logic Programming Proceedings of the *International Conference on Logic Programming*, Sant-Malo, 2004.
11. Walton C. and Robertson D. Flexible multi-agent protocols. *Technical Report EDI-INF-RR-0164*, University of Edinburgh, 2002.
12. Walton C. Multi-Agent Dialogue Protocols. In Proceedings of the *Eighth International Symposium on Artificial Intelligence and Mathematics*, Fort Lauderdale, Florida, January 2004.
13. Walton C. Model Checking Multi-Agent Web Services. In Proceedings *AAAI Spring Symposium on Semantic Web Services*, Stanford, California, 2004.
14. Joseph S., Perreau de Pinninck Bas, A., Robertson, D., Sierra, C., Walton, C. Interaction Model Language Definition. *IJCAI 2007 Workshop AOMS Agent Organizations Models and Simulations*. Dignum V., Dignum F., Matson E. and Edmonds B. eds., pp. 49-61, 2007.